

About the author



- Dr. Tim Menzies (tim@menzies.us) has worked on advanced modeling + AI since 1986.
 - PhD from Uni. New South Wales, Sydney, Oz
 - Assoc/prof at WVU CS &EE
- Former research chair for NASA
- Author of 190 refereed papers: <http://menzies.us/papers.php>
- Co-founder and organizer of the PROMISE conferences on repeatable experiments in SE
- For more, see <http://menzies.us>

http://promisedata.org/data

PROMISE
The 2010 International Conference on Predictive Models in Software Engineering

PROMISE data sets
... AND COUNTING

108
107
106

Home About Data Papers People PROMISE '06 PROMISE '07 PROMISE '08 PROMISE '09 PROMISE '10

Welcome to the Promise Data Repository!

DefectPrediction
EffortEstimation
General
Model-BasedSE
TextMining

In 2006, the repository held 23 data sets.
In 2008, at last update, the repository holds 100 data sets in the following areas:

- Defect Prediction (57)
- Effort Prediction (18)
- General (9)
- Model-based SE (7)
- Text Mining (9)

PROMISE
2010

The 6th International Conference on Predictive Models in Software Engineering
Co-located with ICSM'10, at Timisoara, Romania
Sept 12-13, 2010

Home | Program | Venue | Registration | Call for papers | Dates | Committees | Keynotes | Submit | Promote

Important Dates

Papers:

- Abstract Submission Deadline: May 28, 2010
- Paper Submission Deadline: June 4, 2010
- Student Symposium Submission Deadline: June 4, 2010
- Notification of Results: July 9th, 2010
- Camera Ready Copy Submission Deadline: July 23th, 2010

Registration:

- Early registration deadline: August 16, 2010

Conference:

- Main conference: September 12 and 13, 2010
- Student symposium: September 13, afternoon/evening.
(Note: symposium attendees must register for the main conference.)

Special issue:

- Invitations to submit: October 1, 2010
- Paper submission deadline: Dec 31, 2010
- Notifications of first round reviewing: March 31, 2011
- Publication: late 2011 (planned)

Hotel. PROMISE: 2005, 2006, 2007, 2008, 2009 | Contact: mail (at) promisedata.org

Repository + annual
conference.
See you there?

For other view on DM + SE

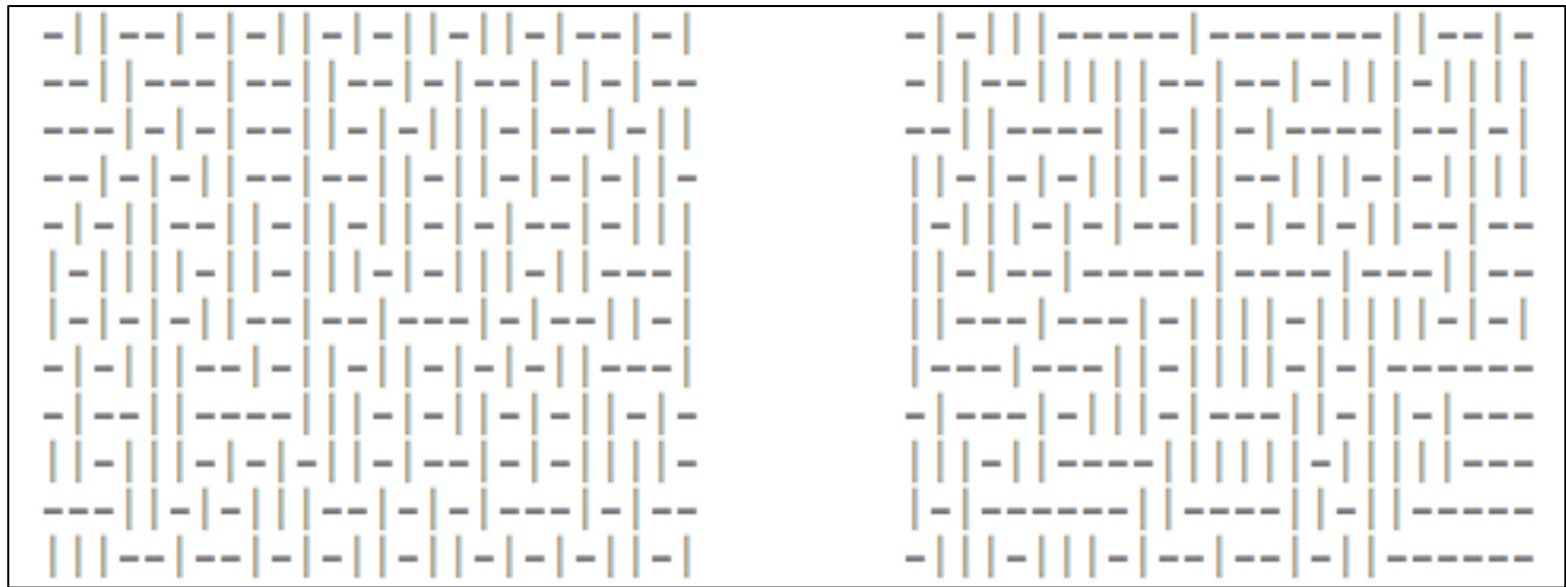
- ICSE 2010 Tutorial T18 Tuesday, 4 May 2010 (afternoon)
- Mining Software Engineering Data
 - Ahmed E. Hassan: Queen's University, Canada
 - Tao Xie: North Carolina State University, USA



- Tutorial Slides:
 - <https://sites.google.com/site/asergp/dmse/dmse-icse08-tutorial.ppt?attredirects=0>

Exercise #1

- One these these things is not like the other
 - One was generating by selecting “-” or “|” at random, 300 times.
- Which one?




Exercise #2

- A little experiment from http://www.youtube.com/v/vjG698U2Mvo&hl=en_US&fs=1&rel=0
- Rules
 - No one talks for the next 4 minutes
 - If you know what is about to happen, see (I)
- This is a selective attention test
 - Count the number of times the team with the white shirt passes the ball.



What have we learned?

- Lesson #1:
 - Algorithms can be pretty dumb
 - If they don't focus on X, they see any Y, at random.
- Lesson #2:
 - Humans can be pretty dumb
 - If they mono-focus on X, you can miss Y
- Maybe, any induction process is a guess
 - And while guessing can be useful
 - Guesses can also be wrong
- Lets us a create community of agents, each with novel insights and limitations
 - Data miners working with humans
 - Maybe in combination, we can see more that separately



Wikipedia:
List of cognitive biases
http://en.wikipedia.org/wiki/List_of_cognitive_biases

- 38 decision making biases
- 30 biases in probability
- 18 social biases,
- 10 memory biases



Applications

- Effort estimation
- Defect prediction
- Optimization of discrete systems
- Test case generation
- Fault localization
- Text mining
- Temporal sequence mining
 - Learning software processes
 - Learning APIs
- Etc
- Welcome to Empirical SE, Version 2.0

Applications

- Effort estimation
- Defect prediction
- Optimization of discrete systems
- Test case generation
- Fault localization
- Text mining
- Temporal sequence mining
 - Learning software processes
 - Learning APIs
- Etc
- Welcome to Empirical SE, Version 2.0

Data mining applications explored by me since 2007.

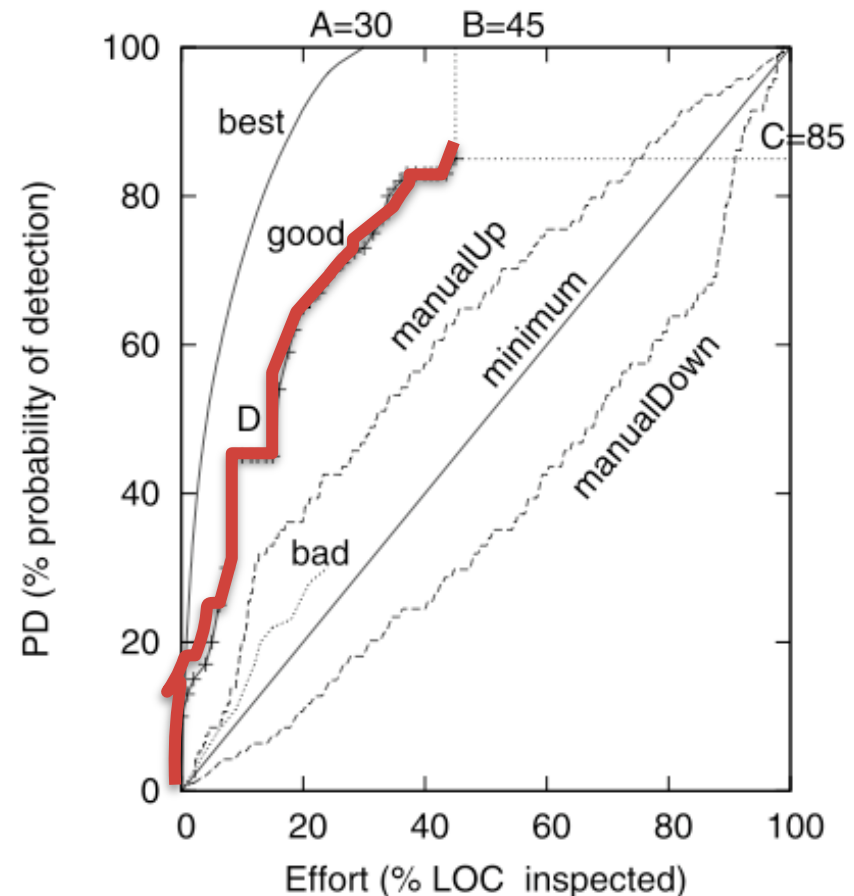
A career in data mining is a very diverse career, indeed

Application: Effort estimation

- Can we predict development effort (time * staff)?
- E.g. using linear regression; effort = $a * KLOC^b c$
 - Boehm, B. W. 1981 Software Engineering Economics
 - Boehm, B. W., Clark, Horowitz, Brown, Reifer, Chulani, Madachy, R., and Steece, B. 2000 Software Cost Estimation with Cocomo II
 - Sunita Chulani, Barry W. Boehm, Bert Steece: Bayesian Analysis of Empirical Software Engineering Cost Models IEEE Trans. Software Eng. 25(4): 573-583 (1999)
- E.g. using analogy
 - Describe past projects according to N dimensions
 - Float all known projects in an N-dimensional space
 - To estimate a project, insert into that space; query its nearest neighbors
 - For the classic estimation via analogy, see
 - Martin J. Shepperd, Chris Schofield: Estimating Software Project Effort Using Analogies IEEE Trans. Software Eng. 23(11): 736-743 (1997)
 - For 12,000+ variants to that process, see
 - Fig1 of <http://menzies.us/pdf/I0stable.pdf>
- E.g. using other methods:
 - See 154 variants in <http://menzies.us/pdf/I0stable.pdf>

Application: Defect Prediction

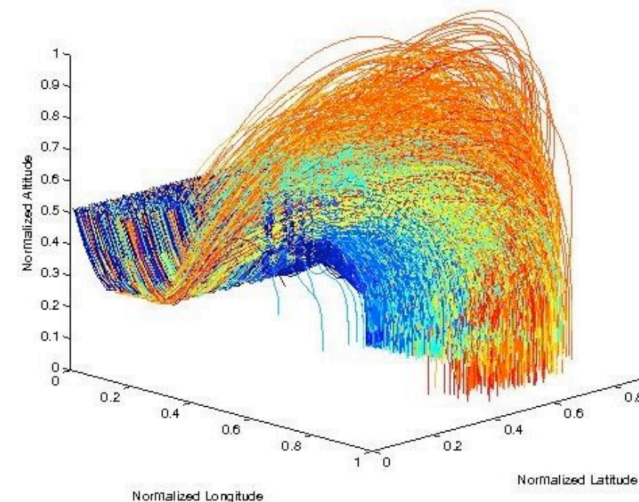
- Limited QA budgets, can't check everything.
 - Where should we place our inspection effort?
- For a review, see Section Two of
 - <http://menzies.us/pdf/I0which.pdf>
- Practical value:
 - How to inspect less, and find more bugs



Application: Optimizations of discrete systems

- Standard numeric optimizers assume continuous, possibly even linear, equations
- Data miners much happier to work in discrete spaces.
- What factors predict for landing closest to the target?
 - State-of-the-art optimizer
 - Simulated annealing
 - the TAR3 data miner
 - TAR3 45 times faster, found better solutions

<http://menzies.us/pdf/10keys.pdf>

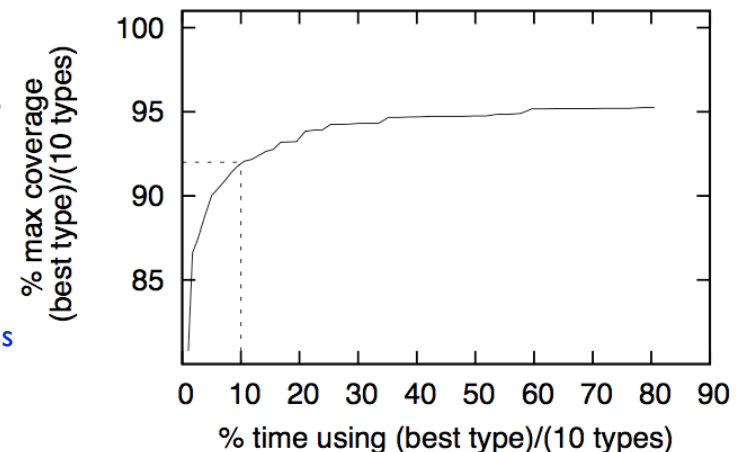


Application: Test Case Generation

- NIGHTHAWK: A genetic algorithm that mutates sequences of method calls in order to maximize code coverage.
- RELIEF: a data mining technique to find “interesting features”
 - Same attribute same values in all classes?
 - Boring
 - Same Attribute, different values in different classes?
 - Interesting
- RELIEF found that 90% of NIGHTHAWK’s mutators were “boring”
 - Order of magnitude speed up in test generation
- [James H. Andrews, Tim Menzies, Felix C.H. Li, "Genetic Algorithms for Randomized Unit Testing," IEEE Transactions on Software Engineering, 25 Mar. 2010.](#)

Rank	Gene type <i>t</i>	<i>avgMerit</i>
1	numberOfCalls	85
2	valuePoolActivityBitSet	83
3	upperBound	64
4	chanceOfTrue	50
5	methodWeight	50
6	numberOfValuePools	49
7	lowerBound	44
8	chanceOfNull	40
9	numberOfValues	40
10	candidateBitSet	34

java.util classes



Application: Fault Localization

- 100,000 JAVA methods
 - In a matrix $T \times D$
 - T = “terms” = all the method calls in each method
 - D = “documents” = all the methods
- Bug report
 - Replace text with just the method calls it mentions
 - Add edited report as row $D+one$ in the matrix
 - Compute similarity of $D+one$ to other rows (cosine similarity)
 - The actual buggy method is in the closest 100 methods
 - Use relevancy feedback to narrow down the search
- [Gregory Gay, Sonia Haiduc, Andrian Marcus Tim Menzies: On the use of relevance feedback in IR-based concept location ICSM 2009: 351-360](#)



Application: Text Mining

- 80% of data in organizations is unstructured
 - Not in databases, or XML schemas
 - But in the natural language of (say) Word documents
- Given enough of these seemingly unstructured documents, structures can be discovered
- E.g.
 - Thousands of natural language bug reports from NASA
 - Used “feature reduction” to find the top 100 most important words
 - Used standard data mining to learn predictors for defect severity from that top-100
 - [Tim Menzies, Andrian Marcus: Automated severity assessment of software defect reports. ICSM 2008: 346-355](#)



Application: Temporal Sequence Mining

- Learning software process descriptions
 - No more prescriptions of what we think goes on inside software projects
 - Lets look at see at what actually happens
 - Li, Mingshu and Boehm, Barry and Osterweil, Leon and Jensen, Chris and Scacchi, Walt “Experiences in Discovering, Modeling, and Reenacting Open Source Software Development Processes”, *Unifying the Software Process Spectrum, Lecture Notes in Computer Science*, 2006, page 449 to 462
- Learning APIs from method sequence calls
 - Tao Xie and Jian Pei. MAPO: Mining API Usages from Open Source Repositories. In *Proceedings of the 3rd International Workshop on Mining Software Repositories (MSR 2006)*, Shanghai, China, pp. 54-57, May 2006
- Learning patches from method sequence calls
 - Suresh Thummalapenta and Tao Xie. Mining exception-handling rules as sequence association rules. In *ICSE '09: Proceedings of the 31st International Conference on Software Engineering*, pages 496– 506, Washington, DC, USA, 2009. IEEE Computer Society.
- Obtaining sequence miners:
 - <https://illimine.cs.uiuc.edu/>
 - Another tool set is at <http://himalaya-tools.sourceforge.net/>
 - See more tools at <https://sites.google.com/site/asergpr/dmse/resources>



Application: etc etc etc

- Data mining + SE a very active area
 - PROMISE conference
 - Mining Software Repository conference
- See also
 - ESEM conference
 - Search-based software engineering
- Hint: to get ahead of the curve...
 - ... learn sequence mining
- Welcome to Empirical SE, version 2.0

Empirical SE, Version 2.0

- Open Science movement
 - Open Data
 - Everyone places their data on-line, all the time
 - Open Access publishing
 - Death to subscription-based services
- [Shneiderman, B. \(2008\) "Science 2.0" Science 319\(5868\):1349-50](#)
 - Science meets web 2.0
 - International team of researchers posting and analyzing data
 - Research at internet speed
- Anda, Markus et al (*) distinguish between
 - **Case studies**: that collect new context variables from project data
 - **Experiments**: that explore case study data
 - Currently, very few case studies generating publicly available data
 - But very many researchers wanting to experiment on that data
 - Perfect setting for data mining
- (*) [Bente Anda Audris Mockus and Dag I.K. Sjoberg. Experiences from replicating a case study to investigate reproducibility of software development. In First International Workshop on Replication in Empirical Software Engineering Research, ICSE'09,](#)

Q: Why Empirical SE 2.0?

A: Increasing pace of change

- New developments are radically changing SE: open source toolkits, agile development, cloud-based computing, etc.
- 20th century Empirical SE used “big science”
 - Research questions, data collection, analysis took years
 - Big science is too slow to keep up with changes to contemporary SE. e.g.
 - Increasing pace of organization change at NASA was fatal to the “big science” approach of Victor Basili’s Software Engineering Laboratory (*)
 - [V. Basili, F. McGarry, R. Pajerski, and M. Zelkowitz. Lessons learned from 25 years of process improvement: The rise and fall of the NASA software engineering laboratory. In Proceedings of the 24th International Conference on Software Engineering \(ICSE\) 2002, Orlando, Florida, 2002.](#)
- Data mining is one response to the open and urgent issue of
 - how to reason faster about SE data.

Q: Why Empirical SE 2.0?

A: Changing nature of SE theories

- 20th century SE: the struggle for the single theory
 - E.g. Boehm's COCOMO effort estimation project
 - E.g. SEI capability maturity model [130];
- 21st century: faster pace = more diversity
 - Less likely that there exists a single over-arching grand theory of SE
- Recent reports [1,2,3,4,5] say that while such generality may elude us, we can still find important local lessons
 1. Rombach A. Endres, H.D. A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories. Addison Wesley, 2003.
 2. B. Kitchenham D. Budgen, P. Brereton. Is evidence based software engineering mature enough for practice & policy? In 33rd Annual IEEE Software Engineering Workshop 2009 (SEW-33), Skvde, Sweden, 2009.
 3. B. A. Kitchenham, E. Mendes, and G. H. Travassos. Cross- vs. within-company cost estimation studies: A systematic review. IEEE Transactions on Software Engineering, pages 316–329, May 2007.
 4. Tim Menzies and Forrest Shull. The quest for convincing evidence. In A. Oram and G. Wilson, editors, Making Software: What Really Works, and Why We Believe It. O'Reilly, 2010.
 5. H. Gall E. Giger T. Zimmermann, N. Nagappan and B. Murphy. Cross-project defect prediction. In ESEC/FSE'09, August 2009.
- Data mining is one way to rapidly find and verify such local best practices

Q: Why Empirical SE 2.0

A: Changing nature of data

- In the 21st century
 - we can access more data collected by others than we can ever collect by ourselves.
- In the 20th century,
 - research was focused on case studies where researchers collected special purpose data sets for their particular questions.
- In the 21st century,
 - much research is devoted to experimentation with the data generated by the case studies,
 - possibly investigating hypotheses not originally considered when the data was collected.
 - Data mining is one way to experiment with data.



Q: Why Empirical SE 2.0?

A: Changing nature of data analysis

- A contemporary empirical SE paper might explore gigabytes of core dumps looking for the method calls that lead to a crash.
- Faced with such large and complex data, analysis methods are becoming more intricate; e.g.
 - Model trees for multi-model data
 - Latent Dirichlet allocation (LDA) for document clustering
 - Mining sequences to learn exception handling rules
- It is now possible to find new insights in old data, just by applying a new analysis method.
 - E.g. see later, the “W” tool



Why Data Mining for SE?

- Natural tool to help a community:
 - racing to keep up with the pace of change in SE;
 - while finding and verifying local theories ...
 - ... from a new kind data sources ...
 - ... using a large menagerie of new data analysis tools.

Empirical Science 2.0 adjusts its questions to the available data

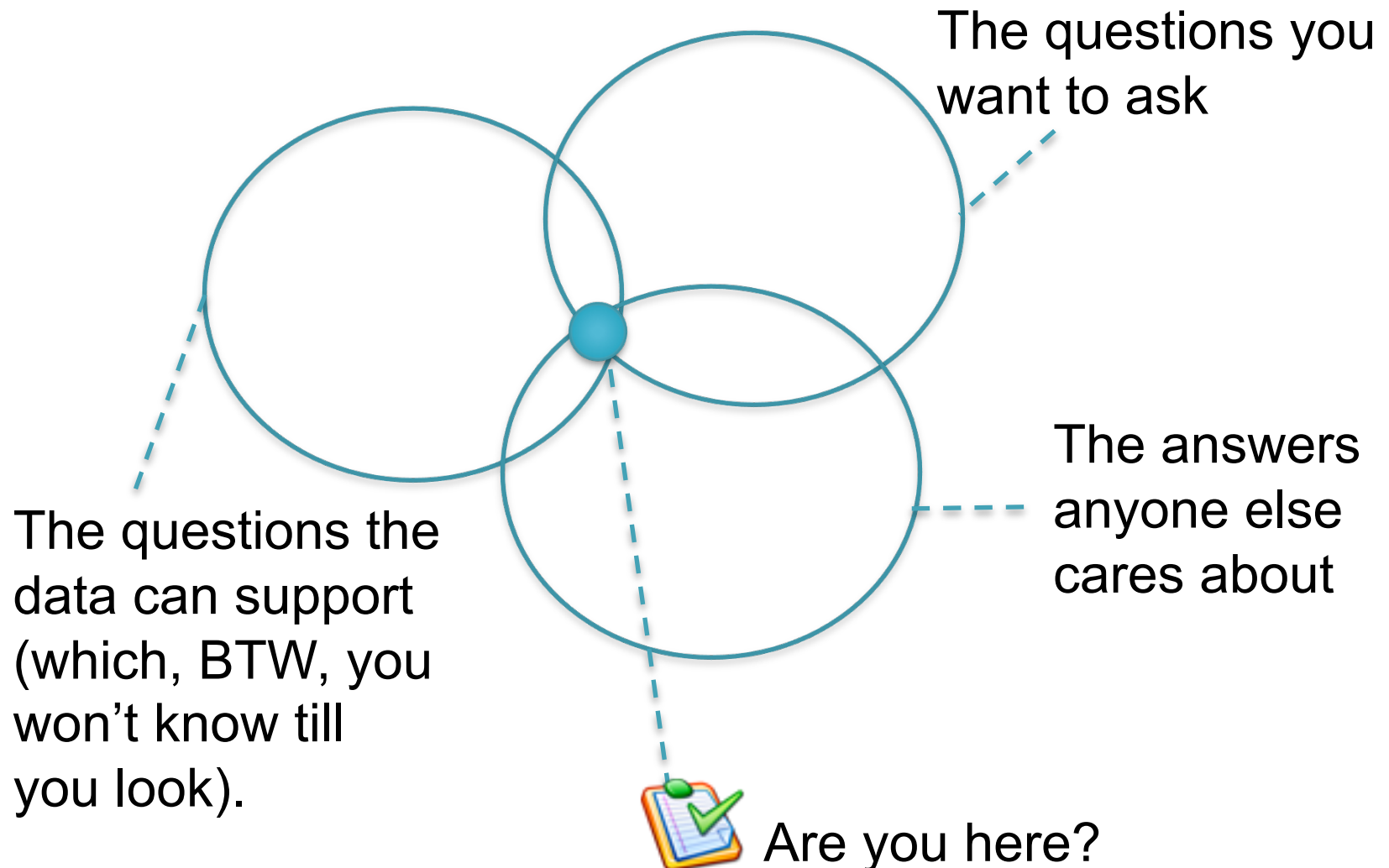


Table 1.2 The weather data.

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

```

If outlook = sunny and humidity= high then play= no
If outlook = rainy and windy= true then play= no
If outlook = overcast then play= yes
If humidity = normal then play= yes
If none of the above then play= yes

```

These rules are meant to be interpreted in order: the first one, then if it doesn't apply the second, and so on. A set of rules that are intended to be interpreted in sequence is called a *decision list*. Interpreted as a decision list, the rules correctly classify all of the examples in the table, whereas taken individually, out of context, some of the rules are incorrect. For example, the rule `if humidity = normal then play = yes` gets one of the examples wrong (check which one). The meaning of a set of rules depends on how it is interpreted—not surprisingly!

In the slightly more complex form shown in Table 1.3, two of the attributes—temperature and humidity—have numeric values. This means that any learning method must create inequalities involving these attributes rather than simple equality tests, as in the former case. This is called a *numeric-attribute problem*—in this case, a *mixed-attribute problem* because not all attributes are numeric.

Now the first rule given earlier might take the following form:

```

If outlook = sunny and humidity > 83 then play = no

```

A slightly more complex process is required to come up with rules that involve numeric tests.

Table 1.3 Weather data with some numeric attributes.

Outlook	Temperature	Humidity	Windy	Play
sunny	85	85	false	no
sunny	80	90	true	no
overcast	83	86	false	yes
rainy	70	96	false	yes
rainy	68	80	false	yes
rainy	65	70	true	no
overcast	64	65	true	yes
sunny	72	95	false	no
sunny	69	70	false	yes
rainy	75	80	false	yes
sunny	75	70	true	yes
overcast	72	90	true	yes
overcast	81	75	false	yes
rainy	71	91	true	no

The rules we have seen so far are *classification rules*: they predict the classification of the example in terms of whether to play or not. It is equally possible to disregard the classification and just look for any rules that strongly associate different attribute values. These are called *association rules*. Many association rules can be derived from the weather data in Table 1.2. Some good ones are as follows:

```

If temperature = cool                then humidity = normal
If humidity = normal and windy = false then play = yes
If outlook = sunny and play = no     then humidity = high
If windy = false and play = no       then outlook = sunny
                                      and humidity = high.

```

All these rules are 100% correct on the given data; they make no false predictions. The first two apply to four examples in the dataset, the third to three examples, and the fourth to two examples. There are many other rules: in fact, nearly 60 association rules can be found that apply to two or more examples of the weather data and are completely correct on this data. If you look for rules that are less than 100% correct, then you will find many more. There are so many because unlike classification rules, association rules can “predict” any of the attributes, not just a specified class, and can even predict more than one thing. For example, the fourth rule predicts both that *outlook* will be *sunny* and that *humidity* will be *high*.

Contact lenses: An idealized problem

The contact lens data introduced earlier tells you the kind of contact lens to prescribe, given certain information about a patient. Note that this example is intended for illustration only: it grossly oversimplifies the problem and should certainly not be used for diagnostic purposes!

The first column of Table 1.1 gives the age of the patient. In case you're wondering, *presbyopia* is a form of longsightedness that accompanies the onset of middle age. The second gives the spectacle prescription: *myope* means shortsighted and *hypermetrope* means longsighted. The third shows whether the patient is astigmatic, and the fourth relates to the rate of tear production, which is important in this context because tears lubricate contact lenses. The final column shows which kind of lenses to prescribe: *hard*, *soft*, or *none*. All possible combinations of the attribute values are represented in the table.

A sample set of rules learned from this information is shown in Figure 1.1. This is a rather large set of rules, but they do correctly classify all the examples. These rules are complete and deterministic: they give a unique prescription for every conceivable example. Generally, this is not the case. Sometimes there are situations in which no rule applies; other times more than one rule may apply, resulting in conflicting recommendations. Sometimes probabilities or weights

```
If tear production rate = reduced then recommendation = none
If age = young and astigmatic = no and
    tear production rate = normal then recommendation = soft
If age = pre-presbyopic and astigmatic = no and
    tear production rate = normal then recommendation = soft
If age = presbyopic and spectacle prescription = myope and
    astigmatic = no then recommendation = none
If spectacle prescription = hypermetrope and astigmatic = no and
    tear production rate = normal then recommendation = soft
If spectacle prescription = myope and astigmatic = yes and
    tear production rate = normal then recommendation = hard
If age = young and astigmatic = yes and
    tear production rate = normal then recommendation = hard
If age = pre-presbyopic and
    spectacle prescription = hypermetrope and astigmatic = yes
then recommendation = none
If age = presbyopic and spectacle prescription = hypermetrope
and astigmatic = yes then recommendation = none
```

Figure 1.1 Rules for the contact lens data.

may be associated with the rules themselves to indicate that some are more important, or more reliable, than others.

You might be wondering whether there is a smaller rule set that performs as well. If so, would you be better off using the smaller rule set and, if so, why? These are exactly the kinds of questions that will occupy us in this book. Because the examples form a complete set for the problem space, the rules do no more than summarize all the information that is given, expressing it in a different and more concise way. Even though it involves no generalization, this is often a very useful thing to do! People frequently use machine learning techniques to gain insight into the structure of their data rather than to make predictions for new cases. In fact, a prominent and successful line of research in machine learning began as an attempt to compress a huge database of possible chess endgames and their outcomes into a data structure of reasonable size. The data structure chosen for this enterprise was not a set of rules but a decision tree.

Figure 1.2 shows a structural description for the contact lens data in the form of a decision tree, which for many purposes is a more concise and perspicuous representation of the rules and has the advantage that it can be visualized more easily. (However, this decision tree—in contrast to the rule set given in Figure 1.1—classifies two examples incorrectly.) The tree calls first for a test on *tear production rate*, and the first two branches correspond to the two possible outcomes. If *tear production rate* is *reduced* (the left branch), the outcome is *none*. If it is *normal* (the right branch), a second test is made, this time on *astigmatism*. Eventually, whatever the outcome of the tests, a leaf of the tree is reached

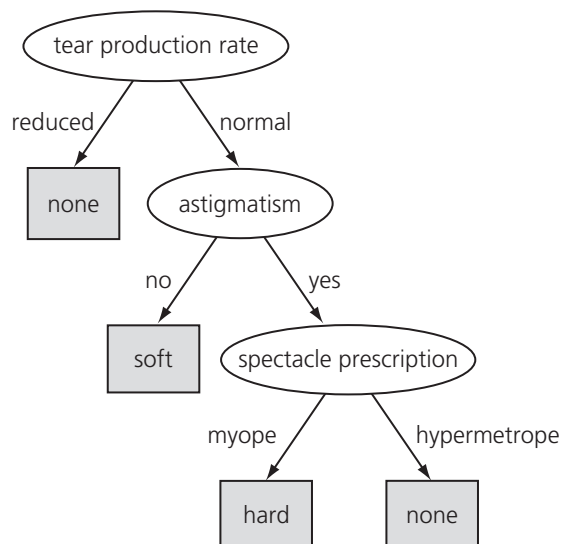


Figure 1.2 Decision tree for the contact lens data.

Table 1.6 The labor negotiations data.

Attribute	Type	1	2	3	...	40
duration	years	1	2	3		2
wage increase 1st year	percentage	2%	4%	4.3%		4.5
wage increase 2nd year	percentage	?	5%	4.4%		4.0
wage increase 3rd year	percentage	?	?	?		?
cost of living adjustment	{none, tcf, tc}	none	tcf	?		none
working hours per week	hours	28	35	38		40
pension	{none, ret-allw, empl-cntr}	none	?	?		?
standby pay	percentage	?	13%	?		?
shift-work supplement	percentage	?	5%	4%		4
education allowance	{yes, no}	yes	?	?		?
statutory holidays	days	11	15	12		12
vacation	{below-avg, avg, gen}	avg	gen	gen		avg
long-term disability assistance	{yes, no}	no	?	?		yes
dental plan contribution	{none, half, full}	none	?	full		full
bereavement assistance	{yes, no}	no	?	?		yes
health plan contribution	{none, half, full}	none	?	full		half
acceptability of contract	{good, bad}	bad	good	good		good

Figure 1.3(b) is a more complex decision tree that represents the same dataset. In fact, this is a more accurate representation of the actual dataset that was used to create the tree. But it is not necessarily a more accurate representation of the underlying concept of good versus bad contracts. Look down the left branch. It doesn't seem to make sense intuitively that, if the working hours exceed 36, a contract is bad if there is no health-plan contribution or a full health-plan contribution but is good if there is a half health-plan contribution. It is certainly reasonable that the health-plan contribution plays a role in the decision but not if half is good and both full and none are bad. It seems likely that this is an artifact of the particular values used to create the decision tree rather than a genuine feature of the good versus bad distinction.

The tree in Figure 1.3(b) is more accurate on the data that was used to train the classifier but will probably perform less well on an independent set of test data. It is “overfitted” to the training data—it follows it too slavishly. The tree in Figure 1.3(a) is obtained from the one in Figure 1.3(b) by a process of pruning, which we will learn more about in Chapter 6.

Soybean classification: A classic machine learning success

An often-quoted early success story in the application of machine learning to practical problems is the identification of rules for diagnosing soybean diseases. The data is taken from questionnaires describing plant diseases. There are about

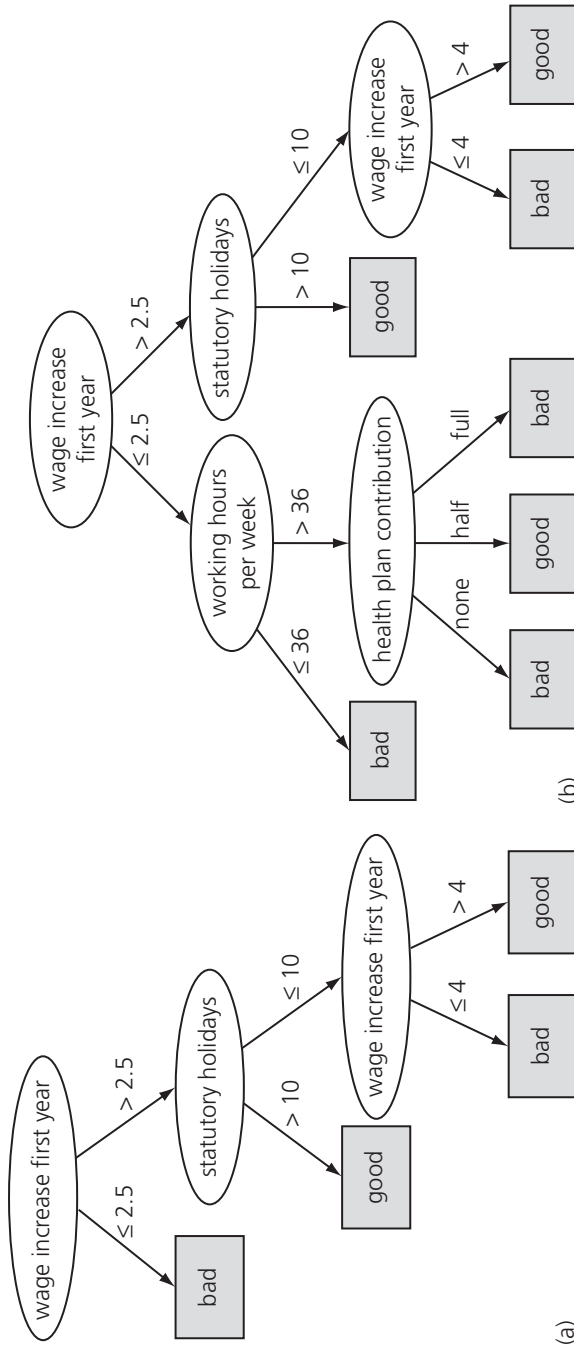


Figure 1.3 Decision trees for the labor negotiations data.

3.7 Trees for numeric prediction

The kind of decision trees and rules that we have been looking at are designed for predicting categories rather than numeric quantities. When it comes to predicting numeric quantities, as with the CPU performance data in Table 1.5, the same kind of tree or rule representation can be used, but the leaf nodes of the tree, or the right-hand side of the rules, would contain a numeric value that is the average of all the training set values to which the leaf, or rule, applies. Because statisticians use the term *regression* for the process of computing an expression that predicts a numeric quantity, decision trees with averaged numeric values at the leaves are called *regression trees*.

Figure 3.7(a) shows a regression equation for the CPU performance data, and Figure 3.7(b) shows a regression tree. The leaves of the tree are numbers that represent the average outcome for instances that reach the leaf. The tree is much larger and more complex than the regression equation, and if we calculate the average of the absolute values of the errors between the predicted and the actual CPU performance measures, it turns out to be significantly less for the tree than for the regression equation. The regression tree is more accurate because a simple linear model poorly represents the data in this problem. However, the tree is cumbersome and difficult to interpret because of its large size.

It is possible to combine regression equations with regression trees. Figure 3.7(c) is a tree whose leaves contain linear expressions—that is, regression equations—rather than single predicted values. This is (slightly confusingly) called a *model tree*. Figure 3.7(c) contains the six linear models that belong at the six leaves, labeled LM1 through LM6. The model tree approximates continuous functions by linear “patches,” a more sophisticated representation than either linear regression or regression trees. Although the model tree is smaller and more comprehensible than the regression tree, the average error values on the training data are lower. (However, we will see in Chapter 5 that calculating the average error on the training set is not in general a good way of assessing the performance of models.)

3.8 Instance-based representation

The simplest form of learning is plain memorization, or *rote learning*. Once a set of training instances has been memorized, on encountering a new instance the memory is searched for the training instance that most strongly resembles the new one. The only problem is how to interpret “resembles”: we will explain that shortly. First, however, note that this is a completely different way of representing the “knowledge” extracted from a set of instances: just store the instances themselves and operate by relating new instances whose class is

PRP =
 -56.1
 +0.049 MYCT
 +0.015 MMIN
 +0.006 MMAX
 +0.630 CACH
 -0.270 CHMIN
 +1.46 CHMAX

(a)

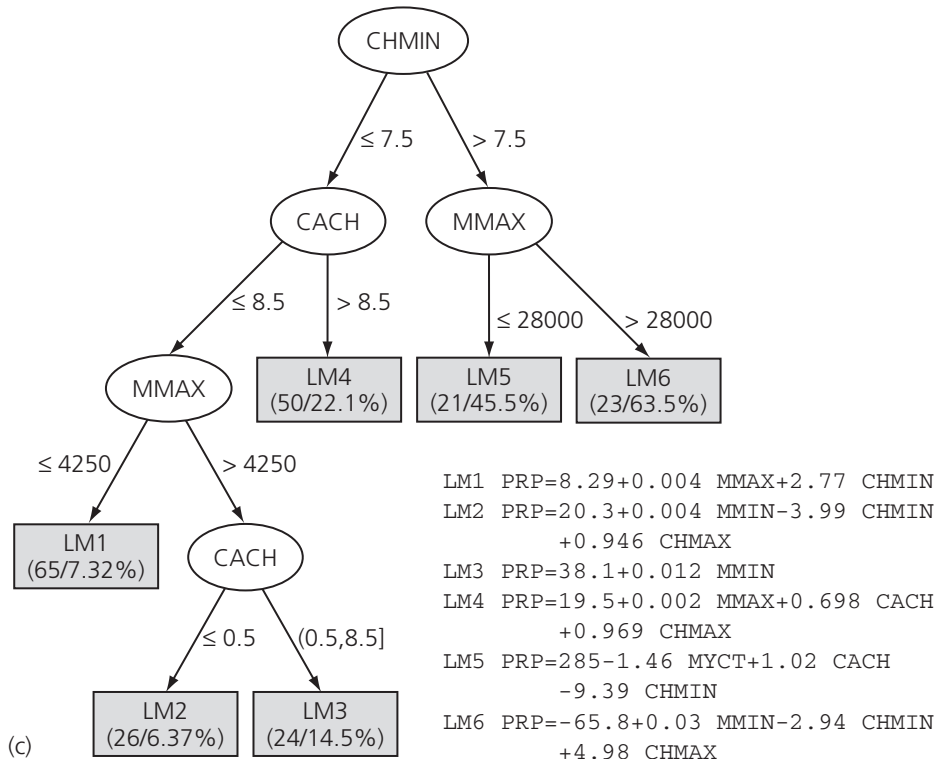
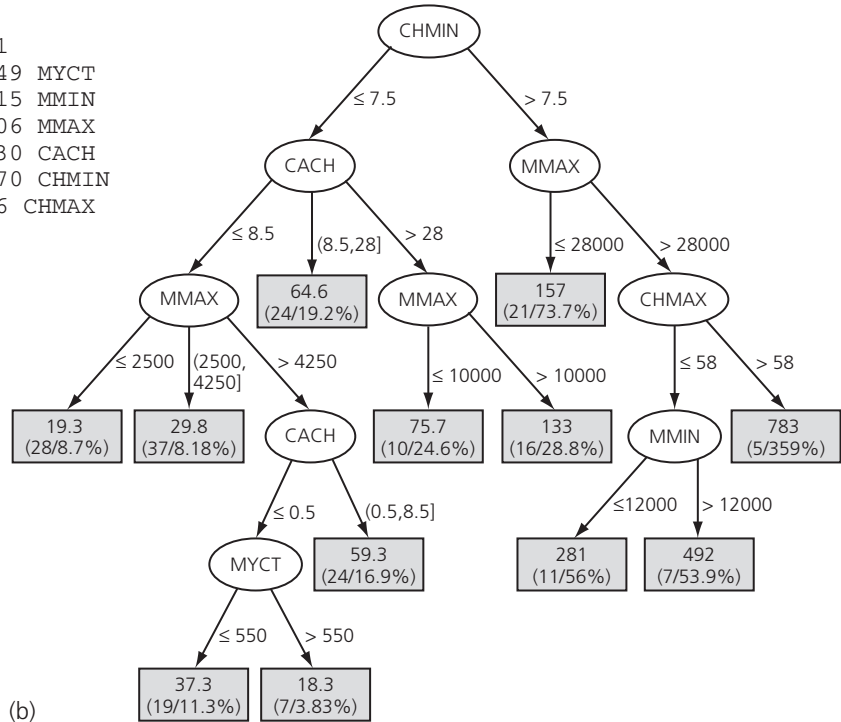


Figure 3.7 Models for the CPU performance data: (a) linear regression, (b) regression tree, and (c) model tree.

few exemplars are needed inside stable regions. For example, you might expect the required density of exemplars that lie well inside class boundaries to be much less than the density that is needed near class boundaries. Deciding which instances to save and which to discard is another key problem in instance-based learning.

An apparent drawback to instance-based representations is that they do not make explicit the structures that are learned. In a sense this violates the notion of “learning” that we presented at the beginning of this book; instances do not really “describe” the patterns in data. However, the instances combine with the distance metric to carve out boundaries in instance space that distinguish one class from another, and this is a kind of explicit representation of knowledge. For example, given a single instance of each of two classes, the nearest-neighbor rule effectively splits the instance space along the perpendicular bisector of the line joining the instances. Given several instances of each class, the space is divided by a set of lines that represent the perpendicular bisectors of selected lines joining an instance of one class to one of another class. Figure 3.8(a) illustrates a nine-sided polygon that separates the filled-circle class from the open-circle class. This polygon is implicit in the operation of the nearest-neighbor rule.

When training instances are discarded, the result is to save just a few prototypical examples of each class. Figure 3.8(b) shows as dark circles only the examples that actually get used in nearest-neighbor decisions: the others (the light gray ones) can be discarded without affecting the result. These prototypical examples serve as a kind of explicit knowledge representation.

Some instance-based representations go further and explicitly generalize the instances. Typically, this is accomplished by creating rectangular regions that enclose examples of the same class. Figure 3.8(c) shows the rectangular regions that might be produced. Unknown examples that fall within one of the rectangles will be assigned the corresponding class; ones that fall outside all rectangles will be subject to the usual nearest-neighbor rule. Of course this produces

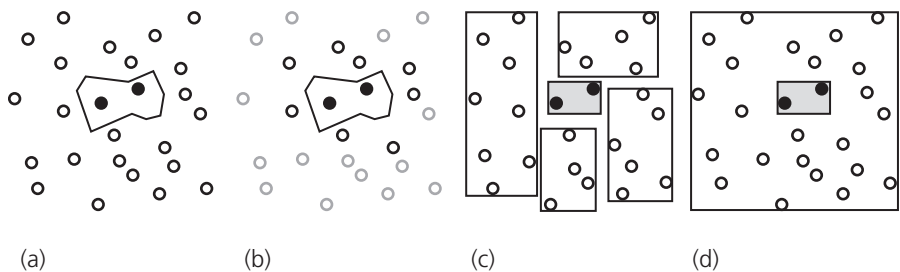


Figure 3.8 Different ways of partitioning the instance space.

restrict ourselves to just one style of learner, the space of options is very large. For example, Figure 1 lists over 12,000 instance-based methods for effort-based estimation.

In instance-based learning, conclusions are drawn from instances near the test instance. The distance measure can be constructed many ways. Mendes et al. [10] discuss three:

M_1 : A simple Euclidean measure;

M_2 : A “maximum distance” measure that focuses on the single feature that maximizes inter-project distance.

M_3 : More elaborate kernel estimation methods.

Once the nearest neighbors are found, they must be used to generate an effort estimate via...

R_1 : Reporting the median effort value of the analogies;

R_2 : Reporting the mean dependent value;

R_3 : Reporting a weighted mean where the nearer analogies are weighted higher than those further away [10];

$R_4..R_6$: Summarize the neighborhood with regression [11], model trees [12] or a neural network [13].

Prior to running an instance-based learning, it is sometimes recommended to handle anomalous rows by:

N_1 : Doing nothing at all;

N_2 : Using outlier removal [14];

N_3 : *Prototype generation*; i.e. replace the data set with a smaller set of most representative examples [15].

When computing distances between pairs, some feature weighting scheme is often applied:

W_1 : All features have uniform weights;

$W_2..W_9$: Some pre-processing scores the relative value of the features. Keung [14], Li et al. [16], and Hall & Holmes [17] review eight different pre-processors. Note that these pre-processors may require *discretization* (discussed below).

Discretization breaks up continuous ranges at points b_1, b_2, \dots , each containing counts of c_1, c_2, \dots of numbers [18].

Discretization methods include:

D_1 : Equal-frequency, where $c_i = c_j$;

D_2 : Equal-width, where $b_{i+1} - b_i$ is a constant;

D_3 : Entropy [19];

D_4 : PKID [20];

D_5 : Do nothing at all.

Finally, there is the issue of how many k neighbors should be used:

K_1 : $k = 1$ is used by Lipowezky et al. [21] and Walkerden & Jeffery [22];

K_2 : $k = 2$ is used by Kirsopp & Shepperd [23]

K_3 : $k = 1, 2, 3$ is used by Mendes et al. [10]

K_4 : Li et al. use $k = 5$ [16];

K_5 : Baker tuned k to a particular training set using an experimental method [11].

Fig. 1. There are many methods for effort estimation. One paper cannot hope to survey them all (e.g. this paper just explores 158). But to get a feel for the space of possibilities, consider the above list of design options for instance-based effort estimation. If we try all the following $N * W * D * M * R * K$ possibilities, this generates a space of $3 * 9 * 5 * 3 * 6 * 5 > 12,000$ methods.

Prior attempts to rank and prune different methods have been inconclusive. Shepperd and Kadoda [24] compared the effort models learned from a variant of regression, rule induction, case-based reasoning (CBR), and neural networks. Their results exhibited much *conclusion instability* where the performance results:

- Differed markedly across different data sets;
- Differed markedly when they repeated their runs using different random seeds.

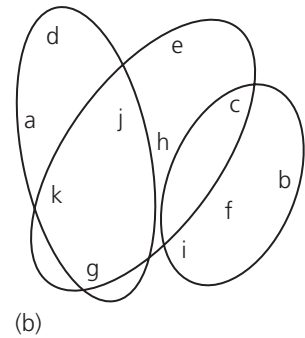
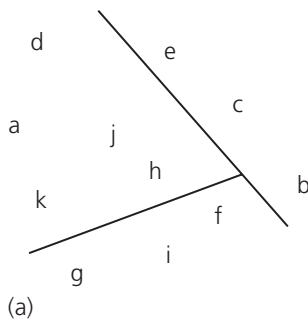
Overall, while no single best method was “best” they found weak evidence that two methods were generally inferior (rule induction and neural nets) [24, p1020].

The genesis of this paper was two observations suggesting that it might be worthwhile revisiting the Shepperd & Kadoda results. Firstly, our data sets are expressed in terms of the COCOMO features [1]. These features were selected by Boehm (a widely-cited and experienced researcher with much industrial experience; e.g. see [25]) and subsequently tested by a large research and industrial community (since 1985, the annual COCOMO forum has met to debate and review the value of those features). Perhaps, we speculated, conclusion instability might be tamed by the use of better features.

3.9 Clusters

When clusters rather than a classifier is learned, the output takes the form of a diagram that shows how the instances fall into clusters. In the simplest case this involves associating a cluster number with each instance, which might be depicted by laying the instances out in two dimensions and partitioning the space to show each cluster, as illustrated in Figure 3.9(a).

Some clustering algorithms allow one instance to belong to more than one cluster, so the diagram might lay the instances out in two dimensions and draw overlapping subsets representing each cluster—a Venn diagram. Some algorithms associate instances with clusters probabilistically rather than categorically. In this case, for every instance there is a probability or degree of membership with which it belongs to each of the clusters. This is shown in Figure 3.9(c). This particular association is meant to be a probabilistic one, so the numbers for each example sum to one—although that is not always the case. Other algorithms produce a hierarchical structure of clusters so that at the top level the instance space divides into just a few clusters, each of which divides into its own subclusters at the next level down, and so on. In this case a diagram such as the one in Figure 3.9(d) is used, in which elements joined together at lower levels are more tightly clustered than ones joined together at



	1	2	3
a	0.4	0.1	0.5
b	0.1	0.8	0.1
c	0.3	0.3	0.4
d	0.1	0.1	0.8
e	0.4	0.2	0.4
f	0.1	0.4	0.5
g	0.7	0.2	0.1
h	0.5	0.4	0.1

(c)

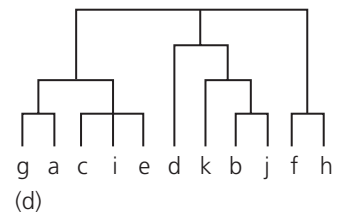
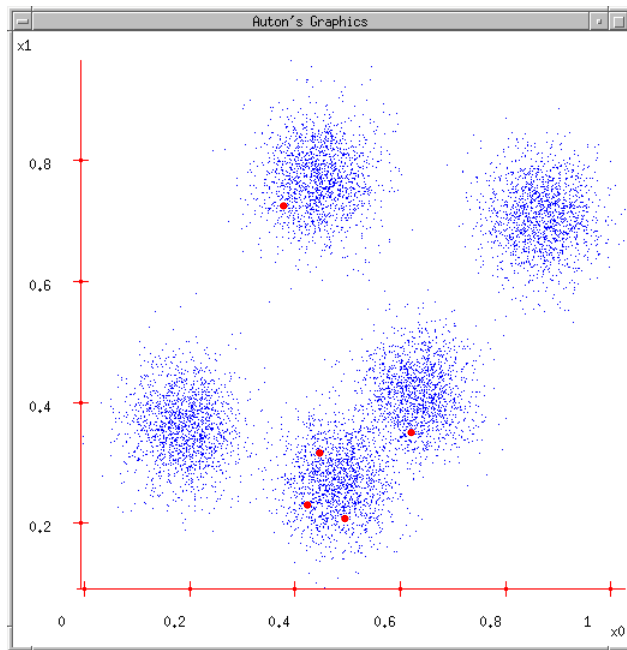


Figure 3.9 Different ways of representing clusters.

K-means

1. Ask user how many clusters they'd like. (e.g. $k=5$)
2. Randomly guess k cluster Center locations

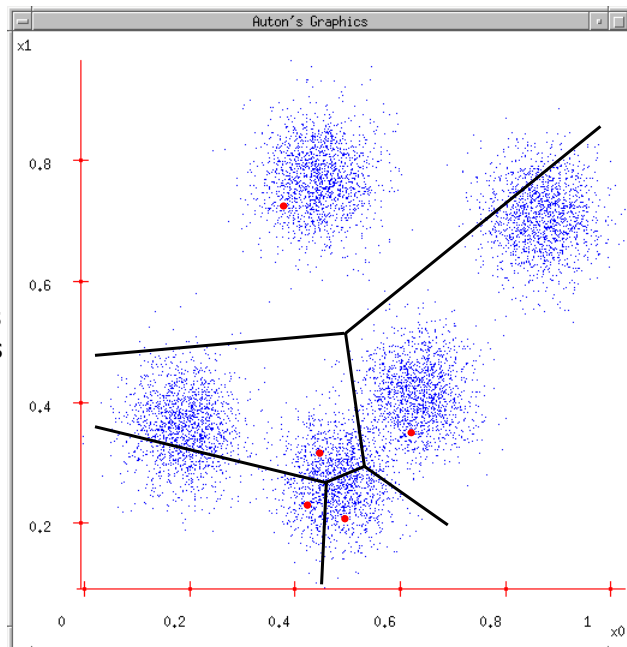


Copyright © 2001, 2004, Andrew W. Moore

K-means and Hierarchical Clustering: Slide 7

K-means

1. Ask user how many clusters they'd like. (e.g. $k=5$)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)

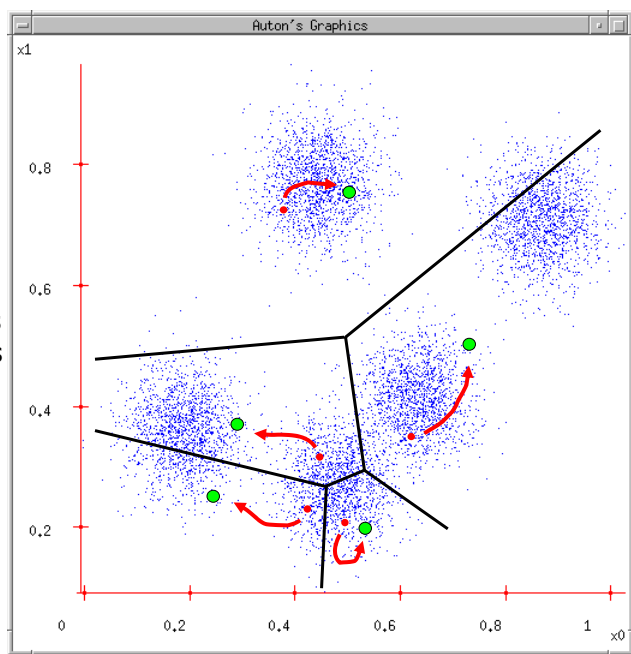


Copyright © 2001, 2004, Andrew W. Moore

K-means and Hierarchical Clustering: Slide 8

K-means

1. Ask user how many clusters they'd like.
(e.g. $k=5$)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns

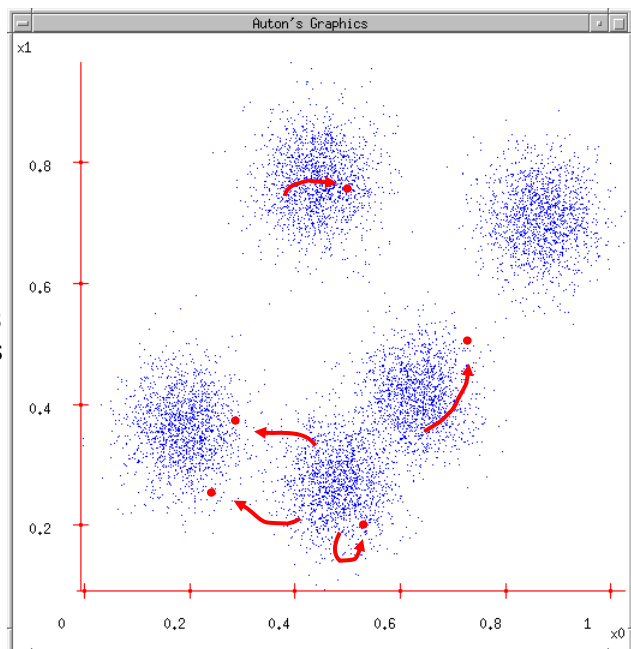


Copyright © 2001, 2004, Andrew W. Moore

K-means and Hierarchical Clustering: Slide 9

K-means

1. Ask user how many clusters they'd like.
(e.g. $k=5$)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns...
5. ...and jumps there
6. ...Repeat until terminated!



Copyright © 2001, 2004, Andrew W. Moore

K-means and Hierarchical Clustering: Slide 10

ai @ wvu

Modeling Intelligence Lab ("the MILL")

Monday, January 25, 2010

Simple single pass k-means

Farnstrom, F., Lewis, J., and Elkan, C. 2000. [Scalability for clustering algorithms revisited](#). SIGKDD Explor. Newsl. 2, 1 (Jun. 2000), 51-57. <http://portal.acm.org/citation.cfm?id=360419>

Model clusters as gaussians:

Clusters are modeled by the sum, sumSquared, and n of each feature.

When clusters combine, their new stats are the sum of the of the old sums, the sum of the old symSquareds, and the n

Reminder: when x is added to a Gaussian

```
n++
sum = sum + x
sumSquared = sumSquared + x*x
mean = sum/n
sd= sqrt((sumSquared-((sum*sum)/n))/(n-1))
```

Do not add a point to a cluster if any of the its attributes falls outside of $\text{mean} \pm 1.96\text{sd}$ (for 95% confidence)

Algorithm:

Initialize k random clusters C1... Ck

Now we have old stats and new stats (initially, old stats is nil)

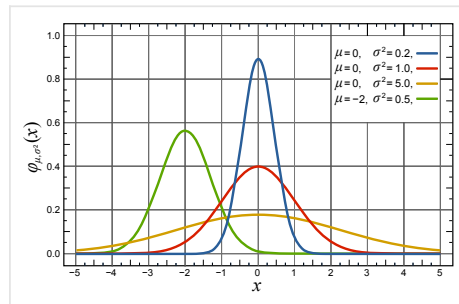
LOOP: Take 1% of the data and add it to a buffer.

For all remaining items in the buffer, run k-means, moving the centroids (until they converge). If can add to a cluster (according to old confidence intervals), then add it (updating sum, sumSquared, and n of new stats).

Here, each cluster is treated as being a point at the mean values, weighted with the number of points in the that cluster.

When no more items can be added, then..

If any cluster has no items, seed it with the most



Site Map

[About us](#)
[Data](#)
[Employment](#)
[Papers](#)
[People](#)
[Projects](#)
[Software](#)
[Starting](#)
[Subscribe to news feed](#)
[Talks](#)
[Teaching](#)
[Tools](#)



Search

powered by Google™

- ▼ 2010 (55)
 - ▶ August (4)
 - ▶ June (7)
 - ▶ May (1)
 - ▶ April (9)
 - ▶ March (10)
 - ▶ February (15)
 - ▼ January (9)
 - [1987 Evett Fiber Model](#)
 - [Why ``W'' Can't Make Up Its Mind](#)
 - [A Simple Approach to RDR](#)
 - [Simple single pass k-means](#)
 - [Searching For Ceiling Effects in NASA MDP Componen...](#)
 - [Kernel estimation](#)
 - [Defect distributions across software components in...](#)
 - [Checking prior results in student retention estima...](#)
 - [ssh keys](#)

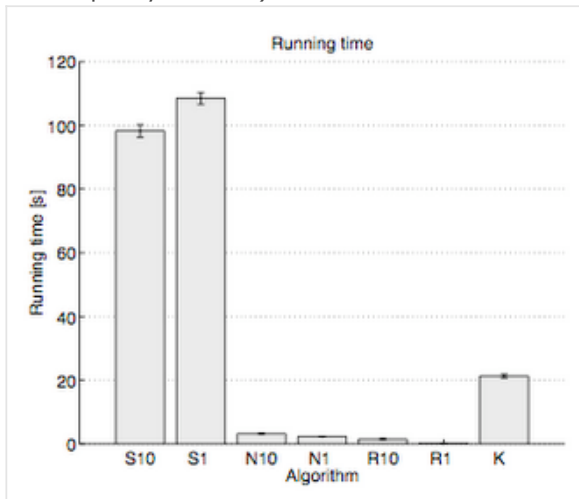
distant data point

Now clear the buffer

Add the new stats to the old stats.

If there is no more new data, stop. Otherwise, go to LOOP

Runs nearly twice as fast as old k-means. Here's the results from using various clustering algorithms on the KDD 1998 cup data (95412 rows, 481 fields). This algorithm is "N1". Normal batch k-means is "K". "R1" is a random sampling k-means (that, btw, generated low quality clusters).



Note that clusters generated by this algorithms, working in increments of 1% of the data, produces clusters of almost the same quality as multiple-pass K-means.

Other advantages:

Small enough memory that you could have multiple versions going, and pick the one with the lowest variances in the generated clusters.

Very simple implementation.

Satisfies the data mining desiderata:

Require one scan (or less) of the database if possible: a single data scan is considered costly, early termination if appropriate is highly desirable.

On-line "anytime" behavior: a "best" answer is always available, with status information on progress, expected remaining time, etc. provided Suspendable, stoppable, resumable; incremental progress saved to resume a stopped job.

Ability to incrementally incorporate additional data with existing models efficiently.

Work within confines of a given limited RAM buffer.

Utilize variety of possible scan modes: sequential, index, and sampling scans if available.

Ability to operate on forward-only cursor over a view of the database. This is necessary since the database view may be a result of an expensive join

Labels

[GregG](#) (18)
[TimM](#) (11)
[Software](#) (8)
[FayolaP](#) (7)
[Paper](#) (7)
[ZachM](#) (7)
[Admin](#) (6)
[AndrewB](#) (6)
[Tar](#) (6)
[Conference](#) (5)
[treatment learning](#) (5)
[BryanL](#) (4)
[AdamN](#) (3)
[KEYS](#) (3)
[Projects](#) (3)
[presentation](#) (3)
[Misc](#) (2)
[RDR](#) (2)
[SEESAW](#) (2)
[tools](#) (2)
[AaronR](#) (1)
[Active Learning](#) (1)
[AdamB](#) (1)
[AndresO](#) (1)
[BojanC](#) (1)
[Clustering](#) (1)
[DDP](#) (1)
[Fastmap](#) (1)
[HyperPipes](#) (1)
[Nova](#) (1)
[OmidJ](#) (1)
[Ous](#) (1)
[OusR](#) (1)
[POM2](#) (1)
[PhillipG](#) (1)
[SBSE](#) (1)
[StevenW](#) (1)
[Talk](#) (1)
[Teaching](#) (1)
[Toolkits](#) (1)
[YueJ](#) (1)
[awk](#) (1)
[clojure](#) (1)
[clump](#) (1)
[cool tools](#) (1)
[fwpsabe](#) (1)
[gawk](#) (1)
[gnuplot tools Timm](#) (1)
[humor](#) (1)
[information retrieval](#) (1)
[interesting](#) (1)
[kernel estimation](#) (1)
[latex](#) (1)
[meetings](#) (1)

Scalability for Clustering Algorithms Revisited

Fredrik Farnstrom
Computer Science
and Engineering
Lund Institute of Technology
Sweden
fredrikf@mail.com

James Lewis
Computer Science
and Engineering
University of California
San Diego
jlewis@cs.ucsd.edu

Charles Elkan
Computer Science
and Engineering
University of California
San Diego
elkan@cs.ucsd.edu

ABSTRACT

This paper presents a simple new algorithm that performs k -means clustering in one scan of a dataset, while using a buffer for points from the dataset of fixed size. Experiments show that the new method is several times faster than standard k -means, and that it produces clusterings of equal or almost equal quality. The new method is a simplification of an algorithm due to Bradley, Fayyad and Reina that uses several data compression techniques in an attempt to improve speed and clustering quality. Unfortunately, the overhead of these techniques makes the original algorithm several times slower than standard k -means on materialized datasets, even though standard k -means scans a dataset multiple times. Also, lesion studies show that the compression techniques do not improve clustering quality. All results hold for 400 megabyte synthetic datasets and for a dataset created from the real-world data used in the 1998 KDD data mining contest. All algorithm implementations and experiments are designed so that results generalize to datasets of many gigabytes and larger.

1. INTRODUCTION

Clustering is the task of grouping together similar items in a dataset. Similar data items can be seen as being generated from the same component of a mixture of probability distributions. The clustering problem is to determine the parameters of the mixture distribution that generated a set of observed data items, where for each item its component is an unobserved feature.

The k -means algorithm is a heuristic solution to the clustering problem based on the assumption that data points are drawn from a fixed number k of spherical Gaussian distributions. The algorithm is an iterative process of assigning cluster memberships and re-estimating cluster parameters. It terminates when the data points no longer change membership due to changes in the re-estimated cluster parameters.

Under the assumption that datasets tend to be small, research on clustering algorithms has traditionally focused on improving the quality of clusterings [4]. However, many datasets now are large and cannot fit into main memory. Scanning a dataset stored on disk or tape repeatedly is time-consuming, but the standard k -means algorithm typically requires many iterations over a dataset to converge to a so-

lution, with each element needing to be accessed on each iteration. Therefore, considerable recent research has focused on designing clustering algorithms that use only one pass over a dataset [9; 6]. These methods all assume that only a portion of the dataset can reside in memory, and require only a single pass through the dataset.

The starting point of this paper is a single pass k -means algorithm proposed by Bradley, Fayyad, and Reina [1]. This method uses several types of compression to limit memory usage. However, the compression techniques make the algorithm complicated. We investigate the tradeoffs involved by comparing several variants of the algorithm of Bradley *et al.* experimentally with a simple new single pass k -means method. Our overall conclusion is that the simple method is superior in speed, and at least equal in the quality of clusterings produced.

2. SINGLE PASS K -MEANS ALGORITHMS

The algorithm of Bradley *et al.* [1] is intended to increase the scalability of k -means clustering for large datasets. The central idea is to use a buffer where points from the dataset are saved in compressed form. First, the means of the clusters are initialized, as with standard k -means. Then, all available space in the buffer is filled with points from the dataset. The current model is updated on the buffer contents in the usual way. The buffer contents are then compressed in two steps.

The first step, called *primary compression*, finds and discards points that are unlikely ever to move to a different cluster. There are two methods to do this. The first method measures the Mahalanobis distance from each point to the cluster mean it is associated with, and discards a point if it is within a certain radius. For the second method, confidence intervals are computed for each cluster mean. Then, for each point, a worst case scenario is created by perturbing the cluster means within the confidence intervals. The cluster mean that is associated with the point is moved away from the point, and the cluster means of all other clusters are moved towards the point. If the point is still closest to the same cluster mean after the perturbations, then it is deemed unlikely ever to change cluster membership.

Points that are unlikely to change membership are removed from the buffer, and are placed in a *discard set*. Each of the main clusters has a discard set, represented by the sufficient statistics for all points belonging to that cluster that have been removed.

On the remaining points in the buffer, another k -means clus-

tering is performed, with a larger number of clusters than for the main clustering. This phase is called *secondary compression*. The aim is to save buffer space by storing some auxiliary clusters instead of individual points. In order to replace points in the buffer by a secondary cluster, the cluster must satisfy a tightness criterion, meaning that its standard deviation in each dimension must be below a certain threshold β . Secondary clusters are combined using hierarchical agglomerative clustering [7], as long as the combined clusters satisfy the tightness criterion.

After primary and secondary compression, the space in the buffer that has become available is filled with new points, and the whole procedure is repeated. The algorithm ends after one scan of the dataset, or if the centers of the main clusters do not change significantly as more points are added.

2.1 Implementation issues

We have coded a new C++ implementation of the algorithm of Bradley *et al.* All the algorithms we compare experimentally are implemented as variants of the same code. The platform for our experiments is a dual 450 MHz Pentium II workstation with 256 megabytes of main memory, running Linux. Our program is not multithreaded, so only one of the processors is directly used in the experiments. The program is compiled with all optimizations turned on. All datasets are stored on disk as Linux binary files. Regardless of the size of any dataset, each pass of each algorithm reads the dataset afresh from disk. Therefore, our experimental conclusions generalize to very large datasets.

Some details of the implementation of their algorithm are not given by Bradley *et al.* For each primary cluster, a Mahalanobis radius must be determined that causes a certain fraction p of buffer points in that cluster to be discarded. Our implementation computes the distance between each buffer point and the cluster it is assigned to. For each cluster, the list of distances is sorted. Then it is easy to find a radius such that a certain fraction of points is discarded. However, sorting can change the time complexity of the whole algorithm. It may be possible to determine each Mahalanobis radius more efficiently, especially when the fraction of discarded points is small.

Our implementation stores the sufficient statistics (sum of elements, squared sum of elements, number of points) as well as the mean and standard deviation in each dimension of all main and secondary clusters. Means are stored so that the distance between old and new means (the new mean is computed from the sum of the elements) can be computed when doing k -means clustering. Standard deviations are stored to speed up primary compression. Representing one cluster uses four times as much space as one data point. Therefore, if a secondary cluster contains four or fewer points, the points themselves are retained instead of a representation of the cluster.

For our purposes, the sufficient statistics of a cluster are two vectors, Sum and $SumSq$, and one integer, n . The vectors store the sum and the sum of squares of the elements of the points in the cluster, and the integer records the number of points in the cluster. From these statistics, the mean and variance along each dimension can be calculated. Let the sufficient statistics of a cluster A be $(Sum^{(A)}, SumSq^{(A)}, n^{(A)})$. If a point x is added to the cluster,

the sufficient statistics are updated as follows:

$$\begin{aligned} Sum_j^{(A)} &:= Sum_j^{(A)} + x_j \\ SumSq_j^{(A)} &:= SumSq_j^{(A)} + x_j^2 \\ n^{(A)} &:= n^{(A)} + 1. \end{aligned}$$

If clusters A and B are merged, the sufficient statistics for the resulting cluster C are

$$\begin{aligned} Sum_j^{(C)} &= Sum_j^{(A)} + Sum_j^{(B)} \\ SumSq_j^{(C)} &= SumSq_j^{(A)} + SumSq_j^{(B)} \\ n^{(C)} &= n^{(A)} + n^{(B)}. \end{aligned}$$

2.2 A simple single pass k -means method

A special case of the algorithm of Bradley *et al.*, not mentioned in their paper, would be when all points in the buffer are discarded each time. This algorithm is:

1. Randomly initialize cluster means. Let each cluster have a discard set in the buffer that keeps track of the sufficient statistics for all points from previous iterations.
2. Fill the buffer with points.
3. Perform iterations of k -means on the points and discard sets in the buffer, until convergence. For this clustering, each discard set is treated like a regular point placed at the mean of the discard set, but weighted with the number of points in the discard set.
4. For each cluster, update the sufficient statistics of the discard set with the points assigned to the cluster. Remove all points from the buffer.
5. If the dataset is exhausted, then finish. Otherwise, repeat from Step 2.

This algorithm is called the *simple single pass k -means* method. Compared to the more complicated algorithm above, it does much less computation each time the buffer is filled, and the whole buffer can be filled with new points at every fill. Following Bradley *et al.* [1], if a cluster ever becomes empty, it is reinitialized with the point in the buffer that is most distant from the centers of all other clusters. However, with a large dataset and a small number of clusters, reinitialization is almost never necessary.

Like the more complicated algorithm above, the simple method uses only one scan over the dataset and a fixed size buffer. It also satisfies all the other desiderata listed by Bradley *et al.* [1]: incremental production of better results given additional data, ease of stopping and resuming execution, and ability to use many different database scan modes, including forward-only scanning over a database view that is never materialized completely.

3. LESION EXPERIMENTS

To evaluate the contribution of each of the data compression methods, we report the results of experimental lesion studies. Comparisons are made between four variants of the algorithm of Bradley *et al.*, the standard k -means algorithm, and the simple single pass k -means algorithm described above.

Parameter	Value
Confidence level for cluster means	95%
Max std. dev. for tight clusters (β)	1.5
Number of secondary clusters	20
Fraction of points discarded (p)	20%

Table 1: Parameter settings used for the lesion studies of the k -means algorithm of Bradley *et al.*

Three variants involve adding one of the data compression methods described above to the previous variant. The first variant uses none of the data compression techniques. This variant runs until convergence on the first fill of the buffer, and then stops. This variant is similar to clustering on a small random sample of the dataset. In the second variant, the first primary compression technique is used. This involves moving to the discard set each point within a certain Mahalanobis distance from its associated cluster mean. In the third variant, the second primary compression technique is added. Confidence intervals are used to discard data points deemed unlikely ever to change cluster membership. The fourth variant includes the data compression technique of determining secondary clusters. All parameter settings used in the experiments reported here are shown in Table 1.

3.1 Synthetic datasets

The lesion experiments use synthetic datasets. Using artificial data allows the clusters found by each algorithm to be compared with known true probability distribution components. In each synthetic dataset, points are drawn from a mixture of a fixed number of Gaussian distributions. Each Gaussian is assigned a random weight that determines the probability of generating a data point from that component. Following Bradley *et al.* [1], the mean and variance of each Gaussian are uniformly sampled, for each dimension, from the intervals $[-5, 5]$ and $[0.7, 1.5]$ respectively.

In order to measure the accuracy of a clustering, the true cluster means must be compared with the estimated cluster means. The problem of discovering which true cluster mean corresponds to which estimated mean must be solved. If the number of clusters k is small, then it is possible to use the one of the $k!$ permutations that yields the highest accuracy. We do this, and for this reason the number of clusters $k = 5$ is small in our experiments. Results with a much larger number of clusters might be different.

The synthetic datasets have 100 dimensions and 1,000,000 data points. They are stored on disk in 400 megabyte files. This size is chosen to guarantee that the operating system cannot buffer a dataset in main memory. Except for the standard k -means algorithm, each clustering algorithm uses a limited buffer large enough to contain approximately 1% of the data points.

The experiments use 30 different synthetic datasets. For each dataset each algorithm generates five different clusterings from different initial conditions. The best of these five models is retained for a comparison of the accuracy of the algorithms. The best of five runs is used because k -means algorithms are known to be sensitive to how cluster centers are initialized. In applications where a good clustering is wanted, it is therefore natural to use the best of several runs.

In general one of two different situations occurs with each clustering. Either, one cluster mean in the model is close to each true Gaussian center, or, two cluster means in the model are trapped near the same center. As we measure the distance between the true and the estimated cluster means, if a center is trapped then the distance measure will be much larger than otherwise. Therefore, the cluster quality in Figure 1 is based only on datasets for which every algorithm produced at least one clustering where no center is trapped.

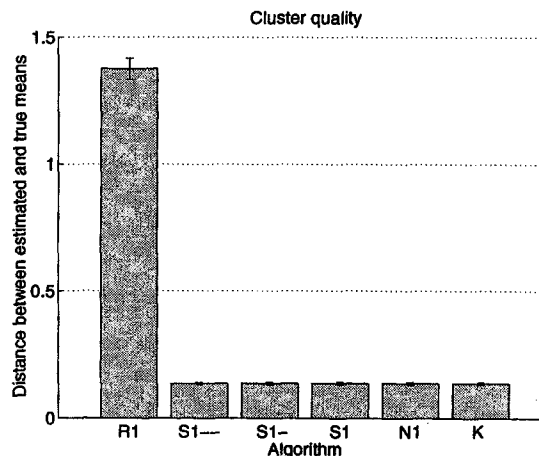


Figure 1: The graph shows the mean sum of the distances between the estimated and true cluster means, for synthetic datasets of 1,000,000 points, 100 dimensions, and five clusters. The algorithms are random sampling k -means (R1), single pass k -means with the first primary compression technique only (S1--), with both primary compression techniques (S1-), with primary and secondary compression (S1), the simple single k -means method (N1), and the standard k -means algorithm operating on the whole dataset (K). Error bars show standard errors.

3.2 Lesion experiment results

Figure 1 shows that even the simplest single pass algorithm achieves the same clustering quality as the full k -means method. Random sampling k -means is less accurate because it uses only 1% of the total data points.

A clustering where no centers are trapped is highly desirable. Therefore, we also measure the fraction of clusterings where no centers are trapped, counting clusterings from all five random initial conditions. We call this fraction the *reliability* of an algorithm. Surprisingly, Figure 2 shows that the single pass algorithms are more reliable than the standard k -means algorithm, and this difference is statistically significant.

Throughout this paper, the difference between x and y is called statistically significant if $x + s_x < y - s_y$ or $y + s_y < x - s_x$, where s_x and s_y are the standard errors of x and y respectively. If x is the mean of n observations then its standard error is the standard deviation of the n observations divided by \sqrt{n} . For the special case where x is a proportion, its standard error is $\sqrt{x(1-x)/n}$. If n is sufficiently large then a Gaussian approximation is valid, so the null hypothesis that the true values of x and y are the same can be rejected with confidence $p < 0.05$, if $x + s_x < y - s_y$ or $y + s_y < x - s_x$. Numerical p values from specific statisti-

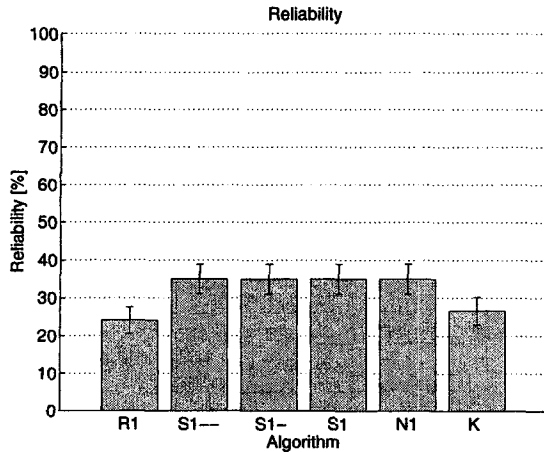


Figure 2: The graph shows the reliability of the different algorithms on the synthetic datasets. Reliability is defined as the fraction of all runs where no centers are trapped. Error bars show standard errors.

cal tests are not reported because their precision could be misleading, since the assumptions on which standard tests are based are often not valid when comparing performance metrics for data mining methods [3].

Figure 2 shows surprisingly that the standard k -means algorithm is not significantly more reliable than random sampling k -means. This fact indicates that the standard algorithm has difficulty escaping from a bad initialization, regardless of how many data points are available. Similarly, the more complicated single pass methods are not more reliable than the simple single pass method. This fact indicates that the more complicated methods do not have any improved ability to escape from a bad initialization.

The average running time of each algorithm is shown in Figure 3. Reported times are averages over 135 runs for each algorithm. The full algorithm of Bradley *et al.*, identified as S1 in Figure 3, is about four times slower than the standard k -means algorithm, while the simple single pass method is about 40% faster.

With the method of Bradley *et al.*, each additional data compression technique allows more points to be discarded from the buffer. Doing so should make the algorithm run faster, because then fewer refills of the buffer are needed. A balance must be maintained between the time taken to identify points to discard and the speedup gained from discarding those points. Figure 3 shows that compression based on confidence interval perturbation causes a net decrease in speed, while compression based on secondary clustering is beneficial.

4. EXPERIMENTS WITH REAL DATA

In order to experiment with real-world data, the dataset from the 1998 KDD (Knowledge Discovery and Data Mining Conference) contest is used. This dataset contains information about people who have made charitable donations in response to direct mailing requests. In principle, clustering can be used to identify groups of donors who can be targeted with specialized solicitations in order to maximize donation

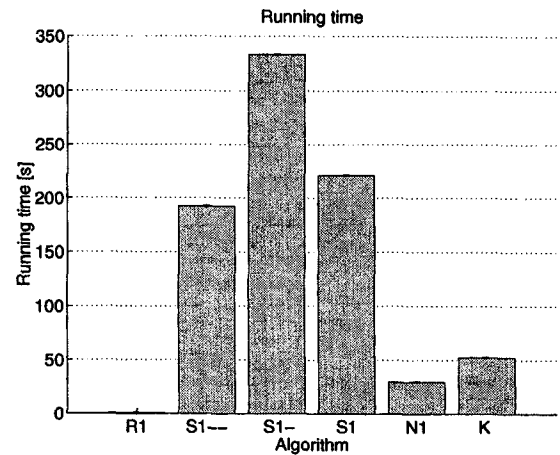


Figure 3: The graph shows the average running time of each k -means algorithm variant. Error bars show standard errors.

profits.

The dataset contains 95412 records, each of which has 481 fields. We take a subset of these fields and code each record as a real-valued vector. Numerical fields (e.g. amounts of past donations, income, age) are directly represented by a single element in the vector. Date values (e.g. donation dates, date of birth) are stored as the number of months from a fixed date. Fields with discrete values, such as an income category, are converted into several binary elements. Each vector has 56 elements in total, of which 18 are binary. To give equal weight to each feature, each feature is normalized to have zero mean and unit variance. The records in the original KDD dataset are converted to this format and saved to a binary file of about 21.4 megabytes. As mentioned in Section 2.1, the implementation of the standard k -means algorithm reads the dataset from disk at each iteration, even though the dataset is small enough to be saved in memory.

The purpose of this experiment is to compare the running time and clustering quality of standard k -means, operating on the whole dataset or on samples, the algorithm of Bradley *et al.* using all types of compression, and the simple single pass method. Experiments are performed with samples and buffers of 10% and 1% of the size of the whole dataset. The number of clusters is always 10.

First, the dataset is randomly reordered. Then it is clustered five times by each algorithm, each time with different randomly chosen initial conditions. All algorithms use the same five initial conditions. The quality of each clustering is measured as the sum of the squared distances between each point and the cluster mean it is associated with. Of the five clusterings for each algorithm, the one with the best quality is used. As above, the best of five is chosen because k -means algorithms are highly sensitive to initial conditions. The whole procedure is repeated 52 times with different random orderings of the dataset.

It is difficult to discover good parameter values for the algorithm of Bradley *et al.*, especially for the parameters that control the number of points removed by secondary compression. The values used here are given in Table 2. Note that it is difficult for a secondary cluster to have standard

Parameter	Value
Confidence level for cluster means	95%
Max std. dev. for tight clusters (β)	1.1
Number of secondary clusters	40
Fraction of points discarded (p)	20%

Table 2: Parameter settings used for the algorithm of Bradley *et al.* with the KDD dataset.

deviation $\beta \leq 1.1$ in *every* dimension, even though the whole dataset is normalized to have standard deviation 1.0 in each dimension.

Figure 4 shows the average quality of the best of five clusterings, for each algorithm. Random sampling k -means operating on a 1% sample performs much worse than all other methods. Standard k -means performs best, followed by the simple single pass method using a buffer of size 1%, followed by the algorithm of Bradley *et al.* All differences mentioned here are statistically significant.

There is no “true” clustering of the KDD dataset that can be used to define reliability in a way similar to how reliability is defined for the synthetic datasets. Therefore, the reliability of an algorithm is defined here to be the fraction of all clusterings that have a quality measure of less than $3.9 \cdot 10^6$. This number is chosen somewhat arbitrarily based on Figure 4 as a threshold for what constitutes an acceptable clustering. A reliable algorithm is one that is less sensitive to how cluster centers are initialized, and that produces a good clustering more often.

Figure 5 shows that the standard k -means method and the simple single pass method with a buffer of size 1% are the most reliable. All other methods are statistically significantly less reliable. It is surprising that the simple single pass algorithm using a buffer of size 1% of the entire dataset outperforms the same method using a 10% buffer. Similar results were found by Bradley *et al.* [1] when they varied the buffer size used by their algorithm. The reason why a smaller buffer can be better remains to be discovered.

Figure 6 shows the average running time of each method. Compared to the standard k -means method, the algorithm of Bradley *et al.* is over four times slower, while the simple single pass method is over five times faster.

5. COMPUTATIONAL COMPLEXITY

In the discussion here of the asymptotic efficiency of the algorithms, we use the following notation:

- m number of k -means passes over entire dataset
- m' number of k -means passes over one buffer refill
- d number of dimensions
- n number of data points
- b size of buffer, as fraction of n
- r number of buffer refills
- k number of main clusters
- k_2 number of secondary clusters
- m_2 number of passes for each secondary clustering.

The time complexity of the standard k -means algorithm is $O(nkdm)$, where empirically m grows very slowly with n , k , and d .

For the simple single pass k -means algorithm, the time complexity of clustering the buffer contents once is $O(nbkdm')$.

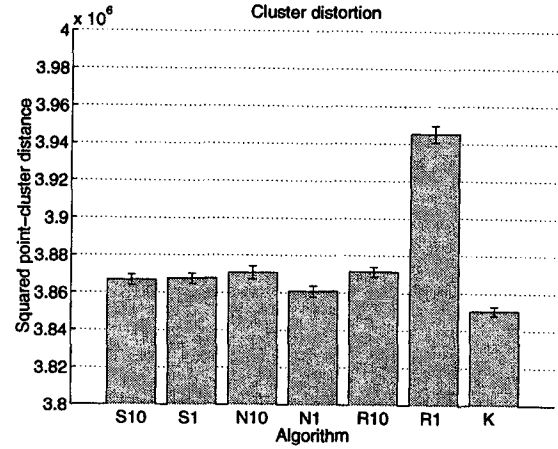


Figure 4: The graph shows the sum of the squared distances between each point in the dataset and the cluster mean it is associated with, on the KDD contest dataset of 95412 points with 10 clusters. The algorithms are due to Bradley *et al.* (S10 and S1), the simple single pass method (N10 and N1), random sampling k -means (R10 and R1), and standard k -means working on the whole dataset. Algorithms with names ending in 10 use a buffer or sample of size 10% of the whole dataset, while those with names ending with 1 use a 1% buffer or sample. Error bars show standard errors.

Algorithm	Time	Space	I/O
Standard	$nkdm$	kd	ndm
Bradley <i>et al.</i>	$nbrk_2dm_2$	$nbd + k_2^2d$	nd
Simple single pass	$nkdm'$	nbd	nd

Table 3: Order of magnitude time, memory, and disk input/output complexity for different k -means algorithms.

Because the buffer is emptied completely before each refill, the number of refills is $1/b$, so the time complexity of clustering the whole dataset is $O(nbkdm' \cdot 1/b) = O(nkdm')$. Interestingly, m' tends to be less than m because clustering is performed over fewer data points than for standard k -means. In fact, m' tends towards one for large datasets, because when the model has stabilized, new points are simply placed in the nearest cluster. This observation is true for all the single pass algorithms.

The complicated nature of the method of Bradley *et al.* makes it difficult to analyze. The main clustering takes $O(nbkdm')$ time per fill. Measuring the Mahalanobis distance to the closest cluster for the points in the buffer is an $O(nbd)$ operation. Finding the discard radius for all main clusters takes $O(nb \log nb)$ time if sorting is used; the worst case is when essentially all points belong to one cluster. The total time complexity of the first method of primary compression is thus $O(nb(d + \log nb))$. The second method of primary compression, where the cluster means are perturbed, has time complexity $O(nbkdm')$.

In the secondary compression phase, m_2 passes with k_2 clusters are performed over the points in each fill of the buffer, giving this phase $O(nbk_2dm_2)$ complexity for one fill of the buffer. Then, hierarchical agglomerative clustering is per-

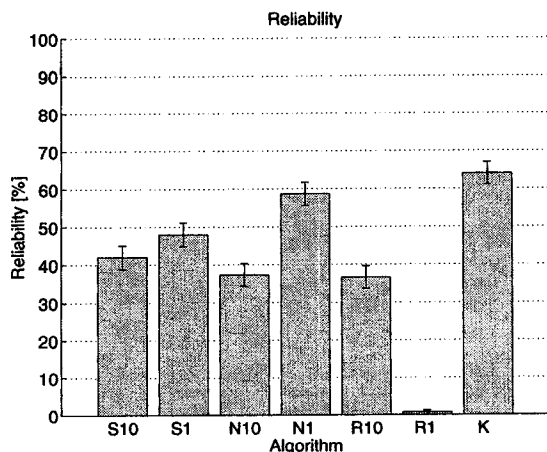


Figure 5: The graph shows the reliability of each different algorithm on the KDD contest dataset, defined as the fraction of clusterings having a distortion less than $3.9 \cdot 10^6$. Error bars show standard errors.

formed on the k_2 clusters. This can be done with $O(k_2^2 d)$ time and space complexity [7].

The steps described above must be repeated r times to scan through the whole dataset. Typically $r > 1/b$ since the whole buffer cannot be filled at each fill. So, the algorithm of Bradley *et al.* has a total time complexity of

$$O\left(nbr(kdm' + d + \log nb + k_2 dm_2 + \frac{k_2^2 d}{nb})\right).$$

In general $k_2 > k$ and $m_2 > m'$, so the total time complexity is $O(nbrk_2 dm_2)$. An assumption here is that the clustering is not stopped until the whole dataset has been processed. This assumption is true in all our experiments.

The time, memory, and disk I/O complexities of the three algorithms are summarized in Table 3. The simple single pass algorithm is superior asymptotically in both time and space complexity to the algorithm of Bradley *et al.*

6. DISCUSSION

The main positive result of this paper is that a simple single pass k -means algorithm, with a buffer of size 1% of the input dataset, can produce clusterings of almost the same quality as the standard multiple pass k -means method, while being several times faster.

Being faster than the standard k -means algorithm is not a trivial accomplishment, because the standard algorithm is already quite scalable. Its running time is close to linear in the size of the input dataset, since the number of passes required is empirically almost independent of the size of the dataset. In addition, at each pass the dataset is scanned sequentially, so a good operating system and disk array can easily provide access to the dataset with high bandwidth.

Although it is called scalable, the algorithm of Bradley *et al.* is much slower in our experiments than the standard k -means method. Bradley *et al.* did not report this fact because their paper contains no comparisons with standard k -means, and no running times. Moreover, the paper gives no measures of statistical significance for differences in clustering qual-

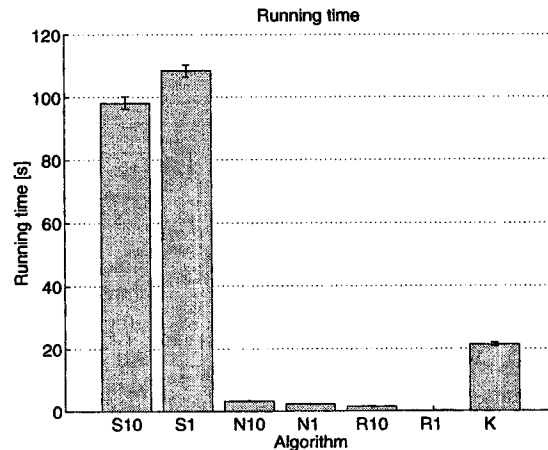


Figure 6: The graph shows the average time taken by each method to perform one clustering of the KDD dataset. Error bars show standard errors.

ity between algorithms, and the largest dataset used in the paper whose size can be computed from information in the paper occupies only 10 megabytes when stored as a floating point binary file. The operating system of any modern workstation can cache a dataset of this size in main memory. We may not have found optimal settings for the parameters of the algorithm of Bradley *et al.* However, we have searched informally for good parameter settings. In general, algorithms that have many parameters with few guidelines about how to choose values for them are difficult to use effectively.

Compared to the standard k -means algorithm, the method of Bradley *et al.* is slower on the KDD dataset than on the synthetic datasets. The opposite is true for the simple single pass method: it is relatively faster on the KDD dataset. The reason is that the KDD dataset has clusters that are separated less well, and the method of Bradley *et al.* is sensitive to clusters not being separated well. The standard algorithm requires 13 passes on average to converge on the KDD dataset, but only 3.3 passes on the synthetic datasets. As explained in Section 5, the number m' of iterations per refill of the buffer tends to 1 for the simple algorithm for all datasets. But for the method of Bradley *et al.*, the number m_2 of iterations in the secondary clustering for each refill of the buffer may remain high.

With all k -means algorithms, both single pass and multiple pass, it is possible to update several clusterings in parallel, where each clustering starts from different initial conditions. We did not do so for our experiments. If we did so, the average running time per clustering of all methods would presumably decrease. There is no reason to think that the relative speeds of the methods would change.

If a dataset to be clustered does not already exist as a single table in a relational database or as a flat file, then materializing it can be expensive. Materializing a dataset may be especially expensive if it consists of a join of tables in a distributed or heterogeneous data warehouse. In this case, all single pass clustering methods can be faster than the standard k -means algorithm. However, the ranking of different single pass methods according to speed is likely to be still

the same.

The results of this paper are complementary to those of Pelleg and Moore [8], who show how to use a sophisticated data structure to increase the speed of k -means clustering for datasets of low dimensionality ($d < 8$). Our simple single pass method is effective regardless of dimensionality. The results here are also complementary to those of Guha, Mishra, Motwani, and O'Callaghan [5], who present single pass clustering algorithms that are guaranteed to achieve clusterings with quality within a constant factor of optimal.

We have not tested other single pass clustering algorithms, notably the BIRCH method [9]. The authors of BIRCH have shown convincingly that it is faster than k -means on large datasets. A comparison of the simple single pass method of this paper with BIRCH would be interesting. Also, all the single pass methods discussed in this paper can be extended to apply to other iterative clustering approaches, and in particular to expectation maximization (EM) [2]. It would be interesting to repeat the experiments of this paper in the EM context.

Acknowledgments: The authors are grateful to Nina Mishra and Bin Zhang of Hewlett Packard Laboratories and to the anonymous referees for valuable comments.

7. REFERENCES

- [1] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 9–15. AAAI Press, 1998.
- [2] P. Bradley, U. Fayyad, and C. Reina. Scaling EM (expectation maximization) clustering to large databases. Technical Report MSR-TR-98-35, Microsoft Research, Redmond, WA, November 1998.
- [3] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924, 1998.
- [4] V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining very large databases. *Computer*, 32(8):38–45, 1999.
- [5] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proceedings of the Annual Symposium on Foundations of Computer Science*. IEEE, November 2000. To appear.
- [6] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 73–84. ACM, 1998.
- [7] M. Meila and D. Heckerman. An experimental comparison of several clustering and initialization methods. Technical Report MSR-TR-98-06, Microsoft Research, Redmond, WA, February 1998.
- [8] D. Pelleg and A. Moore. Accelerating exact k -means algorithms with geometric reasoning. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1999.
- [9] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 103–114. ACM, 1996.

GenIc: A Single Pass Generalized Incremental Algorithm for Clustering

Chetan Gupta*

Department Of Mathematics, Statistics and Computer Science
University of Illinois at Chicago

Robert Grossman†

National Center For Data Mining
University of Illinois at Chicago
and Open Data Partners

This is a draft of a paper from the 2004 SIAM International Conference on Data Mining (SDM 04).

Abstract

In this paper we introduce a new single pass clustering algorithm called GenIc designed with the objective of having low overall cost. We examine some of the properties of GenIc and compare it to windowed k-means. We also study its performance using experimental data sets obtained from network monitoring.

1 Introduction

Developing data mining algorithms for streaming data has emerged as an important problem. For streaming data, the assumption is that the data records can be examined only once. More precisely, given n data records, we would like algorithms with $O(n)$ time complexity and $O(1)$ space complexity.

In this paper we introduce a new algorithm called GenIc, which is a single pass **Generalized Incremental** algorithm for clustering. We also describe simulation results using GenIc, as well as experimental results applying GenIc to network monitoring data.

Perhaps the most common clustering streaming data is a windowed version of the k-means algorithms [6]. One of our main applications is to cluster large amounts of distributed network monitoring data. Simply integrating the distributed data in real time is expensive and for this reason we were interested in low cost clustering algorithms for streaming data. As we

describe in the section on experimental results, GenIc is significantly faster than windowed k-means.

2 Related Work

There is a large research literature on clustering algorithms [4],[15],[16]. Perhaps the most common approach to clustering is to minimize the sums of the squares (SSQ) of the distances between data points and the centers of clusters. The K-means algorithm is a popular heuristic for finding a solution which is a local minimum to this problem since it always converges in a finite number of steps. A wide variety of different algorithms have been proposed to minimize the SSQ, a wide variety of criteria other than SSQ have been proposed, and a wide variety of clustering algorithms adapted to specific applications have been studied.

Another approach is to view the data set as a collection of points and then in a greedy fashion choose the best way to divide the points into two groups. Proceeding inductively in this way leads to what are called hierarchical clustering algorithms and to tree like structures. Similarly, one can work from the bottom up and decide which two best points or clusters to combine. Again either SSQ or other metrics can be used for these types of algorithms.

Recently, clustering algorithms for large data sets and streaming data sets have been developed. The assumption when clustering large data sets is that the data is so large that it resides on secondary storage. BIRCH (Balanced Iterative Reducing and Clustering) clusters large data sets by using specialized trees structures to work with out of memory data [23]. CLARANS (Clustering Large Applications based on RANdom Search) identifies candidate cluster centroids through analysis of repeated random samples of the original data [20].

*gupta@math.uic.edu

†grossman@uic.edu

The assumption when clustering streaming data is that a data record can be examined only once [12]. One approach to clustering streaming data is to perform local clustering on the data that fits in the local memory. The k-mediod problem is a variation of k-means. Guha, et. al. [13] have presented streaming algorithms using local clustering to solve the k-mediod problem. They use a procedure called LOCALSEARCH. The data stream is divided into chunks and a LOCALSEARCH is done on a weighted representation of the chunks. This is done for all such chunks. Additional work done by Bradley, et. al. [5] and the improvements by Farnstrom [10] repeatedly take k-weighted centers (initially chosen at random with a weight of 1) with as much data that can fit in the main memory to compute k cluster centers. The new centers are weighted by the number of points assigned. The data in the memory is discarded and the process is repeated again until all of the data has streamed through.

GenIc is also a windowed algorithms for clustering streaming data. Unlike the windowed algorithms of Guha et. al. [13] Bradley et. al. [5], and Franstorm [10], GenIC employs incremental clustering. GenIc also uses evolutionary techniques to improve its search for global optimal solutions to the SSQ problem required to find cluster centers. Each window in GenIc is viewed as generation for this purpose.

Incremental clustering has also been used for clustering streaming data and large data sets by others [4],[8],[17]. Hartigan’s leader algorithm [15] uses a one-pass, incremental approach. If a point lies within a threshold it is added to the cluster. Otherwise it is used to make a new cluster center. Charikar, et. al. [8] gave theoretical limits for some existing incremental hierarchical clustering algorithms and suggested new approaches. Unlike the incremental clustering described by Hartigan [15] and Charikar [8], GenIC updates each center with each new data point and merges clusters only at the end of a generation (i.e. window of data). DIGNET [21],[22] uses a similar approach in that the cluster centers are pushed or pulled towards a new point, with each new data point; however, the approach used by DIGNET depends strongly on the data ordering. ART [7] is similar to DIGNET. Other incremental clustering algorithms include the Cobweb system of Fisher [11].

Evolutionary techniques have been used in clustering for sometime. They have been used as part of an optimization to reduce overall sum of the squares error by Babu et. al. [2],[3]. Another example is the GGA (Genetically Guided Algorithm) used for fuzzy and hard k-means by Hall et al. [14]. Simulated annealing techniques have likewise been used [18] in clustering.

Cowgill has also introduced a evolutionary clustering algorithm [9]. GenIC is different than these techniques since it views each consecutive set of n data points as a generation rather than looking at all the points before applying evolutionary techniques. For this reason, GenIc is better suited to streaming data.

3 The GenIc Algorithm

In this section, we describe the GenIc algorithm in detail.

An *incremental clustering algorithm* maintains a list of k centers. Each new point which is presented is either i) assigned to one of the clusters or ii) is used to start a new cluster and two of the existing clusters are merged.

By a *generalized incremental algorithm*, we mean an incremental algorithm which can move a center in the list of centers using a weighted sum of the existing center and the new point presented.

Our approach is to divide the stream into chunks or windows as is common with streaming algorithms. We view each chunk of n data points as a generation and think of the “fitness” of a center as being measured by the number of points assigned to it. In general, the fittest centers survive to the next generation, but occasionally new centers are selected and old centers are killed off.

Here are the details:

1. Select parameters.

- Fix the number of centers k .
- Fix the number of initial points m .
- Fix the size of a generation n .

2. Initialize

- Select m points, c_1, \dots, c_m to be the initial candidate centers.
- Assign a weight of $w_i = 1$ to each of these candidate centers.

3. Incremental Clustering.

For each subsequent data point p in the stream: do

- $Count = Count + 1$.
- Find the nearest candidate center c_i to the point p
- Move the nearest candidate center using the formula:

$$c_i = \frac{(w_i * c_i + p)}{(w_i + 1)}$$

- Increment the corresponding weight

$$w_i = w_i + 1$$

- When $Count \equiv 0 \pmod n$, goto Step 4.

4. Generational Update of Candidate Centers.

When $Count$ equals $n, 2n, 3n, \dots$, for every center c_i in the list L of centers, do:

- Calculate its probability of survival using the formula

$$p_i = \frac{w_i}{\sum_{i=1}^m w_i}$$

- Select a random number δ uniformly from $[0, 1]$. If $p_i > \delta$, retain the center c_i in the list L of centers and use it in the next generation of n points.
- If $p_i < \delta$, kill the center c_i and select a new random point from the current generation to replace it as a center in the list L of centers.
- Set the weight $w_i = 1$ back to one. Although some of the points in the stream will be implicitly assigned to other centers now, we do not use this information to update any of the other existing weights.
- Goto step 3 and continue processing the input stream.

5. **Calculate Final Clusters.** The list L contains the m centers. These m centers can be grouped into the final k centers based on their Euclidean distances.

Note that every point can be assigned to a cluster center during the incremental clustering phase of the algorithm. This information can be used in the final phase of the algorithm which computes the final clusters to assign each point to one of the final clusters. For this to work, those points assigned to a cluster center which is dropped can be reassigned to the closest cluster at the end of each generation.

We found that taking m to be a small multiple of k , say $m = 3, 4, 5$ seem to produce the best results.

For our tests we used a slight variant of this algorithm. We worked with a list L of length $2m$. In Step 3, only the first half of the list L was used when testing for the nearest candidate center. In Step 4, we tested the first half of the list to decide whether to drop a center. If the center was dropped in this test, we then tested the second half of the list L to see whether its weight was higher than the lowest weight in the second half. If so, then it was added to the second half of the list L , replacing the center with lowest weight.

For some of the applications, it is useful to update the list L in a different way: when a center say A , is “killed” we look to see if it lies within a certain distance

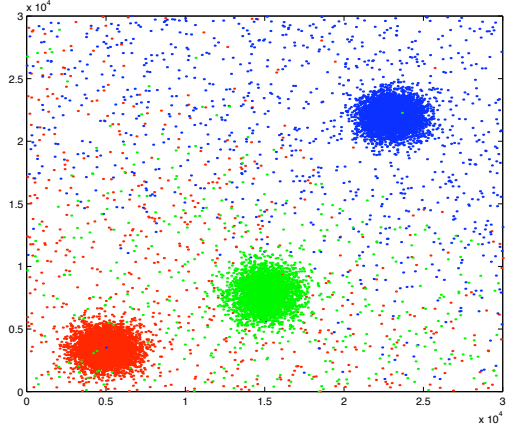


Figure 1: 2-d view of Clustering with GenIc.

from one of the existing centers in the list L . If it lies within this distance, a new center is created and its coordinates are the weighted mean of the two. The weight of the new center is the sum of the two initial centers. If the center A does not lie within a certain distance and if its weight is greater than the lightest weight on the list, it will be added to the list.

4 Experimental Results

We conducted a series of experiments with simulated data using GenIc. Several different data distributions used, including random, normal, Poisson and Cauchy, as well as a variety of different parameters. For these experiments, we randomly chose the centers, distributed data around the centers using the chosen distribution, and then distributed data randomly in the space. Some typical results are shown in Figures 1 and 2.

Accurately identifying cluster centers. In the first series of simulations, we created approximately 50 data sets containing data points distributed around three clusters, as well as other randomly place points, and used GenIc to try to identify the centers of the clusters. The results for some of these experiments are contained in Table 1. For every simulation, the upper row contains the true centers, while the lower row contains the centers as calculated by GenIc.

In most cases, the difference between the calculated centers and the true centers was less than one percent. Occasionally, anomalous centers were identified. This happened more often for points distributed using the Cauchy distribution, when the clusters overlapped, or when one cluster contained almost all of the points.

Quality of clusters. In the second series of simulations, we compared the quality of the clusters by comparing the SSQ of errors of GenIc and windowed k-

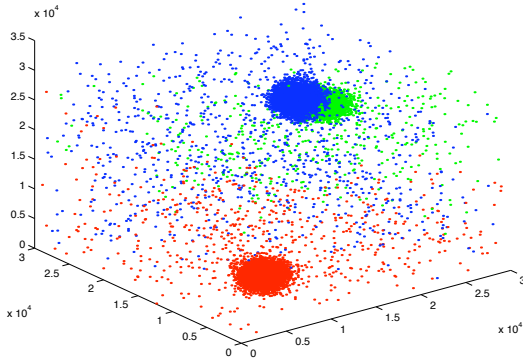


Figure 2: 3-d view of Clustering with GenIc

Exp. No.	Cluster1	Cluster2	Cluster3
1	5000,3400,7600 5092,3404,7667	15000,7800,30000 14884,8167,29747	23000,22000,20000 22823,22039,20084
2	5000,3400,7600 5080,3698,7687	15000,7800,30000 14932,7634,29572	23000,22000,20000 22792,21831,19766
3	5000,3400,7600 5185,3692,7635	15000,7800,30000 14895,7779,30017	23000,22000,20000 23102,22199,19835
4	15000,13400,17600 14645,13210,17064	15000,7800,30000 14726,7803,29942	23000,22000,20000 23153,22066,20044
5	15000,13400,17600 15044,13349,17596	1111,2222,3333 1094,2281,3304	23000,22000,20000 23347,21797,19483
6	6781,9989,10600 6738,9859,10818	4561,8976,7899 4449,9076,8050	23000,22000,20000 22902,21863,19755
7	16781,9989,10600 15114,9737,10149	4561,18976,7899 4664,18875,8105	25000,6780,11232 25324,6884,11307
8	16781,9989,10600 15932,9664,10304	4561,18976,7899 4792,19498,8112	25000,6780,11232 24908,6805,11281
9	16781,9989,10600 16798,10065,10599	9561,18976,7899 9499,19013,7914	25000,8780,11232 25001,8864,11137
10	16781,9989,10600 16203,9580,11198	9561,18976,7899 9399,19549,7883	25000,8780,11232 24698,8886,10872
11	5000,3400,7600 4997,3497,7594	15000,7800,30000 15124,7731,29879	23000,22000,20000 22625,22115,20087
12	5000,3400,7600 5116,3458,7582	15000,7800,30000 14896,7794,29771	23000,22000,20000 23022,21979,19989
13	5000,3400,7600 6957,4223,11811	15000,7800,30000 15163,7699,29913	23000,22000,20000 23039,22027,19903
14	5000,3400,7600 6908,22879,6721	15000,7800,30000 14719,7710,29212	23000,22000,20000 23016,21073,18908
15	5000,3400,7600 5611,4021,7516	15000,7800,30000 14560,7922,27874	23000,22000,20000 22993,22007,20055
16	5000,3400,7600 5655,4434,8596	15000,7800,30000 16018,7492,28847	23000,22000,20000 23006,22074,20251
17	5000,3400,7600 5148,3590,7550	15000,7800,30000 0.0,0	23000,22000,20000 0.0,0
18	5000,3400,7600 5041,3433,7596	15000,7800,30000 14955,7861,29996	23000,22000,20000 23264,22177,20262
19	15000,3400,17600 14993,3487,17544	15000,17800,30000 15080,17836,29883	23000,22000,20000 15080,17836,29883
20	5000,3400,7600 5033,3400,7646	15000,7800,30000 15102,7756,29885	23000,22000,20000 22969,22003,19963
21	5000,3400,7600 0.0,0	15000,7800,30000 0.0,0	23000,22000,20000 23014,22133,19987
22	5000,3400,7600 5251,4268,6499	15000,7800,30000 14724,7639,28289	23000,22000,20000 23199,21920,20014
23	14000,5400,2800 14071,5405,2841	15000,7800,30000 15021,7778,29965	23000,22000,20000 23020,21999,20000
24	14000,5400,2800 14491,6743,28764	15000,7800,30000 14491,6743,28764	23000,22000,20000 22861,21798,20216
25	4000,15400,28000 3783,15893,27742	15000,7800,3000 14111,6875,3204	13000,6000,2900 12282,5951,3414
26	24000,5400,8000 24056,5421,8095	15000,7800,3000 14975,7925,3041	3000,6001,2910

Table 1: Simulation Results for test for finding cluster centers using GenIc

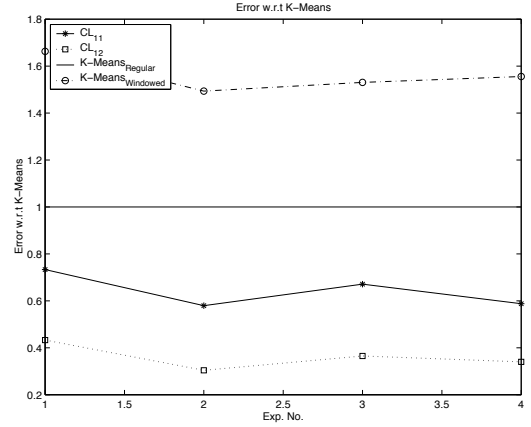


Figure 4: Comparing the ratio of errors between two variants of GenIc and windowed k-means vs. k-means using data as suggested in [23].

means to the SSQ of the errors of standard k-means. For this series, we created 24 data sets: 8 each using the Normal, Cauchy and Poisson distributions. Each set was comprised of 50000 points in three dimensions and with $k = 3$ cluster centers. Each center had a varying number of points assigned to it and there was up to 10 percent noise in each data set. The results can be seen in Figure 3, which compares the ratio of error of an algorithm over the error of k-means. GenIc generally has less error than windowed k-means, and sometimes has less error than k-means.

Changing the number of clusters. For the third series of experiments, we examined the effect of the number of clusters on GenIc. We created four data sets as suggested by the BIRCH paper [23]. There were $k = 100$ centers distributed on a two-dimensional grid. There was no noise and all of the centers had an equal number of points assigned to them. The total number of points for each set was 150000. The results are shown in Figure 4. Again, we used the error of k-means as the standard and plotted the ratio of an algorithm error to the k-means error.

Effect of initial starting points. In this series of experiments we tried to understand how the effect of starting points effected GenIc and k-means. We created 18 data sets of 100000 points in 2 dimensions. We ran experiments using 100 centers, 90 centers, ..., 10 centers, 9 centers, 8 centers, ..., and 2 centers. Since in these experiments the error in the k-means depended fairly strongly on the choice of starting points, we ran k-means 10 times and computed both the minimum error and a trimmed error. To calculate the trimmed error,

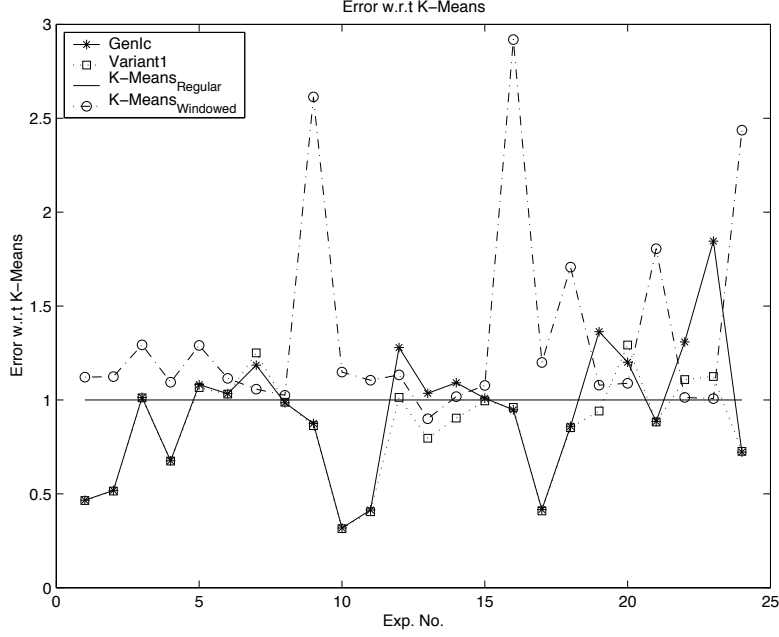


Figure 3: Comparing the ratio of errors between two variants of GenIc and windowed k-means vs. k-means.

we removed the two largest errors and took the mean of the other eight. To calculate the minimum error we took the minimum of the 10 runs.

Figure 5 plots the sum of square distances between the points and the assigned cluster centers. As seen on the graph, the error of GenIc is generally less than the error of k-means when there are a large number of centers. For a sufficiently small number of centers, the minimum error of K-means become less than that of GenIc. On the other hand, the trimmed error is larger than GenIc's, showing that for these experiments k-means is more strongly dependent on the choice of initial center than GenIc.

Independence of data order. The next series of simulations tested the sensitivity of GenIc to the order of the data, a problem faced by some incremental clustering algorithms. Sixteen data sets, grouped into four sets of four, were used. Let A be a set of all points belonging to cluster 1. Let B be the set of all points belonging to cluster 2. Let C be the set of all points belonging to the cluster 3. Each group contained the following four orders: A+B,C;A,B,C;A,B+C;A+B+C, for a total of sixteen data sets. The results are contained in Figure 6 where the relative error between GenIc and windowed k-means as defined by

$$PercentError = 100 \cdot \frac{error_{k-means} - error_{GenIc}}{error_{k-means}}$$

is plotted. Again GenIc does very well.

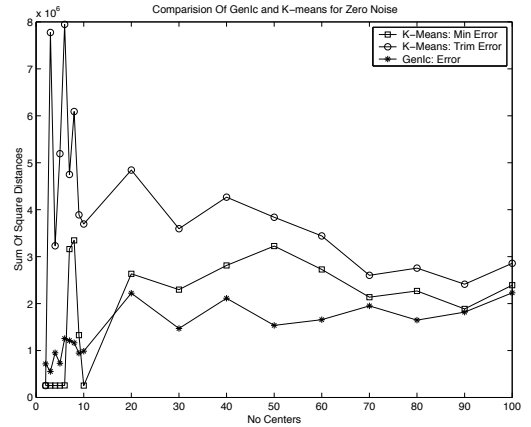


Figure 5: Comparing the ratio of errors between GenIc and k-means. The min error and trimmed error of k-means are plotted.

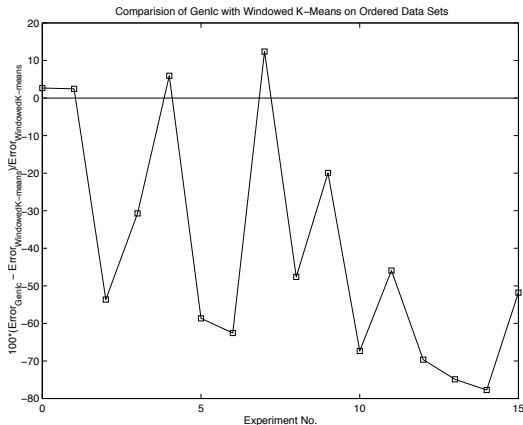


Figure 6: Comparing the effect of data ordering on the error of GenIc and windowed k-means.

GenIc	Windowed k-means
338	14,543

Table 2: Time in seconds using GenIc and windowed k-means to find 4 clusters in 3 million four dimensional feature vectors extracted from 3 million Snort alerts.

Clustering experimental data generated by Snort. Our main interest in developing GenIc was to cluster large experimental data sets, such as those which arise in network monitoring, intrusion detection, threat detection, and related applications.

As a simple test of its speed, we compared GenIc to windowed k-means [6] on a data set 3 million four dimensional feature vectors extracted from 3 million Snort alerts. For GenIc, we used $m = 16$ and $n = 2000$. In order for windowed k-means to obtain a realistic execution time, the window size was set at 100000 points, and for every window, up to 30 iterations were permitted.

5 Summary and Discussion

There is no end to the creation of new clustering algorithms, just a weariness of the flesh. Perhaps the only justification that can be provided for a new clustering algorithm, such as GenIc, is that we have found it useful in applications, such as the clustering of network data, and have found it to be fast.

Here are some of the reasons that we have found GenIc to be useful in practice.

- GenIc is fast. Indeed in some of our experiments it was over 40 times faster than windowed k-means.
- GenIc usually identifies well separated clusters

quite well, as confirmed by our simulation studies.

- GenIc is flexible in the face of the unknown values of k . It does not need an explicit initial input of k . Indeed, it can predict the value of k at the end of the run. Previously, genetic techniques had been used for improving the value of k itself [19].
- Previous incremental algorithms ART [7],[21],[22] are dependent upon the order in which the data comes. GenIc overcomes this by taking every batch of n points independently of the previous n points. As previously shown, we conducted tests by varying the order of streaming data, taking care that the order was not random.
- GenIc is less affected by the choice of initial centers than k-Means.

Any clustering algorithm faced with certain data sets does the wrong thing and GenIc is no exception. If one of the centers has an overwhelming concentration of points near to it, points around it will occupy all of the m points in our list L , and GenIc will perform poorly. One obvious remedy is to exclude points too near to each other. L .

References

- [1] N. Alon, Y. Matias, and M. Szegedy "The Space Complexity of Approximating the Frequency Moments," Journal of Computer and system sciences, 58, 1999.
- [2] G. P. Babu, and M. N. Murty, "A Near Optimal Initial Seed Value Selection in K-means Algorithm Using a Genetic Algorithm," Pattern Recogn. Lett. , 14(10), 1993.
- [3] G. P. Babu, and M. N. Murty, "Clustering with Evolutionary Strategies," Pattern Recognition, 27(2), 321-329,1994.
- [4] P. Berkhin, "Survey Of Clustering Data Mining Techniques," to appear.
- [5] P. S. Bradley, U. M. Fayyad, and C. A. Reina, "Scaling Clustering Algorithms to Large Databases," In proceedings of the 4th Int'l Conference on Knowledge Discovery and Data Mining, New York, NY, Aug 27-31, pages 9-15, 1998.
- [6] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, "Streaming-Data Algorithms for High-Quality Clustering," Proceedings of IEEE International Conference on Data Engineering, March 2002.
- [7] G. Carpenter, and S. Grossberg ART3: "Hierarchical Search Using Chemical Transmitters in Self-Organizing Pattern Recognition Architectures," Neural Networks 3, 129-152. 1990.

- [8] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, "Incremental Clustering and Dynamic Information Retrieval," In proceedings of the 29th annual ACM Symposium on Theory of Computing, 1997.
- [9] M. C. Cowgill, R. J. Harvey, and L. T. Watson, "A genetic algorithm approach to cluster analysis," Computational Mathematics and its Applications, Volume 37, 1999, page 99–108.
- [10] F. Farnstrom, J. Lewis, and C. Elkan, "True Scalability of Clustering Algorithms," SIGKDD Explorations, 2000.
- [11] D. Fisher, "Knowledge Acquisition via Incremental Conceptual Clustering," Mach. Learn. 2, 139-172. 513. 1987.
- [12] P. B. Gibbons, and Y. Matias, "Synopsis Data Structures for Massive Data Sets," Series in Discrete Mathematics and Theoretical Computer Science: Special Issue on External Memory Algorithms and Visualization, A, 1999.
- [13] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering Data Streams," In the Annual Symposium on Foundations of Computer Science, IEEE, 2000.
- [14] L. O. Hall, I. B. Ozyurt, and J. C. Bezdek, "Clustering with a Genetically Optimized Approach," IEEE Transactions on Evolutionary Computation, Volume 3, No. 2, pp. 103-112, 1999.
- [15] J. Hartigan, "Clustering Algorithms," John Wiley and Sons, New York, 1975.
- [16] A. K. Jain, R. C. Dubes, "Algorithms for Clustering Data," New Jersey, Prentice Hall, 1988.
- [17] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," ACM Computing Surveys, vol. 31, no. 3, pp. 264-323, 1999.
- [18] R. W. Klein, and R. C. Dubes, "Experiments in Projection and Clustering by Simulated Annealing," Pattern Recogn. 22, 213-220, 1989.
- [19] C-Y Lee, and E. K. Anotonnson, "Dynamic Partitional Using Evolutionary Strategies," In Proceedings of the Third Asia-Pacific conference on Simulated Evolution and Learning, Nagoya, Japan, 2000.
- [20] R. Ng, and J. Han, "Very Large Data Bases," In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94, Santiago, Chile, Sept.), VLDB Endowment, Berkeley, CA, 144-155. 1994.
- [21] S. C. A. Thomopoulos, D. K. Bougoulas, and C. Wann, "Digent: An Unsupervised-Learning Clustering Algorithm for Clustering and Data Fusion," IEEE transactions on aerospace and electronic systems, 31(1):21-38, 1995.
- [22] C-D Wann, and S.C. A. Thomopoulos, "A Comparative Study of Self Organizing Clustering Algorithm Dignet and ART2," Neural Networks, 10(4) 737-743, 1997.
- [23] T. Zhang, R. Ramkrishnan and M. Linvy, "BIRCH: An Efficient Data Clustering Method for Very Large Databases," SIGMOD Rec. 25, 2, 103-114. 1996.
- [24] www.snort.org, The Open Source Network Intrusion

contribute independently and equally to the final outcome. A third might have a simple logical structure, involving just a few attributes that can be captured by a decision tree. In a fourth, there may be a few independent rules that govern the assignment of instances to different classes. A fifth might exhibit dependencies among different subsets of attributes. A sixth might involve linear dependence among numeric attributes, where what matters is a weighted sum of attribute values with appropriately chosen weights. In a seventh, classifications appropriate to particular regions of instance space might be governed by the distances between the instances themselves. And in an eighth, it might be that no class values are provided: the learning is unsupervised.

In the infinite variety of possible datasets there are many different kinds of structure that can occur, and a data mining tool—no matter how capable—that is looking for one class of structure may completely miss regularities of a different kind, regardless of how rudimentary those may be. The result is a baroque and opaque classification structure of one kind instead of a simple, elegant, immediately comprehensible structure of another.

Each of the eight examples of different kinds of datasets sketched previously leads to a different machine learning method well suited to discovering it. The sections of this chapter look at each of these structures in turn.

4.1 Inferring rudimentary rules

Here's an easy way to find very simple classification rules from a set of instances. Called *1R* for *1-rule*, it generates a one-level decision tree expressed in the form of a set of rules that all test one particular attribute. 1R is a simple, cheap method that often comes up with quite good rules for characterizing the structure in data. It turns out that simple rules frequently achieve surprisingly high accuracy. Perhaps this is because the structure underlying many real-world datasets is quite rudimentary, and just one attribute is sufficient to determine the class of an instance quite accurately. In any event, it is always a good plan to try the simplest things first.

The idea is this: we make rules that test a single attribute and branch accordingly. Each branch corresponds to a different value of the attribute. It is obvious what is the best classification to give each branch: use the class that occurs most often in the training data. Then the error rate of the rules can easily be determined. Just count the errors that occur on the training data, that is, the number of instances that do not have the majority class.

Each attribute generates a different set of rules, one rule for every value of the attribute. Evaluate the error rate for each attribute's rule set and choose the best. It's that simple! Figure 4.1 shows the algorithm in the form of pseudocode.

```

For each attribute,
  For each value of that attribute, make a rule as follows:
    count how often each class appears
    find the most frequent class
    make the rule assign that class to this attribute-value.
  Calculate the error rate of the rules.
Choose the rules with the smallest error rate.

```

Figure 4.1 Pseudocode for 1R.**Table 4.1** Evaluating the attributes in the weather data.

	Attribute	Rules	Errors	Total errors
1	outlook	sunny → no overcast → yes rainy → yes	2/5 0/4 2/5	4/14
2	temperature	hot → no* mild → yes cool → yes	2/4 2/6 1/4	5/14
3	humidity	high → no normal → yes	3/7 1/7	4/14
4	windy	false → yes true → no*	2/8 3/6	5/14

* A random choice was made between two equally likely outcomes.

To see the 1R method at work, consider the weather data of Table 1.2 (we will encounter it many times again when looking at how learning algorithms work). To classify on the final column, *play*, 1R considers four sets of rules, one for each attribute. These rules are shown in Table 4.1. An asterisk indicates that a random choice has been made between two equally likely outcomes. The number of errors is given for each rule, along with the total number of errors for the rule set as a whole. 1R chooses the attribute that produces rules with the smallest number of errors—that is, the first and third rule sets. Arbitrarily breaking the tie between these two rule sets gives:

```

outlook: sunny   → no
          overcast → yes
          rainy   → yes

```

Table 4.2 The weather data with counts and probabilities.

	Outlook		Temperature		Humidity		Windy		Play				
	<i>yes</i>	<i>no</i>	<i>yes</i>	<i>no</i>	<i>yes</i>	<i>no</i>	<i>yes</i>	<i>no</i>	<i>yes</i>	<i>no</i>			
sunny	2	3	hot	2	2	high	3	4	false	6	2	9	5
overcast	4	0	mild	4	2	normal	6	1	true	3	3		
rainy	3	2	cool	3	1								
sunny	2/9	3/5	hot	2/9	2/5	high	3/9	4/5	false	6/9	2/5	9/14	5/14
overcast	4/9	0/5	mild	4/9	2/5	normal	6/9	1/5	true	3/9	3/5		
rainy	3/9	2/5	cool	3/9	1/5								

Table 4.3 A new day.

Outlook	Temperature	Humidity	Windy	Play
sunny	cool	high	true	?

makes real-life datasets interesting is that the attributes are certainly not equally important or independent. But it leads to a simple scheme that again works surprisingly well in practice.

Table 4.2 shows a summary of the weather data obtained by counting how many times each attribute–value pair occurs with each value (*yes* and *no*) for *play*. For example, you can see from Table 1.2 that *outlook* is *sunny* for five examples, two of which have *play* = *yes* and three of which have *play* = *no*. The cells in the first row of the new table simply count these occurrences for all possible values of each attribute, and the *play* figure in the final column counts the total number of occurrences of *yes* and *no*. In the lower part of the table, we rewrote the same information in the form of fractions, or observed probabilities. For example, of the nine days that *play* is *yes*, *outlook* is *sunny* for two, yielding a fraction of 2/9. For *play* the fractions are different: they are the proportion of days that *play* is *yes* and *no*, respectively.

Now suppose we encounter a new example with the values that are shown in Table 4.3. We treat the five features in Table 4.2—*outlook*, *temperature*, *humidity*, *windy*, and the overall likelihood that *play* is *yes* or *no*—as equally important, independent pieces of evidence and multiply the corresponding fractions. Looking at the outcome *yes* gives:

$$\text{likelihood of } \textit{yes} = 2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14 = 0.0053.$$

The fractions are taken from the *yes* entries in the table according to the values of the attributes for the new day, and the final 9/14 is the overall fraction

representing the proportion of days on which *play* is *yes*. A similar calculation for the outcome *no* leads to

$$\text{likelihood of } no = 3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = 0.0206.$$

This indicates that for the new day, *no* is more likely than *yes*—four times more likely. The numbers can be turned into probabilities by normalizing them so that they sum to 1:

$$\text{Probability of } yes = \frac{0.0053}{0.0053 + 0.0206} = 20.5\%,$$

$$\text{Probability of } no = \frac{0.0206}{0.0053 + 0.0206} = 79.5\%.$$

This simple and intuitive method is based on Bayes’s rule of conditional probability. Bayes’s rule says that if you have a hypothesis H and evidence E that bears on that hypothesis, then

$$\Pr[H|E] = \frac{\Pr[E|H] \Pr[H]}{\Pr[E]}.$$

We use the notation that $\Pr[A]$ denotes the probability of an event A and that $\Pr[A|B]$ denotes the probability of A conditional on another event B . The hypothesis H is that *play* will be, say, *yes*, and $\Pr[H|E]$ is going to turn out to be 20.5%, just as determined previously. The evidence E is the particular combination of attribute values for the new day, *outlook* = *sunny*, *temperature* = *cool*, *humidity* = *high*, and *windy* = *true*. Let’s call these four pieces of evidence E_1 , E_2 , E_3 , and E_4 , respectively. Assuming that these pieces of evidence are independent (given the class), their combined probability is obtained by multiplying the probabilities:

$$\Pr[yes|E] = \frac{\Pr[E_1|yes] \times \Pr[E_2|yes] \times \Pr[E_3|yes] \times \Pr[E_4|yes] \times \Pr[yes]}{\Pr[E]}.$$

Don’t worry about the denominator: we will ignore it and eliminate it in the final normalizing step when we make the probabilities of *yes* and *no* sum to 1, just as we did previously. The $\Pr[yes]$ at the end is the probability of a *yes* outcome without knowing any of the evidence E , that is, without knowing anything about the particular day referenced—it’s called the *prior probability* of the hypothesis H . In this case, it’s just 9/14, because 9 of the 14 training examples had a *yes* value for *play*. Substituting the fractions in Table 4.2 for the appropriate evidence probabilities leads to

$$\Pr[yes|E] = \frac{2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14}{\Pr[E]},$$



ourmine
data mining toolkit

 Search projects

[Project Home](#) [Downloads](#) [Wiki](#) [Issues](#) [Source](#) [Administer](#)

[New page](#) | Search for | [Edit this page](#) | [Delete this page](#)

★ **LectureTrees**

Learning Trees

Updated Oct 29, 2009 by [menzies.tim](#)

In this lecture, we discuss "iterative dichotomization"

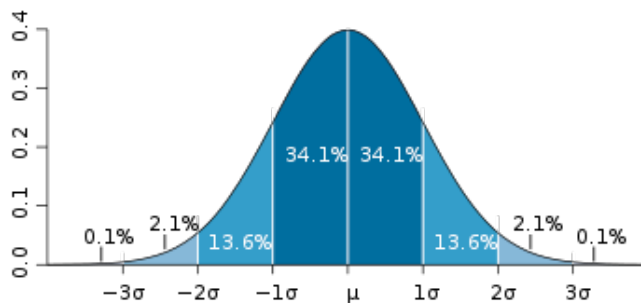
- Split things up
- Recurse on each split

How to generate a tree

- Given a bag of mixed-up stuff.
 - Need a measure of "mixed-up"
- Split: Find something that divides up the bag in two new sub-bags
 - And each sub-bag is less mixed-up;
 - Each split is the root of a sub-tree.
- Recurse: repeat for each sub-bag
 - i.e. on just the data that falls into each part of the split
 - Need a Stop rule
 - Condense the instances that fall into each sub-bag
- Prune back the generated tree.

Different tree learners result from different selections of

- CART: (regression trees)
 - measure: standard deviation
 - Three "normal" curves with [different standard deviations](#)
 - Expected values under the normal curve



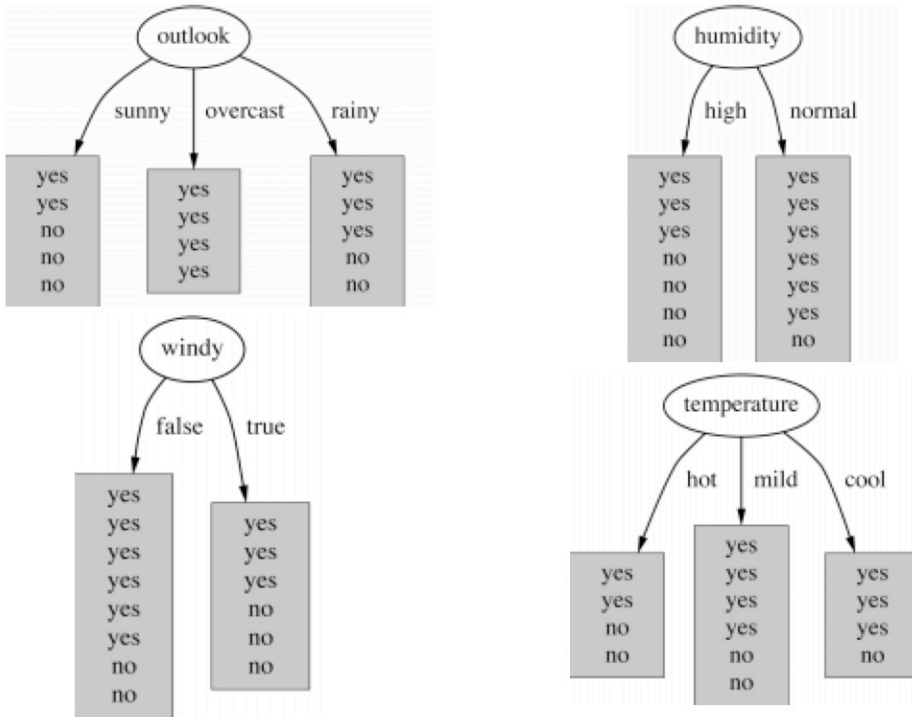
- condense: report the average of the instances in each bag.

- M5prime: (model trees)
 - measure: standard deviation
 - condense: generate a linear model of the form $a + b * x_1 + c * x_2 + d * x_3 + \dots$
- J48: (decision trees)
 - measure: "entropy"

$$\begin{aligned}
 H(X) = E(I(X)) &= \sum_{i=1}^n p(x_i) \log_2 (1/p(x_i)) \\
 &= -\sum_{i=1}^n p(x_i) \log_2 p(x_i)
 \end{aligned}$$

- condense: report majority class

Example: C4.5 (a.k.a. J48)



Q: which attribute is the best to split on?

A: the one which will result in the smallest tree:

Heuristic: choose the attribute that produces the "purest" nodes (purity = not-mixed-up)

e.g. Outlook= sunny

- $\text{info}((2,3)) = \text{entropy}(2/5, 3/5) = -2/5 * \log(2/5) - 3/5 * \log(3/5) = 0.971$ bits

Outlook = overcast

- $\text{info}((4,0)) = \text{entropy}(1,0) = -1 * \log(1) - 0 * \log(0) = 0$ bits

Outlook = rainy

- $\text{info}((3,2)) = \text{entropy}(3/5, 2/5) = -3/5 * \log(3/5) - 2/5 * \log(2/5) = 0.971$ bits

Expected info for Outlook = Weighted sum of the above

- $\text{info}((3,2), (4,0), (3,2)) = 5/14 * 0.971 + 4/14 * 0 + 5/14 * 0.971 = 0.693$

Computing the information gain

- e.g. information before splitting minus information after splitting
- e.g. gain for attributes from weather data:
- $\text{gain}(\text{"Outlook"}) = \text{info}(9,5) - \text{info}(2,3,4,0,3,2) = 0.940 - 0.693 = 0.247$ bits
- $\text{gain}(\text{"Temperature"}) = 0.247$ bits
- $\text{gain}(\text{"Humidity"}) = 0.152$ bits
- $\text{gain}(\text{"Windy"}) = 0.048$ bits

Problem: Numeric Variables

No problem:

- use [cliff learning](#) to split the numerics

- Standard method proposed by [Fayyad](#).

Problem: Highly-branching attributes

Problematic:

- attributes with a large number of values (extreme case: ID code)
- Subsets are more likely to be pure if there is a large number of values
- Information gain is biased towards choosing attributes with a large number of values
- This may result in over fitting (selection of an attribute that is non-optimal for prediction); e.g.

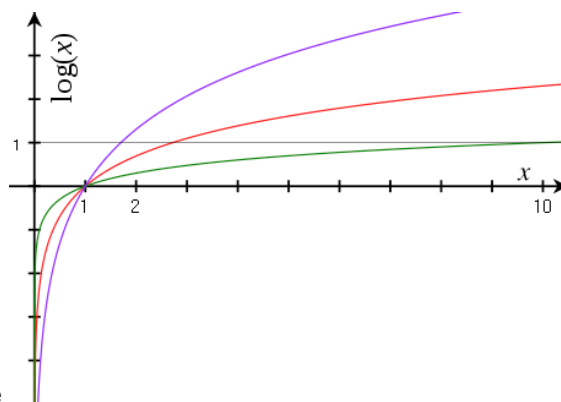
ID code	Outlook	Temp.	Humidity	Windy	Play
A	Sunny	Hot	High	False	No
B	Sunny	Hot	High	True	No
C	Overcast	Hot	High	False	Yes
D	Rainy	Mild	High	False	Yes
E	Rainy	Cool	Normal	False	Yes
F	Rainy	Cool	Normal	True	No
G	Overcast	Cool	Normal	True	Yes
H	Sunny	Mild	High	False	No
I	Sunny	Cool	Normal	False	Yes
J	Rainy	Mild	Normal	False	Yes
K	Sunny	Mild	Normal	True	Yes
L	Overcast	Mild	High	True	Yes
M	Overcast	Hot	Normal	False	Yes
N	Rainy	Mild	High	True	No %%

- If we split on ID we get N sub-trees with one class in each;
- $\text{info}(\text{"ID code"}) = \text{info}((0,1)) + \text{info}((0,1)) + \dots + \text{info}((0,1)) = 0$ bits
- So the info gain is 0.940 bits

Solution 1: Ignore columns with more than N unique values (where $N = m \cdot \text{NumberOfRows}$?). Solution 1: the gain ratio

- Gain ratio: a modification of the information gain that reduces its bias
- Gain ratio takes number and size of branches into account when choosing an attribute
- It corrects the information gain by taking the intrinsic information of a split into account
- Intrinsic informations: entropy of distribution of instances into branches (i.e. how much info do we need to tell which branch an instance belongs to)

Problem: Conclusion Instability



Recall the shape of a log curve

Now play with small values of $-p \cdot \log(p)$; consider what happens if I change those values by plus/minus 30%.

The great ones say:

- It is well known that the error rate of a tree on the cases from which it was constructed (the re-substitution error rate) is much lower than the error rate on unseen cases (the predictive error rate).
- For example, on a well-known letter recognition dataset with 20,000 cases, the re-substitution error rate for C4.5 is 4%, but the error rate from a leave-one-out (20,000-fold) cross-validation is 11.7%.
- As this demonstrates, leaving out a single case from 20,000 often affects the tree that is constructed!

Problem: Comprehension

Trees can hide simple rules

Naive conversion

- One rule for each leaf:
- Antecedent contains a condition for every node on the path from the root to the leaf

Problems with Naive Conversion

Example 1: Simple rules, complex tree

- Rules:

```
If a and b then x
If c and d then x
```

- Tree:

```
if a == y
then if b == y
    then x
    else if b == n
        then if c == y
            then if d == y
                then x
        else if b == n
            then if c == y
                then if d == y
                    then x
```

Example 2: Simple tree, complex rules

- Tree:

```
if x == 1
then if y == 1
    then b
    else a
else if y == 1
    then a
    else b
```

- Rules (unaware of structure)

```
if x == 1 and y == 0 then a
if x == 0 and y == 1 then a
if x == 0 and y == 0 then b
if x == 1 and y == 1 then b
```

Example 3: the replicated subtree problem

- Rules:

```
if x == 1 and y == 1 then a
if z == 1 and w == 1 then a
otherwise b
```

- Tree:

```
if x == 1
then if y == 1
    then a
    else if y == 2
        then SUB
        else if y == 3
            then SUB
else if y == 2
    then SUB
```

```

else if y == 3
then SUB
SUB : ( if z == 1
then if w == 1
then a
else if w == 2
then b
else if w == 3
then b
else if z == 2
then b
else if z == 3
then b
)

```

Other tree learning

Regression trees

- Differences to decision trees:
- Splitting criterion: minimizing intra-subset variation
- Pruning criterion: based on numeric error measure
- Leaf node predicts average class values of training instances reaching that node
- Can approximate piecewise constant functions
- Easy to interpret:

```

curb-weight <= 2660 :
|   curb-weight <= 2290 :
|   |   curb-weight <= 2090 :
|   |   |   length <= 161 : price=6220
|   |   |   length > 161 : price=7150
|   |   |   curb-weight > 2090 : price=8010
|   |   |   curb-weight > 2290 :
|   |   |   |   length <= 176 : price=9680
|   |   |   |   length > 176 :
|   |   |   |   |   normalized-losses <= 157 : price=10200
|   |   |   |   |   normalized-losses > 157 : price=15800
|   |   |   |   curb-weight > 2660 :
|   |   |   |   |   width <= 68.9 : price=16100
|   |   |   |   |   width > 68.9 : price=25500

```

- More sophisticated version: model trees

Model trees

- Regression trees with linear regression functions at each node

```

curb-weight <= 2660 :
|   curb-weight <= 2290 : LM1
|   curb-weight > 2290 :
|   |   length <= 176 : LM2
|   |   length > 176 : LM3
|   curb-weight > 2660 :
|   |   width <= 68.9 : LM4
|   |   width > 68.9 : LM5
|   .
LM1: price = -5280 + 6.68 * normalized-losses
      + 4.44 * curb-weight
      + 22.1 * horsepower - 85.8 * city-mpg
      + 98.6 * highway-mpg
LM2: price = 9680
LM3: price = -1100 + 91 * normalized-losses
LM4: price = 9940 + 47.5 * horsepower
LM5: price = -19000 + 13.2 * curb-weight

```

- Linear regression applied to instances that reach a node after full regression tree has been built
- Only a subset of the attributes is used for LR

- Attributes occurring in subtree (+maybe attributes occurring in path to the root)
- Fast: overhead for LR not large because usually only a small subset of attributes is used in tree

Building the tree

- Splitting criterion: standard deviation reduction into i bins
- $SDR = sd(T) - \sum (|T_i| / |T| * sd(T_i))$
 - where $|T|$ = number of instances in that tree).
- Termination criteria (important when building trees for numeric prediction):
- Standard deviation becomes smaller than certain fraction of sd for full training set (e.g. 5%)
- Too few instances remain (e.g. less than four)

Smoothing (Model Trees)

- Naive method for prediction outputs value of LR for corresponding leaf node
- Performance can be improved by smoothing predictions using internal LR models
- Predicted value is weighted average of LR models along path from root to leaf
- Smoothing formula: $p' = (np+kq)/(n+k)$
- p' is what gets passed up the tree
- p is what got passed from down the tree
- q is the value predicted by the linear models at this node
- n is the number of examples that fall down to here
- k magic smoothing constant; default=2

Enter a comment:

Submit

Wiki markup help

[hide](#)

```
=Heading1=
==Heading2==
===Heading3===
```

```
*bold*  _italic_
`inline code`
escape: ``
```

```
Indent lists 2 spaces:
* bullet item
# numbered list
```

```
{{
verbatim code block
}}
```

```
Horizontal rule
----
```

```
WikiWordLink
[http://domain/page label]
http://domain/page
```

```
|| table || cells ||
```

[More examples](#) 

“Data Carving”: A geometric view of data mining

- Data is like a block of marble,
 - waiting for a sculptor (that’s you)
 - to find the shape within
- So “data mining” is really “data carving”
 - chipping away the irrelevancies
 - To find what lies beneath.

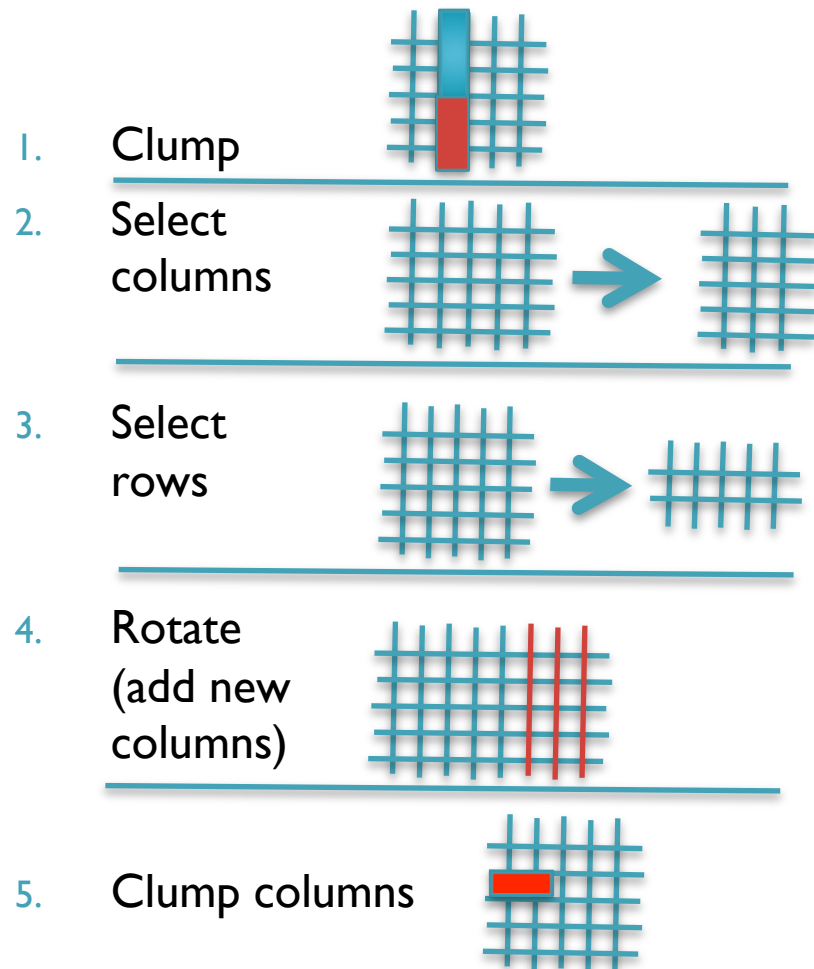


Four operators of data carving

```

@attribute outlook {sunny, overcast, rainy}
@attribute temperature real
@attribute humidity real
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}
@data
sunny,      85,85,FALSE, no
sunny,      80,90,TRUE,  no
overcast,   83,86,FALSE, yes
rainy,      70,96,FALSE, no
rainy,      68,80,FALSE, yes
rainy,      65,70,TRUE,  no
overcast,   64,65,TRUE,  yes
sunny,      72,95,FALSE, no
sunny,      69,70,FALSE, yes
rainy,      75,80,FALSE, yes
sunny,      75,70,TRUE,  yes
overcast,   72,90,TRUE,  yes
overcast,   81,75,FALSE, yes
rainy,      71,91,TRUE,  no
    
```

- Each example is a row in a table
- What can we do to change the table geometry?



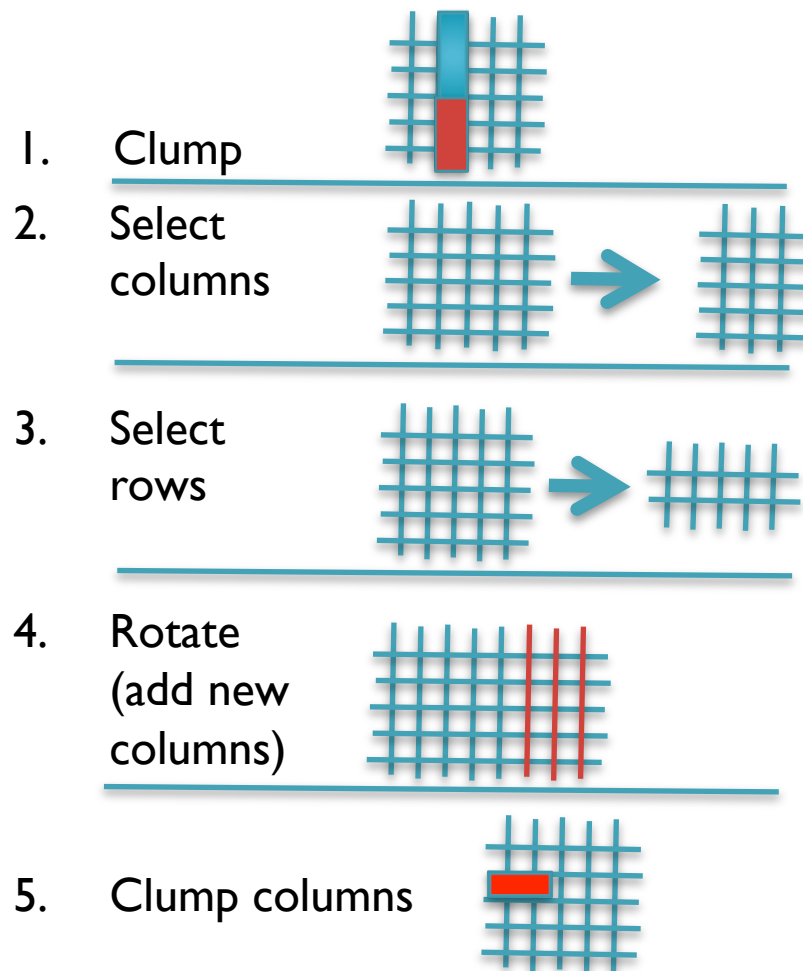
To understand data mining, look at the data, not the algorithms

- Why? We do data mining not to study algorithms.
 - But to study data
- Our results should be insights about data,
 - not trivia about (say) decision tree algorithms
- Besides, the thing that most predicts for performance is the data, not the algorithm,
 - [Pedro Domingos and Michael J. Pazzani, On the Optimality of the Simple Bayesian Classifier under Zero-One Loss, Machine Learning, Volume 29, number 2-3, pages 103-130, 1997](#)

Table 1. Classification accuracies and sample standard deviations, averaged over 20 random training/test splits. “Bayes” is the Bayesian classifier with discretization and “Gauss” is the Bayesian classifier with Gaussian distributions. Superscripts denote confidence levels for the difference in accuracy between the Bayesian classifier and the corresponding algorithm, using a one-tailed paired *t* test: 1 is 99.5%, 2 is 99%, 3 is 97.5%, 4 is 95%, 5 is 90%, and 6 is below 90%.

Data Set	Bayes	Gauss	C4.5	PEBL5	CN2	Def.
Audiology	73.0±6.1	73.0±6.1 ⁶	72.5±5.8 ⁶	75.8±5.4 ³	71.0±5.1 ⁵	21.3
Annealing	95.3±1.2	84.3±3.8 ¹	90.5±2.2 ¹	98.8±0.8 ¹	81.2±5.4 ¹	76.4
Breast cancer	71.6±4.7	71.3±4.3 ⁶	70.1±6.8 ⁵	65.6±4.7 ¹	67.9±7.1 ¹	67.6
Credit	84.5±1.8	78.9±2.5 ¹	85.9±2.1 ³	82.2±1.9 ¹	82.0±2.2 ¹	57.4
Chess endgames	88.0±1.4	88.0±1.4 ⁶	99.2±0.1 ¹	96.9±0.7 ¹	98.1±1.0 ¹	52.0
Diabetes	74.5±2.4	75.2±2.1 ⁶	73.5±3.4 ⁵	71.1±2.4 ¹	73.8±2.7 ⁶	66.0
Echocardiogram	69.1±5.4	73.4±4.9 ¹	64.7±6.3 ¹	61.7±6.4 ¹	68.2±7.2 ⁶	67.8
Glass	61.9±6.2	50.6±8.2 ¹	63.9±8.7 ⁶	62.0±7.4 ⁶	63.8±5.5 ⁶	31.7
Heart disease	81.9±3.4	84.1±2.8 ¹	77.5±4.3 ¹	78.9±4.0 ¹	79.7±2.9 ³	55.0
Hepatitis	85.3±3.7	85.2±4.0 ⁶	79.2±4.3 ¹	79.0±5.1 ¹	80.3±4.2 ¹	78.1
Horse colic	80.7±3.7	79.3±3.7 ¹	85.1±3.8 ¹	75.7±5.0 ¹	82.5±4.2 ²	63.6
Hypothyroid	97.5±0.3	97.9±0.4 ¹	99.1±0.2 ¹	95.9±0.7 ¹	98.8±0.4 ¹	95.3
Iris	93.2±3.5	93.9±1.9 ⁶	92.6±2.7 ⁶	93.5±3.0 ⁶	93.3±3.6 ⁶	26.5
Labor	91.3±4.9	88.7±10.6 ⁶	78.1±7.9 ¹	89.7±5.0 ⁶	82.1±6.9 ¹	65.0
Lung cancer	46.8±13.3	46.8±13.3 ⁶	40.9±16.3 ⁵	42.3±17.3 ⁶	38.6±13.5 ³	26.8
Liver disease	63.0±3.3	54.8±5.5 ¹	65.9±4.4 ¹	61.3±4.3 ⁶	65.0±3.8 ³	58.1
LED	62.9±6.5	62.9±6.5 ⁶	61.2±8.4 ⁶	55.3±6.1 ¹	58.6±8.1 ²	8.0
Lymphography	81.6±5.9	81.1±4.8 ⁶	75.0±4.2 ¹	82.9±5.6 ⁶	78.8±4.9 ³	57.3
Post-operative	64.7±6.8	67.2±5.0 ³	70.0±5.2 ¹	59.2±8.0 ²	60.8±8.2 ⁴	71.2
Promoters	87.9±7.0	87.9±7.0 ⁶	74.3±7.8 ¹	91.7±5.9 ³	75.9±8.8 ¹	43.1
Primary tumor	44.2±5.5	44.2±5.5 ⁶	35.9±5.8 ¹	30.9±4.7 ¹	39.8±5.2 ¹	24.6
Solar flare	68.5±3.0	68.2±3.7 ⁶	70.6±2.9 ¹	67.6±3.5 ⁶	70.4±3.0 ²	25.2
Sonar	69.4±7.6	63.0±8.3 ¹	69.1±7.4 ⁶	73.8±7.4 ¹	66.2±7.5 ⁵	50.8
Soybean	100.0±0.0	100.0±0.0 ⁶	95.0±9.0 ³	100.0±0.0 ⁶	96.9±5.9 ³	30.0
Splice junctions	95.4±0.6	95.4±0.6 ⁶	93.4±0.8 ¹	94.3±0.5 ¹	81.5±5.5 ¹	52.4
Voting records	91.2±1.7	91.2±1.7 ⁶	96.3±1.3 ¹	94.9±1.2 ¹	95.8±1.6 ¹	60.5
Wine	96.4±2.2	97.8±1.2 ³	92.4±5.6 ¹	97.2±1.8 ⁶	90.8±4.7 ¹	36.4
Zoology	94.4±4.1	94.1±3.8 ⁶	89.6±4.7 ¹	94.6±4.3 ⁶	90.6±5.0 ¹	39.4

The rest of this hour

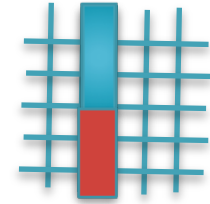


Carving can be dangerous

- While carving the training data is recommended
 - It is a methodological error to carve the test data
- Whatever is learned from the training data
 - Should be assessed on “raw” (i.e. uncarved) test data



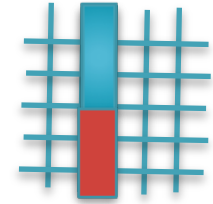
Clumping column data (a.k.a. discretization)



overcast,	64,	65,	TRUE,	yes
Rainy,	65,	70,	TRUE,	no
sunny,	69,	70,	FALSE,	yes
sunny,	75,	70,	TRUE,	yes
overcast,	81,	75,	FALSE,	yes
rainy,	68,	80,	FALSE,	yes
rainy,	75,	80,	FALSE,	yes
sunny,	85,	85,	FALSE,	no
overcast,	83,	86,	FALSE,	yes
overcast,	72,	90,	TRUE,	yes
sunny,	80,	90,	TRUE,	no
rainy,	71,	91,	TRUE,	no
sunny,	72,	95,	FALSE,	no
rainy,	70,	96,	FALSE,	no

- Learning = compression
 - Take a target concept that is spread out across all the data
 - Squeeze it together till it is dense enough to be visible.
- Discretization: clump together observations taken over a continuous range
 - into a small number of regions.
- E.g. "toddlers" If age = 1,2,3
- Discretization improves the performance of a learner
 - Gives a learner a smaller space to reason about,
 - With more examples in each part of the space

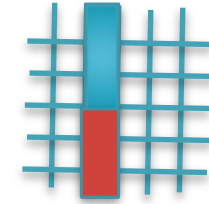
Supervised Discretization



overcast,	64,	65,	TRUE,	yes
Rainy,	65,	70,	TRUE,	no
sunny,	69,	70,	FALSE,	yes
sunny,	75,	70,	TRUE,	yes
overcast,	81,	75,	FALSE,	yes
rainy,	68,	80,	FALSE,	yes
rainy,	75,	80,	FALSE,	yes
sunny,	85,	85,	FALSE,	no
overcast,	83,	86,	FALSE,	yes
overcast,	72,	90,	TRUE,	yes
sunny,	80,	90,	TRUE,	no
rainy,	71,	91,	TRUE,	no
sunny,	72,	95,	FALSE,	no
rainy,	70,	96,	FALSE,	no

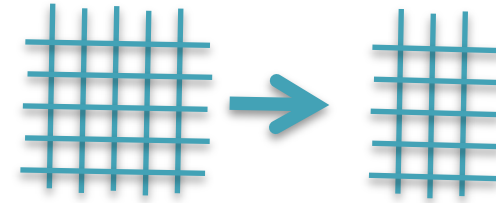
- Standard method:
 - Find a break that most reduces class diversity either side of the break
 - Recurse on data:
 - above break,
 - below break
 - [Fayyad and Irani, Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning IJCAI'93, pp1022-1027](#)

Unsupervised Discretization



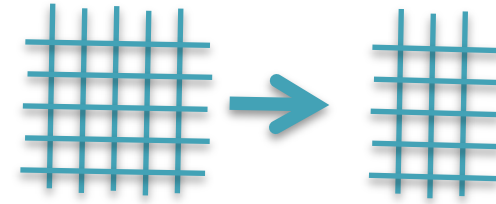
- Divide into “B” bins
 - $(X - \text{Min}) / ((\text{Max} - \text{Min}) / B)$
 - B=3 or 10 very common
- Divide into P percentile groups
 - Each bins contains (say) 25% of the rows
- For Bayesian methods
 - Divide into groups of N items
 - Ying and Webb recommends $N = \sqrt{\text{rows}}$
 - Ying Yang and Geoff Webb, Weighted Proportional k-Interval Discretization of Naïve Bayes classifiers, PAKDD'03, p501-512, 2003

Select columns

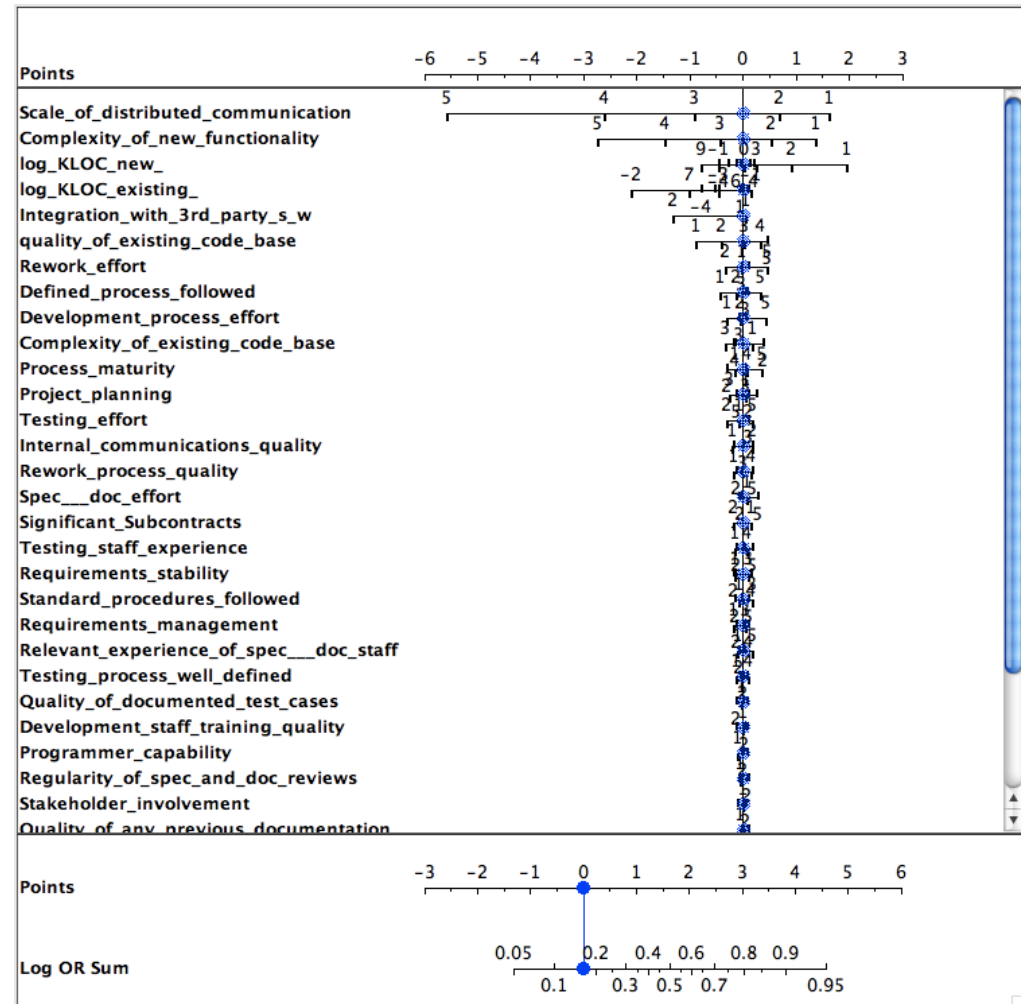


- Occam's Razor - Entia non sunt multiplicanda praeter necessitatem. ("Entities should not be multiplied more than necessary").
 - the fewer features used to explain something, the better
- Log(OR):
 - Discrete every feature. For all pairs of target / other of size C_1, C_2 count frequency of range N_1, N_2 in each class
 - $\text{Log}(\text{odds ratio}) = \log((N_1/C_1) / (N_2/C_2)) > 0$ if more frequent in target
 - "Pivots" are the ranges with high Log (OR)
 - [Možina, M., Demšar, J., Kattan, M., and Zupan, B. 2004. Nomograms for visualization of naive Bayesian classifier. In Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases \(Pisa, Italy, September 20 - 24, 2004\)](#)
- InfoGain:
 - Use Fayyad Irani trick: asses each column by how well it divides up the data
 - Takes linear time : $O(C)$
- Wrapper:
 - Explore 2^C subsets of C columns: takes time $O(2^C)$
 - Call a learner on each subset
 - Use the columns that maximize learner performance
 - Not practical for large data sets
- For more, see [Hall, M. and Holmes, G. \(2003\). Benchmarking attribute selection techniques for discrete class data mining. IEEE Transactions on Knowledge and Data Engineering. 15\(3\), November/December 2003](#)

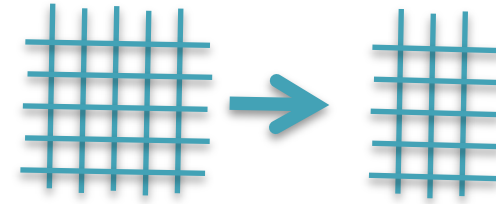
Select columns with log(OR)



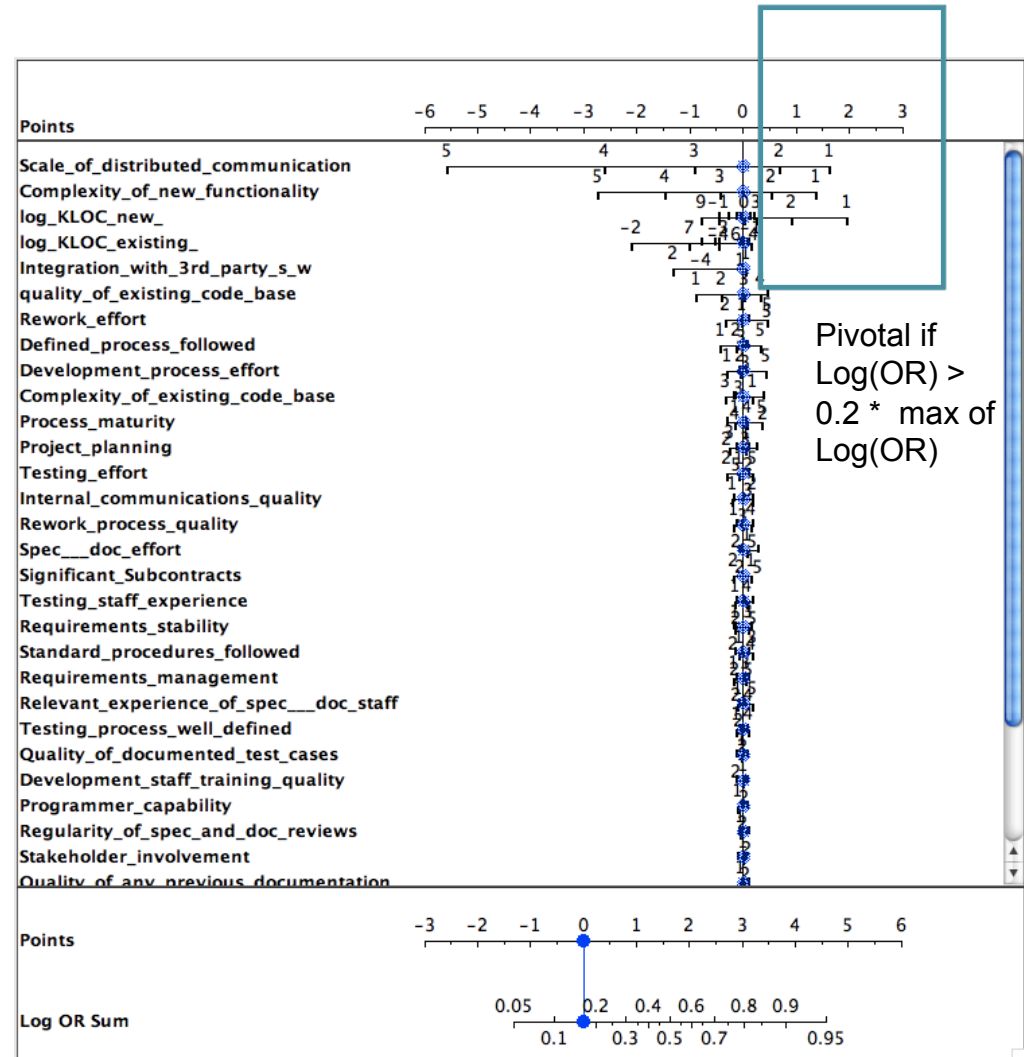
- Data from Norman Fenton's Bayes Net
 - Project Data Incorporating Qualitative Factors for Improved Software Defect Prediction Norman Fenton, Martin Neil, William Marsh, Peter Hearty, Lukasz Radlinski and Paul Krause., PROMISE 2008
- Target class. worse defects
- Only a few features matter
- Only a few ranges of those features matter



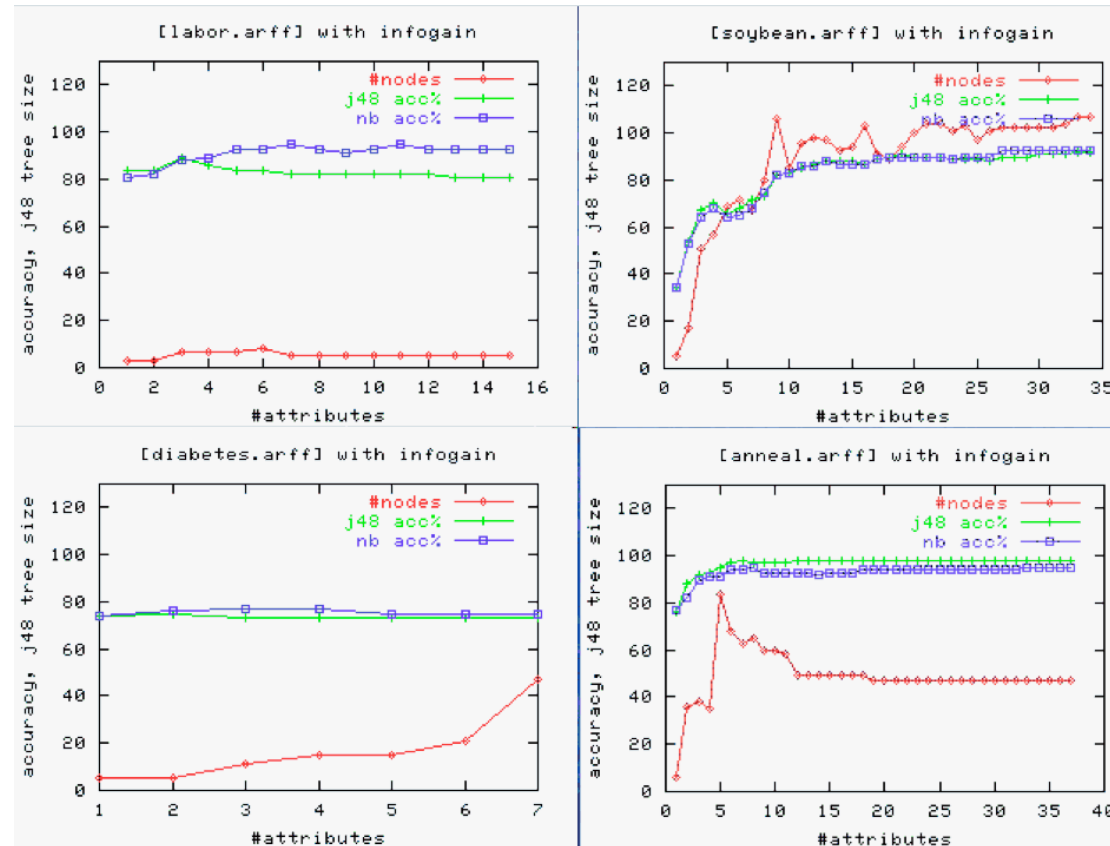
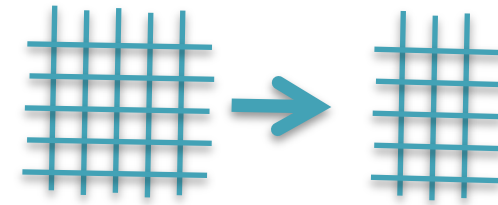
Select columns with log(OR)



- Data from Norman Fenton's Bayes Net
 - Project Data Incorporating Qualitative Factors for Improved Software Defect Prediction Norman Fenton, Martin Neil, William Marsh, Peter Hearty, Lukasz Radlinski and Paul Krause., PROMISE 2008
- Target class. worse defects
- Only a few features matter
- Only a few ranges of those features matter

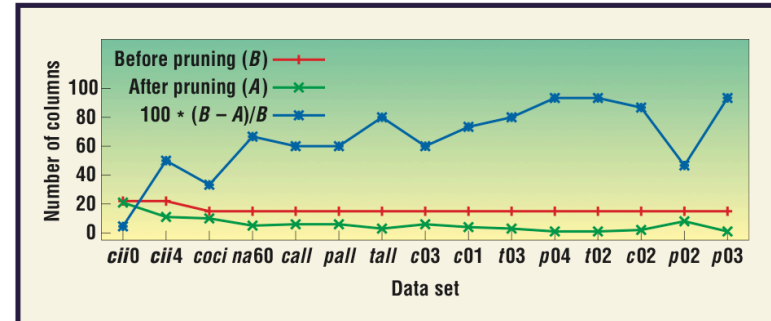
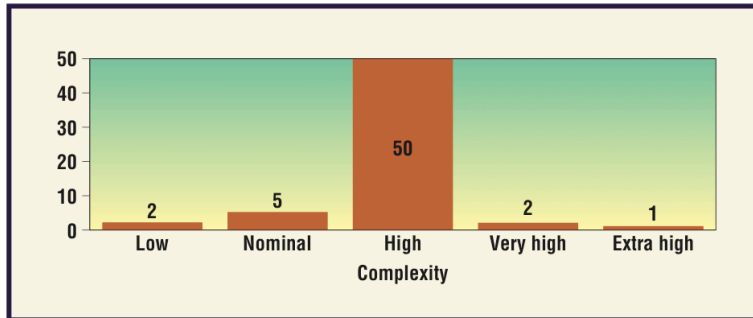
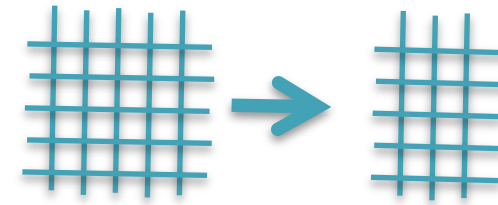


Select columns with InfoGain

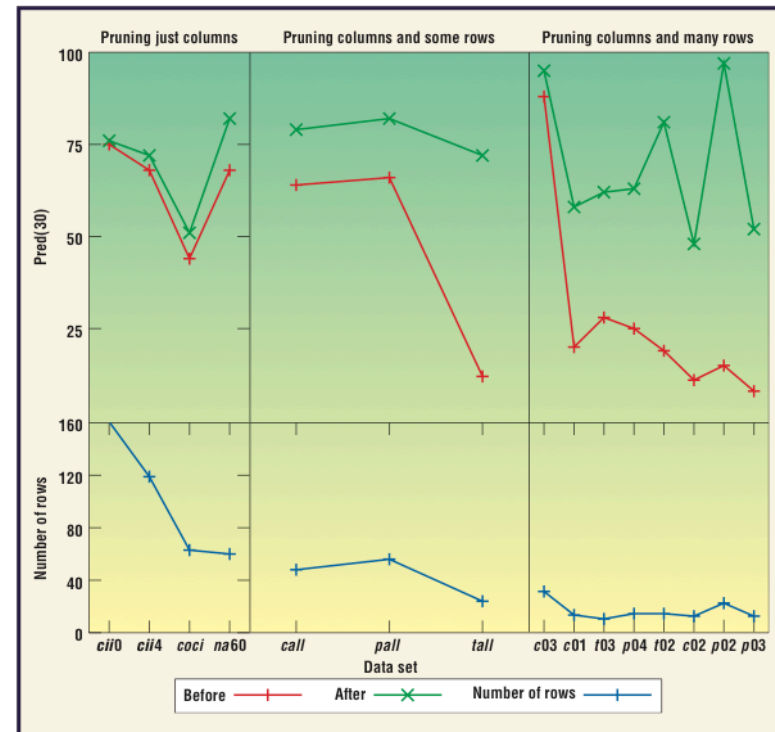


Simpler theories after column selection.
Work just as well as using everything

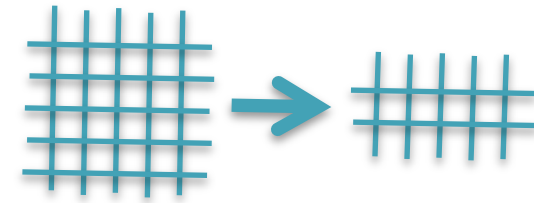
Select columns with WRAPPER



- Finding the Right Data for Software Cost Modeling
Chen, Menzies, Port, Boehm,
IEEE Software Nov/Dec 2005

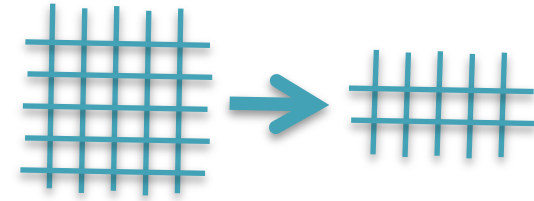


Select rows



- Replace N rows
 - with $M < N$ rows
 - that best exemplify the data
- Typical result:
 - Can throw out 80 to 90% of the rows without losing accuracy
 - C. Chang, "Finding prototypes for nearest neighbor classifiers," IEEE Trans. on Computers, pp. 1179–1185, 1974.
- Benefits:
 - Outlier removal
 - Any downstream processing is faster
 - E.g. any $O(N^2)$ process is 100 times faster on $N/10$ of the data
 - Less errors in conclusions
 - Instance learner: classify according to nearest neighbors
 - If nearest neighbors further away, harder for data collection errors to cause wrong classifications
 - Easier to visualize
 - Fewer things to look at

Select rows

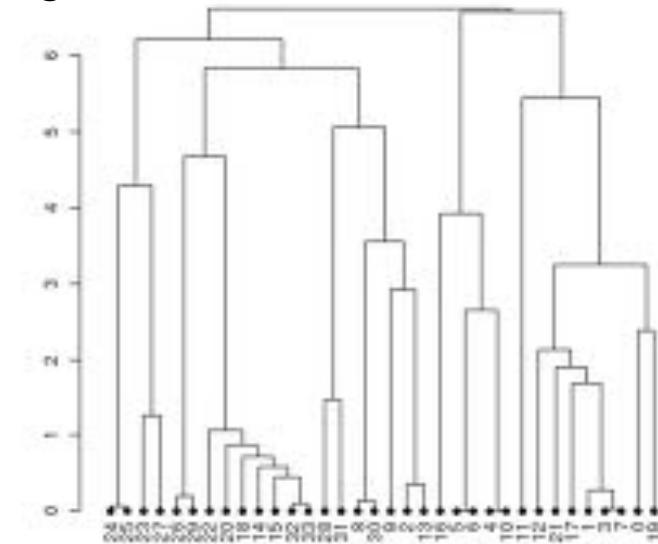


- Exponential time

- Genetic algorithm to explore the 2^R subsets of rows.
 - When more rows than columns, even slower than the WRAPPER's $O(2^C)$ search
 - Y.Li, M.Xie, and T.Goh, "A study of project selection and feature weighting for analogy based software cost estimation," *Journal of Systems and Software*, vol. 82, pp. 241–252, 2009.

- Polynomial time: Greedy agglomerative clustering

- Push every instance to its closest neighbor.
- Build a synthetic example at each pair's median
- Repeat for the synthetic points.
- Prototypes are all nodes at level X of GAC tree
- For R rows, $O(R^2)$



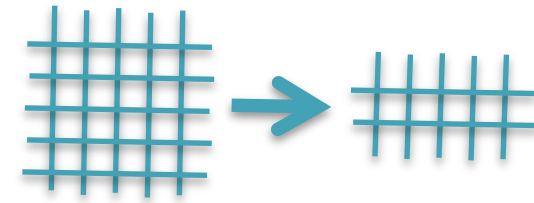
- TEAK = GAC plus ...

- Prune sub-trees with large variance
- [When to Use Data from Other Projects for Effort Estimation](#) Ekrem Kocaguneli, Gregory Gay, Tim Menzies, Ye Yang, Jacky W. Keung, ASE 2010

- Linear-time

- Rank ranges by frequency delta in different classes
- Discard all rows that do not have the top R pivots

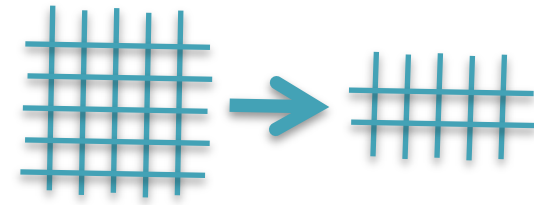
Select rows (with TEAK)



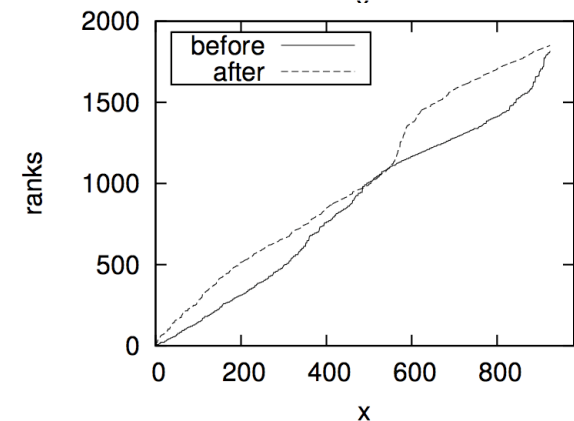
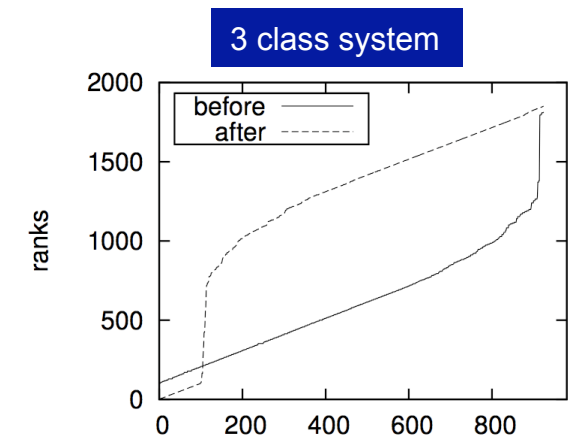
- To effort estimate a test instance, start at root of GAC tree
 - Move to nearest child
 - Stop at leaf or when sub-tree variance greater than super-tree
 - Estimate = median of instances in that sub-tree
- Compared with
 - linear regression,
 - neural nets,
 - analogy methods that use $K=1,2,4$ nearest neighbors (no variance pruning)
- Compared using
 - $20 * \{\text{shuffle rows, 3-way cross-val}\}$
 - $\#\text{wins} - \#\text{losses}$ (in a Wilcoxon, 95%)
 - Count number of times ranked first by this procedure
- Conclusion: row-selection using clustering + variance pruning is a good thing

	TEAK	LR	NNet	Best(K)	k=1	k=16	k=2	k=4	k=8
MRE									
Cocomo81	▲								
Cocomo81e	▲								
Cocomo81o	▲								
Nasa93		▲							
Nasa93c2		▲							
Nasa93c5	▲								
Desharnais		▲							
Sdr	▲								
ISBSG-Banking	▲								
Count	6	3	0	0	0	0	0	0	0
Pred(25)									
Cocomo81	▲								
Cocomo81e			▲						
Cocomo81o	▲								
Nasa93		▲							
Nasa93c2		▲							
Nasa93c5	▲								
Desharnais		▲							
Sdr	▲								
ISBSG-Banking	▲								
Count	5	3	1	0	0	0	0	0	0
AR									
Cocomo81				▲					
Cocomo81e	▲								
Cocomo81o	▲								
Nasa93		▲							
Nasa93c2		▲							
Nasa93c5	▲								
Desharnais		▲							
Sdr	▲								
ISBSG-Banking	▲								
Count	6	3	0	1	0	0	0	0	0

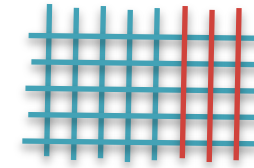
Select rows (with range pruning)



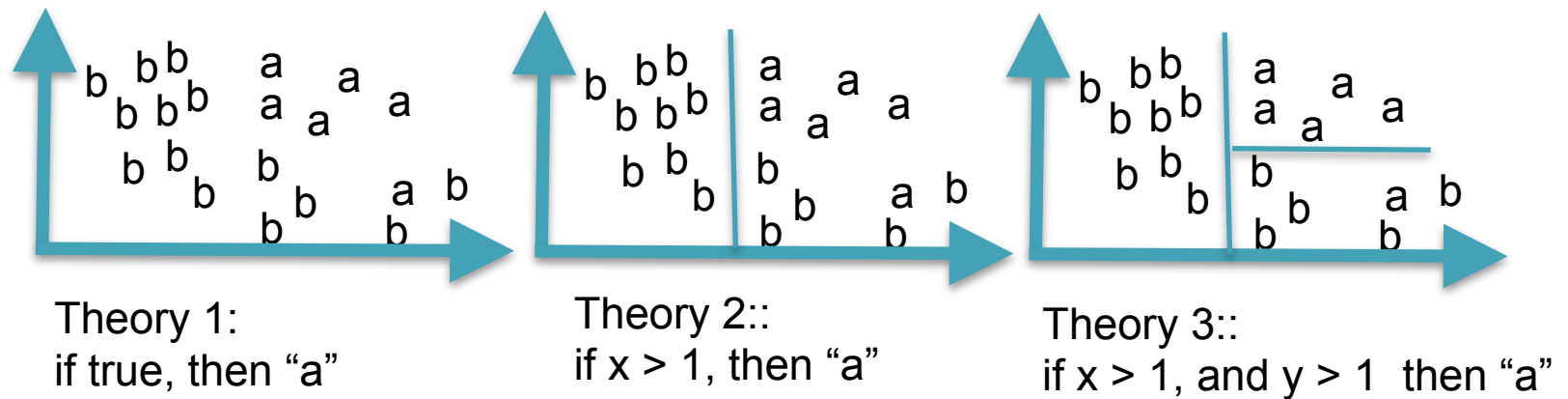
- For K in Classes
 - Let NotK = Classes – K
 - Let N1, N2 be number of rows with K and NotK classes
 - For C in columns
 - For R in range of column C
 - Let F1, F2 = frequency of C.R in K and NotK
 - Let $x = F1 / N1$ and $y = F2 / N2$
 - Let $R.score = x^2 / (x + y)$
;;; pivotal if R far more frequent in K than NotK
- Remove all rows without the top five pivots
 - If accuracy of reduced set decreases, then ABORT.
- For each instance, find distance needed to travel before a K=5 nearest neighbor algorithm changes the classification.
 - In the full data set
 - In the reduced data set
- Result:
 - Much charger to change classification in reduced data set
- Conclusion: if concerned about errors in data collection, use row selection (and less classes)



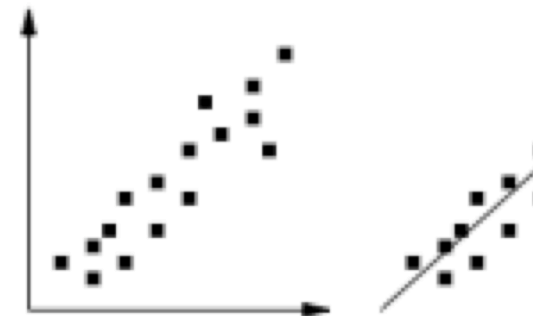
Rotate (add columns)



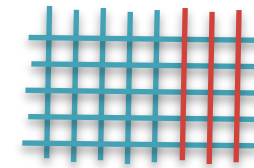
- Sometimes, the data's raw dimensions suffice for isolating the target concept..



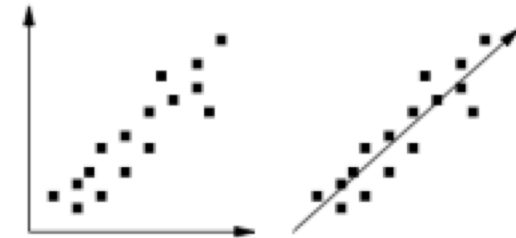
- But what if the target concept falls across and not along, the raw dimensions?



Rotate (add columns)



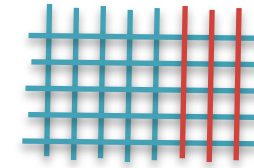
- Synthesize a new dimension that combines the raw into something new
- Apply single-valued decomposition (SVD) to
 - the covariance matrix (principal component analysis, or PCA)
 - or the data table (latent semantic indexing, or LSI)
- PCA that produces a set of orthogonal “components”
 - Transforms C correlated variables into fewer uncorrelated “components”.
 - Component[i] accounts for as much variability as possible.
 - Component[$i+1$] accounts for as much of the remaining variability as possible.



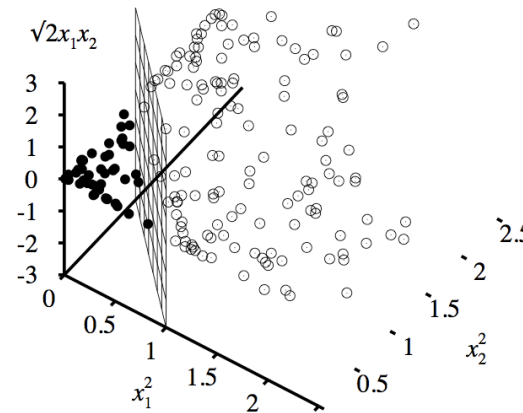
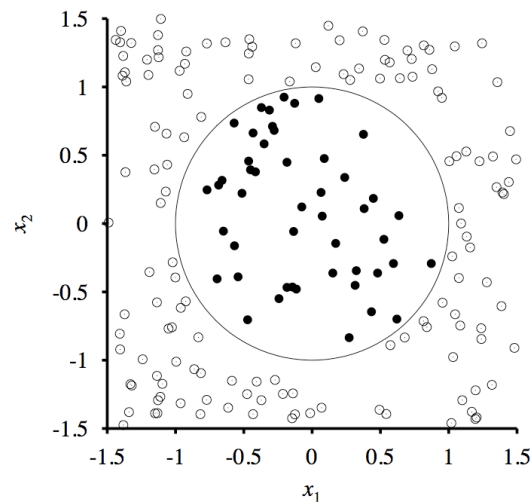
- Much easier to learn rules when dimensions match the data. E.g. a defect predictor:
- if $\text{comp}[1] \leq 0.180$
then NoDefects
else if $\text{comp}[1] > 0.180$
then if $\text{comp}[1] \leq 0.371$ then NoDefects
else if $\text{com}[1] > 0.371$ then Defects
- But it can be hard to explain that predictor:

$$\begin{aligned} \text{Comp}[1] = & 0.236*v(g) + 0.222*ev(g) \\ & + 0.236*iv(g) + 0.241*n + 0.238*v - 0.086*i \\ & + 0.199*d + 0.216*i + 0.225*e + 0.236*b \\ & + 0.221*t + 0.241*IOCode + 0.179*IOComment \\ & + 0.221*IOBlank + 0.158*IOCodeAndComment \\ & + 0.163*uniq_Op + 0.234*uniq_Opnd \\ & + 0.241*total_Op + 0.241*total_Opnd \\ & + 0.236*branchCount \end{aligned}$$

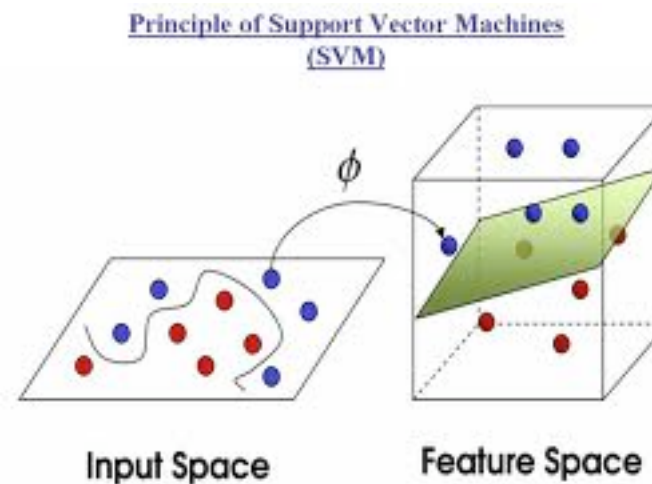
Rotate (add columns)



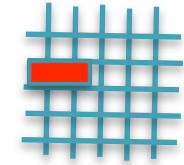
- Special transforms



- Support vector machines: construct a hyper-plane that separates classes



Clump rows (a.k.a. generalize)



- Ever notice that rows and rules have (nearly) the same syntax?
 - Age=young and wealth=rich and iq=high and class=happy
 - If age=old and wealth=rich and iq=high then happy
- But when we write rules, we only do it for frequently occurring patterns in the other rows
- “Clump rows” : replace them with a rule that covers many rows, but many only mention some of the columns
 - **If** age=old and wealth=rich **then** happy
- If you do this after clumping columns and selecting good rows and selecting good columns and (maybe) adding in good columns
 - Then the search space is very small
 - The exploring can be heavily biased by the other steps (e.g. look at great rows before dull ones)
 - And, hey presto, you’ve got a working data miner

Hints and tips (note: only my view)

- Always try clumping with discretization
 - So very simple
 - So experiment with / without discretization
- Always try column selection
 - Usually, massive reduction in the columns
- If the data won't fit in RAM,
 - try column selection first (use a linear-time approach)
 - then you can explore row selection by (say)
 - Era $[i]$: read first 1000 instances and apply row selection
 - Era $[i+1]$: read next 1000 records and ignore instances that fall close to the instances selected at Era $[i]$
- Try these last: PCA / Support vector machines
 - Benefits of PCA often achieved, or beaten by other column selectors
 - [Hall, M. and Holmes, G. \(2003\). Benchmarking attribute selection techniques for discrete class data mining. IEEE Trans on Knowledge and Data Engineering. 15\(3\), November/December 2003](#)
 - The FASTMAP heuristic FASTMAP, can do what PCA does, faster, scalable.
 - [Faloutsos, C. and Lin, K. 1995. FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In Proceedings of the 1995 ACM SIGMOD international Conference on Management of Data](#)
 - For text mining (PCA / LDA) vs TF*IDF never benchmarked



ourmine
data mining toolkit

[Project Home](#) [Downloads](#) [Wiki](#) [Issues](#) [Source](#) [Administer](#)

[New page](#) | Search for | [Edit this page](#) | [Delete this page](#)

★ [LectureNaiveBayes](#)

NaiveBayes Classifiers 101

Updated Oct 27, 2009 by [kelcecil](#)

- [Introduction](#)
- [Example](#)
- [Bayes' rule](#)
 - [Numerical errors](#)
 - [Missing values](#)
 - [The "low-frequencies problem"](#)
- [Pseudo-code](#)
 - [Simple version](#)
 - [More Complex](#)
- [Handling Numerics](#)
- [Simple Extensions](#)
 - [From Naive Bayes to Hyper Pipes](#)
 - [From to Incremental Learning](#)
 - [From to Anomaly Detection](#)
- [The Explanation Problem](#)
- [A Second Look at the Low-Frequency Problem](#)
- [Not so "Naive" Bayes](#)

Introduction

A Bayes classifier is a simple statistical-based learning scheme.

Advantages:

- Tiny memory footprint
- Fast training, fast learning
- Simplicity
- Often works surprisingly well

Assumptions

- Learning is done best via statistical modeling
- Attributes are
 - equally important
 - statistically independent (given the class value)
 - This means that knowledge about the value of a particular attribute doesn't tell us anything about the value of another attribute (if the class is known)
- Although based on assumptions that are almost never correct, this scheme works well in practice
[Domingos97](#)

Table 1. Classification accuracies and sample standard deviations, averaged over 20 random training/test splits. "Bayes" is the Bayesian classifier with discretization and "Gauss" is the Bayesian classifier with Gaussian distributions. Superscripts denote confidence levels for the difference in accuracy between the Bayesian classifier and the corresponding algorithm, using a one-tailed paired *t* test: 1 is 99.5%, 2 is 99%, 3 is 97.5%, 4 is 95%, 5 is 90%, and 6 is below 90%.

Data Set	Bayes	Gauss	C4.5	PEBLS	CN2	Def.
Audiology	73.0±6.1	73.0±6.1 ⁶	72.5±5.8 ⁶	75.8±5.4 ³	71.0±5.1 ⁵	21.3
Annealing	95.3±1.2	84.3±3.8 ¹	90.5±2.2 ¹	98.8±0.8 ¹	81.2±5.4 ¹	76.4
Breast cancer	71.6±4.7	71.3±4.3 ⁶	70.1±6.8 ⁵	65.6±4.7 ¹	67.9±7.1 ¹	67.6
Credit	84.5±1.8	78.9±2.5 ¹	85.9±2.1 ³	82.2±1.9 ¹	82.0±2.2 ¹	57.4
Chess endgames	88.0±1.4	88.0±1.4 ⁶	99.2±0.1 ¹	96.9±0.7 ¹	98.1±1.0 ¹	52.0
Diabetes	74.5±2.4	75.2±2.1 ⁶	73.5±3.4 ⁵	71.1±2.4 ¹	73.8±2.7 ⁶	66.0
Echocardiogram	69.1±5.4	73.4±4.9 ¹	64.7±6.3 ¹	61.7±6.4 ¹	68.2±7.2 ⁶	67.8
Glass	61.9±6.2	50.6±8.2 ¹	63.9±8.7 ⁶	62.0±7.4 ⁶	63.8±5.5 ⁶	31.7
Heart disease	81.9±3.4	84.1±2.8 ¹	77.5±4.3 ¹	78.9±4.0 ¹	79.7±2.9 ³	55.0
Hepatitis	85.3±3.7	85.2±4.0 ⁶	79.2±4.3 ¹	79.0±5.1 ¹	80.3±4.2 ¹	78.1
Horse colic	80.7±3.7	79.3±3.7 ¹	85.1±3.8 ¹	75.7±5.0 ¹	82.5±4.2 ²	63.6
Hypothyroid	97.5±0.3	97.9±0.4 ¹	99.1±0.2 ¹	95.9±0.7 ¹	98.8±0.4 ¹	95.3
Iris	93.2±3.5	93.9±1.9 ⁶	92.6±2.7 ⁶	93.5±3.0 ⁶	93.3±3.6 ⁶	26.5
Labor	91.3±4.9	88.7±10.6 ⁶	78.1±7.9 ¹	89.7±5.0 ⁶	82.1±6.9 ¹	65.0
Lung cancer	46.8±13.3	46.8±13.3 ⁶	40.9±16.3 ⁵	42.3±17.3 ⁶	38.6±13.5 ³	26.8
Liver disease	63.0±3.3	54.8±5.5 ¹	65.9±4.4 ¹	61.3±4.3 ⁶	65.0±3.8 ³	58.1
LED	62.9±6.5	62.9±6.5 ⁶	61.2±8.4 ⁶	55.3±6.1 ¹	58.6±8.1 ²	8.0
Lymphography	81.6±5.9	81.1±4.8 ⁶	75.0±4.2 ¹	82.9±5.6 ⁶	78.8±4.9 ³	57.3
Post-operative	64.7±6.8	67.2±5.0 ³	70.0±5.2 ¹	59.2±8.0 ²	60.8±8.2 ⁴	71.2
Promoters	87.9±7.0	87.9±7.0 ⁶	74.3±7.8 ¹	91.7±5.9 ³	75.9±8.8 ¹	43.1
Primary tumor	44.2±5.5	44.2±5.5 ⁶	35.9±5.8 ¹	30.9±4.7 ¹	39.8±5.2 ¹	24.6
Solar flare	68.5±3.0	68.2±3.7 ⁶	70.6±2.9 ¹	67.6±3.5 ⁶	70.4±3.0 ²	25.2
Sonar	69.4±7.6	63.0±8.3 ¹	69.1±7.4 ⁶	73.8±7.4 ¹	66.2±7.5 ⁵	50.8
Soybean	100.0±0.0	100.0±0.0 ⁶	95.0±9.0 ³	100.0±0.0 ⁶	96.9±5.9 ³	30.0
Splice junctions	95.4±0.6	95.4±0.6 ⁶	93.4±0.8 ¹	94.3±0.5 ¹	81.5±5.5 ¹	52.4
Voting records	91.2±1.7	91.2±1.7 ⁶	96.3±1.3 ¹	94.9±1.2 ¹	95.8±1.6 ¹	60.5
Wine	96.4±2.2	97.8±1.2 ³	92.4±5.6 ¹	97.2±1.8 ⁶	90.8±4.7 ¹	36.4
Zoology	94.4±4.1	94.1±3.8 ⁶	89.6±4.7 ¹	94.6±4.3 ⁶	90.6±5.0 ¹	39.4

It has some drawbacks: it can offer conclusions but it is poor at explaining how those conclusions were reached. But that is something we'll get back to below.

Example

```
weather.symbolic.arff
outlook  temperature  humidity  windy  play
-----  -
rainy    cool           normal   TRUE   no
rainy    mild           high     TRUE   no
sunny    hot            high     FALSE  no
sunny    hot            high     TRUE   no
sunny    mild           high     FALSE  no
overcast cool           normal   TRUE   yes
overcast hot            high     FALSE  yes
overcast hot            normal   FALSE  yes
overcast mild           high     TRUE   yes
rainy    cool           normal   FALSE  yes
rainy    mild           high     FALSE  yes
rainy    mild           normal   FALSE  yes
sunny    cool           normal   FALSE  yes
sunny    mild           normal   TRUE   yes
```

This data can be summarized as follows:

	Outlook		Temperature			Humidity		
	Yes	No	Hot	Yes	No	High	Yes	No
Sunny	2	3	Hot	2	2	High	3	4
Overcast	4	0	Mild	4	2	Normal	6	1
Rainy	3	2	Cool	3	1			

Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5
Overcast	4/9	0/5	Mild	4/9	2/5	Normal	6/9	1/5
Rainy	3/9	2/5	Cool	3/9	1/5			

	Windy		Play	
	Yes	No	Yes	No
False	6	2	9	5
True	3	3		
False	6/9	2/5	9/14	5/14
True	3/9	3/5		

So, what happens on a new day:

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	???

First find the likelihood of the two classes

- For "yes" = $2/9 * 3/9 * 3/9 * 3/9 * 9/14 = 0.0053$
- For "no" = $3/5 * 1/5 * 4/5 * 3/5 * 5/14 = 0.0206$
- Conversion into a probability by normalization:
 - $P(\text{"yes"}) = 0.0053 / (0.0053 + 0.0206) = 0.205$
 - $P(\text{"no"}) = 0.0206 / (0.0053 + 0.0206) = 0.795$

So, we aren't playing golf today.

Bayes' rule

More generally, the above is just an application of Bayes' Theorem.

- Probability of event H given evidence E:

$$\Pr(H | E) = \frac{\Pr(E | H) * \Pr(H)}{\Pr(E)}$$

- A priori probability of H = $\Pr(H)$
 - Probability of event before evidence has been seen
- A posteriori probability of H = $\Pr(H|E)$
 - Probability of event after evidence has been seen
- Classification learning: what's the probability of the class given an instance?
 - Evidence E = instance
 - Event H = class value for instance
- Naive Bayes assumption: evidence can be split into independent parts (i.e. attributes of instance!)

$$\Pr(H | E) = \frac{\Pr(E_1 | H) * \Pr(E_2 | H) * \dots * \Pr(E_n | H) * \Pr(H)}{\Pr(E)}$$

- We used this above. Here's our evidence:

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

- Here's the probability for "yes":

$$\begin{aligned} \Pr(\text{yes} | E) &= \Pr(\text{Outlook} = \text{Sunny} | \text{yes}) * \\ &\quad \Pr(\text{Temperature} = \text{Cool} | \text{yes}) * \\ &\quad \Pr(\text{Humidity} = \text{High} | \text{yes}) * \Pr(\text{yes}) \\ &\quad \Pr(\text{Windy} = \text{True} | \text{yes}) * \Pr(\text{yes}) / \Pr(E) \\ &= (2/9 * 3/9 * 3/9 * 3/9) * 9/14 / \Pr(E) \end{aligned}$$

Return the classification with highest probability

- Probability of the evidence $\Pr(E)$

- Constant across all possible classifications;
- So, when comparing N classifications, it cancels out

Numerical errors

From multiplication of lots of small numbers

- Use the standard fix: don't multiply the numbers, add the logs

Missing values

Missing values are a problem for any learner. Naive Bayes' treatment of missing values is particularly elegant.

- During training: instance is not included in frequency count for attribute value-class combination
- During classification: attribute will be omitted from calculation

Example:	Outlook	Temp.	Humidity	Windy	Play
	?	Cool	High	True	??%

- Likelihood of "yes" = $3/9 * 3/9 * 3/9 * 9/14 = 0.0238$
- Likelihood of "no" = $1/5 * 4/5 * 3/5 * 5/14 = 0.0343$
- $P(\text{"yes"}) = 0.0238 / (0.0238 + 0.0343) = 41\%$
- $P(\text{"no"}) = 0.0343 / (0.0238 + 0.0343) = 59\%$

The "low-frequencies problem"

What if an attribute value doesn't occur with every class value (e.g. "Humidity = high" for class "yes")?

- Probability will be zero!
- $\Pr(\text{Humidity} = \text{High} \mid \text{yes}) = 0$
- A posteriori probability will also be zero! $\Pr(\text{yes} \mid E) = 0$ (No matter how likely the other values are!)

So use an estimators for low frequency attribute ranges

- Add a little "m" to the count for every attribute value-class combination
 - The Laplace estimator
 - Result: probabilities will never be zero!

And use an estimator for low frequency classes

- Add a little "k" to class counts
 - The M-estimate

Magic numbers: m=2, k=1

And we'll return to the low frequency problem, below.

Pseudo-code

Here's the pseudo code of the the Naive Bayes classifier preferred by [Yang03](#) (p4).

```
function train( i ) {
  Instances++
  if (++N[$Klass]==1) Klasses++
  for(i=1;i<=Attr;i++)
    if ( i != Klass )
      if ($i !~ /\?/)
        symbol(i,$i,$Klass)
}
function symbol(col,value,klass) {
  Count[klass,col,value]++;
}
```

When testing, find the likelihood of each hypothetical class and return the one that is most likely.

Simple version


```

function likelihood(l,          class,i,inc,temp,prior,what,like) {
  like = -10000000000;      # smaller than any log
  for(klass in N) {
    prior=N[klass] / Instances;
    temp= prior
    for(i=1;i<=Attr;i++) {
      if (i != Klass)
        if ( $i !~ /\?/ )
          temp *= Count[klass,i,$i] / N[klass]
    }
    l[klass]= temp
    if ( temp >= like ) {like = temp; what=class}
  }
  return what
}

```

More Complex

More realistic version (handles certain low-frequency cases).

```

function likelihood(l,          class,i,inc,temp,prior,what,like) {
  like = -10000000000;      # smaller than any log
  for(klass in N) {
    prior=(N[klass]+K)/(Instances + (K*Klasses));
    temp= log(prior)
    for(i=1;i<=Attr;i++) {
      if (i != Klass)
        if ( $i !~ /\?/ )
          temp += log((Count[klass,i,$i]+M*prior)/(N[klass]+M))
    }
    l[klass]= temp
    if ( temp >= like ) {like = temp; what=class}
  }
  return what
}

```

Handling Numerics

The above code assumes that the attributes are discrete. The usual approximation is to assume a "Gaussian" (i.e. a "normal" or "bell-shaped" curve) for the numerics.

The probability density function for the normal distribution is defined by the mean and standardDev (standard deviation)

Given:

- n: the number of values;
- sum: the sum of the values; i.e. sum = sum + value;
- sumSq: the sum of the square of the values; i.e. sumSq = sumSq + value*value

Then:

```

function mean(sum,n) {
  return sum/n
}
function standardDeviation(sumSq,sum,n) {
  return sqrt((sumSq-((sum*sum)/n))/(n-1))
}
function gaussianPdf(mean,standardDev,x) {
  pi= 1068966896 / 340262731; #: good to 17 decimal places
  return 1/(standardDev*sqrt(2*pi)) ^
    (-1*(x-mean)^2/(2*standardDev*standardDev))
}

```

For example:

```

outlook  temperature  humidity  windy  play
-----  -

```

```

sunny      85      85      FALSE no
sunny      80      90      TRUE  no
overcast   83      86      FALSE yes
rainy      70      96      FALSE yes
rainy      68      80      FALSE yes
rainy      65      70      TRUE  no
overcast   64      65      TRUE  yes
sunny      72      95      FALSE no
sunny      69      70      FALSE yes
rainy      75      80      FALSE yes
sunny      75      70      TRUE  yes
overcast   72      90      TRUE  yes
overcast   81      75      FALSE yes
rainy      71      91      TRUE  no

```

This generates the following statistics:

```

===== Outlook =====
                Yes    No
Sunny           2     3
Overcast        4     0
Rainy           3     2
-----
Sunny           2/9    3/5
Overcast        4/9    0/5
Rainy           3/9    2/5

===== Temperature =====
                Yes    No
83             85
70             80
68             65
-----
mean           73     74.6
std dev        6.2     7.9

===== Humidity =====
                Yes    No
86             85
96             90
80             70
-----
mean           79.1   86.2
std dev        10.2   9.7

===== Windy =====
                Yes    No
False          6     2
True           3     3
-----
False          6/9    2/5
True           3/9    3/5

===== Play =====
                Yes    No
9              5
-----
9/14           5/14

```

Example density value:

- $f(\text{temperature}=66|\text{yes}) = \text{gaussianPdf}(73, 6.2, 66) = 0.0340$
- Classifying a new day:

```

Outlook  Temp.  Humidity  Windy  Play
Sunny    66     90       true  ???

```

- Likelihood of "yes" = $2/9 * 0.0340 * 0.0221 * 3/9 * 9/14 = 0.000036$
- Likelihood of "no" = $3/5 * 0.0291 * 0.0380 * 3/5 * 5/14 = 0.000136$
 - $P(\text{"yes"}) = 0.000036 / (0.000036 + 0.000136) = 20.9\%$
 - $P(\text{"no"}) = 0.000136 / (0.000036 + 0.000136) = 79.1\%$

Note: missing values during training: not included in calculation of mean and standard deviation

BTW, an alternative to the above is apply some discretization policy to the data; e.g. Yang03². Such discretization is good practice since it can dramatically improve the performance of a Naive Bayes classifier (see Dougherty95²).

Simple Extensions

From Naive Bayes to Hyper Pipes

When NaiveBayes² sees a new value, it increments a count.

- When HyperPipes sees a value, it just sets the count for that value to "one".

So NaiveBayes² remembers *how often we see "X" in class "C"*

- While HyperPipes just remembers that *at least once, I've seen "X" in class "K"*

Also:

- For numeric columns, [HyperPipes](#) needs to remember the Min/Max ever seen in column "C" for class "K".

```
function symbol(col,value,klass) {
  # Count[klass,col,value]++;
  if (Numericp[col]) {
    if (value < Min[klass,col]) Min[klass,col] = value;
    if (value > Max[klass,col]) Max[klass,col] = value;
  } else {
    Count[klass,col,value] = 1 ;
  }
}
```

So, to classify a test instance:

- For each class, ask how what percentage of the columns "fall into" that class;
- Return the class with the highest percentage cover

```
function oneColumnFallsIntoKlass(klass,col,value) {
  if (Numericp[col])
    return Max[klass,col] >= val && Min[klass,col] <= val ;
  else {
    return Count[klass,col,value] > 0
  }
}
```

From NaiveBayes? to Incremental Learning

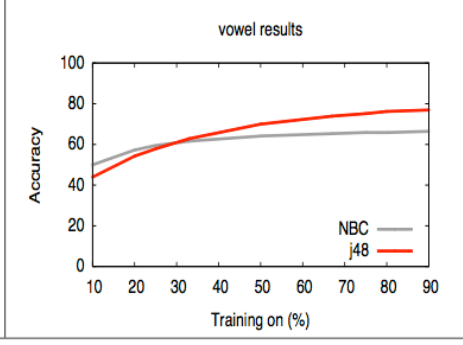
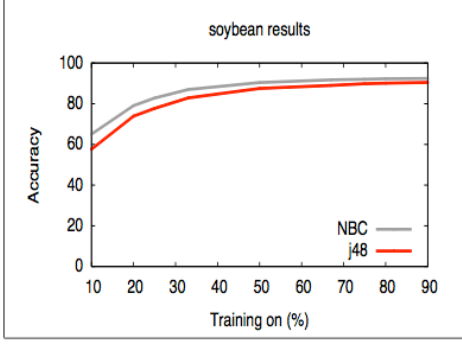
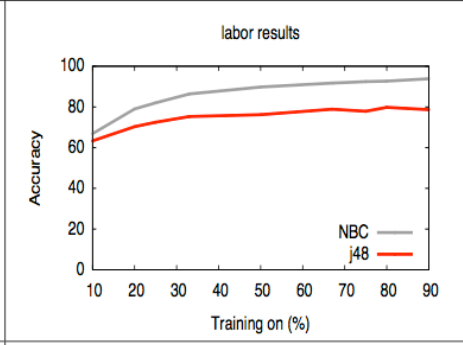
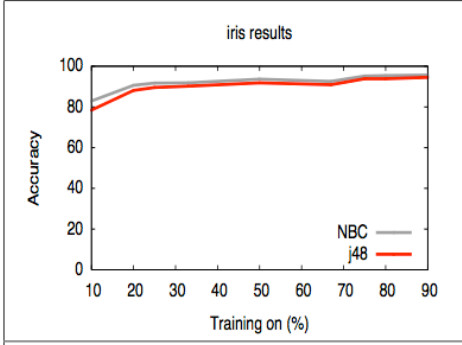
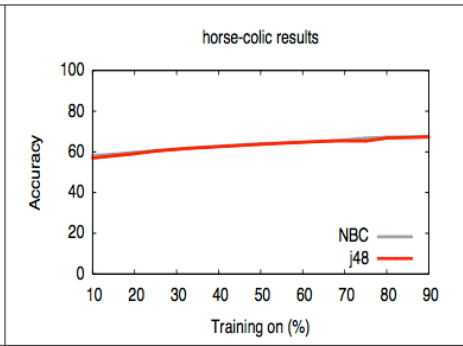
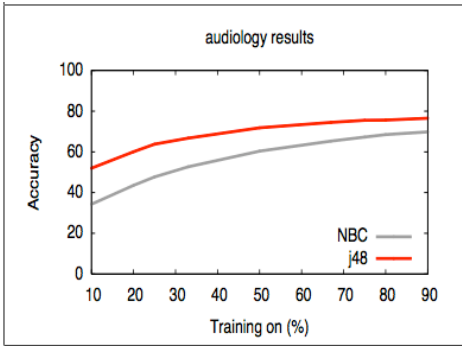
Batch learners read all data, the given performance results.

- Incremental learners give intermediate results

In theory, batch is smarter than incremental

- Since it sees more data before having to make decisions.

In practice, the incremental performance of many learners, including NaiveBayes?, asymptotes to an performance plateau in a few hundred instances:



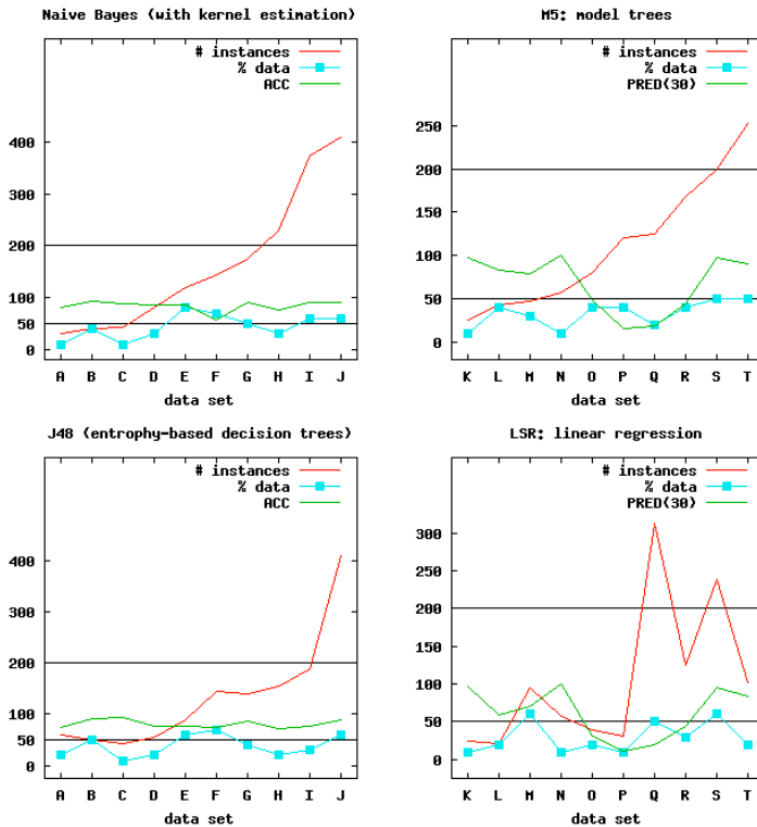


Figure 5.4: $R=10*N=10$ incremental cross validation experiments on 20 UCI data sets [?]. A:heart-c, B:zoo, C:vote, D:heart-statlog, E:lymph, F:autos, G:ionosphere, H:diabetes, I:balance-scale, J:soybean, K:bodyfat. L:cloud, M:fishcatch, N:sensory, O:pwLinear, Q:strike, R:pcb, S:auto-mpg, T:housing. Data sets A..J have discrete classes and are scored via the *accuracy* of the learned theory; i.e % successful classifications. Data sets K..T have continuous classes and are scored by the *PRED(30)* of the learned theory; i.e. what % of the estimated values are within 30% of the actual value. Data sets are sorted according to how many instances were required to reach plateau using nbk and C4.5 (left-hand side) or M5' and LSR (right-hand side).

Building an incremental version of NaiveBayes? is a trivial task:

- Just classify each new instance *before* its counts are added to the frequency tables.

Batch process:

```
Pass="train" { train() }
Pass="test"  { Klass=likelihood(L); print $NF ", " L[Klass] }
```

Incremental process:

```
# don't classify till after seeing a few instances
NR>=30 { Klass = likelihood(L)
        print "want= " $NF " got= " Klass
        }
# but always train
{ train() }
```

Note that the above scheme applies to [HyperPipes](#) as well as NaiveBayes?.

From NaiveBayes? to Anomaly Detection

In anomaly detection, we report "we have not seen this before".

Inside a NaiveBayes? classifier, there is a loop that combines frequency counts for different ranges. If we have not seen "it" before, then some of those frequency counts will drop.

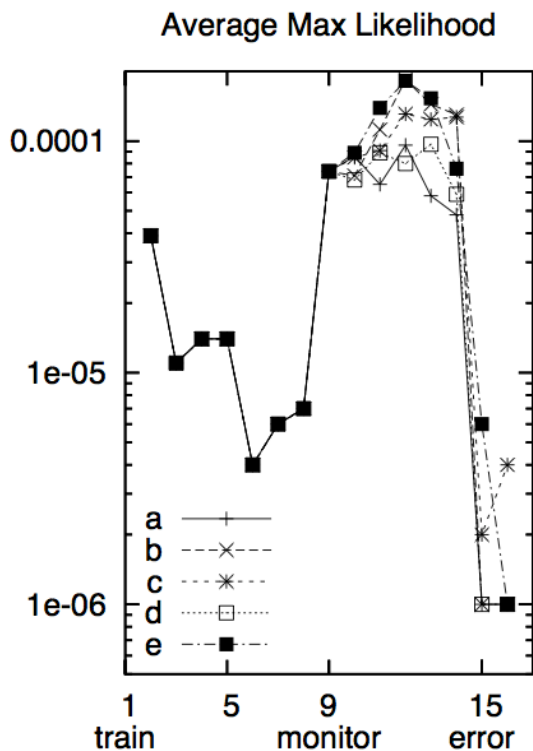
If we take (say) 10 attributes, and three of them halve their frequency count (cause we have not seen this before) then the product of their likelihoods will decrease to $0.5^3=12.5\%$. So all we need to do is:

- Assign each train instance to some class "All"
- Track the likelihoods of "All".
- Complain if this number drops by orders of magnitudes.

```
# don't classify till after seeing a few instances
NR>=30 { Klass = likelihood(L)
        print "want= " $NF " got= " Klass " track= " L["All"]
        }
# but always train
        { train();
          $NF="All";
          train()
        }
```

Here's a trace of the log of the "All" likelihood for five flight simulators, divided into "eras" of 100 samples.

- From era=1 to era=8, all planes fly the same commissioning exercise;
- Form era=9 to era=14, all planes fly different missions, repeating the same maneuver over and over again.
- At era=15, all the planes were hit with different faults (engine flame up, flaps not working, etc)
- In all cases, at era=15, $\log(\text{likelihood}(\text{"All"}))$ dropped two orders of magnitude.
- Note that, at no time, was the classier given a list of specific error conditions of the plane.
 - So it learned "I have not seen *that* before", even though we never told it what was *that*.
 - Magic! (but so simple)



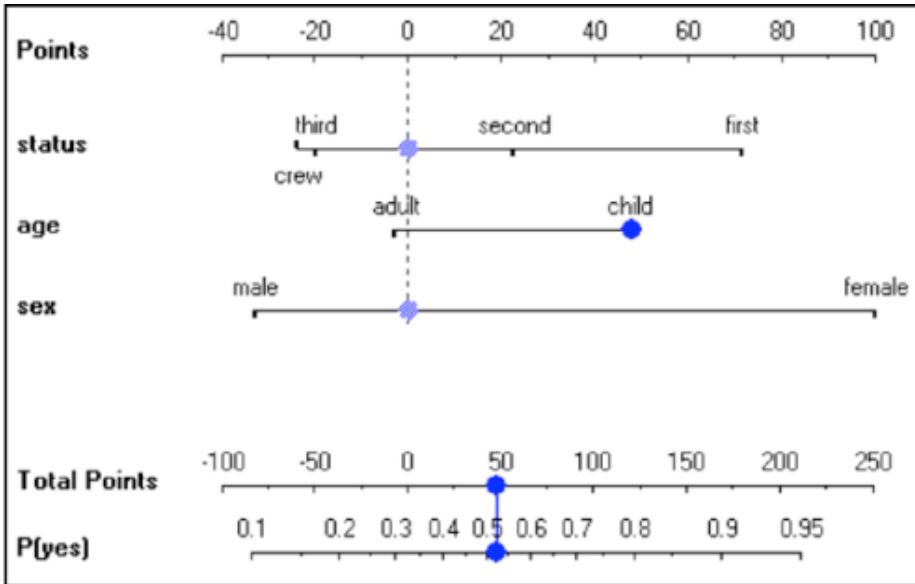
The Explanation Problem

All learners are performance systems that make conclusions. But only some can explain how they reach those conclusions.

The internal data structures of a Bayes classifier are not very pleasant to look at when listed in a table. Moreover,

the particulars of that data are not what really matter.

[Mozina04](#) argue that what really matters is the effect of a cell on the output variables. With knowledge of this goal, it is possible to design a visualization, called a nomogram, of that data. Nomograms do not confuse the user with needless detail. For example, here's nomogram describing who survived and who died on the Titanic:



Of 2201 passengers on Titanic, 711 (32.3%) survived. To predict who will survive, the contribution of each attribute is measured as a point score (topmost axis in the nomogram), and the individual point scores are summed to determine the probability of survival (bottom two axes of the nomogram).

The nomogram shows the case when we know that the passenger is a child; this score is slightly less than 50 points, and increases the posterior probability to about 52%. If we further know that the child traveled in first class (about 70 points), the points would sum to about 120, with a corresponding probability of survival of about 80%.

It is simple to calculate nomogram values for single ranges. All we want is something that is far more probable in a goal class than in other classes.

- Suppose there is a class you like C and a bunch of others you hate.
- Let the bad classes be combined together into a group we'll call notC
- Let the frequencies of C and notC be N1 and N2.
- Let two attribute range appears with frequency F1 and F2 in C1 and notC.
- Then the $\log(\text{OR}) = \log \left(\frac{N1}{H1} \right) / \left(\frac{N2}{H2} \right)$

We use logs since products can be visualized via simple addition. This addition can be converted back to a probability as follows. If the sum is "f" and the target class occurs "N" times out of "I" instances, then the probability of that class is "p=N/I" and the sum's probability is:

```
function points2p(f,p) { return 1 / ( 1 + E^(-1*log(p/(1 - p)) - f) ) }
```

(For the derivation of this expression, see equation 7 of [Mozina04](#). Note that their equation has a one-bracket typo.)

Besides enabling prediction, the nomogram reveals the structure of the model and the relative influences of attribute values on the chances of surviving. For the Titanic data set:

- Gender is an attribute with the biggest potential influence on the probability of survival: being female increases the chances of survival the most (100 points), while being male decreases it (about 30 points). The corresponding line in the nomogram for this attribute is the longest.
- Age is apparently the least influential, where being a child increases the probability of survival.
- Most lucky were also the passengers of the first class for which, considering the status only, the probability of survival was much higher than the prior.

Therefore, with nomograms, we can play cost-benefit games. Consider the survival benefits of

- sex=female (72%)
- sex=female and class=first (92%)

greatly reduced. In practical implementations, we can use an ad hoc test to guard against splitting on such a useless attribute.

Unfortunately, in some situations the gain ratio modification overcompensates and can lead to preferring an attribute just because its intrinsic information is much lower than that for the other attributes. A standard fix is to choose the attribute that maximizes the gain ratio, provided that the information gain for that attribute is at least as great as the average information gain for all the attributes examined.

Discussion

The divide-and-conquer approach to decision tree induction, sometimes called *top-down induction of decision trees*, was developed and refined over many years by J. Ross Quinlan of the University of Sydney, Australia. Although others have worked on similar methods, Quinlan's research has always been at the very forefront of decision tree induction. The method that has been described using the information gain criterion is essentially the same as one known as ID3. The use of the gain ratio was one of many improvements that were made to ID3 over several years; Quinlan described it as robust under a wide variety of circumstances. Although a robust and practical solution, it sacrifices some of the elegance and clean theoretical motivation of the information gain criterion.

A series of improvements to ID3 culminated in a practical and influential system for decision tree induction called C4.5. These improvements include methods for dealing with numeric attributes, missing values, noisy data, and generating rules from trees, and they are described in Section 6.1.

4.4 Covering algorithms: Constructing rules

As we have seen, decision tree algorithms are based on a divide-and-conquer approach to the classification problem. They work from the top down, seeking at each stage an attribute to split on that best separates the classes; then recursively processing the subproblems that result from the split. This strategy generates a decision tree, which can if necessary be converted into a set of classification rules—although if it is to produce effective rules, the conversion is not trivial.

An alternative approach is to take each class in turn and seek a way of covering all instances in it, at the same time excluding instances not in the class. This is called a *covering* approach because at each stage you identify a rule that “covers” some of the instances. By its very nature, this covering approach leads to a set of rules rather than to a decision tree.

The covering method can readily be visualized in a two-dimensional space of instances as shown in Figure 4.6(a). We first make a rule covering the a 's. For

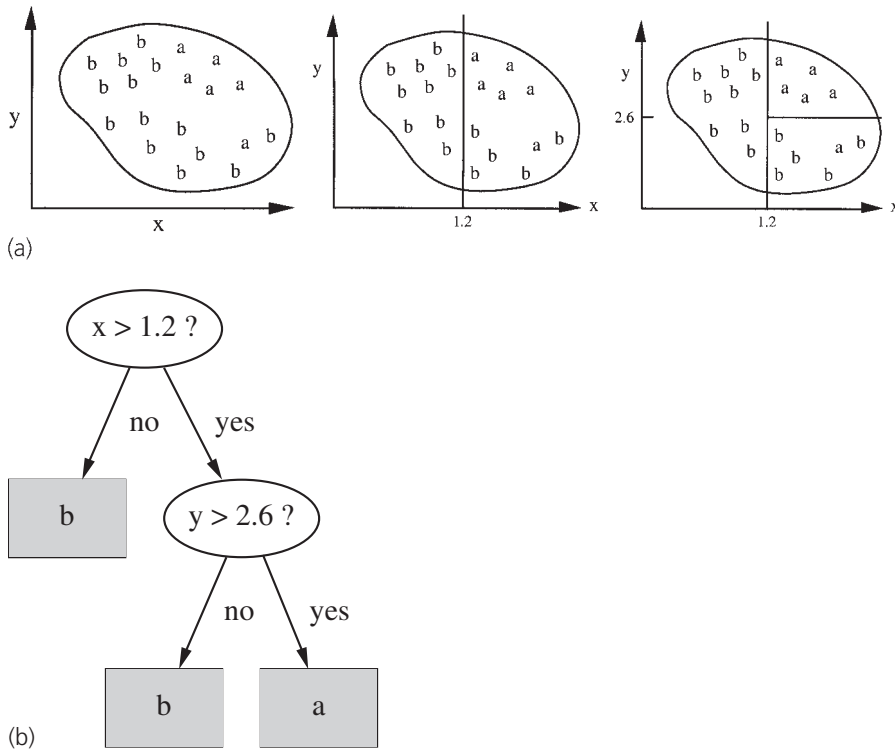


Figure 4.6 Covering algorithm: (a) covering the instances and (b) the decision tree for the same problem.

the first test in the rule, split the space vertically as shown in the center picture. This gives the beginnings of a rule:

If $x > 1.2$ then class = a

However, the rule covers many *b*'s as well as *a*'s, so a new test is added to the rule by further splitting the space horizontally as shown in the third diagram:

If $x > 1.2$ and $y > 2.6$ then class = a

This gives a rule covering all but one of the *a*'s. It's probably appropriate to leave it at that, but if it were felt necessary to cover the final *a*, another rule would be necessary—perhaps

If $x > 1.4$ and $y < 2.4$ then class = a

The same procedure leads to two rules covering the *b*'s:

If $x \leq 1.2$ then class = b

If $x > 1.2$ and $y \leq 2.6$ then class = b

Again, one a is erroneously covered by these rules. If it were necessary to exclude it, more tests would have to be added to the second rule, and additional rules would need to be introduced to cover the b 's that these new tests exclude.

Rules versus trees

A top-down divide-and-conquer algorithm operates on the same data in a manner that is, at least superficially, quite similar to a covering algorithm. It might first split the dataset using the x attribute and would probably end up splitting it at the same place, $x = 1.2$. However, whereas the covering algorithm is concerned only with covering a single class, the division would take both classes into account, because divide-and-conquer algorithms create a single concept description that applies to all classes. The second split might also be at the same place, $y = 2.6$, leading to the decision tree in Figure 4.6(b). This tree corresponds exactly to the set of rules, and in this case there is no difference in effect between the covering and the divide-and-conquer algorithms.

But in many situations there *is* a difference between rules and trees in terms of the perspicuity of the representation. For example, when we described the replicated subtree problem in Section 3.3, we noted that rules can be symmetric whereas trees must select one attribute to split on first, and this can lead to trees that are much larger than an equivalent set of rules. Another difference is that, in the multiclass case, a decision tree split takes all classes into account, trying to maximize the purity of the split, whereas the rule-generating method concentrates on one class at a time, disregarding what happens to the other classes.

A simple covering algorithm

Covering algorithms operate by adding tests to the rule that is under construction, always striving to create a rule with maximum accuracy. In contrast, divide-and-conquer algorithms operate by adding tests to the tree that is under construction, always striving to maximize the separation among the classes. Each of these involves finding an attribute to split on. But the criterion for the best attribute is different in each case. Whereas divide-and-conquer algorithms such as ID3 choose an attribute to maximize the information gain, the covering algorithm we will describe chooses an attribute–value pair to maximize the probability of the desired classification.

Figure 4.7 gives a picture of the situation, showing the space containing all the instances, a partially constructed rule, and the same rule after a new term has been added. The new term restricts the coverage of the rule: the idea is to include as many instances of the desired class as possible and exclude as many instances of other classes as possible. Suppose the new rule will cover a total of t instances, of which p are positive examples of the class and $t - p$ are in other

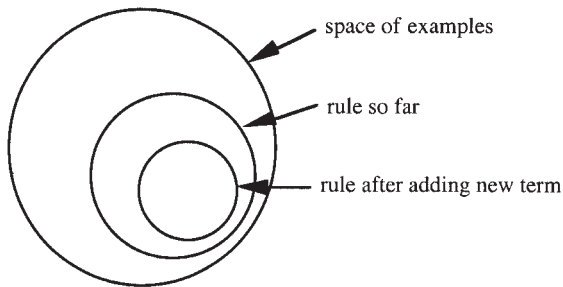


Figure 4.7 The instance space during operation of a covering algorithm.

classes—that is, they are errors made by the rule. Then choose the new term to maximize the ratio p/t .

An example will help. For a change, we use the contact lens problem of Table 1.1. We will form rules that cover each of the three classes, *hard*, *soft*, and *none*, in turn. To begin, we seek a rule:

If ? then recommendation= hard

For the unknown term ?, we have nine choices:

age = young	2/8
age = pre-presbyopic	1/8
age = presbyopic	1/8
spectacle prescription= myope	3/12
spectacle prescription= hypermetrope	1/12
astigmatism= no	0/12
astigmatism= yes	4/12
tear production rate= reduced	0/12
tear production rate= normal	4/12

The numbers on the right show the fraction of “correct” instances in the set singled out by that choice. In this case, *correct* means that the recommendation is *hard*. For instance, *age = young* selects eight instances, two of which recommend hard contact lenses, so the first fraction is 2/8. (To follow this, you will need to look back at the contact lens data in Table 1.1 on page 6 and count up the entries in the table.) We select the largest fraction, 4/12, arbitrarily choosing between the seventh and the last choice in the preceding list, and create the rule:

If astigmatism= yes then recommendation= hard

This rule is an inaccurate one, getting only 4 instances correct out of the 12 that it covers, shown in Table 4.8. So we refine it further:

If astigmatism= yes and ? then recommendation= hard

Table 4.8 Part of the contact lens data for which *astigmatism = yes*.

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	yes	reduced	none
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	yes	reduced	none
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	yes	reduced	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	yes	reduced	none
presbyopic	hypermetrope	yes	normal	none

Considering the possibilities for the unknown term ? yields the seven choices:

age = young	2/4
age = pre-presbyopic	1/4
age = presbyopic	1/4
spectacle prescription= myope	3/6
spectacle prescription= hypermetrope	1/6
tear production rate= reduced	0/6
tear production rate= normal	4/6

(Again, count the entries in Table 4.8.) The last is a clear winner, getting four instances correct out of the six that it covers, and corresponds to the rule

```
If astigmatism= yes and tear production rate= normal
then recommendation= hard
```

Should we stop here? Perhaps. But let's say we are going for exact rules, no matter how complex they become. Table 4.9 shows the cases that are covered by the rule so far. The possibilities for the next term are now

age = young	2/2
age = pre-presbyopic	1/2
age = presbyopic	1/2
spectacle prescription= myope	3/3
spectacle prescription= hypermetrope	1/3

We need to choose between the first and fourth. So far we have treated the fractions numerically, but although these two are equal (both evaluate to 1), they have different coverage: one selects just two correct instances and the other

Table 4.9 Part of the contact lens data for which *astigmatism = yes* and *tear production rate = normal*.

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
young	myope	yes	normal	hard
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	yes	normal	none

selects three. In the event of a tie, we choose the rule with the greater coverage, giving the final rule:

```
If astigmatism= yes and tear production rate= normal
    and spectacle prescription= myope then recommendation= hard
```

This is indeed one of the rules given for the contact lens problem. But it only covers three of the four *hard* recommendations. So we delete these three from the set of instances and start again, looking for another rule of the form:

```
If ? then recommendation= hard
```

Following the same process, we will eventually find that *age = young* is the best choice for the first term. Its coverage is seven; the reason for the seven is that 3 instances have been removed from the original set, leaving 21 instances altogether. The best choice for the second term is *astigmatism = yes*, selecting 1/3 (actually, this is a tie); *tear production rate = normal* is the best for the third, selecting 1/1.

```
If age = young and astigmatism= yes and
    tear production rate= normal then recommendation= hard
```

This rule actually covers three of the original set of instances, two of which are covered by the previous rule—but that’s all right because the recommendation is the same for each rule.

Now that all the hard-lens cases are covered, the next step is to proceed with the soft-lens ones in just the same way. Finally, rules are generated for the *none* case—unless we are seeking a rule set with a default rule, in which case explicit rules for the final outcome are unnecessary.

What we have just described is the PRISM method for constructing rules. It generates only correct or “perfect” rules. It measures the success of a rule by the accuracy formula p/t . Any rule with accuracy less than 100% is “incorrect” in

that it assigns cases to the class in question that actually do not have that class. PRISM continues adding clauses to each rule until it is perfect: its accuracy is 100%. Figure 4.8 gives a summary of the algorithm. The outer loop iterates over the classes, generating rules for each class in turn. Note that we reinitialize to the full set of examples each time round. Then we create rules for that class and remove the examples from the set until there are none of that class left. Whenever we create a rule, start with an empty rule (which covers all the examples), and then restrict it by adding tests until it covers only examples of the desired class. At each stage choose the most promising test, that is, the one that maximizes the accuracy of the rule. Finally, break ties by selecting the test with greatest coverage.

Rules versus decision lists

Consider the rules produced for a particular class, that is, the algorithm in Figure 4.8 with the outer loop removed. It seems clear from the way that these rules are produced that they are intended to be interpreted in order, that is, as a decision list, testing the rules in turn until one applies and then using that. This is because the instances covered by a new rule are removed from the instance set as soon as the rule is completed (in the third line from the end of the code in Figure 4.8): thus subsequent rules are designed for instances that are *not* covered by the rule. However, although it appears that we are supposed to check the rules in turn, we do not have to do so. Consider that any subsequent rules generated for this class will have the same effect—they all predict the same class. This means that it does not matter what order they are executed in: either a rule will

```
For each class C
  Initialize E to the instance set
  While E contains instances in class C
    Create a rule R with an empty left-hand side that predicts class C
    Until R is perfect (or there are no more attributes to use) do
      For each attribute A not mentioned in R, and each value v,
        Consider adding the condition A=v to the LHS of R
        Select A and v to maximize the accuracy p/t
        (break ties by choosing the condition with the largest p)
      Add A=v to R
    Remove the instances covered by R from E
```

Figure 4.8 Pseudocode for a basic rule learner.

Using the Triangle Inequality to Accelerate k -Means

Charles Elkan

Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California 92093-0114

ELKAN@CS.UCSD.EDU

Abstract

The k -means algorithm is by far the most widely used method for discovering clusters in data. We show how to accelerate it dramatically, while still always computing exactly the same result as the standard algorithm. The accelerated algorithm avoids unnecessary distance calculations by applying the triangle inequality in two different ways, and by keeping track of lower and upper bounds for distances between points and centers. Experiments show that the new algorithm is effective for datasets with up to 1000 dimensions, and becomes more and more effective as the number k of clusters increases. For $k \geq 20$ it is many times faster than the best previously known accelerated k -means method.

1. Introduction

The most common method for finding clusters in data used in applications is the algorithm known as k -means. k -means is considered a fast method because it is not based on computing the distances between all pairs of data points. However, the algorithm is still slow in practice for large datasets. The number of distance computations is nke where n is the number of data points, k is the number of clusters to be found, and e is the number of iterations required. Empirically, e grows sublinearly with k , n , and the dimensionality d of the data.

The main contribution of this paper is an optimized version of the standard k -means method, with which the number of distance computations is in practice closer to n than to nke .

The optimized algorithm is based on the fact that most distance calculations in standard k -means are redundant. If a point is far away from a center, it is not necessary to calculate the exact distance between the point and the center in order to know that the point should not be assigned to

this center. Conversely, if a point is much closer to one center than to any other, calculating exact distances is not necessary to know that the point should be assigned to the first center. We show below how to make these intuitions concrete.

We want the accelerated k -means algorithm to be usable wherever the standard algorithm is used. Therefore, we need the accelerated algorithm to satisfy three properties. First, it should be able to start with any initial centers, so that all existing initialization methods can continue to be used. Second, given the same initial centers, it should always produce exactly the same final centers as the standard algorithm. Third, it should be able to use any black-box distance metric, so it should not rely for example on optimizations specific to Euclidean distance.

Our algorithm in fact satisfies a condition stronger than the second one above: after each iteration, it produces the same set of center locations as the standard k -means method. This stronger property means that heuristics for merging or splitting centers (and for dealing with empty clusters) can be used together with the new algorithm. The third condition is important because many applications use a domain-specific distance metric. For example, clustering to identify duplicate alphanumeric records is sometimes based on alphanumeric edit distance (Monge & Elkan, 1996), while clustering of protein structures is often based on an expensive distance function that first rotates and translates structures to superimpose them. Even without a domain-specific metric, recent work shows that using a non-traditional L_p norm with $0 < p < 1$ is beneficial when clustering in a high-dimensional space (Aggarwal et al., 2001).

This paper is organized as follows. Section 2 explains how to use the triangle inequality to avoid redundant distance calculations. Then Section 3 presents the new algorithm, and Section 4 discusses experimental results on six datasets of dimensionality 2 to 1000. Section 5 outlines possible improvements to the method, while Section 6 reviews related work, and Section 7 explains three open research issues.

2. Applying the triangle inequality

Our approach to accelerating k -means is based on the triangle inequality: for any three points x , y , and z , $d(x, z) \leq d(x, y) + d(y, z)$. This is the only “black box” property that all distance metrics d possess.

The difficulty is that the triangle inequality gives upper bounds, but we need lower bounds to avoid calculations. Let x be a point and let b and c be centers; we need to know that $d(x, c) \geq d(x, b)$ in order to avoid calculating the actual value of $d(x, c)$.

The following two lemmas show how to use the triangle inequality to obtain useful lower bounds.

Lemma 1: Let x be a point and let b and c be centers. If $d(b, c) \geq 2d(x, b)$ then $d(x, c) \geq d(x, b)$.

Proof: We know that $d(b, c) \leq d(b, x) + d(x, c)$. So $d(b, c) - d(x, b) \leq d(x, c)$. Consider the left-hand side: $d(b, c) - d(x, b) \geq 2d(x, b) - d(x, b) = d(x, b)$. So $d(x, b) \leq d(x, c)$. ■

Lemma 2: Let x be a point and let b and c be centers. Then $d(x, c) \geq \max\{0, d(x, b) - d(b, c)\}$.

Proof: We know that $d(x, b) \leq d(x, c) + d(b, c)$, so $d(x, c) \geq d(x, b) - d(b, c)$. Also, $d(x, c) \geq 0$. ■

Note that Lemmas 1 and 2 are true for any three points, not just for a point and two centers, and the statement of Lemma 2 can be strengthened in various ways.

We use Lemma 1 as follows. Let x be any data point, let c be the center to which x is currently assigned, and let c' be any other center. The lemma says that if $\frac{1}{2}d(c, c') \geq d(x, c)$, then $d(x, c') \geq d(x, c)$. In this case, it is not necessary to calculate $d(x, c')$.

Suppose that we do not know $d(x, c)$ exactly, but we do know an upper bound u such that $u \geq d(x, c)$. Then we need to compute $d(x, c')$ and $d(x, c)$ only if $u > \frac{1}{2}d(c, c')$.

If $u \leq \frac{1}{2} \min_{c' \neq c} d(c, c')$ where the minimum is over all $c' \neq c$, then the point x must remain assigned to the center c , and all distance calculations for x can be avoided.

Lemma 2 is applied as follows. Let x be any data point, let b be any center, and let b' be the previous version of the same center. (That is, suppose the centers are numbered 1 through k , and b is center number j ; then b' is center number j in the previous iteration.) Suppose that in the previous iteration we knew a lower bound l' such that $d(x, b') \geq l'$. Then we can infer a lower bound l for the current iteration:

$$\begin{aligned} d(x, b) &\geq \max\{0, d(x, b') - d(b, b')\} \\ &\geq \max\{0, l' - d(b, b')\} = l. \end{aligned}$$

Informally, if l' is a good approximation to the previous

distance between x and the j th center, and this center has moved only a small distance, then l is a good approximation to the updated distance.

The algorithm below is the first k -means variant that uses lower bounds, as far as we know. It is also the first algorithm that carries over varying information from one k -means iteration to the next. According to the authors of (Kanungo et al., 2000): “The most obvious source of inefficiency in [our] algorithm is that it passes no information from one stage to the next. Presumably in the later stages of Lloyd’s algorithm, as the centers are converging to their final positions, one would expect that the vast majority of the data points have the same closest center from one stage to the next. A good algorithm would exploit this coherence to improve running time.” The algorithm in this paper achieves this goal. One previous algorithm also re-uses information from one k -means iteration in the next, but that method, due to (Judd et al., 1998), does not carry over lower or upper bounds.

Suppose $u(x) \geq d(x, c)$ is an upper bound on the distance between x and the center c to which x is currently assigned, and suppose $l(x, c') \leq d(x, c')$ is a lower bound on the distance between x and some other center c' . If $u(x) \leq l(x, c')$ then $d(x, c) \leq u(x) \leq l(x, c') \leq d(x, c')$ so it is necessary to calculate neither $d(x, c)$ nor $d(x, c')$. Note that it will never be necessary in this iteration of the accelerated method to compute $d(x, c')$, but it may be necessary to compute $d(x, c)$ exactly because of some other center c'' for which $u(x) \leq l(x, c'')$ is not true.

3. The new algorithm

Putting the observations above together, the accelerated k -means algorithm is as follows.

First, pick initial centers. Set the lower bound $l(x, c) = 0$ for each point x and center c . Assign each x to its closest initial center $c(x) = \operatorname{argmin}_c d(x, c)$, using Lemma 1 to avoid redundant distance calculations. Each time $d(x, c)$ is computed, set $l(x, c) = d(x, c)$. Assign upper bounds $u(x) = \min_c d(x, c)$.

Next, repeat until convergence:

1. For all centers c and c' , compute $d(c, c')$. For all centers c , compute $s(c) = \frac{1}{2} \min_{c' \neq c} d(c, c')$.
2. Identify all points x such that $u(x) \leq s(c(x))$.
3. For all remaining points x and centers c such that
 - (i) $c \neq c(x)$ and
 - (ii) $u(x) > l(x, c)$ and
 - (iii) $u(x) > \frac{1}{2}d(c(x), c)$:

3a. If $r(x)$ then compute $d(x, c(x))$ and assign $r(x) = \text{false}$. Otherwise, $d(x, c(x)) = u(x)$.

3b. If $d(x, c(x)) > l(x, c)$
or $d(x, c(x)) > \frac{1}{2}d(c(x), c)$ then

 Compute $d(x, c)$

 If $d(x, c) < d(x, c(x))$ then assign $c(x) = c$.

4. For each center c , let $m(c)$ be the mean of the points assigned to c .

5. For each point x and center c , assign

$$l(x, c) = \max\{l(x, c) - d(c, m(c)), 0\}.$$

6. For each point x , assign

$$u(x) = u(x) + d(m(c(x)), c(x))$$

$$r(x) = \text{true}.$$

7. Replace each center c by $m(c)$.

In step (3), each time $d(x, c)$ is calculated for any x and c , its lower bound is updated by assigning $l(x, c) = d(x, c)$. Similarly, $u(x)$ is updated whenever $c(x)$ is changed or $d(x, c(x))$ is computed. In step (3a), if $r(x)$ is true then $u(x)$ is out-of-date, i.e. it is possible that $u(x) \neq d(x, c(x))$. Otherwise, computing $d(x, c(x))$ is not necessary. Step (3b) repeats the checks from (ii) and (iii) in order to avoid computing $d(x, c)$ if possible.

The fundamental reason why the algorithm above is effective in reducing the number of distance calculations is that at the start of each iteration, the upper bounds $u(x)$ and the lower bounds $l(x, c)$ are tight for most points x and centers c . If these bounds are tight at the start of one iteration, the updated bounds tend to be tight at the start of the next iteration, because the location of most centers changes only slightly, and hence the bounds change only slightly.

The initialization step of the algorithm assigns each point to its closest center immediately. This requires relatively many distance calculations, but it leads to exact upper bounds $u(x)$ for all x and to exact lower bounds $l(x, c)$ for many (x, c) pairs. An alternative initialization method is to start with each point arbitrarily assigned to one center. The initial values of $u(x)$ and $l(c, x)$ are then based on distances calculated to this center only. With this approach, the initial number of distance calculations is only n , but $u(x)$ and $l(c, x)$ are much less tight initially, so more distance calculations are required later. (After each iteration each point is always assigned correctly to its closest center, regardless of how inaccurate the lower and upper bounds are at the start of the iteration.) Informal experiments suggest that both initialization methods lead to about the same total number of distance calculations.

Logically, step (2) is redundant because its effect is achieved by condition (iii). Computationally, step (2) is

beneficial because if it eliminates a point x from further consideration, then comparing $u(x)$ to $l(x, c)$ for every c separately is not necessary. Condition (iii) inside step (3) is beneficial despite step (2), because $u(x)$ and $c(x)$ may change during the execution of step (3).

We have implemented the algorithm above in Matlab. When step (3) is implemented with nested loops, the outer loop can be over x or over c . For efficiency in Matlab and similar languages, the outer loop should be over c since $k \ll n$ typically, and the inner loop should be replaced by vectorized code that operates on all relevant x collectively.

Step 4 computes the new location of each cluster center c . Setting $m(c)$ to be the mean of the points assigned to c is appropriate when the distance metric in use is Euclidean distance. Otherwise, $m(c)$ may be defined differently. For example, with k -medians the new center of each cluster is a representative member of the cluster.

4. Experimental results

This section reports the results of running the new algorithm on six large datasets, five of which are high-dimensional. The datasets are described in Table 1, while Table 2 gives the results.

Our experimental design is similar to the design of (Moore, 2000), which is the best recent paper on speeding up the k -means algorithm for high-dimensional data. However, there is only one dataset used in (Moore, 2000) for which the raw data are available and enough information is given to allow the dataset to be reconstructed. This dataset is called “covtype.” Therefore, we also use five other publicly available datasets. None of the datasets have missing data.

In order to make our results easier to reproduce, we use a fixed initialization for each dataset X . The first center is initialized to be the mean of X . Subsequent centers are initialized according to the “furthest first” heuristic: each new center is $\text{argmax}_{x \in X} \min_{c \in C} d(x, c)$ where C is the set of initial centers chosen so far (Dasgupta, 2002).

Following the practice of past research, we measure the performance of an algorithm on a dataset as the number of distance calculations required. All algorithms that accelerate k -means incur overhead to create and update auxiliary data structures. This means that speedup compared to k -means is always less in clock time than in number of distance calculations. Our algorithm reduces the number of distance calculations so dramatically that its overhead time is often greater than the time spent on distance calculations. However, the total execution time is always much less than the time required by standard k -means. The overhead of the l and u data structures will be much smaller with a C implementation than with the Matlab implementation used

name	cardinality	dimensionality	description
birch	100000	2	10 by 10 grid of Gaussian clusters, DS1 in (Zhang et al., 1996)
covtype	150000	54	remote soil cover measurements, after (Moore, 2000)
kddcup	95413	56	KDD Cup 1998 data, un-normalized
mnist50	60000	50	random projection of NIST handwritten digit training data
mnist784	60000	784	original NIST handwritten digit training data
random	10000	1000	uniform random data

Table 1. Datasets used in experiments.

		$k = 3$	$k = 20$	$k = 100$
birch	iterations	17	38	56
	standard	5.100e+06	7.600e+07	5.600e+08
	fast	4.495e+05	1.085e+06	1.597e+06
	speedup	11.3	70.0	351
covtype	iterations	18	256	152
	standard	8.100e+06	7.680e+08	2.280e+09
	fast	9.416e+05	7.147e+06	7.353e+06
	speedup	8.60	107	310
kddcup	iterations	34	100	325
	standard	9.732e+06	1.908e+08	3.101e+09
	fast	6.179e+05	3.812e+06	1.005e+07
	speedup	15.4	50.1	309
mnist50	iterations	38	178	217
	standard	6.840e+06	2.136e+08	1.302e+09
	fast	1.573e+06	9.353e+06	3.159e+07
	speedup	4.35	22.8	41.2
mnist784	iterations	63	60	165
	standard	1.134e+07	7.200e+07	9.900e+08
	fast	1.625e+06	7.396e+06	3.055e+07
	speedup	6.98	9.73	32.4
random	iterations	52	33	18
	standard	1.560e+06	6.600e+06	1.800e+07
	fast	1.040e+06	3.020e+06	5.348e+06
	speedup	1.50	2.19	3.37

Table 2. Rows labeled “standard” and “fast” give the number of distance calculations performed by the unaccelerated k -means algorithm and by the new algorithm. Rows labeled “speedup” show how many times faster the new algorithm is, when the unit of measurement is distance calculations.

for the experiments reported here. For this reason, clock times are not reported.

Perhaps the most striking observation to be made from Table 2 is that the relative advantage of the new method increases with k . The number of distance calculations grows only slowly with k and with e (the number of passes over the data, called “iterations” in Table 2). So much redundant computation is eliminated that the total number of distance calculations is closer to n than to nke as for standard k -means.

A related observation is that for $k \geq 20$ we obtain a much better speedup than with the anchors method (Moore, 2000). The speedups reported by Moore for the “covtype” dataset are 24.8, 11.3, and 19.0 respectively for clustering with 3, 20, and 100 centers. The speedups we obtain are 8.60, 107, and 310. We conjecture that the improved speedup for $k \geq 20$ arises in part from using the actual cluster centers as adaptive “anchors,” instead of using a set of anchors fixed in preprocessing. The worse speedup for $k = 3$ remains to be explained.

Another striking observation is that the new method remains effective even for data with very high dimensionality. Moore writes “If there *is* no underlying structure in the data (e.g. if it is uniformly distributed) there will be little or no acceleration in high dimensions no matter what we do. This gloomy view, supported by recent theoretical work in computational geometry (Indyk et al., 1999), means that we can only accelerate datasets that have interesting internal structure.” While this claim is almost certainly true asymptotically as the dimension of a dataset tends to infinity, our results on the “random” dataset suggest that worthwhile speedup can still be obtained up to at least 1000 dimensions. As expected, the more clustered a dataset is, the greater the speedup obtained. Random projection makes clusters more Gaussian (Dasgupta, 2000), so speedup is better for the “mnist50” dataset than for the “mnist784” dataset.

5. Limitations and extensions

During each iteration of the algorithm proposed here, the lower bounds $l(x, c)$ are updated for all points x and centers c . These updates take $O(nk)$ time, so the time complexity of the algorithm remains at least $O(nke)$ even though the number of distance calculations is roughly $O(n)$ only. It may be possible to avoid updating many lower bounds in most iterations, and hence to reduce the nominal complexity of the algorithm. Note that if a point x is eliminated from further consideration in step (2), then $l(x, c)$ is not used at all.

In some clustering applications, $k \gg d$. This is the case in particular for vector quantization for image compress-

ion. For these applications the memory required to store the lower bounds $l(x, c)$ may be the dominant storage cost. However, the entire matrix $l(x, c)$ never needs to be kept in main memory. If the data are streamed into memory at each iteration from disk, then the $l(x, c)$ matrix can also be streamed into memory in synchrony.

Moreover, the algorithm remains beneficial even if lower bounds are not used, so condition (ii) becomes $u(x) > d(x, c)$, where $d(x, c)$ is computed if necessary.

When the algorithm is used with a distance function that is fast to evaluate, such as an L_p norm, then in practice the time complexity of the algorithm is dominated by the bookkeeping used to avoid distance calculations. Therefore, future work should focus on reducing this overhead.

The point above is especially true because a Euclidean distance (or other L_p distance) in d dimensions can often be compared to a known minimum distance in $o(d)$ time. The simple idea is to stop evaluating the new squared distance when the sum of squares so far is greater than the known squared minimum distance. (This suggestion is usually ascribed to (Bei & Gray, 1985), but in fact it is first mentioned in (Cheng et al., 1984).) Distance calculations can be stopped even quicker if axes of high variation are considered first. Axes of maximum variation may be found by principal component analysis (PCA) (McNames, 2000), but the preprocessing cost of PCA may be prohibitive.

At the end of each iteration, centers must be recomputed. Computing means takes $O(nd)$ time independent of k . This can be reduced to $O((k + b)d)$ time where b is the number of points assigned to a different center during the iteration. Typically $b \ll n$ in all except the first few iterations. As mentioned above, the algorithm of this paper can also be used when centers are not recomputed as means.

During each iteration, distances between all centers must be recomputed, so the minimum number of distance computations per iteration is $k(k - 1)/2$. For large k , as in vector quantization, this may be a dominant expense. Future research should investigate the best way to reduce this cost by computing approximations for inter-center distances that are large.

6. Related work

Many papers have been published on the topic of accelerating the k -means algorithm, in several different research communities. Some of the most important of these papers are described briefly in this section. Most of the papers cited below only cite previous papers from the same research community, so one of the contributions of this paper is an attempt to collect references that otherwise cannot be found in one place. All the relevant papers that we know

of can be found by following chains of citations from the papers mentioned here.

A version of the k -means algorithm was first published by (MacQueen, 1965). The history of different variants of the algorithm is discussed by (Faber, 1994). The basic algorithm used most commonly today, and used in this paper, where centers are recomputed once after each pass through the data, is usually attributed to a paper written by Lloyd in 1957 but not published until 1982 (Lloyd, 1982). However, that paper only discusses quantization (i.e. clustering) for some special one-dimensional cases.

The central operation in the k -means algorithm is to find the nearest center for each data point. At least three general approaches have been developed for accelerating this operation.

One general approach is based on locality-sensitive hashing (Indyk & Motwani, 1998), but these methods are not well-suited for finding exact nearest neighbors. A second general approach organizes points into trees where nearby points are in the same subtree. Approaches using kd -trees or similar have been proposed independently by several authors (Ramasubramanian & Paliwal, 1990; Deng & Moore, 1993; Pelleg & Moore, 1999; Alsabti et al., 1998; Kanungo et al., 2000), but these methods are not effective for $d > 10$ about. By using metric trees Moore’s “anchors” method is effective for much larger d (Moore, 2000).

The third general approach to the nearest neighbor task is to use triangle inequalities to eliminate unnecessary distance calculations. Using Lemma 1 above appears to have been proposed first by (Hodgson, 1988), then again independently by (Orchard, 1991; Montolio et al., 1992; Phillips, 2002) among others. Our application of Lemma 1 is more fine-grained than previous applications. The lemma says that if $d(x, c) < \frac{1}{2}d(c, c')$, then $d(x, c) < d(x, c')$. The algorithm of (Hodgson, 1988) only considers the center c' that is closest to c . If $d(x, c) < d(c, c')/2$ for this c' then x remains assigned to c . Otherwise, no distance calculations are eliminated. Our algorithm applies the lemma for every center different from c , so for most x some distance calculations are avoided, even if others must be performed.

Variants of Lemma 2 have been used by many authors, starting with (Burkhard & Keller, 1973; Vidal, 1986), but using the lemma to update a lower bound on the distance between moving points appears to be novel.

The triangle inequality applies to all distance metrics. Many papers have also been published on speeding up k -means or nearest neighbor search using inequalities that are specific for Euclidean distance, for example (Wu & Lin, 2000; Mielikainen, 2002).

Many papers have been published on approximating k -

means quickly; well-known papers include (Zhang et al., 1996; Farnstrom et al., 2000). However, the exact algorithm presented here is so fast that it is not clear when an approximate algorithm is necessary.

7. Open issues

A basic open theoretical question is whether one can find a lower bound on how many distance calculations are needed by any implementation of exact k -means. Can one construct an adversary argument showing that if any algorithm omits certain distance computations, then an opponent can choose values for these distances that, together with all other distances, satisfy the triangle inequality, yet also make the output of the algorithm incorrect?

Perhaps the most fundamental practical question for future work is how to find better clusterings, i.e. better local optima. Now that we can run k -means fast, how can we use additional computation to get answers of better quality?

One common approach to finding better local optima is to run k -means with many different initializations. The algorithm above allows many more initializations to be tried in the same total time. Another widespread heuristic for finding better clusterings is to run k -means with a large value for k , and then to merge or prune the clusters obtained into a good clustering with smaller k . Since our algorithm makes the running time of k -means sublinear in k , it is especially useful for this approach.

A third important open question is how to accelerate clustering methods that use soft assignment of points to centers. Two important methods in this class are Gaussian expectation-maximization (EM) (Dempster et al., 1977) and harmonic k -means (Hamerly & Elkan, 2002). In these methods each center is recomputed as the weighted average of all points, where weights are related to distances. Can triangle inequalities (or other inequalities!) be applied to obtain upper bounds on weights that are close to zero, and hence to obtain approximate soft assignment solutions quickly?

Acknowledgments

Thanks to Sanjoy Dasgupta, Ari Frank, Greg Hamerly, Doug Turnbull, and others for providing useful comments and datasets.

References

- Aggarwal, C. C., Hinneburg, A., & Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional spaces. *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings* (pp. 420–434). Springer.

- Alsabti, K., Ranka, S., & Singh, V. (1998). An efficient k -means clustering algorithm. *IPPS/SPDP Workshop on High Performance Data Mining*. IEEE Computer Society Press.
- Bei, C.-D., & Gray, R. M. (1985). An improvement of the minimum distortion encoding algorithm for vector quantization. *IEEE Transactions on Communications*, 33, 1132–1133.
- Burkhard, W. A., & Keller, R. M. (1973). Some approaches to best-match file searching. *Communications of the ACM*, 16, 230–236.
- Cheng, D.-Y., Gersho, A., Ramamurthi, B., & Shoham, Y. (1984). Fast search algorithms for vector quantization and pattern matching. *International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 9.11.1–9.11.4). IEEE Computer Society Press.
- Dasgupta, S. (2000). Experiments with random projection. *Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI'00)* (pp. 143–151). Morgan Kaufmann.
- Dasgupta, S. (2002). Performance guarantees for hierarchical clustering. *Fifteenth Annual Conference on Computational Learning Theory (COLT'02)* (pp. 351–363). Springer Verlag.
- Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39, 1–38.
- Deng, K., & Moore, A. W. (1993). Multiresolution instance-based learning. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1233–1239). San Francisco: Morgan Kaufmann.
- Faber, V. (1994). Clustering and the continuous k -means algorithm. *Los Alamos Science*, 138–144.
- Farnstrom, F., Lewis, J., & Elkan, C. (2000). Scalability for clustering algorithms revisited. *ACM SIGKDD Explorations*, 2, 51–57.
- Hamerly, G., & Elkan, C. (2002). Alternatives to the k -means algorithm that find better clusterings. *Proceedings of the Eleventh International Conference on Information and Knowledge Management* (pp. 600–607). McLean, Virginia, USA: ACM Press.
- Hodgson, M. E. (1988). Reducing computational requirements of the minimum-distance classifier. *Remote Sensing of Environments*, 25, 117–128.
- Indyk, P., Amir, A., Efrat, A., & Samet, H. (1999). Efficient algorithms and regular data structures for dilation, location and proximity problems. *Proceedings of the Annual Symposium on Foundations of Computer Science* (pp. 160–170).
- Indyk, P., & Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. *Proceedings of the Annual ACM Symposium on the Theory of Computing* (pp. 604–613).
- Judd, D., McKinley, P. K., & Jain, A. K. (1998). Large-scale parallel data clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20, 871–876.
- Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., & Wu, A. Y. (2000). The analysis of a simple k -means clustering algorithm. *ACM Symposium on Computational Geometry* (pp. 100–109). ACM Press.
- Lloyd, S. P. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28, 129–137.
- MacQueen, J. B. (1965). On convergence of k -means and partitions with minimum average variance. *Annals of Mathematical Statistics*, 36, 1084. Abstract only.
- McNames, J. (2000). Rotated partial distance search for faster vector quantization encoding. *IEEE Signal Processing Letters*, 7, 244–246.
- Mielikainen, J. (2002). A novel full-search vector quantization algorithm based on the law of cosines. *IEEE Signal Processing Letters*, 9, 175–176.
- Monge, A. E., & Elkan, C. P. (1996). The field matching problem: Algorithms and applications. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (pp. 267–270). Portland, Oregon: AAAI Press (distributed by MIT Press).
- Montolio, P., Gasull, A., Monte, E., Torres, L., & Marques, F. (1992). Analysis and optimization of the k -means algorithm for remote sensing applications. In A. Sanfeliu (Ed.), *Pattern recognition and image analysis*, 155–170. World Scientific.
- Moore, A. W. (2000). The anchors hierarchy: Using the triangle inequality to survive high dimensional data. *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence* (pp. 397–405). Morgan Kaufmann.
- Orchard, M. T. (1991). A fast nearest-neighbor search algorithm. *International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 2297–2300). IEEE Computer Society Press.
- Pelleg, D., & Moore, A. (1999). Accelerating exact k -means algorithms with geometric reasoning. *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99)* (pp. 277–281).
- Phillips, S. J. (2002). Acceleration of k -means and related clustering algorithms. *Fourth International Workshop on Algorithm Engineering and Experiments (ALENEX)* (pp. 166–177). Springer Verlag.
- Ramasubramanian, V., & Paliwal, K. K. (1990). A generalized optimization of the $k - d$ tree for fast nearest-neighbour search. *Fourth IEEE Region 10 International Conference (TENCON'89)* (pp. 565–568). IEEE Computer Society Press.
- Vidal, E. (1986). An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognition Letters*, 4, 145–157.
- Wu, K.-S., & Lin, J.-C. (2000). Fast VQ encoding by an efficient kick-out condition. *IEEE Transactions on Circuits and Systems for Video Technology*, 10, 59–62.
- Zhang, T., Ramakrishnan, R., & Livny, M. (1996). BIRCH: an efficient data clustering method for very large databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 103–114). ACM Press.

When the dataset is known in advance to contain correlated attributes, the independence assumption no longer holds. Instead, two attributes can be modeled jointly using a bivariate normal distribution, in which each has its own mean value but the two standard deviations are replaced by a “covariance matrix” with four numeric parameters. There are standard statistical techniques for estimating the class probabilities of instances and for estimating the means and covariance matrix given the instances and their class probabilities. Several correlated attributes can be handled using a multivariate distribution. The number of parameters increases with the square of the number of jointly varying attributes. With n independent attributes, there are $2n$ parameters, a mean and a standard deviation for each. With n covariant attributes, there are $n + n(n + 1)/2$ parameters, a mean for each and an $n \times n$ covariance matrix that is symmetric and therefore involves $n(n + 1)/2$ different quantities. This escalation in the number of parameters has serious consequences for overfitting, as we will explain later.

To cater for nominal attributes, the normal distribution must be abandoned. Instead, a nominal attribute with v possible values is characterized by v numbers representing the probability of each one. A different set of numbers is needed for every class; kv parameters in all. The situation is very similar to the Naïve Bayes method. The two steps of expectation and maximization correspond exactly to operations we have studied before. Expectation—estimating the cluster to which each instance belongs given the distribution parameters—is just like determining the class of an unknown instance. Maximization—estimating the parameters from the classified instances—is just like determining the attribute–value probabilities from the training instances, with the small difference that in the EM algorithm instances are assigned to classes probabilistically rather than categorically. In Section 4.2 we encountered the problem that probability estimates can turn out to be zero, and the same problem occurs here too. Fortunately, the solution is just as simple—use the Laplace estimator.

Naïve Bayes assumes that attributes are independent—that is why it is called “naïve.” A pair of correlated nominal attributes with v_1 and v_2 possible values, respectively, can be replaced with a single covariant attribute with v_1v_2 possible values. Again, the number of parameters escalates as the number of dependent attributes increases, and this has implications for probability estimates and overfitting that we will come to shortly.

The presence of both numeric and nominal attributes in the data to be clustered presents no particular problem. Covariant numeric and nominal attributes are more difficult to handle, and we will not describe them here.

Missing values can be accommodated in various different ways. Missing values of nominal attributes can simply be left out of the probability calcula-

tions, as described in Section 4.2; alternatively they can be treated as an additional value of the attribute, to be modeled as any other value. Which is more appropriate depends on what it means for a value to be “missing.” Exactly the same possibilities exist for numeric attributes.

With all these enhancements, probabilistic clustering becomes quite sophisticated. The EM algorithm is used throughout to do the basic work. The user must specify the number of clusters to be sought, the type of each attribute (numeric or nominal), which attributes are modeled as covarying, and what to do about missing values. Moreover, different distributions than the ones described previously can be used. Although the normal distribution is usually a good choice for numeric attributes, it is not suitable for attributes (such as weight) that have a predetermined minimum (zero, in the case of weight) but no upper bound; in this case a “log-normal” distribution is more appropriate. Numeric attributes that are bounded above and below can be modeled by a “log-odds” distribution. Attributes that are integer counts rather than real values are best modeled by the “Poisson” distribution. A comprehensive system might allow these distributions to be specified individually for each attribute. In each case, the distribution involves numeric parameters—probabilities of all possible values for discrete attributes and mean and standard deviation for continuous ones.

In this section we have been talking about clustering. But you may be thinking that these enhancements could be applied just as well to the Naïve Bayes algorithm too—and you’d be right. A comprehensive probabilistic modeler could accommodate both clustering and classification learning, nominal and numeric attributes with a variety of distributions, various possibilities of covariation, and different ways of dealing with missing values. The user would specify, as part of the domain knowledge, which distributions to use for which attributes.

Bayesian clustering

However, there is a snag: overfitting. You might say that if we are not sure which attributes are dependent on each other, why not be on the safe side and specify that *all* the attributes are covariant? The answer is that the more parameters there are, the greater the chance that the resulting structure is overfitted to the training data—and covariance increases the number of parameters dramatically. The problem of overfitting occurs throughout machine learning, and probabilistic clustering is no exception. There are two ways that it can occur: through specifying too large a number of clusters and through specifying distributions with too many parameters.

The extreme case of too many clusters occurs when there is one for every data point: clearly, that will be overfitted to the training data. In fact, in the

mixture model, problems will occur whenever any of the normal distributions becomes so narrow that it is centered on just one data point. Consequently, implementations generally insist that clusters contain at least two different data values.

Whenever there are a large number of parameters, the problem of overfitting arises. If you were unsure of which attributes were covariant, you might try out different possibilities and choose the one that maximized the overall probability of the data given the clustering that was found. Unfortunately, the more parameters there are, the larger the overall data probability will tend to be—not necessarily because of better clustering but because of overfitting. The more parameters there are to play with, the easier it is to find a clustering that seems good.

It would be nice if somehow you could penalize the model for introducing new parameters. One principled way of doing this is to adopt a fully Bayesian approach in which every parameter has a prior probability distribution. Then, whenever a new parameter is introduced, its prior probability must be incorporated into the overall likelihood figure. Because this will involve multiplying the overall likelihood by a number less than one—the prior probability—it will automatically penalize the addition of new parameters. To improve the overall likelihood, the new parameters will have to yield a benefit that outweighs the penalty.

In a sense, the Laplace estimator that we met in Section 4.2, and whose use we advocated earlier to counter the problem of zero probability estimates for nominal values, is just such a device. Whenever observed probabilities are small, the Laplace estimator exacts a penalty because it makes probabilities that are zero, or close to zero, greater, and this will decrease the overall likelihood of the data. Making two nominal attributes covariant will exacerbate the problem. Instead of $v_1 + v_2$ parameters, where v_1 and v_2 are the number of possible values, there are now $v_1 v_2$, greatly increasing the chance of a large number of small estimated probabilities. In fact, the Laplace estimator is tantamount to using a particular prior distribution for the introduction of new parameters.

The same technique can be used to penalize the introduction of large numbers of clusters, just by using a prespecified prior distribution that decays sharply as the number of clusters increases.

AutoClass is a comprehensive Bayesian clustering scheme that uses the finite mixture model with prior distributions on all the parameters. It allows both numeric and nominal attributes and uses the EM algorithm to estimate the parameters of the probability distributions to best fit the data. Because there is no guarantee that the EM algorithm converges to the global optimum, the procedure is repeated for several different sets of initial values. But that is not all. AutoClass considers different numbers of clusters and can consider different amounts of covariance and different underlying probability distribution types

for the numeric attributes. This involves an additional, outer level of search. For example, it initially evaluates the log-likelihood for 2, 3, 5, 7, 10, 15, and 25 clusters: after that, it fits a log-normal distribution to the resulting data and randomly selects from it more values to try. As you might imagine, the overall algorithm is extremely computation intensive. In fact, the actual implementation starts with a prespecified time bound and continues to iterate as long as time allows. Give it longer and the results may be better!

Discussion

The clustering methods that have been described produce different kinds of output. All are capable of taking new data in the form of a test set and classifying it according to clusters that were discovered by analyzing a training set. However, the incremental clustering method is the only one that generates an explicit knowledge structure that describes the clustering in a way that can be visualized and reasoned about. The other algorithms produce clusters that could be visualized in instance space if the dimensionality were not too high.

If a clustering method were used to label the instances of the training set with cluster numbers, that labeled set could then be used to train a rule or decision tree learner. The resulting rules or tree would form an explicit description of the classes. A probabilistic clustering scheme could be used for the same purpose, except that each instance would have multiple weighted labels and the rule or decision tree learner would have to be able to cope with weighted instances—as many can.

Another application of clustering is to fill in any values of the attributes that may be missing. For example, it is possible to make a statistical estimate of the value of unknown attributes of a particular instance, based on the class distribution for the instance itself and the values of the unknown attributes for other examples.

All the clustering methods we have examined make a basic assumption of independence among the attributes. AutoClass does allow the user to specify in advance that two or more attributes are dependent and should be modeled with a joint probability distribution. (There are restrictions, however: nominal attributes may vary jointly, as may numeric attributes, but not both together. Moreover, missing values for jointly varying attributes are not catered for.) It may be advantageous to preprocess a dataset to make the attributes more independent, using a statistical technique such as the principal components transform described in Section 7.3. Note that joint variation that is specific to particular classes will not be removed by such techniques; they only remove overall joint variation that runs across all classes.

Our description of how to modify k -means to find a good value of k by repeatedly splitting clusters and seeing whether the split is worthwhile follows

Visually Mining and Monitoring Massive Time Series

Jessica Lin^a Eamonn Keogh^a Stefano Lonard^a Jeffrey P. Lankford^b Donna M. Nystrom^b

^aComputer Science & Engineering Department
University of California, Riverside
Riverside, CA 92521
{jessica, eamonn, stelo}@cs.ucr.edu

^bThe Aerospace Corporation
El Segundo, CA 90245-4691
{Jeffrey.P.Lankford, Donna.M.Nystrom}@aero.org

ABSTRACT

Moments before the launch of every space vehicle, engineering discipline specialists must make a critical *go/no-go* decision. The cost of a false positive, allowing a launch in spite of a fault, or a false negative, stopping a potentially successful launch, can be measured in the tens of millions of dollars not including the cost in morale and other more intangible detriments. The Aerospace Corporation is responsible for providing engineering assessments critical to the *go/no-go* decision for every Department of Defense space vehicle. These assessments are made by constantly monitoring streaming telemetry data in the hours before launch. We will introduce VizTree, a novel time-series visualization tool to aid the Aerospace analysts who must make these engineering assessments. VizTree was developed at the University of California, Riverside and is unique in that the same tool is used for mining archival data and monitoring incoming live telemetry. The use of a single tool for both aspects of the task allows a natural and intuitive transfer of mined knowledge to the monitoring task. Our visualization approach works by transforming the time series into a symbolic representation, and encoding the data in a modified suffix tree in which the frequency and other properties of patterns are mapped onto colors and other visual properties. We demonstrate the utility of our system by comparing it with state-of-the-art batch algorithms on several real and synthetic datasets.

Categories and Subject Descriptors

H.3.3 [Information Search and Retrieval]: *Information filtering, Query formulation, Selection process* H.2.8 [Database Applications]: *Data mining, Scientific databases.*

General Terms

Algorithms, Design, Experimentation, Human Factors.

Keywords

Time Series, Visualization, Motif Discovery, Anomaly Detection, Pattern Discovery

Dr. Keogh is supported by NSF Career Award IIS-0237918

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or to publish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD '04, August 22–25, 2004, Seattle Washington, U.S.A.
Copyright 2004 ACM 158113-888-1/04/0008...\$5.00.

1. INTRODUCTION

One of the crucial responsibilities of The Aerospace Corporation is to provide engineering assessments for the government engineering discipline specialists who make the critical *go/no-go* decision moments before the launch of every space vehicle launched by the DoD. The cost of a false positive, allowing a launch in spite of a fault, or a false negative, stopping a potentially successful launch, can be measured in the tens of millions of dollars, not including the cost in morale and other more intangible detriments to the U.S. defense program.

The launch monitoring facilities at Aerospace are similar to the familiar Hollywood movie recreations [26]. There are several rows of work cells, each with a computer display and a headset. Each work cell is devoted to one analyst, for example, propulsion, guidance, electrical, etc. Each display presents some common data (say vehicle location and orientation), as well as data specific to that discipline.

The analyst making the engineering assessments has access to data from previous launches and must constantly monitor streaming telemetry from the current mission.

Currently, the analysts use electronic strip charts similar to those used to record earthquake shock on paper rolls. However, while these charts illustrate the recent history of each sensor, they do not provide any useful higher-level information that might be valuable to the analyst.

To reduce the possibility of wrong *go/no-go* decisions, Aerospace is continually investing in research. There are two major directions of research in this area.

- Producing better techniques to mine the archival launch data from previous missions. Finding rules, patterns, and regularities from past data can help us *know what to expect* for future missions, and allow more accurate and targeted monitoring, contingency planning, etc [26].
- Producing better techniques to visualize the streaming telemetry data in the hours before launch. This is particularly challenging because an analyst may have to monitor as many as dozens of rapidly changing sensors [26].

Although these two tasks are quite distinct, and are usually tackled separately, the contribution of this work is to introduce a single framework that can address both. Having a single tool for both tasks allows knowledge gleaned in the *mining* stage to be represented in the same visual language of the *monitoring* stage, thus allowing a more natural and intuitive transfer of knowledge.

More concretely, we propose VizTree: a time series pattern discovery and visualization system based on augmenting suffix trees. VizTree simultaneously visually summarizes both the global and local structures of time series data. In addition, it provides novel interactive

solutions to many pattern discovery problems, including the discovery of frequently occurring patterns (motif discovery) [7, 29, 38], surprising patterns (anomaly detection) [9, 24, 36], and query by content [11, 15, 21, 34]. The user-interactive paradigm allows users to visually explore the time series and perform real-time hypotheses testing [1, 19].

We employ the widely referenced *Overview, zoom & filter, details on demand* paradigm of Dr. Ben Shneiderman of the University of Maryland [37]. As we will show in this paper, our work fits neatly into these principles. We give an overview of the global structure of an arbitrarily long time series in constant space, while we allow the user to zoom in on particular local structures and patterns, and provide details on demand for patterns and regularities that the user has tentatively identified.

While there are several systems for visualizing time series in the literature, our approach is unique in several respects. First, almost all other approaches assume highly periodic time series [40, 41], whereas ours makes no such assumption. Other methods typically require space (both memory space, and pixel space) that grows at least linearly with the length of the time series, making them untenable for mining massive datasets. Finally, our approach allows us to visualize a much richer set of features, including global summaries of the differences between two time series, locally repeated patterns, anomalies, etc.

While the evaluation of visualization systems is often subjective, we will evaluate our system with objective experiments by comparing our system with state-of-the-art algorithms on several real and synthetic datasets.

The rest of the paper is organized as follows. In Section 2 we review necessary background material and survey related work. We introduce our system VizTree in Section 3. In Section 4, we extend the idea to further allow comparison and contrast between two time series. Section 5 contains a detailed empirical evaluation of our system. We conclude in Section 6.

We note that all the figures in this text suffer from their small scale and monochromatic printing. We encourage the interested reader to visit [27] to view high-resolution full-color examples.

2. BACKGROUND AND RELATED WORK

We begin this section by briefly reviewing the most important time series data mining tasks. We will then consider current visualization techniques and explain why they are unsuited to the problem at hand.

2.1 Time Series Data Mining Tasks

For lack of space, this brief introduction to the important time series data mining tasks is necessarily subjective and somewhat domain driven. Nevertheless, these three tasks cover the majority of time series data mining research [6, 7, 9, 11, 15, 18, 22, 24, 29, 30, 31, 38].

2.1.1 Subsequence Matching

Sequence matching is perhaps the most widely studied area of time series data mining [11, 15]. The task has long been divided into two categories: whole matching and subsequence matching [11, 21].

- **Whole Matching** a query time series is matched against a database of individual time series to identify the ones similar to the query.
- **Subsequence Matching** a short query subsequence time series is matched against longer time series by sliding it along the longer sequence, looking for the best matching location.

While there are literally hundreds of methods proposed for whole sequence matching (see, e.g., [22] and references therein), in practice, its application is limited to cases where some information about the data is known *a priori*.

Subsequence matching can be generalized to whole matching by dividing sequences into non-overlapping sections. For example, we may wish to take a long electrocardiogram and extract the individual heartbeats. This informal idea has been used by many researchers and is also an optional feature of VizTree. We will therefore formally name this transformation *chunking*, and define it below.

Definition 1 *Chunking* the process where a time series is broken into individual time series by either a specific period or, more arbitrarily, by its shape.

The former usually applies to periodic data, for example consider power usage data provided by a Dutch research facility (this dataset is used as a running example in this work, see Figures 3 and 15): the data can be chunked by days, weeks, etc. The latter applies to data having regular structure or repetitive shape, but not necessarily having the same length for each occurrence. Electrocardiogram data are such an example, and they can be separated into individual heartbeats.

There is increasing awareness that for many data mining and information retrieval tasks, very fast approximate search is preferable to slower exact search [5]. This is particularly true for exploratory purposes and hypotheses testing. Consider the stock market data. While it makes sense to look for approximate patterns, for example, “a pattern that rapidly decreases after a long plateau” it seems pedantic to insist on exact matches. As we will demonstrate in Section 5.1, our application flows rapid approximate sequence matching.

2.1.2 Anomaly Detection

In time series data mining and monitoring, the problem of detecting anomalous/surprising/novel patterns has attracted much attention [9, 30, 36]. In contrast to subsequence matching, anomaly detection is identification of previously *unknown* patterns. The problem is particularly difficult because what constitutes an anomaly can greatly differ depending on the task at hand. In a general sense, an anomalous behavior is one that deviates from “normal” behavior. While there have been numerous definitions given for anomalous or surprising behaviors, the one given by Keogh et al. [24] is unique in that it requires no explicit formulation of what is anomalous. Instead, they simply defined an anomalous pattern as one whose frequency of occurrences differs substantially from that expected, given previously seen data. Their definition was implemented in an algorithm (called “Tarzan”) that was singled out by NASA as an algorithm that has great promise in the long term [17]. As it will become clearer later, a subset of the system that we propose here includes what may be considered a visual encoding of Tarzan.

2.1.3 Time Series Motif Discovery

In bioinformatics, it is well documented that overrepresented DNA sequences often have biological significance [2, 10, 35]. Other applications that rely heavily on overrepresented (and underrepresented) pattern discovery include intrusion detection, fraud detection, web usage prediction, financial analysis, etc.

A substantial body of literature has been devoted to techniques to discover such overrepresented patterns in time series; however, each work considered a different definition of pattern [3, 32]. In previous work, we unified and formalized the problem by defining the concept of “time series motif” [29]. Time series motifs are close analogues of

their discrete cousins, although the definitions must be augmented to prevent certain degenerating solutions. This definition is gaining acceptance, and now being used in animation [4], mining human motion data [38], and several other applications. The naïve algorithm to discover motifs is quadratic in the length of the time series. In [29], we demonstrated a simple technique to mitigate the quadratic complexity by a large constant factor; nevertheless this time complexity is clearly untenable for most real datasets. As we shall demonstrate in Section 5.2, VizTree allows users to visually discover approximate motifs in real time.

2.2 Visualizing Time Series

Time series is perhaps the most common data type encountered in data mining, touching as it does, almost every aspect of human life, including medicine (ECG, EEG data) finance (stock market data, credit card usage data), aerospace (lunch telemetry, satellite sensor data), entertainment (music, movies) [4], etc. Because time series datasets are often massive (in gigabytes or even terabytes), time- and space-complexity is of paramount importance.

Surprisingly, although the human eye is often advocated as the ultimate data-mining tool [19, 37, 39], there has been relatively little work on visualizing massive time series datasets. We have reimplemented the three most referenced approaches in the literature. Below, we will briefly review them and explain why they are not suited to the task at hand.

2.2.1 TimeSearcher

TimeSearcher [14] is a time series exploratory and visualization tool that allows users to retrieve time series by creating queries. This is achieved by use of “TimeBoxes”, which are rectangular query locators that specify the region(s) in which the users are interested within any given time series. In Figure 1, three TimeBoxes have been drawn to specify time series that start low, increase, then fall once more.

The main advantage of this tool is its flexibility. In particular, unlike conventional query-by-content similarity search algorithms, TimeSearcher allows users to specify different regions of interest from a query time series, rather than feeding the entire query for matching. This is useful when users are interested in finding time series that exhibit similar behaviors as the query time series in only specific regions.

While TimeSearcher and VizTree proposed here both serve as visualization and exploratory tools for time series, their functionalities are fundamentally different. For example, TimeSearcher is a query-by-example tool for multiple time series data. Even with its flexibility, users still need to specify the query regions in order to find similar patterns. In other words, some knowledge about the datasets may be needed in advance and users need to have a general idea of what is interesting. On the other hand, VizTree serves as a true pattern discovery tool for a long time series that tentatively identifies and isolates interesting patterns and invites further inspection by the analyst.

The functionality of TimeSearcher for similarity search is implicit in the design of our system: similar patterns are automatically grouped together. Furthermore, TimeSearcher suffers from its limited scalability, which restricts its utility to smaller datasets, and is impractical for the task at hand.

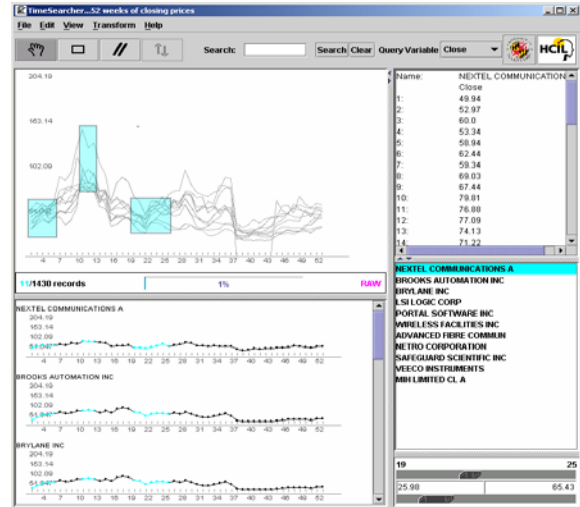


Figure 1: The TimeSearcher visual query interface. A user can filter away sequences that are not interesting by insisting that all sequences have at least one data point within the query boxes.

2.2.2 Cluster and Calendar-Based Visualization

Another time series visualization system is cluster and calendar-based, developed by [40]. The time series data are chunked into sequences of day patterns, and these day patterns are in turn clustered using a bottom-up clustering algorithm. This visualization system displays patterns represented by cluster averages, as well as a calendar with each day color-coded by the cluster that it belongs to. Figure 2 shows just one view of this visualization scheme.

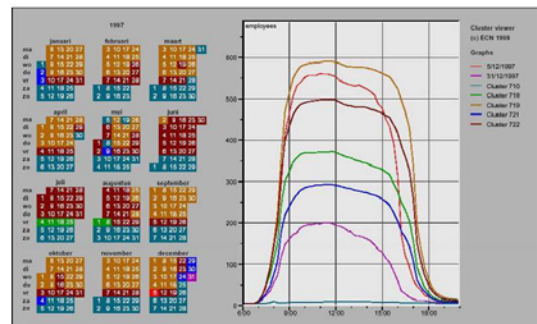


Figure 2: The cluster and calendar-based visualization on employee working hours data. It shows six clusters, representing different working-day patterns.

While the calendar-based approach provides a good overview of the data, its application is limited to calendar-based data, that is to say, data which has some regularity imposed on it by social or financial dependence on the calendar. This approach is of little utility for data without obvious daily/weekly patterns and/or *a priori* knowledge about such patterns. In short, this system works well to find patterns within a specific, known time scale, while our system aims to discover previously unknown patterns with little or no knowledge about the data.

2.2.3 Spiral

Weber et. al developed a tool that visualizes time series on spirals [41]. Each periodic section of time series is mapped onto one “ring” and attributes such as color and line thickness are used to characterize the data values. The main use of this approach is the identification of periodic structures in the data. However, the utility of this tool is

limited for time series that do not exhibit periodic behaviors, or when the period is unknown.

We reimplemented the spiral approach and ran it on the power consumption dataset. A screenshot of the resulting spiral is shown in Figure 3

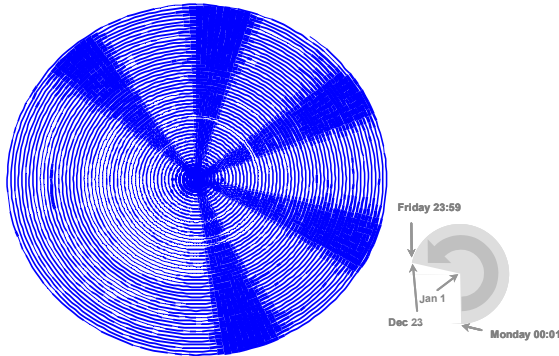


Figure 3: The Spiral visualization approach of Weber et al. applied to the power usage dataset.

Note that one can clearly visualize the normal “9-to-5” working week pattern. In addition, one can see several other interesting events. For example, while it is apparent that one works during weekends in general, on one Saturday in late summer, there was a power demand suggestive of a full days shift. Surprisingly, this idea for visualizing time series predates computers, with elegant hand drawn examples dating back to at least the 1880’s [12, 39].

While the Spiral approach is elegant, it does not meet our requirements for several reasons. As mentioned, it works well only for periodic data (based on the original authors’ claims and our own experiments). More importantly, it requires pixel space linear in the length of the time series; this is simply untenable for our purposes.

3. OUR APPROACH: VIZ-TREE

Our visualization approach works by transforming the time series into a symbolic representation, and encoding the data in a modified suffix tree in which the frequency and other properties of patterns are mapped onto colors and other visual properties. Before explaining our approach in detail, we will present a simple problem that motivates our work.

Two sets of binary sequences of length 200 were generated: the first set by the pseudo-random-number generator by the computer, and the second set by hand by a group of volunteers. The volunteers were asked to try and make the bit strings as random as possible, and were offered a prize to motivate them. Figure 4 shows one sample sequence from each set.

By simply looking at the original bit strings, it’s difficult, if not impossible, to distinguish the computer-generated from the human-constructed numbers. However, if we represent them with a tree structure where the frequencies of subsequences are encoded in the thickness of branches, the distinction becomes clear. For clarity, the trees are pruned at depth three. Each tree represents one sequence from each set, and each node in the tree has exactly two branches: the

upper branch represents 1, and the lower branch represents 0. The tree is constructed as follows: starting from the beginning of each sequence, subsequences of length three are extracted with a sliding window that slides across the sequence one bit at a time. So for the first sequence we get a set of subsequences $\{(0,1,0), (1,0,1), (0,1,1), \dots\}$.

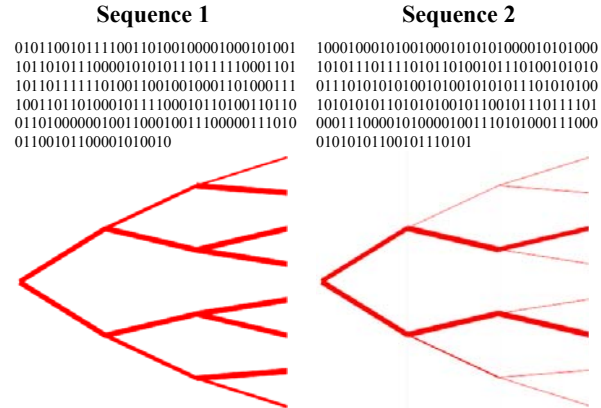


Figure 4: (Left) Computer-generated random bits presented as an augmented suffix tree (Right) Human-constructed bits presented as an augmented suffix tree.

For the tree shown on the left in Figure 4, the branches at any given level have approximately the same thickness, which means that the probabilities of subsequences at any given level are approximately evenly distributed. In contrast, the tree on the right shows that subsequences of alternating 0’s and 1’s dominate the whole sequence. The “motifs” for the sequence, 101 and 010, can be easily identified, since they appear more frequently than the other subsequences.

The non-randomness, which can be seen very clearly in this example, implies that humans usually try to “fake” randomness by alternating patterns [16]. Undoubtedly, there exist other solutions to uncover these “patterns” (entropy, Hidden Markov models, etc.). Nonetheless, what this visualization scheme provides is a straightforward solution that allows users to easily identify and view the patterns in a way intuitive to human perception.

The simple experiment demonstrates how visualizing augmented suffix trees can provide an overall visual summary, and potentially reveal hidden structures in the data. Since the strings represented in the tree are in fact “subsequences” rather than “suffixes,” we call such trees *subsequence trees*.

This simple experiment motivates our work. Although time series are not discrete, they can be discretized with little loss of information, thus allowing the use of suffix/subsequence trees.

Our system is partly inspired by Visualysis [25], a visualization tool for biological sequences. Visualysis uses a suffix tree to store the biological sequences and, through the properties of the tree, such as bushiness, branch distribution, etc. and user navigation, interesting biological information can be discovered [25]. Visualysis incorporates algorithms that utilize suffix trees in computational biology; more specifically, exact sequence matching and tandem repeat algorithms. At a first glance, our visualization system is similar to Visualysis in the sense that it also has the objective of pattern discovery using a tree structure. However, several characteristics that are unique to our application make it more diversely functional than its computational-biology counterpart. First, although the tree structure needs the data to be discrete, the

¹ Of all the figures in this paper, this one suffers the most from the small scale of reproduction. In addition we did not optimize the anti-aliasing and other graphic tricks to make the hard copy reproduction as good as the on screen version. We encourage the interested reader to refer to the original paper [29] for much higher quality images.

original time series data is not. Using a time-series discretization method that we introduced in an earlier work [28], continuous data can be transformed into discrete domain, with certain desirable properties such as lower-bounding distance, dimensionality reduction, etc. Second, instead of using a suffix tree, we use a subsequence tree that maps all subsequences onto the branches of the tree. Thus, given the same parameters, the trees have the same overall shape for any dataset. This approach makes comparing two time series easy and anomaly detection possible.

3.1 The Utility of Discretizing Time Series

In [28], we introduced Symbolic Aggregate approximation (SAX), a novel symbolic representation for time series. It is ideal for this application since, unlike all previously proposed discretization methods for time series, SAX allows lower-bounding distance measures to be defined on the symbolic space. In addition, its dimensionality reduction feature makes approximating large dataset feasible, and its ability to convert the data using merely the local information, without having to access the entire dataset, is especially desirable for streaming time series. The utility of SAX has been demonstrated in [28], and the adaptation or extension of SAX by other researchers further shows its impact in diverse fields such as medical and video [6, 33]. For these reasons, we choose to use SAX as the discretization method for the input time series data.

Before converting a time series to symbols, it should be normalized. The importance of normalization has been extensively documented in the past [22]. Without normalization, many time series data mining tasks have little meaning [22]. After normalization, SAX performs the discretization in two steps. First, a time series T of length m is divided into w equal-sized segments; the values in each segment are then approximated and replaced by a single coefficient, which is their average. Aggregating these w coefficients form the Piecewise Aggregate Approximation (PAA) representation of T .

Next, to convert the PAA coefficients to symbols, we determine the breakpoints that divide the distribution space into α equiprobable regions, where α is the alphabet size specified by the user. In other words, the breakpoints are determined such that the probability of a segment falling into any of the regions is approximately the same. If the symbols were not equiprobable, some of the substrings would be more probable than others. As a consequence, we would inject a probabilistic bias in the process. In [8], Crochemore et. al. showed that a suffix tree automation algorithm is optimal if the letters are equiprobable.

Once the breakpoints are determined, each region is assigned a symbol. The PAA coefficients can then be easily mapped to the symbols corresponding to the regions in which they reside. In [28], the symbols are assigned in a bottom-up fashion so the PAA coefficient that falls in the lowest region is converted to "a," the one above to "b," and so forth. In this paper, for reason that will become clear in the next section we reverse the assigning order, so the regions will be labeled top-down instead (i.e. the top-most region is labeled "a," the one below it "b," and so forth). Figure 5 shows an example of a time series being converted to string `acdcbdba`. Note the general shape of the time series is preserved, in spite of the massive amount of dimensionality reduction, and the symbols are equiprobable.

The discretization technique can be applied to VizTree by calling SAX repeatedly for each subsequence. More specifically, subsequences of specified length are extracted from the input time series and normalized to have a mean of zero and a standard deviation of one. Applying SAX on these subsequences, we obtain a set of strings. From this point on, the steps are identical to the motivating example shown in the beginning of Section 3: the strings are inserted into the subsequence tree one by one.

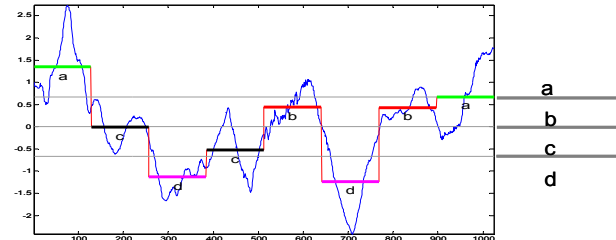


Figure 5: A time series dataset “leleccum” of length 1024 is converted into an eight-symbol string `acdcbdba`. Note that the general shape of the time series is preserved, in spite of the massive amount of dimensionality reduction.

3.2 A First Look at VizTree

Figure 6 shows a screen shot of VizTree. When the program is executed, four blank panels and a parameter-setting area are displayed. To load a time series dataset, the user selects the input file using a familiar dropdown menu. The input time series is plotted in the top left-hand panel. Next to the time series plotting window is the parameter setting area; the analyst can enter the sliding window length, the number of SAX segments per window, and select alphabet size from a dropdown menu. Once the parameters are entered, the user can click on the “Show Tree” button to display the subsequence tree on the bottom left panel.

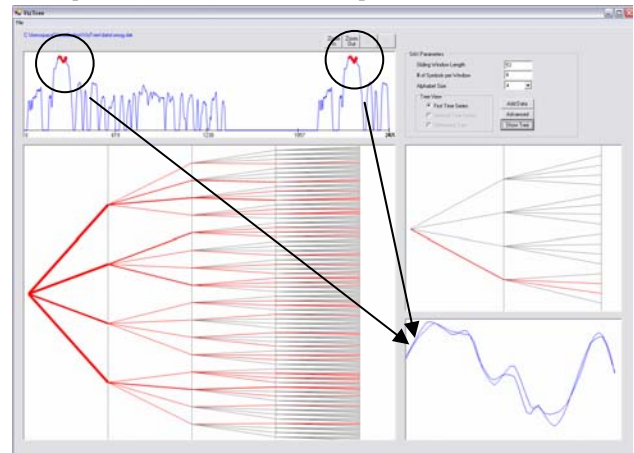


Figure 6: A screenshot of Viztree. The top panel is the input time series. The bottom left panel shows the subsequence tree for the time series. On the right, the very top is the parameter setting area. Next to the subsequence tree panel, the top window shows the zoom-in of the tree, and the bottom window plots the actual subsequences when the analyst clicks on a branch.

The time series used for this example is a real industrial dataset of smog emissions from a motor vehicle. The length of the time series is 2478. The length of the sliding window is set to 53; the number of segments (i.e., the depth of the tree) is four, and the alphabet size (i.e., the number of children for each node) is four.

Each branch represents one pattern. As mentioned in the previous section, we reverse the assigning order of the symbols from bottom-

² In the unusual event where it might be more appropriate not to normalize, for example, when offset and amplitude changes are important, VizTree provides an option to skip the normalization step.

up to top-down. The reason is that when the symbols are arranged this way, it is more consistent with the natural shape of the tree. For example, for any given node, a branch at a higher position denotes segments with higher values. Traversing breadth-first from the top-most branch of any given node, the symbols that represent the branches are a , b , c , and d , respectively. Each level of the tree represents one segment (or one symbol). To retrieve any string, we simply traverse down the appropriate branches.

Definition 2 *Pattern* a pattern p is the SAX representation of a subsequence in the time series, denoted by the strings formed by following any path down the subsequence tree. The frequency of p in time series A is denoted by $f(p_A)$, which is the number of occurrences of p over the number of all occurrences in A .

The frequency of a pattern is encoded in the thickness of the branch. For clarity, the full tree is drawn. Branches with zero frequency are drawn in light gray, while others are drawn in red with varying thicknesses.

On the right hand side of VizTree, there are two panels. The upper one shows the zoom-in of the tree shown in the left panel. This is very useful especially for deep and bushy trees. The user can click on any node (on the subsequence tree window, or recursively, on the zoom-in window) and the sub-tree rooted at this node will be displayed in this upper panel. The sub-tree shown in Figure 6 is rooted at the node representing the string $abxx$, where the xx denotes *don't-care* since we are not at the leaf level. If the user clicks on any branch, then the actual subsequences having the string represented by this particular branch will be displayed in the bottom panel and highlighted in the time series plot window. In the figure, subsequences encoded to $abdb$ are shown.

3.2.1 Parameter Selection

Three parameters need to be determined: the length of the sliding window, the number of segments, and the alphabet size. In [29] we showed the trade-off between the number of segments and the alphabet size. In general, VizTree works very well even with massive dimensionality reduction, as we will demonstrate in Section 5 (in the experiments we used no more than 5 segments). The length of the sliding window is data-dependent; however, the user can drag a range over any pattern of interest on the time series plot window and the window size will be filled in automatically.

3.3 Subsequence Matching

Subsequence matching can be done very efficiently with VizTree. Instead of feeding another time series as query, the user provides the query in an intuitive way. Recall that each branch corresponds to one of the equiprobable regions that are used to convert the PAA coefficients to symbols. The top branch corresponds to the region with the highest values, and the bottom branch corresponds to the region with the lowest values. Therefore, any path can be easily translated into a general shape and can be used as a query. For example, the top-most branch at depth one (i.e., string a^k) represents all subsequences that start with high values, or more precisely, whose values in the first segment have the mean value that resides in the highest region. In the previous example, the user is interested in finding a concave-down pattern (i.e., a U-shape). This particular pattern, according to the domain experts, corresponds to a change of gears in the motor vehicle during the smog emission test. From the U shape, the user can approximate the query to be something that goes down and comes up, or a string that starts and ends with high branches, with low branches in the middle. As a

result, clicking on the branch representing $abdb$ as shown in the figure uncovers the pattern of interest.

3.4 Motif Discovery & Simple Anomaly Detection

VizTree provides a straightforward way to identify motifs. Since the thickness of a branch denotes the frequency of the subsequences having the same, corresponding strings we can identify approximate motifs by examining the subsequences represented by thick tree paths. A feature unique to VizTree is that it allows users to visually evaluate and inspect the patterns returned. This interactive feature is important since different strings can also represent similar subsequences, such as those that differ by only one symbol. In addition, the user can prune off uninteresting or expected patterns to improve the efficiency of the system and reduce false positives. For example, for ECG data, the motif algorithm will mostly likely return normal heart beats as the most important motif, which is correct but non-useful. Allowing user to manually prune off this dominant pattern, secondary yet more interesting patterns may be revealed.

Figure 7 shows such an example. The dataset used here is *real*, industrial dataset, “winding,” which records the angular speed of a reel. The subsequences retrieved in the lower right panel have the string representation $abcb$. Examining the motifs in this dataset allowed us to discover an interesting fact: while the dataset was advertised as real, we noted that repeated patterns occur at every 1000 points. For example, in Figure 7, the two nearly identical subsequences retrieved are located at offsets 599 and 1599, exactly 1000 points apart. We checked with the original author and discovered that this is actually a synthetic dataset composed from parts of a real dataset, a fact that is not obvious from inspection of the original data.

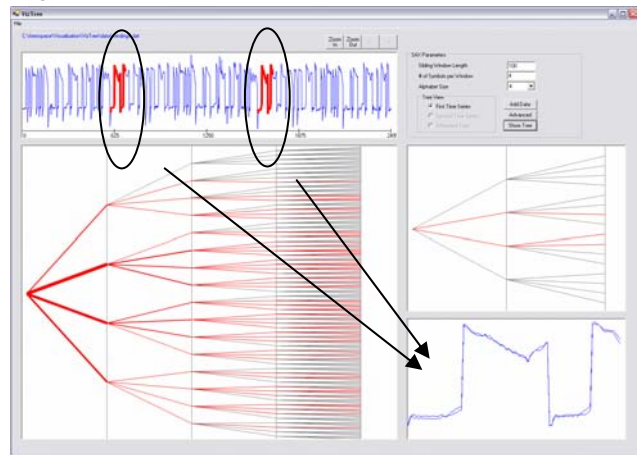


Figure 7: Example of motif discovery on the winding dataset. Two nearly identical subsequences are identified, among the other motifs.

The complementary problem of motif discovery is anomaly detection. While frequently occurring patterns can be detected by thick branches in the Viztree, simple anomalous patterns can be detected by unusually thin branches. Figure 8 demonstrates both motif discovery and simple anomaly detection on an MIT-BIH Noise Stress Test Dataset (ECG recordings) obtained from PhyoBank [13]. Here, motifs can be identified very easily from the thick branches; more remarkably, there is one very thin line straying off on its own (the path that starts with “a”). This line turns out to be an anomalous heart beat, independently annotated by a cardiologist as a premature ventricular contraction.

While anomalies can be detected this way for trivial cases, in more complex cases, the anomalies are usually detected by comparing the time series against a normal, reference time series. Anything that differs substantially from this reference time series can signal anomalies. This is exactly the objective of the Diff-Tree, as described in the next section.

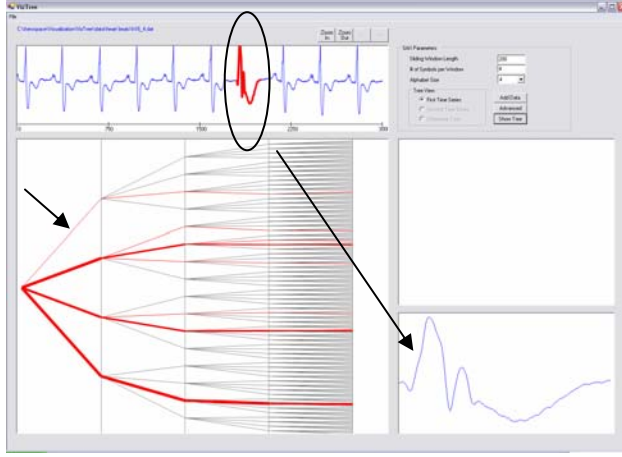


Figure 8: Heart-beat data with anomaly is shown. While the subsequence tree can be used to identify motifs, it can be used for simple anomaly detection as well.

4. DIFF-TREE

We have described how global structures, motifs, and simple anomalies can be identified by a subsequence tree. In this section, we extend these ideas to further allow the comparison of two time series by means of a “diff-tree.” A diff tree is short for “difference tree,” and as the name implies, shows the distinction between two time series. The construction of a diff-tree is fairly straightforward with the use of subsequence tree, since the overall tree shape is the same regardless of the strings, provided that the parameters selected (i.e., alphabet size, number of segment, etc) are the same. The diff-tree is constructed by computing the difference in thickness (i.e., frequency of occurrence) for each branch. Intuitively, time series data with similar structures can be expected to have similar subsequence trees, and in turn, a sparse diff-tree. In contrast, those with dissimilar structures will result in distinctly different subsequence trees and therefore a relatively dense diff-tree.

One or two datasets can be loaded to VizTree simultaneously. If only one is loaded, then its subsequence tree will be shown. If two datasets are loaded, the user has the option of viewing the subsequence tree of either one, or their diff-tree. The branches in the difference tree are color-coded to distinguish between the overrepresented and underrepresented patterns. Given two time series A and B, where A is the basis for comparison (the reference time series), and B is the added time series, we can define the following terms:

Definition 3. Overrepresented Pattern a pattern is over-represented in B if it occurs more frequently in B than it does in A.

Definition 4 Underrepresented Pattern a pattern is under-represented in B if it occurs more frequently in A than it does in B.

Definition 5. Degree of Difference the degree of difference for any pattern p between A and B is defined as follows:

$$D_p = \frac{f(p_B) - f(p_A)}{\max(\max_freq_in_A, \max_freq_in_B)} \quad (1)$$

Simply stated, D_p measures how a pattern (i.e. branch) differs from one time series to another, by computing the difference of frequencies between A and B and dividing by the maximum frequency in A and B. If p occurs less frequently in B than in A, then the pattern is underrepresented and $D_p < 0$, otherwise it is overrepresented and $D_p > 0$.

This is the measure encoded in the diff-tree as the thickness of the branch. Currently, discrete colors are used to distinguish overrepresentation from underrepresentation: overrepresented patterns are drawn in green (same color as the test time series); underrepresented patterns in blue (same color as the basis time series); and if the frequency is the same, then the branch is drawn in red. However, color intensity can be used to further highlight the degrees of difference.

4.1 Anomaly Detection

The datasets used for anomaly detection, constructed independently of the current authors and provided by the Aerospace Corporation for sanity check, are shown in Figure 9. The one on the top is the normal time series, and the one below is similar to a normal time series, except it has a gap in the middle as anomaly. Figure 10 shows a screenshot of the anomaly detection by diff-tree. The tree panel shows the diff-tree between the two datasets. The two thick paths denote the beginning and the end of the anomaly, respectively.

This is a very trivial example for demonstration purpose. However, the effect is similar for more complex cases.

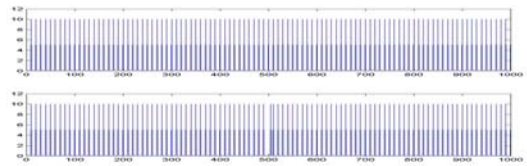


Figure 9 The input files used for anomaly detection by diff-tree. (Top) Normal time series. (Bottom) Anomaly is introduced as a gap in the middle of the dataset.

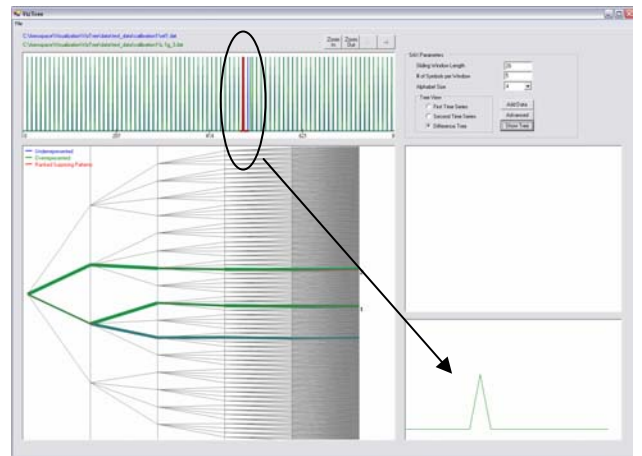


Figure 10: Diff-tree on the datasets shown in the previous figure. The gap is successfully identified.

5. EXPERIMENTAL EVALUATION

In this section we evaluate (and demonstrate) our approach on datasets which are either very intuitive to the average person or have been extensively annotated by domain experts. In particular, we will evaluate our work on human motion data and the power demand

data. Note that all datasets used here are available for free from the UCR archive [20].

5.1 Subsequence Matching

This experiment incorporates *bt* subsequence matching and motif discovery. The dataset used is the human motion data of yoga postures. A model postured yoga routines in front of a green screen, and her motion was captured by various sensors. The motion capture is transformed into a time series by computing the aspect ratio of the minimum-bounding rectangle formed around her body. The length of the time series is approximately 26,000 (i.e. there are approximately these many frames in the original video).

Suppose we are interested in finding a yoga sequence like the one in Figure 11:



Figure 11: A sample yoga sequence for approximate subsequence matching.

Then we would expect the shape of the query to descend rapidly after the first position (the width-to-height-ratio decreases), ascend slowly after the second position, descend again, and finally ascend once more. Assume that we set the number of segments to be five (an arbitrary choice), then a reasonable start would be the branch “adxxx.” Since there are only two paths extending from the node “ad,” the matches are found very quickly without much refinement in the search space. The result is shown in Figure 12 and the actual yoga sequences for the matches are outlined in Figure 13. The subsequence length is 400 (i.e. about 6.5 seconds). As the figure shows, the two sequences are very similar with only very minor distinction.

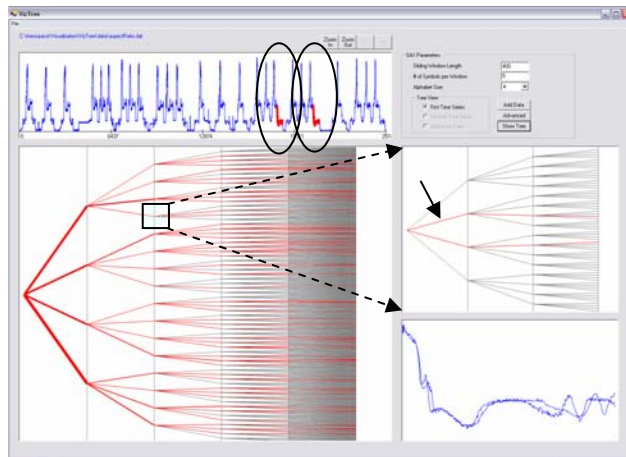


Figure 12: Matches for the yoga sequence in Figure 11. The bottom right corner shows how similar these two subsequences are.



Figure 13: Outline of the actual yoga sequences that match the query.

There are several advantages of the approximate subsequence matching by VizTree. One is that this feature is built-in to the application, and it is relatively easy to specify the query without

explicitly providing it. More importantly, the system retrieves the results very efficiently since the information is already stored in the tree. With the current state-of-the-art *exact* subsequence matching algorithms, retrieval is much too slow for a real time interaction.

5.2 Motif Discovery

For the motif discovery experiment, we will continue with the previous human-motion example. There are obviously some noticeable motifs such as the long spikes that occur throughout the sequence (see the time series plot in Figure 12). They denote the posture where the model is lying flat on the ground, when the aspect ratio is at its maximum. However, one of the desirable features of VizTree is that it allows users to visually identify secondary yet more interesting motifs. The matches found in the previous section are such example. We can zoom-in on these subsequences and examine their similarity.

From Figure 14 we can see that these two subsequences are indeed very similar to each other. Note that they both have a small dip towards the end of the sequence. However, there is a slight difference there – the dip for the first sequence occurs before that for the second sequence, and is followed by a plateau. Examining the motion captures we discover that the dip corresponds to the 6th position shown in Figure 13, right before the model stretched her arms straight in front of her. In addition, for the first sequence, the model held that last position for a longer period of time, thus the plateau following the dip. These subtle differences are difficult to notice without the motif discovery and/or the subsequence matching features in VizTree.

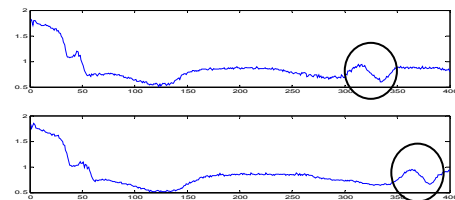


Figure 14 Zoom-ins of the two matches found in the yoga subsequence match example. Note that they both have a dip towards the end of the sequences.

For comparison, we ran the fastest known *exact* motif discovery algorithm [29]. Although the same motif can also be successfully identified, it takes minutes to compute, while VizTree gives instant (less than one second) feedback on the results. Even with the *approximate* motif discovery algorithm [7], it takes tens of seconds to complete. In addition, the visualization power of VizTree allows the user to see exactly where the motif occurs and how it maps to the original time series.

5.3 Anomaly Detection

For anomaly detection, we used the power demand data that was also used in Figure 3. Electricity consumption is recorded every 15 minutes; therefore, for the year of 1997, there are 35,040 data points. Figure 15 shows the resulting tree with the sliding window length set to 672 (exactly one week of data), and both alphabet size and number of segments to 3. The majority of the weeks follow the regular Monday-Friday, 5-working-day pattern, as shown by the thick branches. The thin branches denote the anomalies. The one circled is from the branch “bab.” The zoom-in shows the beginning of the three-day week during Christmas (Thursday and Friday off). The

other thin branches denote other anomalies³ such as New Year’s Day, Good Friday, Queen’s Birthday, etc.

While other anomaly detection algorithms such as the TSA-Tree Wavelet-based algorithm by Shahabi et. al. [36] and the Immunology-based algorithm (IMM) by Dasgupta and Forrest [9] can potentially find these anomalies as well given the right parameters, both are much more computationally intensive. While VizTree requires input of parameters the results are almost instant. In the contrary, the TSA-Tree takes tens of seconds, and IMM needs re-training its data with every adjustment of parameters, with each training session taking several minutes. This is clearly untenable for massive datasets.

In addition to the fast computational time, anomaly detection by VizTree does not always require a training dataset. As demonstrated, simple anomalies can be identified as an inverse to the motifs.

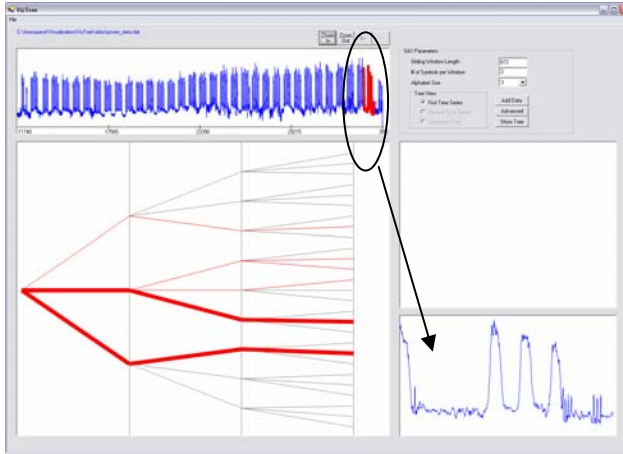


Figure 15 Anomaly detection on power consumption data. The anomaly shown here is a short week during Christmas.

5.4 Scalability

The pixel space of the subsequence tree is determined solely by the number of segments and alphabet size. In particular, we note that the pixel size of the tree is *constant* and independent to the length of time series. We have already shown that large amounts of dimensionality reduction do not greatly affect the accuracy of our results (in Section 5.3, the dimensionality is reduced from 672 to 3, a compression ratio of 224-to-1). However, the size of the dataset plays a role in memory space, since each node in the tree stores the offsets of its subsequences. However, SAX allows efficient numerosity reduction to reduce the number of subsequences being included into the tree, in addition to alleviating the problem associated with trivial matches (see below) [23, 29].

5.4.1 Numerosity Reduction

In [29] we showed that the best matches for a subsequence tend to be its immediate neighbors: the subsequence one point to the right and the subsequence one point to the left. We defined these matches to be the “trivial matches.” In the smooth regions of the time series, the amount of trivial matches might be large. If we include them in any sliding-window algorithms, the trivial matches will dominate over the true patterns due to over-counting and the results will likely be distorted, or worse, become meaningless [23]. Therefore, when

extracting subsequences from the time series by a sliding window, the trivial matches should be excluded.

Different definitions can be used to identify trivial matches. The easiest way is to compare the SAX strings and only record a subsequence if its string is different from the last one recorded. In other words, no two consecutive strings should be the same.

Additionally, we can check two strings symbol-by-symbol and consider them trivial matches of one another if no pair of symbols is more than one alphabet apart. This extra check is based on the same idea as the previous numerosity reduction option, that similar subsequences have the same SAX representation. However, it is also likely that similar subsequences do not have exactly the same SAX representations; rather, they might have alphabets that differ by at most one at any given position (i.e. the values could be very close but reside on different sides of a breakpoint).

Furthermore, the second option can be extended to also exclude non-monotonic strings. Depending on the nature of the datasets, users might only be interested in finding patterns with ups and downs.

Finally, the ultimate numerosity reduction can be achieved by chunking, which allows no overlapping subsequences. This has been used for many approaches; however we would like to note that it is only useful if the dataset exhibits regular patterns, either by shape or by period. For example, if we use chunking for the power consumption data used in Section 5.3, then we get an even more distinctive tree.

6. CONCLUSIONS AND FUTURE WORK

We proposed VizTree, a novel visualization framework for time series that summarizes the global and local structures of the data. We demonstrated how pattern discovery can be achieved very efficiently with VizTree.

As mentioned, VizTree will be formally evaluated by The Aerospace Corp in the summer of 2004, and we will incorporate the feedback into the system. We believe that researchers from other sectors of the industry can greatly benefit from our system as well. For example, it could potentially be used for indexing and editing video sequences. We plan to have domain experts in other fields such as medicine and animation evaluate our system.

In the beginning of the paper we mention that the system can be used for monitoring and mining time series data. While we mainly focus on the “mining” aspect in this paper, we will extend VizTree to accept online streaming data for monitoring purposes.

Reproducible Research Statement All datasets and code used in this work will be freely available. For higher-quality images and more information, please visit <http://www.cs.ucr.edu/~jessica/VizTree.htm>.

7. ACKNOWLEDGMENTS

Thanks to Victor Zordan and Bhriгу Celly for providing the yoga postures data.

8. REFERENCES

- [1] Aggarwal, C. (2002). Towards Effective and Interpretable Data Mining by Visual Interaction. In *SIGKDD Explorations* Jan, 2002.
- [2] Apostolico, A., Bock, M. E. & Lonardi, S. (2002). Monotony of Surprise in Large-Scale Quest for Unusual Words. In *Proceedings of the 6th Int'l conference on Research in Computational Molecular Biology* Washington, D.C., Apr 18-21. pp. 22-31.

³ Anomalies in the sense that the electricity consumption is abnormal given the day of the week.

- [3] Caraca-Valente, J. P. & Lopez-Chavarrias, I. (2000). Discovering Similar Patterns in Time Series. *In proceedings of the 6th Int'l Conference on Knowledge Discovery and Data Mining* Boston, MA. pp. 497-505.
- [4] Cardle, M. (2004). Ph.D Thesis, in progress. University of Cambridge.
- [5] Chang, C. L. E., Garcia-Molina, H. & Wiederhold, G. (2002). Clustering for Approximate Similarity Search in High-Dimensional Space. *IEEE Transactions on Knowledge and Data Engineering* vol. 14(4), July-August. pp. 792-808.
- [6] Chen, L., Ozsu, T. & Oria, V. (2003). Symbolic Representation and Retrieval of Moving Object Trajectories. University of Waterloo. 2003.
- [7] Chiu, B., Keogh, E. & Lonardi, S. (2003). Probabilistic Discovery of Time Series Motifs. *In proceedings of the 9th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining* Washington DC, USA, Aug 24-27. pp. 493-498.
- [8] Crochemore, M., Czumaj, A., Gasieniec, L., Jarominek, S., Lecroq, T., Plandowski, W. & Rytter, W. (1994). Speeding Up Two String-Matching Algorithms. *Algorithmica* vol. 12(4/5). pp. 247-267.
- [9] Dasgupta, D. & Forrest, S. (1999). Novelty Detection in Time Series Data Using Ideas from Immunology. *In proceedings of the 8th Int'l Conference on Intelligent Systems* Denver, CO, Jun 24-26.
- [10] Durbin, R., Eddy, S., Krogh, A. & Mitchison, G. (1998). Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge University Press.
- [11] Faloutsos, C., Ranganathan, M. & Manolopoulos, Y. (1994). Fast Subsequence Matching in Time-Series Database. *SIGMOD Record* vol. 23(2), June. pp. 419-429.
- [12] Gabaglio, A. (1888). *Theoria Generale Della Statistica*, 2nd ed. Milan.
- [13] Goldberger, A. L., Amaral, L. A., Gibb, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C. K. & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. *Circulation* vol. 101(23), June 13. pp. e215-e220 [Circulation Electronic Pages; <http://circ.ahajournals.org/gi/content/full/101/23/e215>]
- [14] Hochheiser, H. & Shneiderman, B. (2001). Interactive Exploration of Time-Series Data. *In proceedings of the 4th Int'l Conference on Discovery Science* Washington D.C., Nov 25-28. pp. 441-446.
- [15] Huang, Y. W. & Yu, P. S. (1999). Adaptive Query Processing for Time-Series Data. *In proceedings of the 5th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining* San Diego, CA, Aug 15-18. pp. 282-286.
- [16] Huettel, S., Mack, P. B. & McCarthy, G. (2002). Perceiving Patterns in Random Series: Dynamic Processing of Sequence in Prefrontal Cortex. *Nature Neuroscience* vol. 5. pp. 485-490.
- [17] Isaac, D. & Lynnes, C. (2003). Automated Data Quality Assessment in the Intelligent Archive, White Paper prepared for the Intelligent Data Understanding program. 2003. pp. 17.
- [18] Jin, X., Wang, L., Lu, Y. & Shi, C. (2002). Indexing and Mining of the Local Patterns in Sequence Database. *In proceedings of the 3rd Int'l Conference on Intelligent Data Engineering and Automated Learning* Manchester, UK, Aug 12-14. pp. 68-73.
- [19] Keim, D. A. (2002). Information Visualization and Visual Data Mining. *IEEE Transactions on Visualization and Computer Graphics* vol. 8(1). pp. 1-8.
- [20] Keogh, E. The UCR Time Series Data Mining Archive. <http://www.cs.ucr.edu/~eamonn/tsdms/index.html>
- [21] Keogh, E., Chakrabarti, K. & Pazzani, M. (2001). Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. *SIGMOD Record* vol. 30(2), June. pp. 151-162.
- [22] Keogh, E. & Kasetty, S. (2002). On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. *In proceedings of the 8th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining* Edmonton, Alberta, Canada, July 23-26. pp. 102-111.
- [23] Keogh, E. & Lin, J. (2004). Clustering of Time Series Subsequences is Meaningless: Implications for Previous and Future Research. *Knowledge and Information Systems Journal* To Appear
- [24] Keogh, E., Lonardi, S. & Chiu, B. (2002). Finding Surprising Patterns in a Time Series Database in Linear Time and Space. *In proceedings of the 8th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining* Edmonton, Alberta, Canada, Jul 23-26. pp. 550-556.
- [25] Kim, S., Kim, Y., Ahn, T., Nam, H. K., Han, B. J. & Kim, S. M. (2000). Visualysis: A Tool for Biological Sequence Analysis. *In proceedings of the 4th Int'l Conference on Computational Molecular Biology* Tokyo, Japan, Apr 8-11.
- [26] Lankford, J. P. & Quan, A. (2002). Evolution of Knowledge-Based Applications for Launch Support. *In proceedings of Ground System Architecture Workshop* El Segundo, CA.
- [27] Lin, J. VizTree Website <http://www.cs.ucr.edu/~jessica/viztree.htm>
- [28] Lin, J., Keogh, E., Lonardi, S. & Chiu, B. (2003). A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. *In Workshop on Research Issues in Data Mining and Knowledge Discovery the 8th ACM SIGMOD* San Diego, CA. June 13, 2003.
- [29] Lin, J., Keogh, E., Patel, P. & Lonardi, S. (2002). Finding Motifs in Time Series. *In the 2nd Workshop on Temporal Data Mining the 8th ACM Int'l Conference on Knowledge Discovery and Data Mining* Edmonton, Alberta, Canada. July 23-26, 2002.
- [30] Ma, J. & Perkins, S. (2003). Online Novelty Detection on Temporal Sequences. *In proceedings of the 9th Int'l Conference on Knowledge Discovery and Data Mining* Washington D.C., Aug 24-27.
- [31] Oates, T. (1999). Identifying Distinctive Subsequences in Multivariate Time Series by Clustering. *In proceedings of the 5th Int'l Conference on Knowledge Discovery and Data Mining* San Diego, CA, Aug 15-18. pp. 322-326.
- [32] Oates, T., Schmill, M. & Cohen, P. (2000). A Method for Clustering the Experiences of a Mobile Robot that Accords with Human Judgements. *In proceedings of the 17th National Conference on Artificial Intelligence* pp. 846-851.
- [33] Ohsaki, M., Sato, Y., Yokoi, H. & Yamaguchi, T. (2003). A Rule Discovery Support System for Sequential Medical Data, in the Case Study of a Chronic Hepatitis Dataset. *In Discovery Challenge Workshop the 14th European Conference on Machine Learning/the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases* Cavtat-Dubrovnik, Croatia. Sep 22-26, 2003.
- [34] Park, S., Chu, W., Yoon, J. & Hsu, C. (2000). Efficient Searches for Similar Subsequences of Different Lengths in Sequence Databases. *In proceedings of the 16th IEEE Int'l Conference on Data Engineering* San Diego, CA, Feb 28 - Mar 3. pp. 22-32.
- [35] Reinert, G., Schbath, S. & Waterman, M. S. (2000). Probabilistic and Statistical Properties of Words: An Overview. *Journal of Computational Biology* vol. 7. pp. 1-46.
- [36] Shahabi, C., Tian, X. & Zhao, W. (2000). TSA-Tree: A Wavelet-Based Approach to Improve the Efficiency of Multi-Level Surprise and Trend Queries. *In proceedings of the 12th Int'l Conference on Scientific and Statistical Database Management* Berlin, Germany, Jul 26-28. pp. 55-68.
- [37] Shneiderman, B. (1996). The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. *In proceedings of the IEEE Symposium on Visual Languages* Boulder, CO, Sep 3-6. pp. 336-343.
- [38] Tanaka, Y. & Uehara, K. (2003). Discover Motifs in Multi Dimensional Time-Series Using the Principal Component Analysis and the MDL Principle. *In proceedings of the 3rd Int'l Conference on Machine Learning and Data Mining in Pattern Recognition* Leipzig, Germany, Jul 5-7. pp. 252-265.
- [39] Tufte, E. R. (1983). *The Visual Display of Quantitative Information*. Cheshire, CT. Graphics Press.
- [40] van Wijk, J. J. & van Selow, E. R. (1999). Cluster and Calendar Based Visualization of Time Series Data. *In proceedings of 1999 IEEE Symposium on Information Visualization* San Francisco, CA, Oct 24-29. pp. 4-9.
- [41] Weber, M., Alexa, M. & Muller, W. (2001). Visualizing Time Series on Spirals. *In proceedings of 2001 IEEE Symposium on Information Visualization* San Diego, CA, Oct 21-26. pp. 7-14.

Towards Parameter-Free Data Mining

Eamonn Keogh

Stefano Lonardi

Chotirat Ann Ratanamahatana

Department of Computer Science and Engineering

University of California, Riverside

Riverside, CA 92521

{eamonn, stelo, ratana}@cs.ucr.edu

ABSTRACT

Most data mining algorithms require the setting of many input parameters. Two main dangers of working with parameter-laden algorithms are the following. First, incorrect settings may cause an algorithm to fail in finding the true patterns. Second, a perhaps more insidious problem is that the algorithm may report spurious patterns that do not really exist, or greatly overestimate the significance of the reported patterns. This is especially likely when the user fails to understand the role of parameters in the data mining process.

Data mining algorithms should have as few parameters as possible, ideally none. A parameter-free algorithm would limit our ability to impose our prejudices, expectations, and presumptions on the problem at hand, and would let the data itself *speak* to us. In this work, we show that recent results in bioinformatics and computational theory hold great promise for a parameter-free data-mining paradigm. The results are motivated by observations in Kolmogorov complexity theory. However, as a practical matter, they can be implemented using any off-the-shelf compression algorithm with the addition of just a dozen or so lines of code. We will show that this approach is competitive or superior to the state-of-the-art approaches in anomaly/interestingness detection, classification, and clustering with empirical tests on time series/DNA/text/video datasets.

Keywords

Kolmogorov Complexity, Parameter-Free Data Mining, Anomaly Detection, Clustering.

1. INTRODUCTION

Most data mining algorithms require the setting of many input parameters. There are many dangers of working with parameter-laden algorithms. We may fail to find true patterns because of poorly chosen parameter settings. A perhaps more insidious problem is that we may find patterns that do not exist [21], or greatly overestimate the significance of a pattern because of a failure to understand the role of parameter searching in the data mining process [5][7]. In addition, as we will show, it can be very difficult to compare the results across methods or even to reproduce the results of heavily parameterized algorithms.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD '04, August 22–25, 2004, Seattle, WA, U.S.A.

Copyright 2004 ACM 1-58113-000-0/00/0004...\$5.00.

Data mining algorithms should have as few parameters as possible, ideally none. A parameter-free algorithm prevents us from imposing our prejudices and presumptions on the problem at hand, and let the data itself *speak* to us.

In this work, we introduce a data mining paradigm based on compression. The work is motivated by results in bioinformatics and computational theory that are not well known outside those communities. As we will demonstrate here, our approach allows parameter-free or parameter-light solutions to many classic data mining tasks, including clustering, classification, and anomaly detection.

Our approach has the following advantages, which we will empirically demonstrate with extensive experiments:

- 1) It allows true *exploratory* data mining, rather than forcing us to impose our presumptions on the data.
- 2) The accuracy of our approach can be greatly superior to those of parameter-laden algorithms, even if we allow these algorithms to search exhaustively over their parameter spaces.
- 3) Our approach is based on compression as its cornerstone, and compression algorithms are typically space and time efficient. As a consequence, our method is generally much more efficient than other algorithms, in some cases by three or four orders of magnitude.
- 4) Many parameterized algorithms require the data to be in a special format. For concreteness, consider time series data mining [14][20]. Here, the Euclidean distance requires that the dimensionality of two instances being compared is exactly the same, and Dynamic Time Warping (DTW) is not defined if a single data point is missing [30]. In contrast, our approach works for time series of different lengths, sampling rates, dimensionalities, with missing values, etc.

In this work, we decided to take the unusual step of reproducing our entire actual *code*, rather than just the pseudocode. There are two reasons for doing this. First, free access to the actual code combined with our policy of making all data freely available allows independent confirmation of our results. Second, it reinforces our claim that our methods are very simple to implement.

The rest of the paper is organized as follows. In Section 2, we discuss the results in bioinformatics and computational theory that motivate this work. In Section 3, we consider the minor changes and extensions necessary to extend these results to the classic data mining tasks of anomaly/interestingness detection, classification, and clustering. Section 4 sees an exhaustive empirical comparison, in which we compare dozens of algorithms to our approach, on dozens of datasets from several domains, including time series, video, DNA, and text. Finally, in Section 5, we discuss many avenues for possible extensions.

2. BACKGROUND AND RELATED WORK

We begin this section by arguing that a contribution made by a parameter-laden algorithm can be difficult to evaluate. We review some background material on Kolmogorov complexity, which motivates the parameter-free Compression-based Dissimilarity Measure (CDM), the technique at the heart of this paper.

2.1 The Perils of Parameter-Laden Algorithms

A recent paper in a top-tier journal introduced a new machine-learning framework and noted that it “...abounds with parameters that *can* be tuned” (our emphasis). It is surely more accurate to state that the approach has parameters that *must* be tuned. When surveying the literature on this topic, we noted that while there are many techniques for automatically tuning parameters, many of these techniques themselves have parameters, possibly resulting in an infinite regression.

An additional problem of parameter-laden algorithms is that they make it difficult to reproduce published experimental results, and to understand the contribution of a proposed algorithm.

A recently published paper introduced a new time series distance measure. The algorithm requires the setting of two parameters, and the authors are to be commended for showing the results of the cross-product: sixteen by four possible parameter choices. Of the sixty-four settings, eleven are slightly better than DTW, and the authors conclude that their approach is superior to DTW. However, the authors did not test over different parameters for DTW, and DTW does allow a single parameter, the maximum temporal distortion (the “*warping window*” [30]). The authors kindly provided us with the exact data they used in the experiment, and we reproduced the experiment, this time allowing a search over DTW’s single parameter. We discovered that over a wide range of parameter choices, DTW produces a near perfect accuracy, outperforming all sixty-four choices of the proposed algorithm.

Although the above is only one anecdotal piece of evidence, it does help make the following point. It is very difficult to evaluate the contribution of papers that introduce a parameter-laden algorithm.

In the case above, the authors’ commendable decision to make their data public allows the community to discover that DTW is probably a better distance measure, but only at the expense of some effort on the readers’ behalf. In general, the potential asymmetry in parameter tuning effort effectively prevents us from evaluating the contribution of many papers. Here, the problem is compounded by the fact that the authors created the dataset in question. Creating a dataset may be regarded as a form of meta parameter tuning, since we don’t generally know if the very first dataset created was used in the paper, or many datasets were created and only the most satisfactory one was used. In any case, there are clearly problems in setting parameters (training) and reporting results (testing) on the *same* dataset [32]. In the field of neural networks, Flexer [11] noted that 93% of papers did just that. While no such statistics are published for data mining, an informal survey suggests a similar problem may exist here. In Section 4.2.2, we will empirically reinforce this point by showing that in the context of anomaly detection, parameter-laden algorithms can have their parameters tuned to achieve excellent performance on one dataset, but completely fail to generalize to a new but very similar dataset.

Before leaving this section, it would be remiss of us not to note that many papers by the authors of this manuscript also feature

algorithms that have (too) many parameters. Indeed, the frustration of using such algorithms is one inspiration for this work.

2.2 Kolmogorov Complexity

The proposed method is based on the concept of Kolmogorov complexity. Kolmogorov complexity is a measure of randomness of strings based on their information content. It was proposed by A.N. Kolmogorov in 1965 to quantify the randomness of strings and other objects in an objective and absolute manner.

The Kolmogorov complexity $K(x)$ of a string x is defined as the length of the shortest program capable of producing x on a universal computer — such as a Turing machine. Different programming languages will give rise to distinct values of $K(x)$, but one can prove that the differences are only up to a fixed additive constant. Intuitively, $K(x)$ is the minimal quantity of information required to generate x by an algorithm.

Hereafter, we will follow the notation of [23], which was the main inspiration of this work. The conditional Kolmogorov complexity $K(x|y)$ of x to y is defined as the length of the shortest program that computes x when y is given as an auxiliary input to the program. The function $K(xy)$ is the length of the shortest program that outputs y concatenated to x .

In [22], the authors consider the distance between two strings x and y , defined as

$$d_k(x, y) = \frac{K(x|y) + K(y|x)}{K(xy)} \quad (1)$$

which satisfies the triangle inequality, up to a small error term. A more mathematically precise distance was proposed in [23].

Kolmogorov complexity is without a doubt the ultimate lower bound among all measures of information content. Unfortunately, it cannot be computed in the general case [24]. As a consequence, one must approximate this distance.

It is easy to realize that universal compression algorithms give an upper bound to the Kolmogorov complexity. In fact, $K(x)$ is the best compression that one could possibly achieve for the text string x . Given a data compression algorithm, we define $C(x)$ as the size of the compressed size of x and $C(x|y)$ as the compression achieved by first training the compression on y , and then compressing x . For example, if the compressor is based on a textual substitution method, one could build the dictionary on y , and then use that dictionary to compress x .

We can approximate (1) by the following distance measure

$$d_c(x, y) = \frac{C(x|y) + C(y|x)}{C(xy)} \quad (2)$$

The better the compression algorithm, the better the approximation of d_c for d_k is.

In [23], Li *et al.* have shown that d_c is a similarity metric, and can be successfully applied to clustering DNA and text. However, the measure would require hacking the chosen compression algorithm in order to obtain $C(x|y)$ and $C(y|x)$. We therefore decided to simplify the distance even further. In the next section, we will show that a simpler measure can be just as effective.

The idea of using data compression to classify sequences is not new. In the early days of computational biology, lossless compression was used to classify DNA sequences. We refer to,

e.g., [1][10][12][26][27], and references therein for a sampler of the rich literature existing on this subject.

Recently, Benedetto *et al.* [2] have shown how to use a compression-based measure to classify fifty languages. The paper was featured in several scientific (and less-scientific) journals, including Nature, Science, and Wired. It has also generated some controversies (see, e.g., [16]).

Finally, the idea of using compression to classify sequences is tightly connected with the minimum description length (MDL) principle. The principle was introduced by the late '70 by Rissanen [31], and has generated a very extensive body of literature in the machine learning community (see, e.g., [29])

2.3 Compression-Based Dissimilarity Measure

Given two strings, x and y , we define the Compression-based Dissimilarity Measure (CDM) as follows

$$CDM(x, y) = \frac{C(xy)}{C(x) + C(y)} \quad (3)$$

The CDM dissimilarity is close to 1 when x and y are not related, and smaller than one if x and y are related. The smaller the $CDM(x, y)$, the more closely related x and y are. Note that $CDM(x, x)$ is not zero.

The dissimilarity measure can be easily implemented. The entire Matlab code is shown in Table 1.

Table 1: Compression-based Dissimilarity Measure (CDM)

```
function dist = CDM(A, B)
save A.txt A -ASCII % Save variable A as A.txt
zip('A.zip', 'A.txt'); % Compress A.txt
A_file = dir('A.zip'); % Get file information

save B.txt B -ASCII % Save variable B as B.txt
zip('B.zip', 'B.txt'); % Compress B.txt
B_file = dir('B.zip'); % Get file information

A_n_B = [A; B]; % Concatenate A and B
save A_n_B.txt A_n_B -ASCII % Save A_n_B.txt
zip('A_n_B.zip', 'A_n_B.txt'); % Compress A_n_B.txt
A_n_B_file = dir('A_n_B.zip'); % Get file information
% Return CDM dissimilarity
dist = A_n_B_file.bytes / (A_file.bytes + B_file.bytes);
```

The inputs are the two matrices to be compared. These matrices can be time series, DNA strings, images, natural language text, midi representations of music, etc. The algorithm begins by saving the two objects to disk, compressing them, and obtaining file information. The next step is to concatenate the two objects ($A_n_B = [A; B]$); the resulting matrix is also saved, compressed, and the file information is retrieved. At this point we simply return the size of the compressed concatenation over the size of the sum of the two individual compressed files.

One could argue that also CDM has several parameters. In fact, CDM depends on the choice of the specific compressor (gzip, compress, bzip2, etc.), and on the compression parameters (for example, the sliding window size in gzip). But because we are trying to get the best approximation of the Kolmogorov

complexity, one should just choose the *best* combination of compression tool and compression parameters for the data. There is, in fact, no freedom in the choice to be made. We simply run these compression algorithms on the data to be classified and choose the one that gives the highest compression.

2.4 Choosing the Representation of the Data

As we noted above, the only objective in CDM is to obtain good compression. There are several ways to achieve this goal. First, one should try several compressors. If we have domain knowledge about the data under study, and specific compressors are available for that type of data, we use one of those. For example, if we are clustering DNA we should consider a compression algorithm optimized for compressing DNA (see, e.g., [3]).

There is another way we can help improve the compression; we can simply ensure that the data to be compared is in a format that can be readily and meaningfully compressed. Consider the following example; Figure 1 shows the first ten data points of three Electrocardiograms from PhysioNet [15] represented in textual form.

A	B	C
0.13812500000000	0.51250000000000	0.49561523437690
0.04875000000000	0.50000000000000	0.49604248046834
0.10375000000000	0.50000000000000	0.49653076171875
0.17875000000000	0.47562500000000	0.49706481933594
0.24093750000000	0.45125000000000	0.49750732421875
0.29875000000000	0.45125000000000	0.49808715820312
0.37000000000000	0.47656250000000	0.49875854492187
0.48375000000000	0.50000000000000	0.49939941406230
0.55593750000000	0.48281250000000	0.50007080078125
0.64625000000000	0.48468750000000	0.50062011718750
0.70125000000000	0.46937500000000	0.50123046875826

Figure 1: The first ten data points of three ECG s

It happens to be the case that sequences **A** and **C** are both from patients with supraventricular escape beats. If we are allowed to see a few hundred additional data points from these sequences, we can correctly group the sequences (**(A,C),B**) by eye, or with simple Euclidean distance.

Unfortunately, CDM may have difficulties with these datasets. The problem is that although all sequences are stored with 16-digit precision, sequences **A** and **B** were actually recorded with 8-digit precision and automatically converted by the Rdsamp-O-Matic tool [15]. Note that, to CDM, **A** and **B** may have great similarity, because the many occurrences of 00000000's in both **A** and **B** will compress even better in each other's company. In this case, CDM is finding true similarity between these two sequences, but it is a trivial *formatting* similarity, and not a meaningful measure of the *structure* of the heartbeats. Similar remarks can be made for other formatting conventions and hardware limitations, for example, one sensor's number-rounding policy might produce a surfeit of numbers ending with "5".

Before explaining our simple solution this problem, we want to emphasize that CDM is extremely robust to it. For example, all the anomalies detected in Section 4.2 can be easily discovered on the original data. However, addressing this problem allows us to successfully apply CDM on much smaller datasets.

A simple solution to problem noted above is to convert the data into a discrete format, with a small alphabet size. In this case, every part of the representation contributes about the same amount of information about the shape of the time series. This opens the

question of which symbolic representation of time series to use. In this work, we use the SAX (Symbolic Aggregate Approximation) representation of Lin *et al.* [25]. This representation has been shown to produce competitive results for classifying and clustering time series, which suggest that it preserves meaningful information from the original data. Furthermore, the code is freely available from the authors’ website. While SAX does allow parameters, for all experiments here we use the parameterless version.

Similar remarks can be made for other data types, for example, when clustering WebPages, we may wish to strip out the HTML tags first. Imagine we are trying to cluster WebPages based on authorship, and it happens that some of the WebPages are graphic intensive. The irrelevant (for this task) similarity of having many occurrences of “” may dominate the overall similarity.

3. PARAMETER-FREE DATA MINING

Most data mining algorithms, including classification [5], clustering [13][17][21], anomaly/interestingness detection [4][28][33], reoccurring pattern (motif) discovery, similarly search [35], etc., use some form of similarity/dissimilarity measure as a subroutine. Because of space limitations, we will consider just the first three tasks in this work.

3.1 Clustering

As CDM is a dissimilarity measure, we can simply use it directly in most standard clustering algorithms. For some partitioned algorithms [6], it is necessary to define the concept of cluster “center”. While we believe that we can achieve this by extending the definition of CDM, or embedding it into a metric space [9], for simplicity here, we will confine our attention to hierarchical clustering.

3.2 Anomaly Detection

The task of finding anomalies in data has been an area of active research, which has long attracted the attention of researchers in biology, physics, astronomy, and statistics, in addition to the more recent work by the data mining community [4][28][33]. While the word “anomaly” implies that a radically different subsection of the data has been detected, we may actually be interested in more subtle deviations in the data, as reflected by some of the synonyms for anomaly detection, interestingness/deviation/surprise/novelty detection, etc.

For true parameter-free anomaly detection, we can use a divide-and-conquer algorithm as shown in Table 2. The algorithm works as follows: Both the left and right halves of the entire sequence being examined are compared to the entire sequence using the CDM dissimilarity measure. The intuition is that the side containing the most unusual section will be less similar to the global sequence than the other half. Having identified the most interesting side, we can recursively repeat the above, repeatedly dividing the most interesting section until we can no longer divide the sequence.

This twelve-line algorithm appears trivial, yet as we shall see in Section 4.2, it outperforms four state-of-the-art anomaly detection algorithms on a wide variety of real and synthetic problems. The algorithm has another important advantage; it can handle both single dimensional anomaly detection and multidimensional anomaly detection without changing a single line of code. We will demonstrate this ability in Section 4.2.3.

Table 2: Parameter-Free Anomaly Detection Algorithm

```
function loc_of_anomaly = kolmogorov_anomaly(data)
loc_of_anomaly = 1;
while size(data,1) > 2
left_dist = CDM(data(1:floor(end/2),:),data);
right_dist = CDM(data(ceil(end/2):end,:),data);
if left_dist < right_dist
loc_of_anomaly = loc_of_anomaly + size(data,1) / 2;
data = data(ceil(end/2):end,:);
else
data = data(1:floor(end/2),:);
end
end
```

While the algorithm above easily detects the anomalies in all the datasets described in Section 4.2, there are two simple ways to greatly improve it further. The first is to use the SAX representation when working with time series, as discussed in Section 2.4. The second is to introduce a simple and intuitive way to set parameter. The algorithm in Table 2 allows several potential weaknesses for the sake of simplicity. First, it assumes a single anomaly in the dataset. Second, in the first few iterations, the measure needs to note the difference a small anomaly makes, even when masked by a large amount of surrounding normal data. A simple solution to these problems is to set a parameter W , for number of windows. We can divide the input sequence into W contiguous sections, and assign the anomaly value of the i^{th} window as $CDM(W_i, data)$. In other words, we simply measure how well a small local section can match the global sequence. Setting this parameter is not too burdensome for many problems. For example of the ECG dataset discussed in Section 4.2.3, we found that we could find the objectively correct answer, if the size of the window ranged anywhere from a $\frac{1}{4}$ heartbeat length to four heartbeats. For clarity, we call this slight variation Window Comparison Anomaly Detection (WCAD).

3.3 Classification

Because CDM is a dissimilarity measure, we can trivially use it with a lazy-learning scheme. For simplicity, in this work, we will only consider the one-nearest-neighbor algorithm. Generally speaking, lazy learners using non-metric proximity measures are typically forced to examine the entire dataset. However, one can use an embedding technique such as FASTMAP [9] to map the objects into a metric space, thus allowing indexing and faster classification. For simplicity, we disregard this possibility in this work.

4. EMPIRICAL EVALUATION

While this section shows the results of many experiments, it is actually only a subset of the experiments conducted for this research project. We encourage the interested reader to consult [18] for additional examples.

4.1 Clustering

While CDM can work with most clustering techniques, here we confine our attention to hierarchical clustering, since it lends itself to immediate visual confirmation.

4.1.1 Clustering Time Series

In order to perform convincing experiments, we wanted to test our algorithm against all reasonable alternatives. However, lack of space prevents us from referencing, much less explaining them. So, we re-implemented *every* time series distance/dissimilarity/similarity measure that has appeared in the last decade in any of the following conferences: SIGKDD, SIGMOD, ICDM, ICDE, VLDB, ICML, SSDB, PKDD, and PAKDD. In total, we implemented fifty-one such measures, including the ten mentioned in [20] and the eight variations mentioned in [13]. For fairness, we should note that many of these measures are designed to deal with short time series, and made no claim about their ability to handle longer time series. In addition to the above, we considered the classic Euclidean distance, Dynamic Time Warping (DTW), the L_1 metric, the L_{inf} metric, and the Longest Common Subsequence (LCSS), all of which are more than a decade old. Some of these (Euclidean and the other L_p metrics) are parameter free. For measures that require a single parameter, we did an exhaustive search for the best parameter. For measures requiring more than one parameter (one method required seven!), we spent one hour of CPU time searching for the best parameters using a genetic algorithm and independently spent one hour searching manually for the best parameters. We then considered only the better of the two.

For our first experiment, we examined the UCR Time Series Archive [19] for datasets that come in pairs. For example, in the Foetal-ECG dataset, there are two time series, *thoracic* and *abdominal*, and in the Dryer dataset, there are two time series, *hot gas exhaust* and *fuel flow rate*. We were able to identify eighteen such pairs, from a diverse collection of time series covering the domains of finance, science, medicine, industry, etc. Although our method is able to deal with time series of different lengths, we truncated all time series to length 1,000 to allow comparisons to methods that require equal length time series.

While the correct hierarchical clustering at the top of the tree is somewhat subjective, at the lower level of the tree, we would hope to find a single bifurcation separating each pair in the dataset. Our metric, Q , for the quality of clustering is therefore the number of such correct bifurcations divided by eighteen, the number of datasets. For a perfect clustering, $Q = 1$, and because the number of dendrograms of thirty-six objects is greater than $3 \cdot 10^{49}$, for a random clustering, we would expect $Q = 0$.

For each measure, we clustered using single linkage, complete linkage, group average linkage, and wards methods, and reported only the best performing result. Figure 2 shows the resulting dendrogram for our approach.

Our approach achieved a perfect clustering, with $Q = 1$. Although the higher level clustering is subjective, here too our approach seems to do very well. For example, the appearance of the *Evaporator* and *Furnace* datasets in the same subtree is quite intuitive, and similar remarks can be made for the two *Video* datasets and the two *MotorCurrent* datasets.

More than $\frac{3}{4}$ of the other approaches we tested scored $Q = 0$. Several of the parameter-laden algorithms suffer from the following limitation. Although their parameters could be carefully tuned to do well on one type of data, say the relatively smooth *MotorCurrent* datasets, they achieve poor performance on the more noisy datasets like *Balloon*. We could then tune the parameters to do better on the noisy datasets, but immediately lose discriminatory power on the smooth data.

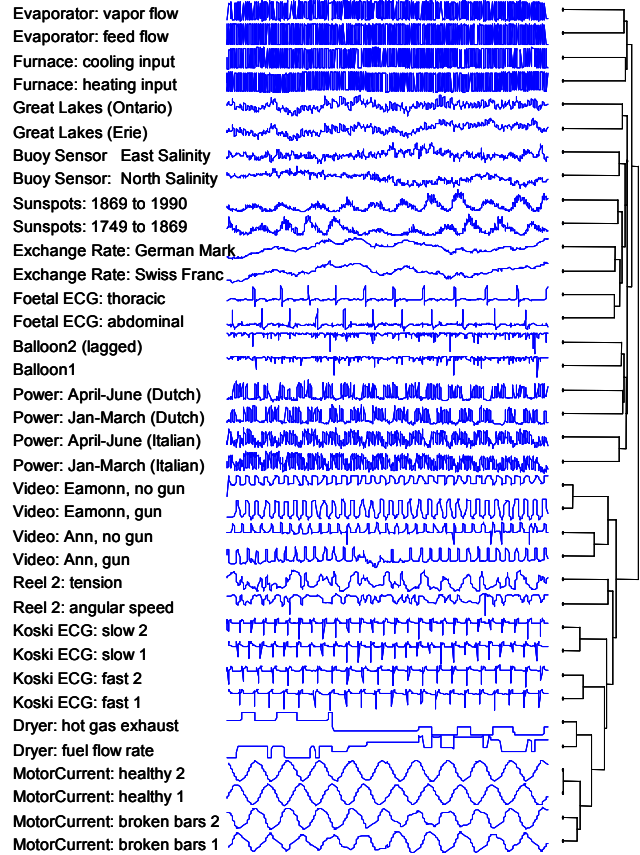


Figure 2: Thirty-six time series (in eighteen pairs) clustered using the approach proposed in this paper

The only measures performing significantly better than random were the following. **Euclidean distance** had $Q = 0.27$. **DTW** was able to achieve $Q = 0.33$ after careful adjustment of its single parameter. The **Hidden Markov Model** approach of [14] achieved $Q = 0$ using the original piecewise linear approximation of the time series. However, when using the SAX representation, its score jumped to $Q = 0.33$. The **LPC Cepstra** approach of [17] and the similar **Autocorrelation** method of [35] both had $Q = 0.16$. **LCSS** had $Q = 0.33$.

Our first experiment measured the quality of the clustering only at the leaf level of the dendrogram. We also designed a simple experiment to test the quality of clustering at a higher level. We randomly extracted ten subsequences of length 2,000 from two ECG databases. For this problem the clustering at the leaf level is subjective, however the first bifurcation of the tree should divide the data into the two classes (the probability of this happening by chance is only 1 in 524,288). Figure 3 shows the two best clusterings obtained.

In a sense, our exhaustive comparison to other similarity methods was unfair to many of them, which can only measure the similarity of a few local shapes, rather than the higher-level structural similarity required.

The following “trick” improved the results of most of the algorithms on both problems above. To compare two time series A and B of length n , we can extract a subsequence of length s from A , and compare it to every location in B , then record the closest match

as the overall distance between A and B . Although this does help the majority of the similarity measures, it has a significant downside. It adds a new (and highly sensitive) parameter to set and increases the time complexity by a factor of $O(n^2)$ and even after this optimization step, none of the competing similarity measures come close to the performance of our method.

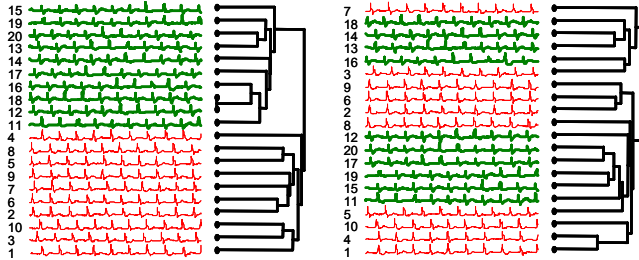


Figure 3: Two clusterings on samples from two records from the MIT-BIH Arrhythmia Database (*Left*) Our approach (*Right*) Euclidean distance

Finally, while the results of these experiments are very promising for our approach, some fraction of the success could be attributed to luck. To preempt this possibility, we conducted many additional experiments, with essentially identical results. These experiments are documented in [18].

4.1.2 Clustering Text

As a test of our ability to cluster text, we began by conducting experiments on DNA strings. We took the first 16,300 symbols from the mitochondrial DNA of twelve primates and one “outlier” species, and hierarchically clustered them. A similar strategy was used in [23] on a different set of organisms. To validate our results, we showed the resulting dendrogram to an expert in primate evolution, Dr. Sang-Hee Lee of UCR. Dr. Lee noted that some of the relevant taxonomy is still the subject of controversy, but informed us that the “*topography of the tree looks correct*”. Figure 4 shows the clustering obtained; Dr. Lee provided the annotation of the internal nodes.

We want to note that using a compressor optimized for DNA [3] was essential here. A standard dictionary-based compressor like gzip, would have resulted in less meaningful distances.

We conducted additional experiments with a more diverse collection of animals; in every case the clustering agreed with the current consensus on evolutionary history [18].

We also examined natural language text. A similar experiment is reported in [2]. Here, we began by clustering the text of various countries’ Yahoo portals. We only considered the first 1,615 characters, the size of the smallest webpage (excluding white spaces). Figure 5 (*left*) shows the resulting clustering. Note that the first bifurcation correctly divides the tree into *Germanic* and *Romance* languages. While we striped out all HTML tags for this experiment, we found that leaving them in made little difference, presumably because they were more or less equally frequent across languages.

Surprisingly, the clustering shown is much better than that achieved by the ubiquitous cosine similarity measure. In retrospect, this is hardly surprising.

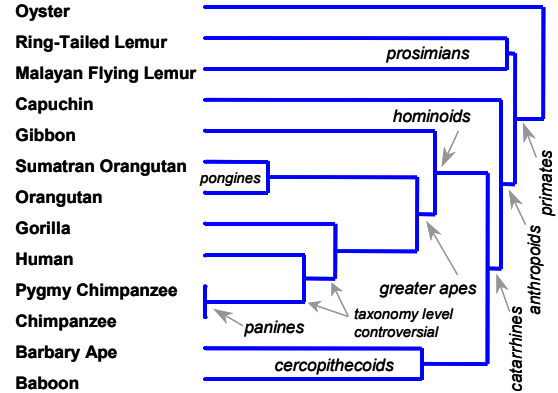


Figure 4: The clustering achieved by our approach on 16,300 symbols from the mitochondrial DNA of twelve primates, and one “outlier” species

Consider the following English, Norwegian and Danish words taken from the Yahoo portals:

English: {England, *information*, **addresses**}
 Norwegian: {Storbritannia, *informasjon*, **adressebok**}
 Danish: {Storbritannien, *informationer*, **adressekartotek**}

Because there is not a single word in common to all (even after applying Porters algorithm), the three vectors are completely orthogonal to each other in vector space. However, any human inspection of the text is likely to correctly conclude that Norwegian and Danish are much more similar to each other than they are to English. Our approach can leverage off the same cues by finding repeated structure within and across texts.

We tried a similar experiment with text from various translations of the first fifty chapters of the bible, this time including what one would expect to be an outlier, the Maori language of the indigenous people of New Zealand. As shown in Figure 5 (*right*) the clustering is subjectively correct, except for an inclusion of French in the *Germanic* subtree.

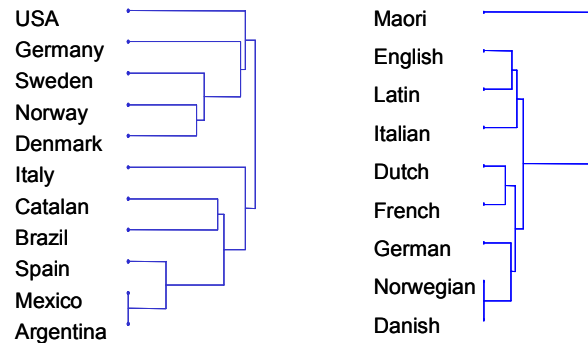


Figure 5: (*Left*) The clustering achieved by our approach on the text from various Yahoo portals (Jan-15th 2004). The smallest webpage had 1,615 characters, excluding white spaces. (*Right*) The clustering achieved by our approach on the text from the first fifty chapters of Genesis. The smallest file had 132,307 characters, excluding white spaces. Maori, a Malayo-Polynesian language, is clearly identified as an “outlier”

Once again, we reiterate the following disclaimer. We are not suggesting that our method replace the vector space model for indexing text, or a linguistic aware method for tracing the evolution of languages. Our point is simply to show that given a dataset in which we know nothing about, we can expect our CDM to produce reasonable results that can be a starting point for future study.

4.2 Anomaly Detection

Although our approach can be used to find anomalies in text, video, images, and other data sources, we will confine our attention here to time series, since this domain has attracted the most attention in the data mining community and readily lends itself to visual confirmation.

For all the problems shown below, we can objectively discover the anomaly using the simple algorithm in Table 2. However, that algorithm only tells us the *location* of the anomaly, without telling us anything about the relative *strength* of the anomaly. For this reason, we use the Window Comparison Anomaly Detection (WCAD) variation discussed in Section 2.2. This slight variation allows us to determine the relative strength of the anomaly, which we can visualize by mapping onto the line’s thickness. As noted in Section 3.2, WCAD does have one simple parameter to set, which is W , the approximate size of the window we expect to find anomalies in. In these experiments, we only count an experiment as a success for CDM if the first window size we choose finds the anomaly, and if window sizes four times as large, and one quarter as large, can also find the anomaly.

Because of space limitations, we will consider only four rival techniques. Here, we simply list them, and state the number of parameters each requires in parenthesis. We refer the interested reader to the original papers for more details. We compared our approach to the Support Vector Machine (SVM) based approach of [28] (6), the Immunology (IMM) inspired approach of [4] (5), The Association Rule (AR) based approach of [36] (5), and the TSA-tree Wavelet based approach of [33] (3). As before, for each experiment we spent one hour of CPU time, and one hour of human time trying to find the best parameters and only reported the best results.

4.2.1 A Simple Normalizing Experiment

We begin our experiments with a simple sanity check, repeating the *noisy sine* problem of [28]. Figure 6 shows the results.

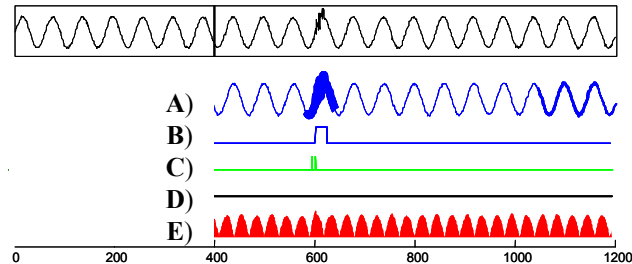


Figure 6: A comparison of five novelty detection algorithms on the *synthetic sine* problem of Ma and Perkins [28]. The first 400 data points are used as training data, an “event” is embedded at time point 600. A) The approach proposed in this work, the thickness of the line encodes the level of novelty. B) SVM. C) IMM. D) AR. E) TSA.

Our approach easily finds the novelty, as did SVM with careful parameter tuning. The IMM algorithm is stochastic, but was able to find the novelty in the majority of runs. We were simply unable to make the AR approach work. Finally, TSA does peak for the novelty, although its discriminatory power appears weak.

The ability of our approach to simply match the prowess of SVM and IMM on this problem may not seem like much of an achievement, even though we did it orders of magnitude faster and

without setting any parameters. However, the real utility of our approach becomes evident when we see how the algorithms generalize, or when we move from toy problems to real world problems. We consider both cases below.

4.2.2 Generalizability Experiment

To illustrate the dangers of working with parameter-laden algorithms, we examined a generalization of the last experiment. As illustrated in Figure 7, the training data remains the same. However, in the test data, we changed the period of the sine wave by a barely perceptible 5%, and added a much more obvious “anomaly”, by replacing a half of a sine wave with its absolute value. To be fair, we modified our algorithm to only use the training data as reference.

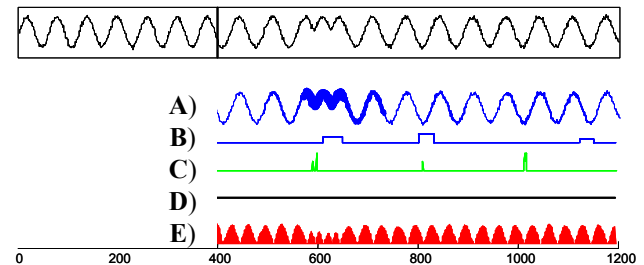


Figure 7: A comparison of five novelty detection algorithms on a generalization of the *synthetic sine* problem. The first 400 data points are used as training data. In the rest of the time series, the period of the sine wave was changed by 5%, and one half of a sine wave was replaced by its absolute value. A) The approach proposed in this work, the thickness of the line encodes the level of novelty. B) SVM. C) IMM. D) AR. E) TSA.

The results show that while our algorithm easily finds the new anomaly, SVM and IMM discover more important “anomalies” elsewhere. It may be argued that the very slight change of period *is the anomaly* and these algorithms did the right thing. However, we get a similar inability to generalize if we instead slightly change the amplitude of the sine waves, or if we add (or remove!) more uniform noise or make any other innocuous changes, including ones that are imperceptible to the human eye.

In case the preceding example was a coincidentally unfortunate dataset for the other approaches, we conducted many other similar experiments. And since creating our own dataset opens the possibility of data bias [20], we considered datasets created by others. We were fortunate enough to obtain a set of 20 time series anomaly detection benchmark problems from the Aerospace Corp. A subset of the data is shown in Figure 8.

The TSA algorithm easily discovered the anomaly in the time series L-1j, but not the other two time series. We found that both SVM and IMM could have their parameters tuned to find the anomaly on any individual one of the three sequences, but once the parameters were tuned on one dataset, they did not generalize to the other two problems.

The objective of these experiments is to reinforce the main point of this work. Given the large number of parameters to fit, it is nearly impossible to avoid overfitting.

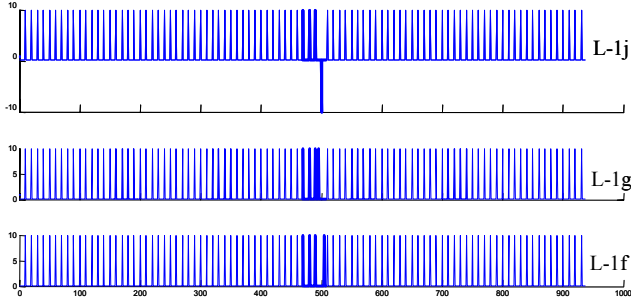


Figure 8: The results of applying our algorithm to (a subset of) a collection of anomaly detection benchmark datasets from the Aerospace Corp. the thickness of the line encodes the level of novelty. In every case, an anomaly was inserted beginning at time point 500

Before leaving this section we would like to briefly relate an anecdote as a further support for our approach. For the above problem, we wrote a simple Matlab script to read in the twenty datasets, run our anomaly detection algorithm, and confirm that the most anomalous section was discovered within twenty-five points of 500. After successfully testing our approach, we modified the script to consider the other approaches but found that it always crashed when working with dataset L-1s. After some careful debugging, we discovered that the artificial anomaly in this sequence is some missing data points, which are encoded in Matlab as the special character “NaN”. While none of the other algorithms are defined for missing values (hence the crashing), and are not trivially extendible, our approach was robust enough not to crash, and to find the right answer.

4.2.3 Real-World Anomaly Detection

We examined annotated datasets from the MIT-BIH Noise Stress Test Database. For the algorithms which need a training/test split, we gave them 1/3 of the dataset which had been annotated as normal. We then asked the algorithms to examine the rest of the data to find the most interesting events, comparing the results to the cardiologists’ annotations. Figure 9 shows the result of one such experiment. Note that only a small excerpt from the full dataset is shown.

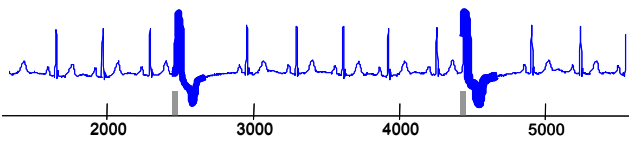


Figure 9: A small excerpt from dataset 118e06 from the MIT-BIH Noise Stress Test Database. The full dataset is 21,600 data points long. Here, we show only a subsection containing the two most interesting events detected by our algorithm (the bolder the line, the more interesting the subsequence). The gray markers are independent annotations by a cardiologist indicating Premature Ventricular Contractions

We only illustrate the performance of our approach in Figure 9 because all the other approaches produced results that were objectively (per the cardiologists’ annotations) and subjectively incorrect, in spite of careful parameter tuning.

Our final example illustrates the flexibility of our approach. None of the approaches for anomaly detection in time series in the

literature are defined for multidimensional time series¹, in spite of an increasing general interest in multidimensional time series [34]. However, we can consider multidimensional time series without changing a single line of code. In order to have some straw man to compare to, each of the four competing methods was adapted as follows. We collected the results on each individual dimension and then we linearly combined them into a single measure of novelty.

We experimented on a 2D time series that was collected for a different purpose (in particular, a classification problem [30]). The 2D time series was extracted from a video of an actor performing various actions with and without a replica gun. Figure 10 (bottom) illustrates a typical sequence. The actor draws a replica gun from a hip mounted holster, aims it at a target, and returns it to the holster.

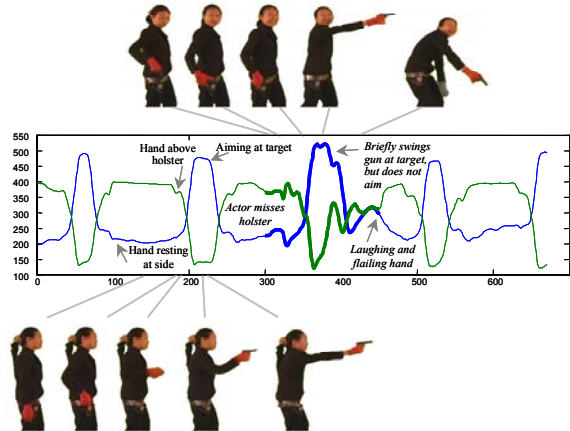


Figure 10: (Bottom) A typical video snippet from the Gun video is mapped onto a two-dimensional time series (Center) by tracking the actor’s right hand. While the vast majority of the dataset looks approximately like the first 200 data points, the section from about 300 to 450 looks somewhat different, and was singled out by our anomaly detection algorithm. Examining the original video (Top), we discovered the cause of the anomaly.

Watching the video we discovered that at about ten seconds into the shoot, the actor misses the holster when returning the gun. An off-camera (inaudible) remark is made, the actor looks toward the video technician, and convulses with laughter. At one point (frame 385), she is literally bent double with laughter. This is the only interesting event in the dataset, and our approach easily finds it. The other techniques returned results that do not seem truly anomalous, given a careful inspection of both the time series and the original video.

We have not considered time efficiency as a metric in these experiments, because we cannot guarantee that our implementations of the rival approaches are as efficient as they might be, given careful optimization. However, our approach is certainly not sluggish, requiring less than ten seconds (on a 2.65 GHz machine) to process a million data points.

4.3 Classification

In this section, we illustrate the utility of CDM for classification with the following simple experiment. We use the following

¹ This includes the 4 rival approaches considered here [4][28][33][36]. While the TSA-Wavelet approach was extended to 2D, this extension is for *spatial* mining.

similarity measures on four datasets (Two each from two databases:- ECG and Gun) and measure their error rates:

- Euclidean Distance [20].
- Dynamic Time Warping (DTW). Here, we exhaustively test all values of its single parameter (warping window size [30]) and report only the best result, and
- Compression-Based Dissimilarity Measure (CDM)

Note that we only compare CDM with Dynamic Time Warping and Euclidean Distance metric in this section for brevity, since it has been shown in [20] that many of the more complex similarity measures proposed in other work have higher error rates than a simple Euclidean Distance metric.

The ECG datasets are four-class problem derived from BIDMC Congestive Heart Failure Database [15] of four patients. Since this original database contains two ECG signals, we separate each signal and create two datasets of one-dimensional time series in the following way. Each instance of 3,200 contiguous data points (about 20 heartbeats) of each signal is randomly extracted from each long ECG signals of each patient. Twenty instances are extracted from each class (patient), resulting in eighty total instances for each dataset.

The Gun datasets are time-series datasets extracted from video sequences of two actors either aiming a gun or simply pointing at a target [30] (see also, Figure 10). We randomly extract twenty instances of 1,000 contiguous data points (about 7 reps) from each of the following long time series:

- Actor 1 with gun
- Actor 1 without gun (point)
- Actor 2 with gun
- Actor 2 without gun (point)

The first dataset is a two-class problem of differentiating Actor 1 from Actor 2 -- (A+B) vs. (C+D). The second dataset is a four-class problem of differentiating each of the acts independently – A vs. B vs. C vs. D. In total, each dataset contains eighty instances. Some samples from both databases are illustrated in Figure 11.

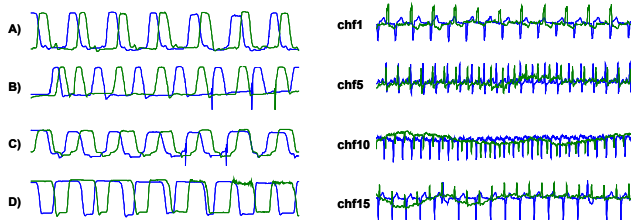


Figure 11. Some extracted time series from the gun datasets (left) and the ECG (sig.1) dataset (right)

We measure the error rates on each dataset, using the one-nearest-neighbor with ‘leaving-one-out’ evaluation method. The lower bounding technique noted in [30] is also integrated in all the DTW calculations to help achieve speedup. The experimental results are summarized in Table 3.

In all four datasets discussed above, Euclidean distance is extremely fast, yet inaccurate. DTW with the best uniform window size greatly reduces the error rates, but took several orders of magnitude longer. However, CDM outperforms both Euclidean and DTW in all datasets. Even though CDM is slower than Euclidean distance, it is much faster than the highly optimized DTW.

Table 3. Classification Error Rates (%) for all four datasets

	Euclidean	DTW (best unif. window)	CDM
ECG: signal 1	42.25 %	16.25 %	6.25 %
ECG: signal 2	47.50 %	11.25 %	7.50 %
Gun: 2 classes	5.00 %	0.00 %	0.00 %
Gun: 4 classes	37.50 %	12.5 %	5.00 %

We do not give exact times here since CDM is implemented in the relatively lethargic Matlab, whereas DTW is implemented in highly optimized C++. Nevertheless, even if we excluded the time taken to find search over DTW’s single (and sensitive, see [30]) parameter, CDM is still about 25 times faster than DTW.

5. CONCLUSIONS AND FUTURE WORK

In this work, we argued that data mining algorithms with many parameters are burdensome to use, and make it difficult to compare results across different methods. We further showed empirically that at least in the case of anomaly detection, parameter-laden algorithms are particularly vulnerable to overfitting. Sometimes they achieve perfect accuracy on one dataset, and then completely fail to generalize to other very similar datasets [7].

As a step towards mitigating these problems, we showed that parameter-free or parameter-light algorithms can compete with or outperform parameter-laden algorithms on a wide variety of problems/data types.

There are many directions in which this work may be extended. We intend to perform a more rigorous theoretical analysis of the CDM measure. For example, CDM is a dissimilarity measure; if it could be modified to be a *distance* measure, or better still, a *distance metric*, we could avail of a wealth of pruning and indexing techniques to speed up classification [30], clustering [6], and similarity search [34]. While it is unlikely that CDM can be transformed in a true metric, it may be possible to prove a weaker version of the triangular inequality, which can be bounded and used to prune the search space [6]. The results in [8] on textual substitution compressors could lead to some insights in the general problem.

Finally, we note that our approach is clearly not suitable for classifying or clustering low dimensionality data (although Figure 2 shows exceptionally good results on time series with only 1,000 data points). We plan to theoretically and empirically investigate the limitations on object sizes that we can meaningfully work with using our proposed approach.

6. ACKNOWLEDGMENTS

Thanks to Ming Li for his feedback on Section 4.1, and to the many donors of datasets. Thanks also to Stephen Bay for his many useful comments, Jessica Lin for the SAX code and the anonymous reviewers for their useful comments.

All datasets used in this paper are available for free download from [18]. For convenience, we also include the Yahoo dataset; however, the copyright remains the property of Yahoo! Inc.

7. REFERENCES

- [1] Allison, L., Stern, L., Edgoose, T., Dix, T.I. Sequence Complexity for Biological Sequence Analysis. *Computers & Chemistry* 24(1): 43-55 (2000)
- [2] Benedetto, D., Caglioti, E., & Loreto, V. Language trees and zipping. *Physical Review Letters* 88, 048702, (2002).
- [3] Chen, X., Kwong, S., & Li, M. A compression algorithm for DNA sequences and its applications in genome comparison. In *Proceedings of RECOMB 2000*: 107
- [4] Dasgupta, D. & Forrest, S. Novelty Detection in Time Series Data using Ideas from Immunology." In *Proc. of the International Conference on Intelligent Systems* (1999).
- [5] Domingos, P. A process-oriented heuristic for model selection. In *Machine Learning Proceedings of the Fifteenth International Conference*, pages 127-135. San Francisco, CA, 1998.
- [6] Elkan, C. Using the triangle inequality to accelerate k-Means. In *Proc. of ICML 2003*. pp 147-153
- [7] Elkan, C. Magical thinking in data mining: lessons from CoIL challenge 2000. *SIGKDD*, 2001. pp 426-431.
- [8] Ergün, F., Muthukrishnan, S., & Sahinalp, S.C. Comparing Sequences with Segment Rearrangements. *FSTTCS 2003*:
- [9] Faloutsos, C., & Lin, K. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proc of 24th ACM SIGMOD*, 1995.
- [10] Farach, M., Noordewier, M., Savari, S., Shepp, L., Wyner, A., & Ziv, J. On the Entropy of DNA: Algorithms and Measurements Based on Memory and Rapid Convergence, *Proc. of the Symp. on Discrete Algorithms*, 1995. pp 48-57.
- [11] Flexer, A. Statistical evaluation of neural networks experiments: Minimum requirements and current practice. In *Proc. of the 13th European Meeting on Cybernetics and Systems Research*, vol. 2, pp 1005-1008, Austria, 1996
- [12] Gatlin, L. *Information Theory and the Living Systems*. Columbia University Press, 1972.
- [13] Gavrilov, M., Anguelov, D., Indyk, P., Motwani, R. Mining the stock market: which measure is best? *Proc. of the 6th ACM SIGKDD*, 2000
- [14] Ge, X. & Smyth, P. Deformable Markov model templates for time-series pattern matching. In *proceedings of the 6th ACM SIGKDD*. Boston, MA, Aug 20-23, 2000. pp 81-90.
- [15] Goldberger, A.L., Amaral, L., Glass, L, Hausdorff, J.M., Ivanov, P.Ch., Mark, R.G., Mietus, J.E., Moody, G.B., Peng, C.K., Stanley, H.E.. *PhysioBank, PhysioToolkit, and PhysioNet: Circulation* 101(23):e215-e220
- [16] Goodman, J. Comment on "Language Trees and Zipping", unpublished manuscript, 2002 (available at [<http://research.microsoft.com/~joshuago/>]).
- [17] Kalpakis, K., Gada, D., & Puttagunta, V. Distance measures for effective clustering of ARIMA time-series. In *proc. of the IEEE ICDM*, 2001. San Jose, CA. pp 273-280.
- [18] Keogh, E. <http://www.cs.ucr.edu/~eamonn/SIGKDD2004>.
- [19] Keogh, E. & Folias, T. *The UCR Time Series Data Mining Archive*. Riverside CA. 2002. [<http://www.cs.ucr.edu/~eamonn/TSDMA/index.html>].
- [20] Keogh, E. & Kasetty, S. On the need for time series data mining benchmarks: A survey and empirical demonstration. In *Proc. of SIGKDD*, 2002.
- [21] Keogh, E., Lin, J., & Truppel, W. Clustering of Time Series Subsequences is Meaningless: Implications for Past and Future Research. In *proc. of the 3rd IEEE ICDM*, 2003. Melbourne, FL. Nov 19-22, 2003. pp 115-122.
- [22] Li, M., Badger, J.H., Chen, X., Kwong, S, Kearney, P., & Zhang, H. An information-based sequence distance and its application to whole mitochondrial genome phylogeny. *Bioinformatics* 17: 149-154, 2001.
- [23] Li, M., Chen, X., Li, X., Ma, B., Vitanyi, P. The similarity metric. *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, 2003. Pages: 863 – 872
- [24] Li, M. & Vitanyi, P. *An Introduction to Kolmogorov Complexity and Its Applications*. Second Edition, Springer Verlag, 1997.
- [25] Lin, J., Keogh, E., Lonardi, S. & Chiu, B. A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. In *proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. San Diego, CA. June 13, 2003
- [26] Loewenstern, D., Hirsh, H., Yianilos, P., & Noordewier, M. DNA Sequence Classification using Compression-Based Induction, *DIMACS Technical Report 95-04*, April 1995.
- [27] Loewenstern, D., & Yianilos, P.N. Significantly lower entropy estimates for natural DNA sequences, *Journal of Computational Biology*, 6(1), 1999.
- [28] Ma, J. & Perkins, S. Online Novelty Detection on Temporal Sequences. *Proc. International Conference on Knowledge Discovery and Data Mining*, August 24-27, 2003.
- [29] Quinlan, J.R. & Rivest, R.L. Inferring Decision Trees Using the Minimum Description Length Principle. *Information and Computation*, 80:227--248, 1989.
- [30] Ratanamahatana, C.A. & Keogh, E. Making Time-series Classification More Accurate Using Learned Constraints. In *proceedings of SIAM International Conference on Data Mining (SDM '04)*, Lake Buena Vista, Florida, April 22-24, 2004.
- [31] Rissanen, J. Modeling by shortest data description. *Automatica*, vol. 14 (1978), pp. 465-471.
- [32] Salzberg, S.L. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3), 1997.
- [33] Shahabi, C., Tian, X., & Zhao, W. TSA-tree: A Wavelet-Based Approach to Improve the Efficiency of Multi-Level Surprise and Trend Queries The 12th Int'l Conf on Scientific and Statistical Database Management (SSDBM 2000)
- [34] Vlachos, M., Hadjieleftheriou, M., Gunopulos, D. & Keogh, E. Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures. In the 9th ACM SIGKDD. August 24 - 27, 2003. Washington, DC, USA. pp 216-225.
- [35] Wang, C. & Wang, X. S. Supporting content-based searches on time series via approximation. In *proceedings of the 12th Int'l Conference on Scientific and Statistical Database Management*. Berlin, Germany, Jul 26-28, 2000. pp 69-81.
- [36] Yairi, T., Kato, Y., & Hori, K. Fault Detection by Mining Association Rules from House-keeping Data, *Proc. of Int'l Sym. on AI, Robotics and Automation in Space*, 2001.

We report here on additional results and information that were left out from the paper due to lack of space.

This work was inspired by an article in the June 2003 issue of Scientific American, by Charles H. Bennett, Ming Li, and Bin Ma. The article, “*Chain Letters and Evolutionary Histories*”, is a beautiful example of popular science writing. The authors have made the data used in this work available here: www.math.uwaterloo.ca/~mli/chain.html

Dr. Ming Li, and Dr. Paul Vitanyi have (together and separately) published many papers that explore compression for clustering, bioinformatics, plagiarism detection etc. Dr. Li’s webpage is www.math.uwaterloo.ca/~mli/ and Dr. Vitanyi’s webpage is <http://homepages.cwi.nl/~paulv/>. There has been enormous interest in this work, as you can gauge from <http://homepages.cwi.nl/~paulv/pop.html>

In addition, Li and Vitanyi have published the definitive book on Kolmogorov Complexity: “*An Introduction to Kolmogorov Complexity and Its Applications*”, Second Edition, Springer Verlag 1997; ISBN 0-387-94868-6.

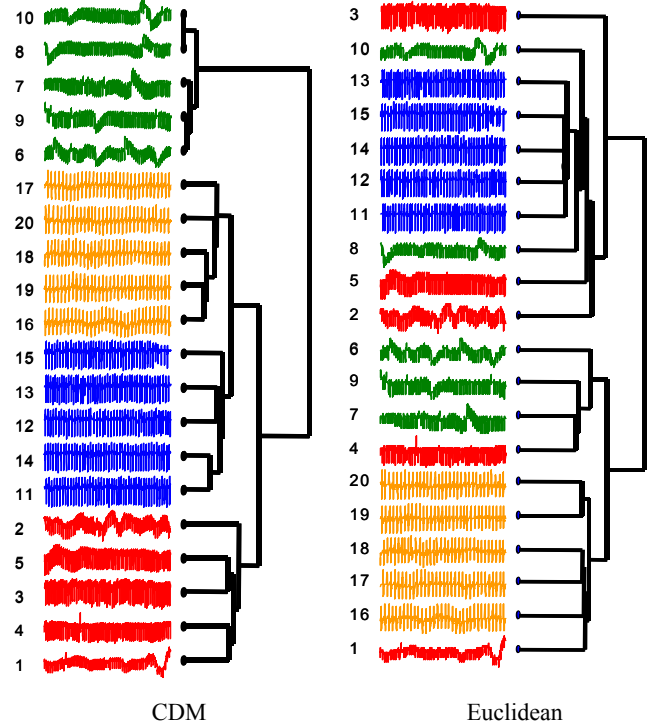
Additional papers that are (to varying degrees) related to this work, but not cited in the full paper due to lack of space (or because they came to our attention too late) include:

- A. Eibe Frank, Chang Chui and Ian H. Witten (2000). Text Categorization Using Compression Models. Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, IEEE Computer Society, pp. 555.
- B. Matthew B. Kennel (2004). Testing time symmetry in time series using data compression dictionaries. Phys. Rev. E 69, 056208 (9 pages).
- C. Matt Mahoney (2003). Space Shuttle Engine Valve Anomaly Detection by Data Compression. Unpublished notes. (Thanks to Stan Salvador for bringing this to our attention).
- D. J. Segen (1990). Graph Clustering and Model Learning by Data Compression. In Proceedings of the Machine Learning Conference, pages 93-101.
- E. Chunyu Kit. 1998. A goodness measure for phrase learning via compression with the MDL principle. In I. Kruijff-Korbayova(ed.), The ELLSSI-98 Student Session, Chapter 13, pp.175-187. Aug. 17-28, Saarbrueken.
- F. P. Grünwald, A Tutorial Introduction to the Minimum Description Length Principle. To appear as Chapters 1 & 2 of *Advances in Minimum Description Length: Theory and Applications*. MIT Press, 04.
- G. A. Ortega, B. Beferull-Lozano, N. Srinivasamurthy, and H. Xie. (2000). Compression for Recognition and Content based Retrieval. In Proc. of the European Signal Processing Conference, EUSIPCO'00, Tampere, Finland.
- H. Andrea Baronchelli, Vittorio Loreto (2004). Data Compression approach to Information Extraction and Classification CoRR cond-mat/0403233: (submitted for publication).
- I. C. Noble and D. J. Cook, Graph-Based Anomaly Detection, Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003.
- J. Teahan, W.J., Wen, Y., McNab, R.J., Witten, I.H.(2000). A compression-based algorithm for Chinese word segmentation. Computational Linguistics 26. 375--393

Both Dr. Stephen Bay and one of the anonymous reviewers noted that one implication of the experiments in Section 4.1.1 is that the Euclidean distance works very well for time series! This is true, we did not emphasize this fact in this work, because we already forcefully made this point in [20] (see Section 4.2).

Finally, we will show some additional experiments that did not make it to the published paper due to lack of space.

The experiment in Figure A is similar to the one shown in Figure 3, but with more classes.



Cluster 1 (datasets 1 ~ 5):
 BIDMC Congestive Heart Failure Database (chfdb): record chf02
 Start times at 0, 82, 150, 200, 250, respectively

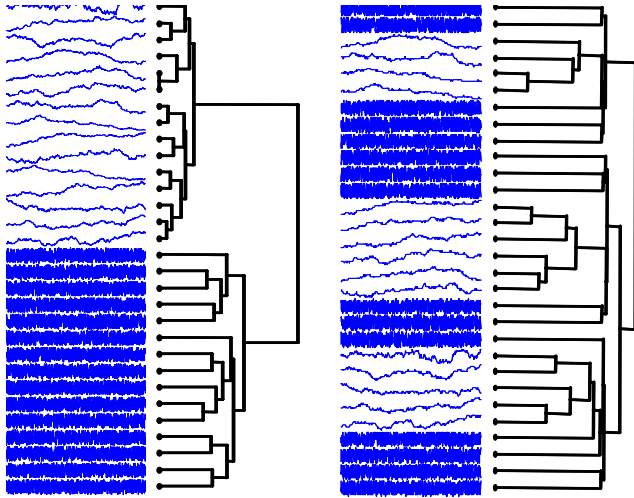
Cluster 2 (datasets 6 ~ 10):
 BIDMC Congestive Heart Failure Database (chfdb): record chf15
 Start times at 0, 82, 150, 200, 250, respectively

Cluster 3 (datasets 11 ~ 15):
 Long Term ST Database (ltstadb): record 20021
 Start times at 0, 50, 100, 150, 200, respectively

Cluster 4 (datasets 16 ~ 20):
 MIT-BIH Noise Stress Test Database (nstdb): record 118e6
 Start times at 0, 50, 100, 150, 200, respectively

Figure A: Two clusterings on samples from four records from the MIT-BIH Arrhythmia Database, (Left) Our approach (Right) Euclidean distance

Several people that viewed an early version of the work suggested that the clustering might only work in highly structured data, but not for more “random” data. As a simple sanity check we tried clustering random data and random walk data, as shown in Figure B.



CDM

Euclidean

Figure B: Two clusterings on 15 samples of random walk, and 15 samples of random data

In Figure C, we add some structured data to the mix, to see if CDM is confused by the presence of random data.

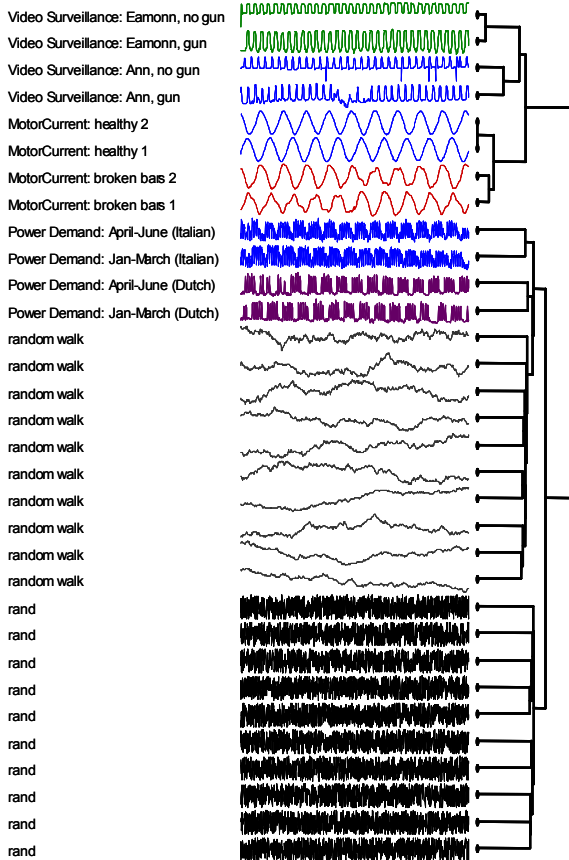
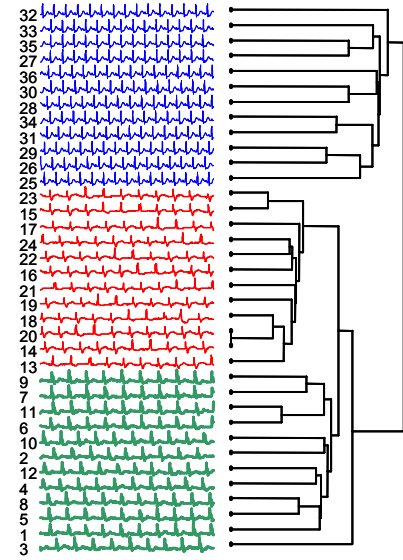


Figure C: The clustering obtained on some random walk data, random data, and some highly structured datasets.

It goes without saying that CDM is by no means perfect, in Figure D, Time Series 3 is incorrectly clustered.



MIT-BIH Arrhythmia Database www.physionet.org/physiobank/database/qtdb

Class 1: Record sel102, Class 2: Record sel104, Class 3: Record sel213

Figure D: The clustering obtained on a 3-class problem. Note that time series 3 (at the bottom of the figure) is not clustered properly

In Figure E we show additional examples from the dataset shown in Figure 8. Although the problems look too simple to be of interest, none of the other four approaches discussed in the paper can find the anomaly in all four examples.

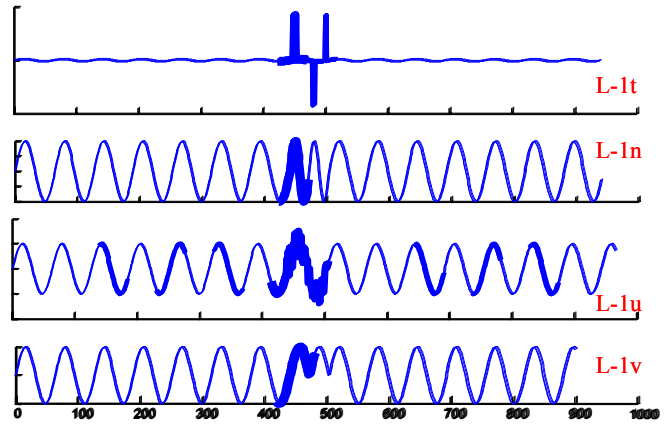


Figure E: Additional examples from the Aerospace anomaly detection problems, the thickness of the line encodes the level of novelty

the X -means algorithm of Moore and Pelleg (2000). However, instead of the MDL principle they use a probabilistic scheme called the Bayes Information Criterion (Kass and Wasserman 1995). The incremental clustering procedure, based on the merging and splitting operations, was introduced in systems called Cobweb for nominal attributes (Fisher 1987) and Classit for numeric attributes (Gennari et al. 1990). Both are based on a measure of category utility that had been defined previously (Gluck and Corter 1985). The AutoClass program is described by Cheeseman and Stutz (1995). Two implementations are available: the original research implementation, written in LISP, and a follow-up public implementation in C that is 10 or 20 times faster but somewhat more restricted—for example, only the normal-distribution model is implemented for numeric attributes.

6.7 Bayesian networks

The Naïve Bayes classifier of Section 4.2 and the logistic regression models of Section 4.6 both produce probability estimates rather than predictions. For each class value, they estimate the probability that a given instance belongs to that class. Most other types of classifiers can be coerced into yielding this kind of information if necessary. For example, probabilities can be obtained from a decision tree by computing the relative frequency of each class in a leaf and from a decision list by examining the instances that a particular rule covers.

Probability estimates are often more useful than plain predictions. They allow predictions to be ranked, and their expected cost to be minimized (see Section 5.7). In fact, there is a strong argument for treating classification learning as the task of learning class probability estimates from data. What is being estimated is the conditional probability distribution of the values of the class attribute given the values of the other attributes. The classification model represents this conditional distribution in a concise and easily comprehensible form.

Viewed in this way, Naïve Bayes classifiers, logistic regression models, decision trees, and so on, are just alternative ways of representing a conditional probability distribution. Of course, they differ in representational power. Naïve Bayes classifiers and logistic regression models can only represent simple distributions, whereas decision trees can represent—or at least approximate—arbitrary distributions. However, decision trees have their drawbacks: they fragment the training set into smaller and smaller pieces, which inevitably yield less reliable probability estimates, and they suffer from the replicated subtree problem described in Section 3.2. Rule sets go some way toward addressing these shortcomings, but the design of a good rule learner is guided by heuristics with scant theoretical justification.

Does this mean that we have to accept our fate and live with these shortcomings? No! There is a statistically based alternative: a theoretically well-founded way of representing probability distributions concisely and comprehensibly in a graphical manner. The structures are called *Bayesian networks*. They are drawn as a network of nodes, one for each attribute, connected by directed edges in such a way that there are no cycles—a *directed acyclic graph*.

In our explanation of how to interpret Bayesian networks and how to learn them from data, we will make some simplifying assumptions. We assume that all attributes are nominal and that there are no missing values. Some advanced learning algorithms can create new attributes in addition to the ones present in the data—so-called hidden attributes whose values cannot be observed. These can support better models if they represent salient features of the underlying problem, and Bayesian networks provide a good way of using them at prediction time. However, they make both learning and prediction far more complex and time consuming, so we will not consider them here.

Making predictions

Figure 6.20 shows a simple Bayesian network for the weather data. It has a node for each of the four attributes *outlook*, *temperature*, *humidity*, and *windy* and one for the class attribute *play*. An edge leads from the *play* node to each of the other nodes. But in Bayesian networks the structure of the graph is only half the story. Figure 6.20 shows a table inside each node. The information in the tables defines a probability distribution that is used to predict the class probabilities for any given instance.

Before looking at how to compute this probability distribution, consider the information in the tables. The lower four tables (for *outlook*, *temperature*, *humidity*, and *windy*) have two parts separated by a vertical line. On the left are the values of *play*, and on the right are the corresponding probabilities for each value of the attribute represented by the node. In general, the left side contains a column for every edge pointing to the node, in this case just the *play* attribute. That is why the table associated with *play* itself does not have a left side: it has no parents. In general, each row of probabilities corresponds to one combination of values of the parent attributes, and the entries in the row show the probability of each value of the node's attribute given this combination. In effect, each row defines a probability distribution over the values of the node's attribute. The entries in a row always sum to 1.

Figure 6.21 shows a more complex network for the same problem, where three nodes (*windy*, *temperature*, and *humidity*) have two parents. Again, there is one column on the left for each parent and as many columns on the right as the attribute has values. Consider the first row of the table associated with the *temperature* node. The left side gives a value for each parent attribute, *play* and

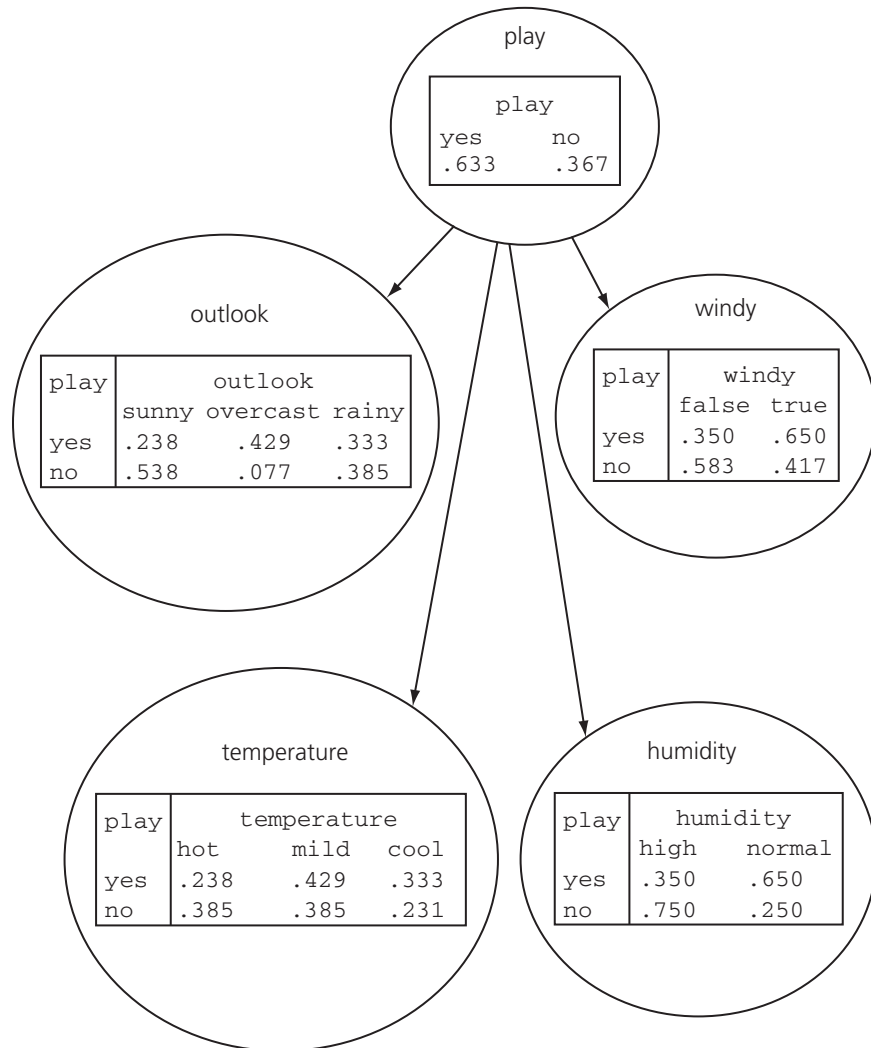


Figure 6.20 A simple Bayesian network for the weather data.

outlook; the right gives a probability for each value of *temperature*. For example, the first number (0.143) is the probability of *temperature* taking on the value *hot*, given that *play* and *outlook* have values *yes* and *sunny*, respectively.

How are the tables used to predict the probability of each class value for a given instance? This turns out to be very easy, because we are assuming that there are no missing values. The instance specifies a value for each attribute. For each node in the network, look up the probability of the node's attribute value based on the row determined by its parents' attribute values. Then just multiply all these probabilities together.

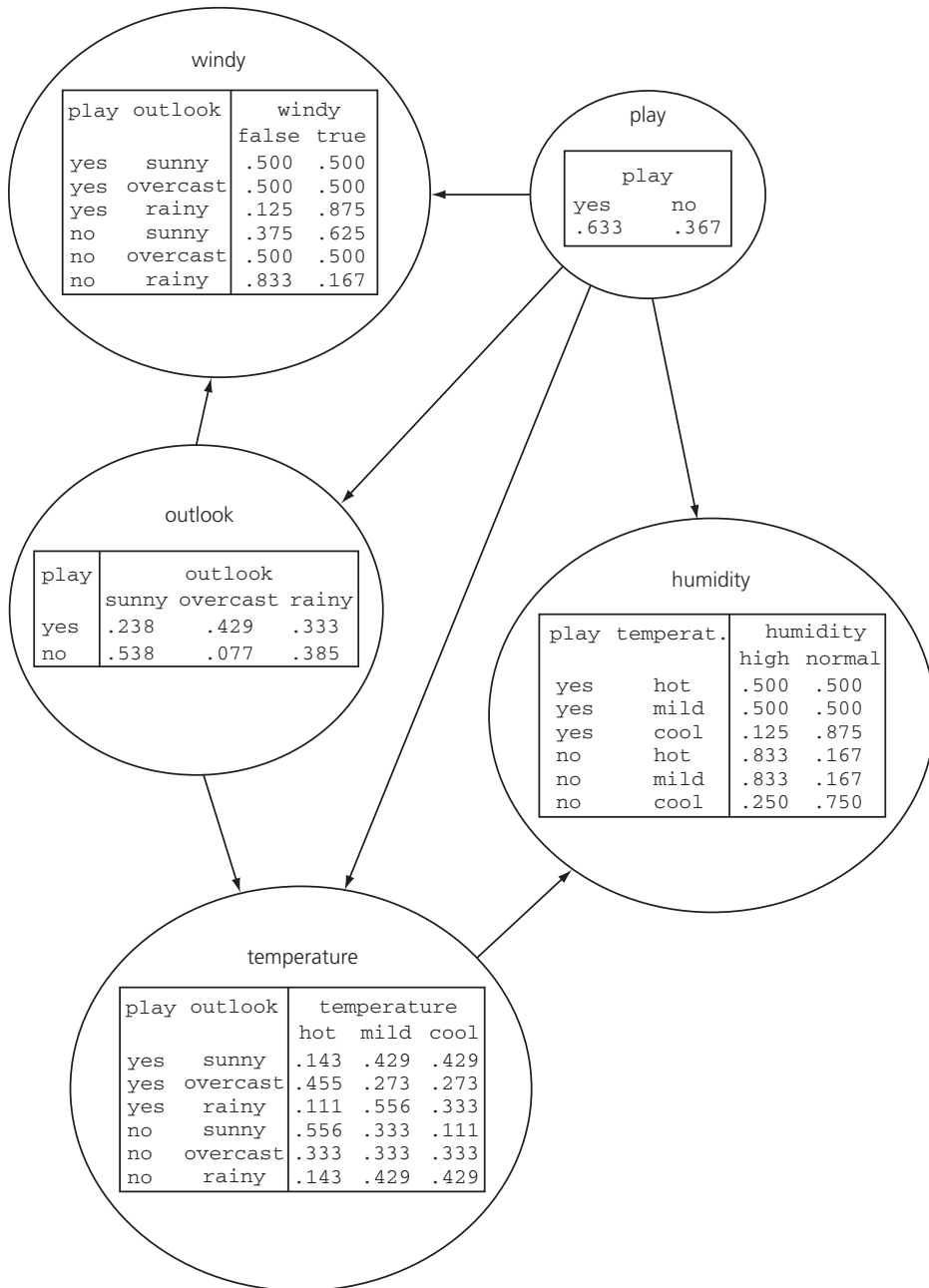


Figure 6.21 Another Bayesian network for the weather data.

For example, consider an instance with values *outlook* = *rainy*, *temperature* = *cool*, *humidity* = *high*, and *windy* = *true*. To calculate the probability for *play* = *no*, observe that the network in Figure 6.21 gives probability 0.367 from node *play*, 0.385 from *outlook*, 0.429 from *temperature*, 0.250 from *humidity*, and 0.167 from *windy*. The product is 0.0025. The same calculation for *play* = *yes* yields 0.0077. However, these are clearly not the final answer: the final probabilities must sum to 1, whereas 0.0025 and 0.0077 don't. They are actually the joint probabilities $\Pr[\textit{play} = \textit{no}, E]$ and $\Pr[\textit{play} = \textit{yes}, E]$, where E denotes all the evidence given by the instance's attribute values. Joint probabilities measure the likelihood of observing an instance that exhibits the attribute values in E as well as the respective class value. They only sum to 1 if they exhaust the space of all possible attribute-value combinations, including the class attribute. This is certainly not the case in our example.

The solution is quite simple (we already encountered it in Section 4.2). To obtain the conditional probabilities $\Pr[\textit{play} = \textit{no} | E]$ and $\Pr[\textit{play} = \textit{yes} | E]$, normalize the joint probabilities by dividing them by their sum. This gives probability 0.245 for *play* = *no* and 0.755 for *play* = *yes*.

Just one mystery remains: why multiply all those probabilities together? It turns out that the validity of the multiplication step hinges on a single assumption—namely that, given values for each of a node's parents, knowing the values for any other ancestors does not change the probability associated with each of its possible values. In other words, ancestors do not provide any information about the likelihood of the node's values over and above the information provided by the parents. This can be written

$$\Pr[\textit{node} | \textit{ancestors}] = \Pr[\textit{node} | \textit{parents}],$$

which must hold for all values of the nodes and attributes involved. In statistics this property is called *conditional independence*. Multiplication is valid provided that each node is conditionally independent of its grandparents, great-grandparents, and so on, given its parents. The multiplication step results directly from the chain rule in probability theory, which states that the joint probability of n attributes a_i can be decomposed into this product:

$$\Pr[a_1, a_2, \dots, a_n] = \prod_{i=1}^n \Pr[a_i | a_{i-1}, \dots, a_1]$$

The decomposition holds for any order of the attributes. Because our Bayesian network is an acyclic graph, its nodes can be ordered to give all ancestors of a node a_i indices smaller than i . Then, because of the conditional independence assumption,

$$\Pr[a_1, a_2, \dots, a_n] = \prod_{i=1}^n \Pr[a_i | a_{i-1}, \dots, a_1] = \prod_{i=1}^n \Pr[a_i | a_i \text{'s parents}],$$

which is exactly the multiplication rule that we applied previously.

The two Bayesian networks in Figure 6.20 and Figure 6.21 are fundamentally different. The first (Figure 6.20) makes stronger independence assumptions because for each of its nodes the set of parents is a subset of the corresponding set of parents in the second (Figure 6.21). In fact, Figure 6.20 is almost identical to the simple Naïve Bayes classifier of Section 4.2. (The probabilities are slightly different but only because each count has been initialized to 0.5 to avoid the zero-frequency problem.) The network in Figure 6.21 has more rows in the conditional probability tables and hence more parameters; it may be a more accurate representation of the underlying domain.

It is tempting to assume that the directed edges in a Bayesian network represent causal effects. But be careful! In our case, a particular value of *play* may enhance the prospects of a particular value of *outlook*, but it certainly doesn't cause it—it is more likely to be the other way round. Different Bayesian networks can be constructed for the same problem, representing exactly the same probability distribution. This is done by altering the way in which the joint probability distribution is factorized to exploit conditional independencies. The network whose directed edges model causal effects is often the simplest one with the fewest parameters. Hence, human experts who construct Bayesian networks for a particular domain often benefit by representing causal effects by directed edges. However, when machine learning techniques are applied to induce models from data whose causal structure is unknown, all they can do is construct a network based on the correlations that are observed in the data. Inferring causality from correlation is always a dangerous business.

Learning Bayesian networks

The way to construct a learning algorithm for Bayesian networks is to define two components: a function for evaluating a given network based on the data and a method for searching through the space of possible networks. The quality of a given network is measured by the probability of the data given the network. We calculate the probability that the network accords to each instance and multiply these probabilities together over all instances. In practice, this quickly yields numbers too small to be represented properly (called *arithmetic underflow*), so we use the sum of the logarithms of the probabilities rather than their product. The resulting quantity is the log-likelihood of the network given the data.

Assume that the structure of the network—the set of edges—is given. It's easy to estimate the numbers in the conditional probability tables: just compute the relative frequencies of the associated combinations of attribute values in the training data. To avoid the zero-frequency problem each count is initialized with a constant as described in Section 4.2. For example, to find the probability that *humidity = normal* given that *play = yes* and *temperature = cool* (the last number

of the third row of the *humidity* node's table in Figure 6.21), observe from Table 1.2 (page 11) that there are three instances with this combination of attribute values in the weather data, and no instances with *humidity = high* and the same values for *play* and *temperature*. Initializing the counts for the two values of *humidity* to 0.5 yields the probability $(3 + 0.5) / (3 + 0 + 1) = 0.875$ for *humidity = normal*.

The nodes in the network are predetermined, one for each attribute (including the class). Learning the network structure amounts to searching through the space of possible sets of edges, estimating the conditional probability tables for each set, and computing the log-likelihood of the resulting network based on the data as a measure of the network's quality. Bayesian network learning algorithms differ mainly in the way in which they search through the space of network structures. Some algorithms are introduced below.

There is one caveat. If the log-likelihood is maximized based on the training data, it will always be better to add more edges: the resulting network will simply overfit. Various methods can be employed to combat this problem. One possibility is to use cross-validation to estimate the goodness of fit. A second is to add a penalty for the complexity of the network based on the number of parameters, that is, the total number of independent estimates in all the probability tables. For each table, the number of independent probabilities is the total number of entries minus the number of entries in the last column, which can be determined from the other columns because all rows must sum to 1. Let K be the number of parameters, LL the log-likelihood, and N the number of instances in the data. Two popular measures for evaluating the quality of a network are the *Akaike Information Criterion* (AIC),

$$\text{AIC score} = -LL + K,$$

and the following *MDL metric* based on the MDL principle:

$$\text{MDL score} = -LL + \frac{K}{2} \log N.$$

In both cases the log-likelihood is negated, so the aim is to minimize these scores.

A third possibility is to assign a prior distribution over network structures and find the most likely network by combining its prior probability with the probability accorded to the network by the data. This is the "Bayesian" approach to network scoring. Depending on the prior distribution used, it can take various forms. However, true Bayesians would average over all possible network structures rather than singling out a particular network for prediction. Unfortunately, this generally requires a great deal of computation. A simplified approach is to average over all network structures that are substructures of a

given network. It turns out that this can be implemented very efficiently by changing the method for calculating the conditional probability tables so that the resulting probability estimates implicitly contain information from all sub-networks. The details of this approach are rather complex and will not be described here.

The task of searching for a good network structure can be greatly simplified if the right metric is used for scoring. Recall that the probability of a single instance based on a network is the product of all the individual probabilities from the various conditional probability tables. The overall probability of the dataset is the product of these products for all instances. Because terms in a product are interchangeable, the product can be rewritten to group together all factors relating to the same table. The same holds for the log-likelihood, using sums instead of products. This means that the likelihood can be optimized separately for each node of the network. This can be done by adding, or removing, edges from other nodes to the node that is being optimized—the only constraint is that cycles must not be introduced. The same trick also works if a local scoring metric such as AIC or MDL is used instead of plain log-likelihood because the penalty term splits into several components, one for each node, and each node can be optimized independently.

Specific algorithms

Now we move on to actual algorithms for learning Bayesian networks. One simple and very fast learning algorithm, called *K2*, starts with a given ordering of the attributes (i.e., nodes). Then it processes each node in turn and greedily considers adding edges from previously processed nodes to the current one. In each step it adds the edge that maximizes the network's score. When there is no further improvement, attention turns to the next node. As an additional mechanism for overfitting avoidance, the number of parents for each node can be restricted to a predefined maximum. Because only edges from previously processed nodes are considered and there is a fixed ordering, this procedure cannot introduce cycles. However, the result depends on the initial ordering, so it makes sense to run the algorithm several times with different random orderings.

The Naïve Bayes classifier is a network with an edge leading from the class attribute to each of the other attributes. When building networks for classification, it sometimes helps to use this network as a starting point for the search. This can be done in *K2* by forcing the class variable to be the first one in the ordering and initializing the set of edges appropriately.

Another potentially helpful trick is to ensure that every attribute in the data is in the *Markov blanket* of the node that represents the class attribute. A node's Markov blanket includes all its parents, children, and children's parents. It can be shown that a node is conditionally independent of all other nodes given

values for the nodes in its Markov blanket. Hence, if a node is absent from the class attribute's Markov blanket, its value is completely irrelevant to the classification. Conversely, if K2 finds a network that does not include a relevant attribute in the class node's Markov blanket, it might help to add an edge that rectifies this shortcoming. A simple way of doing this is to add an edge from the attribute's node to the class node or from the class node to the attribute's node, depending on which option avoids a cycle.

A more sophisticated but slower version of K2 is not to order the nodes but to greedily consider adding or deleting edges between arbitrary pairs of nodes (all the while ensuring acyclicity, of course). A further step is to consider inverting the direction of existing edges as well. As with any greedy algorithm, the resulting network only represents a *local* maximum of the scoring function: it is always advisable to run such algorithms several times with different random initial configurations. More sophisticated optimization strategies such as simulated annealing, tabu search, or genetic algorithms can also be used.

Another good learning algorithm for Bayesian network classifiers is called *tree augmented Naïve Bayes* (TAN). As the name implies, it takes the Naïve Bayes classifier and adds edges to it. The class attribute is the single parent of each node of a Naïve Bayes network: TAN considers adding a second parent to each node. If the class node and all corresponding edges are excluded from consideration, and assuming that there is exactly one node to which a second parent is not added, the resulting classifier has a tree structure rooted at the parentless node—this is where the name comes from. For this restricted type of network there is an efficient algorithm for finding the set of edges that maximizes the network's likelihood based on computing the network's maximum weighted spanning tree. This algorithm is linear in the number of instances and quadratic in the number of attributes.

All the scoring metrics that we have described so far are likelihood based in the sense that they are designed to maximize the joint probability $\Pr[a_1, a_2, \dots, a_n]$ for each instance. However, in classification, what we really want to maximize is the conditional probability of the class given the values of the other attributes—in other words, the conditional likelihood. Unfortunately, there is no closed-form solution for the maximum *conditional*-likelihood probability estimates that are needed for the tables in a Bayesian network. On the other hand, computing the conditional likelihood for a given network and dataset is straightforward—after all, this is what logistic regression does. Hence it has been proposed to use standard maximum likelihood probability estimates in the network, but the conditional likelihood to evaluate a particular network structure.

Another way of using Bayesian networks for classification is to build a separate network for each class value, based on the data pertaining to that class, and combine the predictions using Bayes's rule. The set of networks is called a

when a redundant attribute is about to be added than the backward elimination approach—in conjunction with a very simple, almost “naïve,” metric that determines the quality of an attribute subset to be simply the performance of the learned algorithm on the *training* set. As was emphasized in Chapter 5, training set performance is certainly not a reliable indicator of test-set performance. Nevertheless, experiments show that this simple modification to Naïve Bayes markedly improves its performance on those standard datasets for which it does not do so well as tree- or rule-based classifiers, and does not have any negative effect on results on datasets on which Naïve Bayes already does well. *Selective Naïve Bayes*, as this learning method is called, is a viable machine learning technique that performs reliably and well in practice.

7.2 Discretizing numeric attributes

Some classification and clustering algorithms deal with nominal attributes only and cannot handle ones measured on a numeric scale. To use them on general datasets, numeric attributes must first be “discretized” into a small number of distinct ranges. Even learning algorithms that do handle numeric attributes sometimes process them in ways that are not altogether satisfactory. Statistical clustering methods often assume that numeric attributes have a normal distribution—often not a very plausible assumption in practice—and the standard extension of the Naïve Bayes classifier to handle numeric attributes adopts the same assumption. Although most decision tree and decision rule learners can handle numeric attributes, some implementations work much more slowly when numeric attributes are present because they repeatedly sort the attribute values. For all these reasons the question arises: what is a good way to discretize numeric attributes into ranges before any learning takes place?

We have already encountered some methods for discretizing numeric attributes. The 1R learning scheme described in Chapter 4 uses a simple but effective technique: sort the instances by the attribute’s value and assign the value into ranges at the points that the class value changes—except that a certain minimum number of instances in the majority class (six) must lie in each of the ranges, which means that any given range may include a mixture of class values. This is a “global” method of discretization that is applied to all continuous attributes before learning starts.

Decision tree learners, on the other hand, deal with numeric attributes on a local basis, examining attributes at each node of the tree when it is being constructed to see whether they are worth branching on—and only at that point deciding on the best place to split continuous attributes. Although the tree-building method we examined in Chapter 6 only considers binary splits of continuous attributes, one can imagine a full discretization taking place at that

point, yielding a multiway split on a numeric attribute. The pros and cons of the local versus the global approach are clear. Local discretization is tailored to the actual context provided by each tree node, and will produce different discretizations of the same attribute at different places in the tree if that seems appropriate. However, its decisions are based on less data as tree depth increases, which compromises their reliability. If trees are developed all the way out to single-instance leaves before being pruned back, as with the normal technique of backward pruning, it is clear that many discretization decisions will be based on data that is grossly inadequate.

When using global discretization before applying a learning method, there are two possible ways of presenting the discretized data to the learner. The most obvious is to treat discretized attributes like nominal ones: each discretization interval is represented by one value of the nominal attribute. However, because a discretized attribute is derived from a numeric one, its values are ordered, and treating it as nominal discards this potentially valuable ordering information. Of course, if a learning scheme can handle ordered attributes directly, the solution is obvious: each discretized attribute is declared to be of type “ordered.”

If the learning method cannot handle ordered attributes, there is still a simple way of enabling it to exploit the ordering information: transform each discretized attribute into a set of binary attributes before the learning scheme is applied. Assuming the discretized attribute has k values, it is transformed into $k - 1$ binary attributes, the first $i - 1$ of which are set to *false* whenever the i th value of the discretized attribute is present in the data and to *true* otherwise. The remaining attributes are set to *false*. In other words, the $(i - 1)$ th binary attribute represents whether the discretized attribute is less than i . If a decision tree learner splits on this attribute, it implicitly uses the ordering information it encodes. Note that this transformation is independent of the particular discretization method being applied: it is simply a way of coding an ordered attribute using a set of binary attributes.

Unsupervised discretization

There are two basic approaches to the problem of discretization. One is to quantify each attribute in the absence of any knowledge of the classes of the instances in the training set—so-called *unsupervised* discretization. The other is to take the classes into account when discretizing—*supervised* discretization. The former is the only possibility when dealing with clustering problems in which the classes are unknown or nonexistent.

The obvious way of discretizing a numeric attribute is to divide its range into a predetermined number of equal intervals: a fixed, data-independent yardstick. This is frequently done at the time when data is collected. But, like any unsupervised discretization method, it runs the risk of destroying distinctions that

would have turned out to be useful in the learning process by using gradations that are too coarse or by unfortunate choices of boundary that needlessly lump together many instances of different classes.

Equal-interval binning often distributes instances very unevenly: some bins contain many instances, and others contain none. This can seriously impair the ability of the attribute to help to build good decision structures. It is often better to allow the intervals to be of different sizes, choosing them so that the same number of training examples fall into each one. This method, *equal-frequency binning*, divides the attribute's range into a predetermined number of bins based on the distribution of examples along that axis—sometimes called *histogram equalization*, because if you take a histogram of the contents of the resulting bins it will be completely flat. If you view the number of bins as a resource, this method makes best use of it.

However, equal-frequency binning is still oblivious to the instances' classes, and this can cause bad boundaries. For example, if all instances in a bin have one class, and all instances in the next higher bin have another except for the first, which has the original class, surely it makes sense to respect the class divisions and include that first instance in the previous bin, sacrificing the equal-frequency property for the sake of homogeneity. Supervised discretization—taking classes into account during the process—certainly has advantages. Nevertheless, it has been found that equal-frequency binning can yield excellent results, at least in conjunction with the Naïve Bayes learning scheme, when the number of bins is chosen in a data-dependent fashion by setting it to the square root of the number of instances. This method is called *proportional k-interval discretization*.

Entropy-based discretization

Because the criterion used for splitting a numeric attribute during the formation of a decision tree works well in practice, it seems a good idea to extend it to more general discretization by recursively splitting intervals until it is time to stop. In Chapter 6 we saw how to sort the instances by the attribute's value and consider, for each possible splitting point, the information gain of the resulting split. To discretize the attribute, once the first split is determined the splitting process can be repeated in the upper and lower parts of the range, and so on, recursively.

To see this working in practice, we revisit the example on page 189 for discretizing the temperature attribute of the weather data, whose values are

64	65	68	69	70	71	72	75	80	81	83	85
yes	no	yes	yes	yes	no	no	yes	no	yes	yes	no
						yes	yes				

(Repeated values have been collapsed together.) The information gain for each of the 11 possible positions for the breakpoint is calculated in the usual way. For example, the information value of the test $temperature < 71.5$, which splits the range into four *yes*'s and two *no*'s versus five *yes*'s and three *no*'s, is

$$\text{info}([4, 2], [5, 3]) = (6/14) \times \text{info}([4, 2]) + (8/14) \times \text{info}([5, 3]) = 0.939 \text{ bits}$$

This represents the amount of information required to specify the individual values of *yes* and *no* given the split. We seek a discretization that makes the subintervals as pure as possible; hence, we choose to split at the point where the information value is smallest. (This is the same as splitting where the information *gain*, defined as the difference between the information value without the split and that with the split, is largest.) As before, we place numeric thresholds halfway between the values that delimit the boundaries of a concept.

The graph labeled A in Figure 7.2 shows the information values at each possible cut point at this first stage. The cleanest division—smallest information value—is at a temperature of 84 (0.827 bits), which separates off just the very final value, a *no* instance, from the preceding list. The instance classes are written below the horizontal axis to make interpretation easier. Invoking the algorithm again on the lower range of temperatures, from 64 to 83, yields the graph labeled B. This has a minimum at 80.5 (0.800 bits), which splits off the next two values,

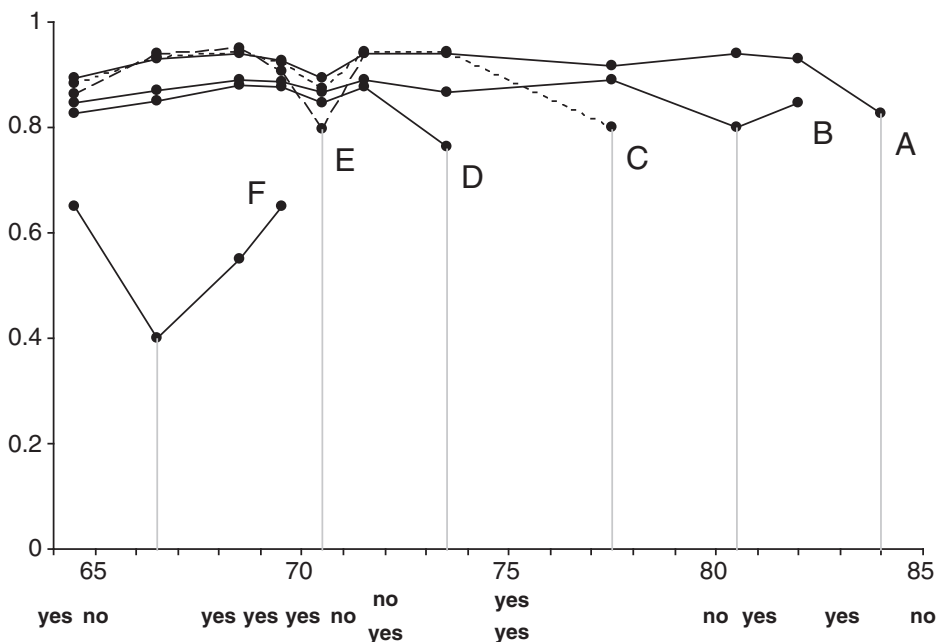


Figure 7.2 Discretizing the *temperature* attribute using the entropy method.

64	65	68	69	70	71	72	75	80	81	83	85
yes	no	yes	yes	yes	no	no	yes	no	yes	yes	no
						yes	yes				
		F			E		D	C	B		A
		66.5			70.5		73.5	77.5	80.5		84

Figure 7.3 The result of discretizing the *temperature* attribute.

both *yes* instances. Again invoking the algorithm on the lower range, now from 64 to 80, produces the graph labeled C (shown dotted to help distinguish it from the others). The minimum is at 77.5 (0.801 bits), splitting off another *no* instance. Graph D has a minimum at 73.5 (0.764 bits), splitting off two *yes* instances. Graph E (again dashed, purely to make it more easily visible), for the temperature range 64 to 72, has a minimum at 70.5 (0.796 bits), which splits off two *nos* and a *yes*. Finally, graph F, for the range 64 to 70, has a minimum at 66.5 (0.4 bits).

The final discretization of the *temperature* attribute is shown in Figure 7.3. The fact that recursion only ever occurs in the first interval of each split is an artifact of this example: in general, both the upper and the lower intervals will have to be split further. Underneath each division is the label of the graph in Figure 7.2 that is responsible for it, and below that is the actual value of the split point.

It can be shown theoretically that a cut point that minimizes the information value will never occur between two instances of the same class. This leads to a useful optimization: it is only necessary to consider potential divisions that separate instances of different classes. Notice that if class labels were assigned to the intervals based on the majority class in the interval, there would be no guarantee that adjacent intervals would receive different labels. You might be tempted to consider merging intervals with the same majority class (e.g., the first two intervals of Figure 7.3), but as we will see later (pages 302–304) this is not a good thing to do in general.

The only problem left to consider is the stopping criterion. In the temperature example most of the intervals that were identified were “pure” in that all their instances had the same class, and there is clearly no point in trying to split such an interval. (Exceptions were the final interval, which we tacitly decided not to split, and the interval from 70.5 to 73.5.) In general, however, things are not so straightforward.

A good way to stop the entropy-based splitting discretization procedure turns out to be the MDL principle that we encountered in Chapter 5. In accordance with that principle, we want to minimize the size of the “theory” plus the size of the information necessary to specify all the data given that theory. In this case, if we do split, the “theory” is the splitting point, and we are comparing the situation in which we split with that in which we do not. In both cases we assume that the instances are known but their class labels are not. If we do not split, the classes can be transmitted by encoding each instance’s label. If we do, we first encode the split point (in $\log_2[N - 1]$ bits, where N is the number of instances), then the classes of the instances below that point, and then the classes of those above it. You can imagine that if the split is a good one—say, all the classes below it are *yes* and all those above are *no*—then there is much to be gained by splitting. If there is an equal number of *yes* and *no* instances, each instance costs 1 bit without splitting but hardly more than 0 bits with splitting—it is not quite 0 because the class values associated with the split itself must be encoded, but this penalty is amortized across all the instances. In this case, if there are many examples, the penalty of having to encode the split point will be far outweighed by the information saved by splitting.

We emphasized in Section 5.9 that when applying the MDL principle, the devil is in the details. In the relatively straightforward case of discretization, the situation is tractable although not simple. The amounts of information can be obtained exactly under certain reasonable assumptions. We will not go into the details, but the upshot is that the split dictated by a particular cut point is worthwhile if the information gain for that split exceeds a certain value that depends on the number of instances N , the number of classes k , the entropy of the instances E , the entropy of the instances in each subinterval E_1 and E_2 , and the number of classes represented in each subinterval k_1 and k_2 :

$$\text{gain} > \frac{\log_2(N-1)}{N} + \frac{\log_2(3^k - 2) - kE + k_1E_1 + k_2E_2}{N}.$$

The first component is the information needed to specify the splitting point; the second is a correction due to the need to transmit which classes correspond to the upper and lower subintervals.

When applied to the temperature example, this criterion prevents any splitting at all. The first split removes just the final example, and as you can imagine very little actual information is gained by this when transmitting the classes—in fact, the MDL criterion will never create an interval containing just one example. Failure to discretize *temperature* effectively disbars it from playing any role in the final decision structure because the same discretized value will be given to all instances. In this situation, this is perfectly appropriate: the *temper-*

ature attribute does not occur in good decision trees or rules for the weather data. In effect, failure to discretize is tantamount to attribute selection.

Other discretization methods

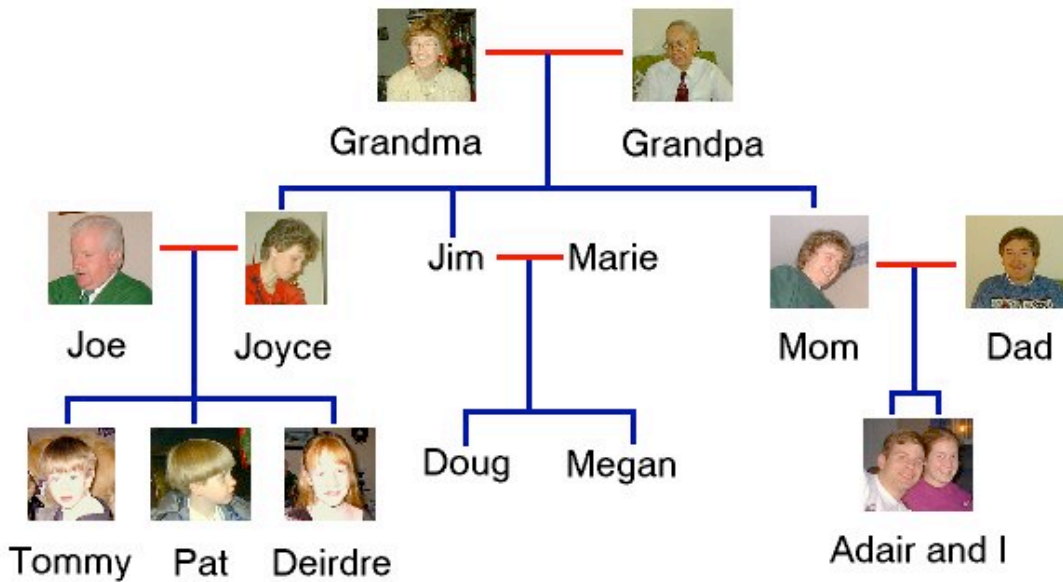
The entropy-based method with the MDL stopping criterion is one of the best general techniques for supervised discretization. However, many other methods have been investigated. For example, instead of proceeding top-down by recursively splitting intervals until some stopping criterion is satisfied, you could work bottom-up, first placing each instance into its own interval and then considering whether to merge adjacent intervals. You could apply a statistical criterion to see which would be the best two intervals to merge, and merge them if the statistic exceeds a certain preset confidence level, repeating the operation until no potential merge passes the test. The χ^2 test is a suitable one and has been used for this purpose. Instead of specifying a preset significance threshold, more complex techniques are available to determine an appropriate level automatically.

A rather different approach is to count the number of errors that a discretization makes when predicting each training instance's class, assuming that each interval receives the majority class. For example, the 1R method described earlier is error based—it focuses on errors rather than the entropy. However, the best possible discretization in terms of error count is obtained by using the largest possible number of intervals, and this degenerate case should be avoided by restricting the number of intervals in advance. For example, you might ask, what is the best way to discretize an attribute into k intervals in a way that minimizes the number of errors?

The brute-force method of finding the best way of partitioning an attribute into k intervals in a way that minimizes the error count is exponential in k and hence infeasible. However, there are much more efficient schemes that are based on the idea of dynamic programming. Dynamic programming applies not just to the error count measure but also to any given additive impurity function, and it can find the partitioning of N instances into k intervals in a way that minimizes the impurity in time proportional to kN^2 . This gives a way of finding the best entropy-based discretization, yielding a potential improvement in the quality of the discretization (but in practice a negligible one) over the recursive entropy-based method described previously. The news for error-based discretization is even better, because there is a method that minimizes the error count in time linear in N .

Entropy-based versus error-based discretization

Why not use error-based discretization, since the optimal discretization can be found very quickly? The answer is that there is a serious drawback to error-based



Family Pets



Numeric Transforms

Pragmatically, the following transforms for numbers are very useful:

- Coarse-grain: if many numbers, bunch them up into a small set; e.g. below mean, above mean
 - More generally, discretize them (see [#Discretization]).
- Log: if the numerics form an exponential distribution, convert all N to log(N)
 - Used in the COCOMO example
 - pragmatics: missing values can't be "zero" (infinitely negative log values)
 - Apply some min value (e.g. 0.0001) and use:
 - new = (if old < min then log(min) else log(old))
- Remove outliers: suspiciously large/small values
- Replace unknowns with most expected value
 - Numerics: use mean or median
- Discretizes: use most common symbol
 - ?cluster first, then fill in from local neighborhood
 - Warning: maybe missing really means missing!
 - Also, some learners can handle missing values.
- Bore: Best or Rest
 - Sort: rename the top X% (best) values "good" and the rest "bad"; eg. X=25%
 - Scales to multi-variables (can be used to replace multiple numeric target classes with one binary classification)
- Time series
 - Add attributes to record the moving average over the last N minutes, 5N minutes, 25N minutes, etc

Sampling

Build data sets by sub-sampling real data

- Column sub-sampling: manual FeatureSelection?
 - Maybe there is domain knowledge that some columns are
 - More costly to use

- Less trustworthy
- E.g. in the MDP data: different measures from modules
- Row sampling: manual [Stratification?]
 - Ignore all but the relevant data (how to judge? domain knowledge? nearest neighbor?).

data	language	(# modules)		
		examples	features	%defective
pc5	C++	17,186	38	3.0
mc1	C++	9,466	38	0.71
pc2	C++	5,589	36	0.41
kc1	C++	2,109	21	15.45
pc3	C++	1,563	37	10.23
pc4	C	1,458	37	12.2
pc1	C++	1,109	21	6.94
kc2	C++	522	21	20.49
cm1	C++	498	21	9.83
kc3	JAVA	458	39	9.38
mw1	C++	403	37	7.69
mc2	C++	61	39	32.29
		40,422		

When the target class is rare,

- Sub-sample: create training sets that contain all the target instances and an equal number of randomly selected non-target instances (stand back! give the little guy some air!).
- Super-sample: take the minority class and repeat it (build yourself up in the crowd).

Some experimental results:

- In one study, under-sampling beat over-sampling Drummond03?
- In another, once again, over-sampling was useless and under-sampling did the same as no-sampling, but with much much less data Menzies08a?. The following results show balance results for the above data sets. NB= naive bayes. J48= a decision tree learner.

treatment	quartiles					
	min	25%	median	75%	max	
NB/ none	21.9	67.7	74.6	81.9	100.0	----- ●-----
NB/ under	19.9	67.1	74.1	81.6	100.0	----- ●-----
J48/ under	21.6	64.8	73.6	82.6	100.0	----- ●-----
NB/ over	17.5	42.0	62.5	72.2	100.0	----- ●-----
J48/ over	0.0	29.3	45.6	56.2	100.0	----- ●-----
J48/ none	0.0	29.3	42.3	54.5	100.0	----- ●-----

Standard Text Transforms

Tokenize (kill white space).

Send to lower case.

Remove stop words (boring words like "a above above.." (see examples at http://www.dcs.gla.ac.uk/idom/irresources/linguisticutils/stop_words

- Warning: don't remove words that are important to your domain.

Stemming

- These words are all "connect": connect, connected, connecting, connection,connections
- PorterStemming?: the standard stemming algorithm, available in multiple languages: <http://www.tartarus.org/~martin/PorterStemmer/>
 - Definition: <http://www.tartarus.org/~martin/PorterStemmer/def.txt>

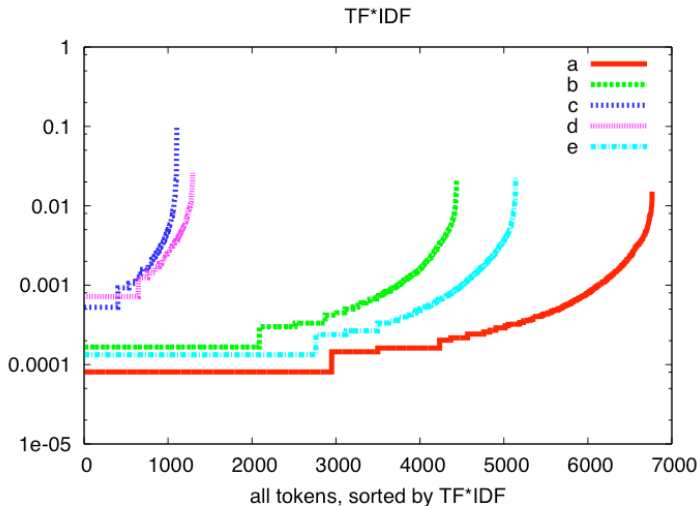
Reject all but the top k most interesting words

- Interesting if frequent OR usually appears in just a few paragraphs
- TF*IDF (term frequency, inverse document frequency)

- Interesting =

$$F(i,j) * \log((\text{Number of documents})/(\text{number of documents including word } i))$$

- $F(i,j)$: frequency of word i in document j
- Often, on a very small percentage of the words are high scorers, so a common transform is to just use the high fliers. e.g. here's data from five bug tracking systems a,b,c,d,e:



Build a symbol table of all the remaining words

- Convert strings to pointers into the symbol table
- So "the cat sat on the cat" becomes 6 pointers to 4 words

Misc

Watch for one-letter typos

- Check all symbols that occur only once in the data.

Add synthetic attributes

- Capture domain knowledge
- e.g. risk is age/weight*temperature

Sample randomly

- Useful when the whole data can't fit into ram.
- Useful when building training/test sets

Sample instances according to the mis-classification rate of its class

- So focus more on the things that are harder to classify
- Also called Boosting: discussed (much) later

Discretization

Think of learning like an accordion- some target concept is spread out across all the data and our task is to squeeze it together till it is dense enough to be visible. That is, learning is like a compression algorithm.

One trick that helps compressions is discretization: i.e. clumping together observations taken over a continuous range into a small number of regions. Humans often discretize real world data. For example, parents often share tips for "toddlers"; i.e. humans found between the breaks of age=1 and age=3.

Many researchers report that discretization improves the performance of a learner since it gives a learner a smaller space to reason about, with more examples in each part of the space ([Dou95](#), [Yang03](#), [Fayyad93](#)). What is Discretization?

Formally, discretization can generally be described as a process of assigning data attribute instances to bins or

buckets that they fit in according to their value or some other score:

- The general concept for discretization as a binning process is dividing up each instance of an attribute to be discretized into a number distinct buckets or bins.
- The number of bins is most often a user-defined, arbitrary value.
- However, some methods use more advanced techniques to determine an ideal number of bins to use for the values.
- While others use the user-defined value as a starting point and expand or contract the number of bins that are actually used (based upon the number of data instances being placed in the bins).
- Each bin or bucket is assigned a range of the attribute values to contain, and discretization occurs when the values that fall within a particular bucket (or bin) are replaced by identifier for the bucket into which they fall.

After [Gama and Pinto](#), we say that

- Discretization is the process of converting a continuous range into a histogram with "k" break points
- $b_1 \dots b_k$ (where for all $i < j$: $\text{not}(b_i = b_j)$).
- The histogram divides a continuous range into bins (one for each break) and many observations from the range may fall between two break points b_i and b_{i+1} at frequency counts c_i .

Simple discretizers are unsupervised methods that build their histograms without exploiting information about the target class; e.g.

- equal width discretization: $(b_i - b_{i-1}) = (b_j - b_{j-1})$
- equal frequency discretization: $c_i = c_j$

How to discretize

Unsupervised discretization: ignore the class variable, just chop each column (this may seem dumb, but often works surprisingly well)

Supervised discretization: separates the numerics according to the class variable

Unsupervised methods

Nbins: divide data into N equal width bins

- Pass1: find min and max of each column. Find bin size for each column $(\text{max} - \text{min})/N$.
- Pass2: convert all numbers X to $\text{floor}(X - \text{min})/\text{binsize}$.
- $N=10$ is a commonly used number (but for Naive Bayes classifiers working on "n" instances, Yang and Webb [Yang03](#) advocate equal frequency with $c_i=c_j=\text{sqrt}(n)$).
- Example:
 - e.g. divide 0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,2,2,2,2,3,4,5,10,20,40,80,100 using 10bins

```
0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,2,2,2,2,3,4,5,10,20,40,80,100
-----|---|---|---|---|
bin1                                     b2 b3 b5 b9 b10
```

- one bucket would get numbers 1 to 25,
- the last 4 numbers would get a bin each.
- So our learner would have 5 bins with nothing in it,
 - one bin with 83% of the data and
 - 4 bins with 3.3% of the data in each.
- Simple variants:
 - [BinLogging2](#): set N via the number of unique numerics $N=\text{max}(1,\text{log2}(\text{uniqueValues}))$
 - Caution, for numbers generated from some random process and if you are using many significant figures, then you may need to round back.
 - Logging filter: hit distributions like the above with $X = \text{log}(X)$. This smoothes out the distributions across more of the buckets.

Percentile chop: diver data into N equal sized bins

- Pass1: collect all numbers for each column. Sort them. Break the sorted numbers into N equal frequency regions (checking that numbers each size of the break are different).
- So the frequency counts in each bin is equal (flat histogram).
- Example: In practice, not quite flat. e.g. 10 equal frequency bins on the above data:

```

0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,2,2,2,2,3,4,5,10,20,40,80,100
-----|-----|-----|-----|-----|-----|-----|-----|-----|
bin1  bin2 bin3  bin4  bin5  bin6  bin7  bin8  bin9

```

- Note the buckets with repeated entries. Its a design choice what to do with those.
- We might squash them together such that there are no repeats in the numbers that are the boundaries between bins.

```

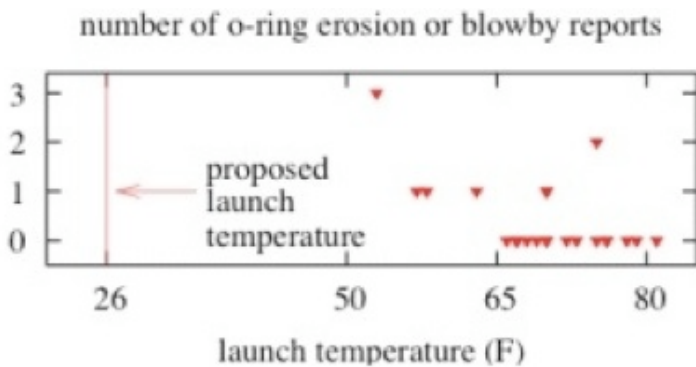
0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,2,2,2,2,3,4,5,10,20,40,80,100
-----|-----|-----|-----|-----|-----|
bin1          bin2          bin3          bin4 bin5          bin6

```

Supervised Discretization

Find a cliff where there most change in the class distribution. For example:

- class=dead (always) if age under 120; class=alive (sometimes) if age under 120
 - 120 is the cliff
 - The following graph shows the number of o-ring damage reports seen in the space shuttle prior to the Challenger launch. There is a cliff at X=65 degrees, below which all launches had o-ring damage.



Some details on cliff learning (from <https://list.scms.waikato.ac.nz/pipermail/wekalist/2004-October/002986.html>). Per numeric attribute, apply the following:

- Sort the instances on the attribute of interest
- Look for potential cut-points.
 - Cut points are points in the sorted list above where the class labels change.
 - Eg. if I had five instances with values for the attribute of interest and labels
 - (1.0,A), (1.4,A), (1.7, A), (2.0,B), (3.0, B), (7.0, A),
 - then there are two cut points of interest (mid-way between the points where the classes change from A to B or vice versa):
 - 1.85
 - 5
- Apply your favorite measure on each of the cuts, and choose the one with the maximum value
 - Weight the measures by the same size; e.g.
 - if 20% and 80% of the data fall left and right of the slot
 - measure = 0.2 * rightMeasure + 0.8 * leftMeasure
 - e.g. variance, info gain etc etc
 - Common practice is to follow the lead of [Fayyad93](#) and use info gain
 - Let classes X and Y occur on the right-hand-side (containing 20%) of the data
 - Let class frequency be F_x and F_y (and $N=F_x + F_y$ numbers)
 - Let $p_x = F_x/N$
 - Let $p_y = F_y/N$
 - Let $\text{entropy}_x = -p_x \cdot \log_2(p_x)$
 - Let $\text{entropy}_y = -p_y \cdot \log_2(p_y)$
 - Let $\text{weighted entropy}_{\text{right}} = -0.2 * (\text{entropy}_x + \text{entropy}_y)$
 - Return the split that minimizes $\text{entropy}_{\text{right}} + \text{entropy}_{\text{left}}$
- Repeat recursively in both subsets (the ones less than and greater than the cut point) until either
 - the subset is "pure" i.e. only contains instances of a single class
 - some stopping criterion is reached. e.g. too few examples to proceed

Incremental Discretization

An interesting variant on discretization is incremental discretization. Suppose we are learning from an infinite data stream so we'll never know "the number of unique symbols" or the "max" and "min" of that data. How might we conduct discretization?

Incremental discretization can be very simple. Below, we describe two schemes:

- D.J. Bolands' RBST method (a local favorite; sees 2007 WVU CSEE masters thesis;
- [Gamma and Pinto's](#) PID method (more widely used).

RBSTs for Incremental Discretization

Consider this binary search tree BST, where everything on the left is less than everything on the right.

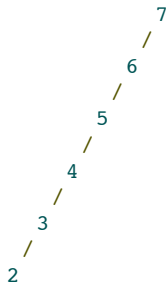


We can add the number "5" in at least two places in this tree, and still preserve the search property that everything on the left is less than the stuff on the right. In case (a), we insert at root and in (b), we insert at leaf.

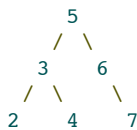


Internally, what we really do is always insert at leaf and sometimes bubble up the leaf to the root, switching sub-trees as we go to preserve the ordering. In random BST (RBST), we insert at root of a sub-tree with probability $1/(N+1)$, where "N" is the number of sub-tree nodes. RBST's tend to generate balanced trees.

Why? Because we are reaching in at random to a distribution, and dividing into two sets below and above a number. Repeat that a few times and you tend to get balanced trees. So you very rarely get this:



Rather, you tend to get this:



So, what has all this got to do with incremental discretization?

- Note that if we had a balanced tree, we could perform discretization by just returning (say) the breaks at level 3 of the tree (below 2, 2 to 4, 4 to 7, above 7). To do this, we'd add a counter to each node and if something arrives T times at node N, then N's counter value is T. So balanced trees can be used for batched discretization.
- But note that RBSTs adjust themselves after each insertion. So if used for an infinite stream of arrivals, they are always self-adjusting. This ability to react to new data and changes in the distribution of the new data reacting is exactly what we want from an incremental discretizer.
- Infinite data streams cause memory problems (cannot store infinite memory). RBSTs support a simple garbage collection algorithm. If we are discretizing at level L (in the above case, $L=3$) then we can periodically throw away the subtrees below level (say) 2^L . Yes, we'll lose some details but those details

are down in the weeds and we can live without them.

Pid

[Gama and and Pinto](#)'s Partition Incremental Discretization (PiD) maintains two sets of "break" points and "counts" of values that fall into each break:

- Layer two: the actual discretized ranges. Layer two is very small and is generated on demand from layer one.
- Layer one: is very large (say, 30 times the number of bins you seek); Layer one just maintains counts on a large number of bins and if one bin gets too big (e.g. $1/(\text{number of bins})$), it is split in two (and all the breaks and counts arrays are pushed up by one index value).

That's nearly all there is too it. Layer one is initialized according to some wild guess about the min and max possible values (and if data arrives outside that range, then a new bin is added bottom or two of "breaks" and "counts"). Layer two could be generated in any number of ways (nbins, logbins, [FayyadIranni](#), etc) and those methods could work by querying the layer one data.

When should we recreate layer2? Here are three policies:

- For equal width discretization: if ever we split a bin, rebuild layer2.
- For equal frequency discretization: if a layer1 bin gets too large, rebuild layer2. If we have seen "n" examples, and our bins have min and max counts of "cmin" and "cmax" then rebuild layer2 when we see an interval with:
 - count below $(1-\text{beta})\text{cmin}/n$ or
 - count above $(1+\text{beta})\text{cmax}/n$
 - Gama and Pinto comment that $\text{beta}=1/100$ seems to be a useful value.
- For other discretization policies, recreate layer2 after seeing N examples (say, $N=100$).

Here's the pseudo-code for updating layer1. Its a little tacky (a linear time operation to increase the size of an array) but it runs so fast that no one cares:

```
Update-Layer1(x, breaks, counts, NrB, alfa, Nr)
  x - observed value of the random variable
  breaks - vector of actual set of break points
  counts - vector of actual set of frequency counts
  NrB - Actual number of breaks
  alfa - threshold for Split an interval
  Nr - Number of observed values

If (x < breaks[1]) k = 1; Min.x = x
Else If (x > breaks[NrB]) k = NrB; Max.x = x
Else k = 2 + integer((x - breaks[1]) / step)

while(x < breaks[k-1]) k <- k - 1
while(x > breaks[k]) k <- k + 1

counts[k] = 1 + counts[k]
Nr = 1 + Nr
If ((1+counts[k])/(Nr+2) > alfa) {
  val = counts[k] / 2
  counts[k] = val
  if (k == 1) {
    breaks = append(breaks[1]-step, breaks)
    counts <- append(val,counts)
  }
  else {
    if(k == NrB) {
      breaks <- append(breaks, breaks[NrB]+step)
      counts <- append(counts,val)
    }
    else {
      breaks <- Insert((breaks[k]+ breaks[k+1])/2, breaks, k)
      counts <- Insert(val, counts, k)
    }
  }
}
NrB = NrB + 1
}
```

Applications of Incremental Discretization: Anomaly Detection and Repair

Curiously, the literature is silent on two obvious applications of incremental discretization:

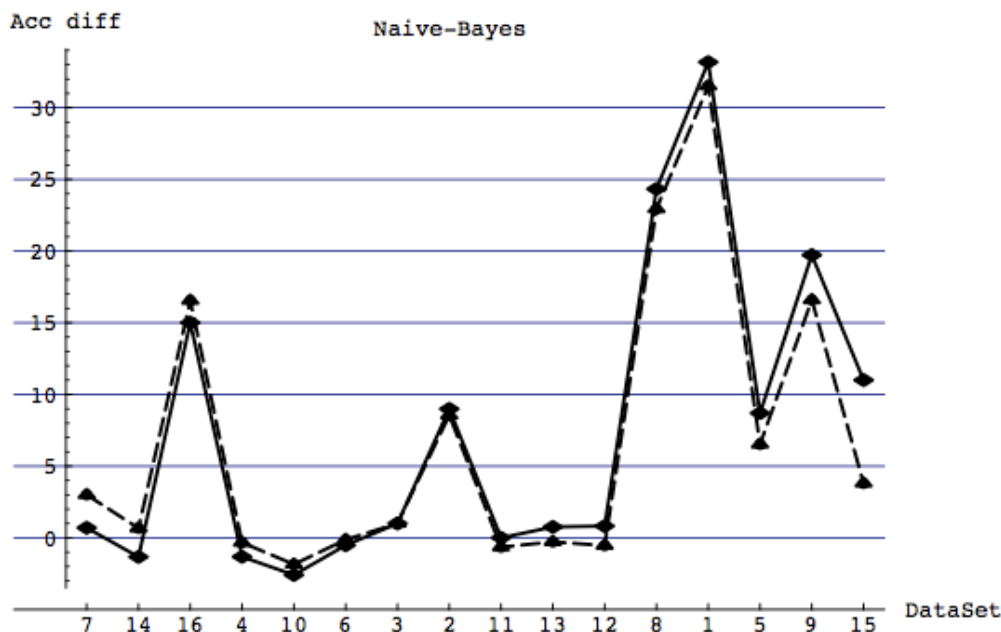
- Anomaly detection: if the discretization boundaries in an incremental discretizer were stable, then start changing, then something is happening to the data generating phenomena. Incremental discretizers could alert when old knowledge needs to be thrown out and new learning initiated.
- Repair: if we could track how discretization ranges changed, then we could take old knowledge, patch its ranges, and test the fixes. If that happened incrementally with changes to the discretization boundaries, then we'd be keeping the knowledge up to date with the underlying data generating phenomena.

What Works Best?

The following graph from [Dou95](#), shows results from three experiments:

- Experiment 1: Running a standard Naive Bayes classifier
- Experiment 2,3: Discretizing the data in one of two ways, then running Naive Bayes.

The y-axis of this graph shows the difference between experiment 1 and the others. Any "y" value greater than one means that discretization increased accuracy. The data sets on the x-axis are sorted by the delta between the experiment 2,3 results (so, on the left, one discretizer is best and, on the right, the other is best).



Note that:

- discretization rarely made things worse;
- often made things much better;
- and the exact nature of the discretizer was not so important.

What	win	loss	ties
fayyadIrani	4	0	0
pkid	2	1	1
disctree3	2	1	1
tbin	1	3	0
cat	0	4	0

Here's another result, from Boland07. These are "win-loss-tie" tables showing a statistical analysis of the difference between several discretization methods:

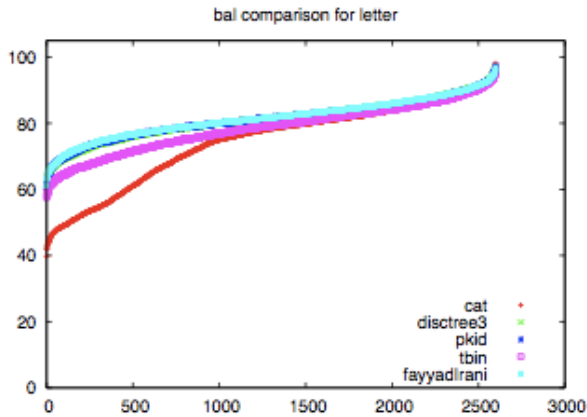
- Some of the undiscretized methods shown below:
 - "tbin" is Nbins (discussed below) with N=10
 - "disctree3" (a neat trick from Boland07);
- A supervised methods (discussed below) called "fayyadIrani" from [Fayyad93](#) (discussed below, see "cliff

- learning").
- And other methods including "cat" (do nothing).

Note that there is an overall winner (fayyadlrani) and this is the discretizer in widest current use.

But if you look at the raw numbers (say, for "balance"), a different picture emerges. This is one result, out of the hundreds explored by Boland07. Note, as before:

- discretization rarely made things worse (i.e. do worse than "cat");
- can make things much better;
- and the exact nature of the discretizer is not so important.



From the above, we conclude that discretization is important and that we not get too tense about exploring better discretizers. Time to move on.

Comment by [menzies.tim](#), Sep 29, 2009

[Delete comment](#)

i want to go home

Enter a comment:

Submit

Wiki markup help

[hide](#)

=Heading1=
 ==Heading2==
 ===Heading3===

bold *_italic_*
 `inline code`
 escape: `*`

Indent lists 2 spaces:
 * bullet item
 # numbered list

{{
 verbatim code block
 }}

Horizontal rule

WikiWordLink
 [http://domain/page label]
 http://domain/page

|| table || cells ||

[More examples](#)

Improving IV&V Techniques Through the Analysis of Project Anomalies: Text Mining PITS issue reports - final report

Tim Menzies

Lane Department of Computer Science and Electrical Engineering, West Virginia University, USA

tim@menzies.us

<http://menzies.us>

Abstract This project is in two parts. The *second part* will try to combine two (or more) of the IV&V data sources into an active monitoring framework where data collected during an active IV&V project will trigger an alert if a project becomes unusual” (and defining “unusual” is one of the goals of this project).

But before we can generalize between sources, we need to study each source in isolation to determine its strengths and weaknesses. Hence, the *first part* of this project aims to gain experience with the various IV&V data sources available to researchers like myself; i.e.

- SILAP, from the IV&V planning and scoping team;
- James Dabney’s Bayes networks that describe the IV&V business practices of the L3 IV&V contractor;
- The PITS issue tracking data;
- The LINKER database project that intends to join PITS to other data sources;
- Balanced score card strategy maps from NASA Langley.
- and the COCOMO data sets from JPL.

This is the second year of a three year project that started in June 2006. The project is data-rich project and much progress has already been achieved.

- At SAS’06, a preliminary report described what had been learned from the SILAP data. A ranking was offered on the most common IV&V work-breakdown structure (WBS) activities. This ranking can be used for (e.g.) identifying what WBS tasks would benefit most from optimization.
- This report on SILAP was finalized in the first part of 2007. In summary, there exists a very strong signal in the SILAP data for issue frequency and severity.
- In October’06, a preliminary report was delivered on the Bayes network. On a limited case study, it was shown that Bayes nets and treatment learning could generate parsimonious explanations for project events.
- A preliminary report on text mining from the PITS issues tracking database that generated an expert system which audited a test engineer’s proposed severity level.

This document updates the preliminary PITS report. Before, the PITS report studied two projects with a limited range of

severities (mostly severity 3 and 4). Here, we explore five projects with a much wider range of severities. The results from the PITS preliminary report is confirmed. Using text mining, PITS can be used to generate an expert system that audits a test engineer’s proposed severity level for an issue.

Credits: This work was made possible due to the heroic efforts of Ken Costello (chief engineering at NASA IV&V) who provided the PITS defect reports. The text mining technology used here was inspired by the trace-ability work of Jane Hayes and Alex Dekhtyar. Alex was particularly helpful in mapping out the ABCs of text mining. Jane also offered extensive advice on how to extend the current system. This research was conducted at West Virginia University under NASA sub-contract project 100005549, task 5e, award 1002193r.

Cautions Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

Revision history : Much of the introductory and exposition text of this document comes from the preliminary report. However, all the results of this document have been recomputed for the five new projects. Also new in this report are the rule minimization results.

Contents

1	Introduction: We don't need another hero	3
2	Concept of Operations	4
3	How it Works	6
3.1	Tokenization	6
3.2	Stop lists	6
3.3	Stemming	6
3.4	Tf*IDF	7
3.5	InfoGain	7
3.6	Rule Learning	7
3.7	Assessing the Results	8
4	Results	8
4.1	Data	8
4.2	Stopping and Stemming	9
4.3	Tf*Idf	9
4.4	Learning	9
5	Discussion	11

List of Figures

1	Heroic successes with PITS	3
2	Severities for robotic missions.	4
3	Severities for human-rated missions.	4
4	Reviewing severity levels	5
5	Tokenization	6
6	Stop words	6
7	Applying a stop-list.	6
8	Some stemming rules.	6
9	Using a downloaded stemmer.	6
10	Preparation.	7
11	tfidf.awk.	7
12	Finding the 100 highest Tf*Idf words	7
13	Sample 10-way classification results.	8
14	Some precision, recall, f-measures from Figure 13.	8
15	Data sets in the this study.	8
16	Effects of stopping and stemming.	9
17	Tf*Idf scoring for the stopped, stemmed tokens	9
18	Top 1 to 50 terms found by TF*IDF, sorted by infogain.	9
19	Top 51 to 100 terms found by TF*IDF, sorted by infogain.	10
20	Re-writing issue reports as frequency counts.	10
21	Frequency counts in the data	10
22	Data set "a"; top 100 tokens; learned rules.	12
23	Data set "a"; top 3 tokens; learned rules.	12
24	Data set "b"; top 100 tokens; learned rules.	12
25	Data set "b"; top 3 tokens; learned rules.	12
26	Data set "c"; top 100 tokens; learned rules.	12
27	Data set "c"; top 3 tokens; learned rules.	13
28	Data set "d"; top 100 tokens; learned rules.	13
29	Data set "d"; top 3 tokens; learned rules.	13
30	Data set "e"; top 100 tokens; learned rules.	13
31	Data set "e"; top 3 tokens; learned rules.	13

1 Introduction: We don't need another hero



NASA's software IV&V Program captures all of its findings in a database called the Project and Issue Tracking System (PITS). The data in PITS has been collected for more than 10 years and includes issues on robotic satellite missions and human-rated systems.

It is difficult, to say the least, to generate conclusions from a moving target like PITS. Several heroic studies have made significant conclusions using PITS data (see Figure 1). These studies were heroic in the sense that the "heroes" reached their goals after tedious and complex struggling. Worse, the extracted data was only accessible with the help of NASA civil servants- a scarce and expensive resource.

The problem with PITS is that there is a lack of consistency in how each of the projects collected issue data. In virtually all instances, the specific configuration of the information captured about an issue was tailored by the IV&V project to meet its needs. This has created consistency problems when metrics data is pulled across projects. While there was a set of required data fields, the majorities of those fields do not provide information in regards to the quality of the issue and are not very suitable for comparing projects.

NASA is very aware of the problems with PITS and is taking active steps to improve it. At the time of this writing, there is an on-going effort to implement a mandatory data set in each IV&V project database to support IV&V effectiveness metrics. This effort has been in development for about a year and is currently being executed by several projects. However, it is too early to make any useful observations from that data.

- Ken Costello (IV&V's chief engineer) compiled statistics for NASA headquarters that showing, in nine IV&V tasks, the majority of issues found by IV&V were found via an analysis of requirements documents.
- Marcus Fisher (IV&V's research lead) applied a "mid-course correction" to one IV&V project after checking the progress of the IV&V against historical records in PITS.
- David Raffo (University of Portland), working with Ken Costello and other civil servants, found enough cost data to partially tune his waterfall-based model of IV&V;
- In a prior report in this project, Melissa Northey (Project Manager) performed some joins across PITS to return costs for different IV&V tasks;

Fig. 1 A partial list of past heroic successes with PITS.

To be fair, PITS is hardly unique. Based on my experience with data mining at other corporations, I assert that PITS is a typical database, useful for storing day-to-day information and generating small-scale tactical reports (e.g. "list the bugs we found last Tuesday"), but difficult to use for high-end business strategic analysis (e.g.. "in the past, what methods have proved most cost effective in finding bugs?"). Like many other databases, it takes heroes to extract information from PITS. Sadly, most of the heroes I know are so busy saving their own part of the world that they have little time to save researchers like me.

Hence, in this report, we try a new approach for extracting general conclusions from PITS data. Unlike previous heroic efforts, our text mining and machine learning methods are low cost, automatic, and rapid. We find we can build an agent to automatically review issue reports and alert when a proposed severity is anomalous. Better, the way we generated the agent means that we have probabilities that the agent is correct. These probabilities can be used to intelligently guide decision making.

An extremely surprising conclusion from this report is that the unstructured text might be a better candidate for generating lessons learned than the structured data base fields.

- While the database fields in PITS keep changing, the nature of the unstructured text remains constant.
- In other words, the reason it is so hard in the past to reason about PITS is that we have been looking at the wrong data.

If we could properly understand unstructured text, this would be a result of tremendous practical importance. A recent study¹ concluded that

- 80 percent of business is conducted on unstructured information;
- 85 percent of all data stored is held in an unstructured format (e.g. the unstructured text descriptions of issues found in PITS);
- Unstructured data doubles every three months;

That is, if we can tame the text mining problem, it would be possible to reason and learn from a much wider range of NASA data than ever before.

¹ <http://www.b-eye-network.com/view/2098>

2 Concept of Operations

NASA uses a five-point scale to score issue severity. The scale ranges one to five, worst to dullest, respectively. A different scale is used for robotic and human-rated missions (see Figure 2 and Figure 3). The data used in this report comes from robotic missions.

Using text mining and machine learning methods, this report shows that it is possible to automatically generate a *review agent* from PITS issue reports via the process of Figure 4. This agent can check the validity of the severity levels assigned to issues:

- After seeing an issue in some *artifact*, a human *analyst* generates some text *notes* and assigned a severity level *severityX*.
- An agent learns a predictor for issue severity level from logs of $\{notes, severityX\}$. A *training* module (a) updates the agent beliefs and (b) determines how much *self-confidence* a *supervisor* might have in the *agent's* conclusions.
- Using the learned knowledge, the *agent* reviews the *analysts's* text and generates its own *severityY* level.
- If the *agent's* proposed *severityY* differs from the *severityX* level of the human *analyst*, then a human *supervisor* can decide to review the human *analyst's severityX*. To help in that process, the *supervisor* can review the *self-confidence* information to decide if they trust the *agent's* recommendations.

This agent would be of useful under the following circumstances:

- When a less-experienced test engineer has assigned the wrong severity levels.
- When experienced test engineers are operating under urgent time pressure demands, they could use the agent to automatically and quickly audit their conclusions.
- For agents that can detect severity one and two-level errors with high probability, the agent could check for the rare, but extremely dangerous case, that an IV&V team has missed a high-severity problem.

Severity 1: Prevent the accomplishment of an essential capability; or jeopardize safety, security, or other requirement designated critical.

Severity 2: Adversely affect the accomplishment of an essential capability and no work-around solution is known ; or adversely affect technical, cost or schedule risks to the project or life cycle support of the system, and no work-around solution is known.

Severity 3: Adversely affect the accomplishment of an essential capability but a work-around solution is known; or adversely affect technical, cost, or schedule risks to the project or life cycle support of the system, but a work-around solution is known.

Severity 4: Results in user/operator inconvenience but does not affect a required operational or mission essential capability; or results in inconvenience for development or maintenance personnel, but does not affect the accomplishment of these responsibilities.

Severity 5: Any other issues.

Fig. 2 Severities for robotic missions.

Severity 1: A failure which could result in the loss of the human-rated system, the loss of flight or ground personnel, or a permanently disabling personnel injury.

Severity 1N: A failure which would otherwise be Severity 1 but where an established mission procedure precludes any operational scenario in which the problem might occur, or the number of detectable failures necessary to result in the problem exceeds requirements.

Severity 2: A failure which could result in loss of critical mission support capability.

Severity 2N: A failure which would otherwise be Severity 2 but where an established mission procedure precludes any operational scenario in which the problem might occur or the number of detectable failures necessary to result in the problem exceeds requirements.

Severity 3: A failure which is perceivable by an operator and is neither Severity 1 nor 2.

Severity 4: A failure which is not perceivable by an operator and is neither Severity 1 nor 2.

Severity 5: A problem which is not a failure but needs to be corrected such as standards violations or maintenance issues.

Fig. 3 Severities for human-rated missions.

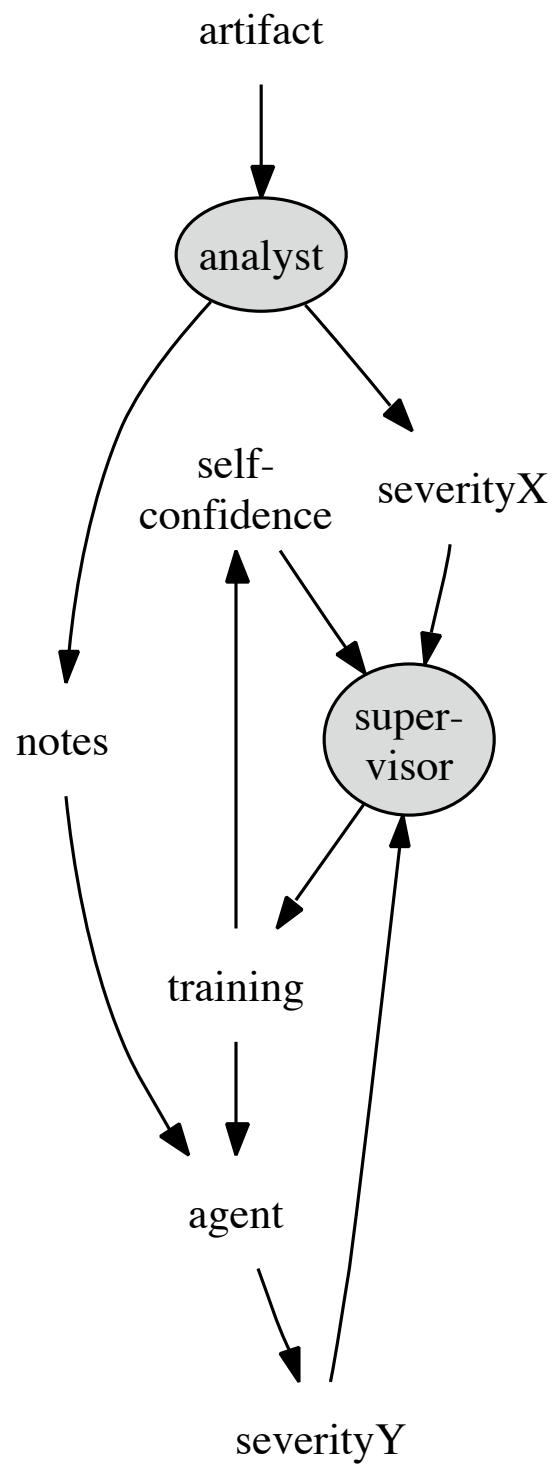


Fig. 4 An agent for reviewing issue severity levels. Gray nodes denote humans.

3 How it Works

The essential problem of text mining is *dimensionality reduction*. Standard machine learners work well for instances that are nearly all fully described using dozens (or fewer) attributes [6]. But text mining applications (e.g. analyzing PITS detect reports) must process thousands of unique words, and any particular paragraph may only mention a few of them [1, 5]. Therefore, before we can apply machine learning to text mining, we have to reduce the number of dimensions (i.e. attributes) in the problem.

There are several standard methods for dimensionality reduction such as *tokenization*, *stop lists*, *stemming*, *Tf*Idf* and *InfoGain*. All these methods are discussed below.

3.1 Tokenization

Figure 5 shows the tokenizer used in this study:

- Clean replaces certain punctuation with blank spaces.
- The file `bad.txt` contains some non-printable escape characters which are removed from the issue reports.
- Finally, `lowerCase` sends all text to lower case.

3.2 Stop lists

Another way to reduce dimensionality is to remove “dull” words via a *stop list* of “dull” words. Figure 6 shows a sample of the stop list used in this study. IV&V’s chief engineer, Ken Costello, reviewed this list and removed “counting words” such as “one”, “every”, etc, arguing that “reasoning about number of events could be an important requirement”. Figure 7 shows code for a stop-list function. Note that our code supports use a *keep list* of words we want to retain (but, in this study, the keep list was empty).

3.3 Stemming

Terms with a common stem will usually have similar meanings. For example, all these words relate to the same concept.

```
clean() {
    gawk '{gsub(Bad,""); print $0}' \
    Bad='cat $Here/bad.txt' $1 |
    sed 's/[\\\".\*\/\\\(\)\{\}\[\]]>\(\)/ /g'
}
lowerCase() {
    tr A-Z a-z $1 ;
}
```

Fig. 5 Tokenization

a	about	across	again	against
almost	alone	along	already	also
although	always	am	among	amongst
amongst	amount	an	and	another
any	anyhow	anyone	anything	anyway
anywhere	are	around	as	at
...

Fig. 6 24 of the 262 stop words used in this study.

```
stops() { gawk '
NR==1 {
    while (getline < Stops) Stop[$0] = 1;
    while (getline < Keeps) Keep[$0] = 1;
}
{ for (I=1;I<=NF;I++)
  if (Stop[$I] && ! Keep[$I])
    $I=" "
  print $0
}' \
  Stops="$Here/stop_words.txt" \
  Keeps="$Here/keep_words.txt" \
  $1
}
```

Fig. 7 Applying a stop-list.

RULE	EXAMPLE
ATIONAL -> ATE	relational -> relate
TIONAL -> TION	conditional -> condition
	rational -> rational
ENCI -> ENCE	valenci -> valence
ANCI -> ANCE	hesitanci -> hesitance
IZER -> IZE	digitizer -> digitize
ABLI -> ABLE	conformabli -> conformable
ALLI -> AL	radicalli -> radical
ENTLI -> ENT	differentli -> different
ELI -> E	vileli -> vile
OUSLI -> OUS	analogousli -> analogous
IZATION -> IZE	vietnamization -> vietnamize
ATION -> ATE	predication -> predicate
ATOR -> ATE	operator -> operate
ALISM -> AL	feudalism -> feudal
IVENESS -> IVE	decisiveness -> decisive
FULNESS -> FUL	hopefulness -> hopeful
OUSNESS -> OUS	callousness -> callous
ALITI -> AL	formaliti -> formal
IVITI -> IVE	sensitiviti -> sensitive
BILITI -> BLE	sensibiliti -> sensible

Fig. 8 Some stemming rules.

```
stemming() { perl $Here/stemming.pl $1 ; }
```

Fig. 9 Using a downloaded stemmer.

- CONNECT
- CONNECTED
- CONNECTING
- CONNECTION
- CONNECTIONS

Porter’s stemming algorithm [4] is the standard stemming tool. It repeatedly replies a set of pruning rules to the end of words until the surviving words are unchanged. The pruning rules ignore the semantics of a word and just perform syntactic pruning (e.g. Figure 8).

Porter’s stemming algorithm has been coded in any number of languages² such as the Perl *stemming.pl* used in this study (see Figure 9).

Stemming is the end of our pre-processing (a sequence that began with the `clean` function shown above). Recall that the complete sequence was

$$clean \rightarrow lowerCase \rightarrow stops \rightarrow stems$$

This full sequence is shown in Figure 10.

² <http://www.tartarus.org/martin/PorterStemmer>

```

selectColumns() {
  gawk -F, ' $3 {OFS=","; print $4 ", " $3}' -
}
cleans() {
  for i in $Files; do
    (cd $Dir; clean tab${i}5.csv 2> /dev/null |
     lowerCase |
     stops |
     stemming |
     selectColumns > $Temp/ok_${i}
    )
  done
}
Files="a b c d e"
Dir=$Root/mine/trunk/doc/07/telling/data/raw
cleans

```

Fig. 10 Preparation.

```

#update counters for all words in the record
function train() {
  Documents++;
  for(I=1;I<NF;I++) {
    if( ++In[$I,Documents]==1)
      Document[$I]++
    Word[$I]++
    Words++
  }
}
# computer tfidf for one word
function tfidf(i) {
  return Word[i]/Words*log(Documents/Document[i])
}

```

Fig. 11 tfidf.awk.

```

tfidf() {
  gawk -f tfidf.awk --source '
  { train() }
  END { OFS=","; for(I in Word) print I, tfidf(I) } ' $1 ;
  } ' $1
}
tfidf | sort -t, -n +0 | tail -100

```

Fig. 12 Finding the 100 highest Tf*Idf words using the *tfidf.awk* code of Figure 11.

3.4 Tf*IDF

Tf*Idf is shorthand for “term frequency times inverse document frequency”. This calculation models the intuition that jargon usually contains technical words that appear a lot, but only in a small number of paragraphs. For example, in a document describing a space craft, the terminology relating to the power supply may be appear frequently in the sections relating to power, but nowhere else in the document.

Calculating Tf*Idf is a relatively simple matter. If there be $Words$ number of document and each word I appear $Word[I]$ number of times inside a set of $Documents$ and if $Document[I]$ be the documents containing I , then:

$$Tf*Id = Word[i]/Words*log(Documents/Document[i])$$

The standard way to use this measure is to cull all but the k top Tf*Idf ranked stopped, stemmed tokens. This study used $k = 100$ (see Figure 11 and Figure 12).

3.5 InfoGain

According to the *InfoGain* measure, the *best* words are those that *most simplifies* the target concept (in our case, the distri-

bution of severities). Concept “simplicity” is measured using information theory. Suppose a data set has 80% severity=5 issues and 20% severity=1 issues. Then that data set has a class distribution C_0 with classes $c(1) = severity5$ and $c(2) = severity1$ with frequencies $n(1) = 0.8$ and $n(2) = 0.2$. The number of bits required to encode an arbitrary class distribution C_0 is $H(C_0)$ defined as follows:

$$\left. \begin{aligned} N &= \sum_{c \in C} n(c) \\ p(c) &= n(c)/N \\ H(C) &= -\sum_{c \in C} p(c) \log_2 p(c) \end{aligned} \right\} \quad (1)$$

If A is a set of attributes, the number of bits required to encode a class after observing an attribute is:

$$H(C|A) = -\sum_{a \in A} p(a) \sum_{c \in C} p(c|a) \log_2(p(c|a))$$

The highest ranked attribute A_i is the one with the largest *information gain*; i.e the one that most reduces the encoding required for the data *after* using that attribute; i.e.

$$InfoGain(A_i) = H(C) - H(C|A_i) \quad (2)$$

where $H(C)$ comes from Equation 1. In this study, we will use InfoGain to find the top $N \in \{100, 50, 25, 12, 6, 3\}$ most informative tokens.

3.6 Rule Learning

A data miner was then called to learn rules that predict for the severity attribute using the terms found above. The learner used here was a JAVA version of Cohen’s RIPPER rule learner [2, 7]. RIPPER is useful for generating very small rule sets. The generated rules are of the form *if* → *then*:

$$\underbrace{Feature_1 = Value_1 \wedge Feature_2 = Value_2 \wedge \dots}_{condition} \rightarrow \underbrace{Class}_{conclusion}$$

RIPPER, is a *covering* algorithm that runs over the data in multiple passes. Rule covering algorithms learns one rule at each pass for the *majority class*. All the examples that satisfy the conditions are marked as *covered* and removed from the data set. The algorithm then recurses on the remaining data. The output of a rule covering algorithm is an ordered *decision list* of rules where $rule_j$ is only tested if all conditions in $rule_{i < j}$ fail.

One way to visualize a covering algorithm is to imagine the data as a table on a piece of paper. If there exists a clear pattern between the features and the class, define that pattern as a rule and cross out all the rows covered by that rule. As covering recursively explores the remaining data, it keeps splitting the data into:

- what is easiest to explain, and
- any remaining ambiguity that requires a more detailed analysis.

a	b	c	d	<-- classified as
321	12	21	0	a = 1
157	41	8	1	b = 2
49	3	259	0	c = 3
21	1	2	2	d = 4

Fig. 13 Sample 10-way classification results.

3.7 Assessing the Results

It is a methodological error to assess the rules learned from a data miner using the data used in training. Such a *self-test* can lead to an over-estimate of the value of that model.

Cross-validation, on the other hand, assesses a learned model using data *not* used to generate it. The data is divided into, say, 10 buckets. Each bucket is set aside as a test set and a model is learned from the remaining data. This learned model is then assessed using the test set. Such cross-validation studies are the preferred evaluation method when the goal is to produce predictors intended to predict future events [7].

Mean results from a 10-way cross-validation can be assessed via a *confusion matrix* such as Figure 13. In that figure, some rule learner has generated predictions for classes {a,b,c,d} which denote issues of severity {1,2,3,4} (respectively). As shown top left of this matrix, the rules correctly classified issue reports of severity=1 as severity=1 321 times (mean results in 10-way cross-val). However, some severity=1 issues were incorrectly classified as severity=2 and severity=3 in 12 and 21 cases (respectively).

A confusion matrices can be summarized as follows. Let {A, B, C, D} denote the true negatives, false negatives, false positives, and true positives (respectively). When predicting for class “a”, then for Figure 13:

- A are all the examples where issues of severity=1 were classified as severity=1; i.e. A=321.
- B are all the examples where lower severity issues were classified as severity=1; i.e. B=157+49+21;
- C are all the examples where severity=1 issues were classified as something else; i.e. C=21+12
- D are the remaining examples; i.e. D=41+8+1=2+259+0+1+2+2.

A, B, C, D can be combined in many ways. Recall (or *pd*) comments on how much of the target was found.

$$pd = recall = D / (B + D) \tag{3}$$

Precision (or *prec*) comments on how many of the instances that triggered the detector actually containing the target concept.

$$prec = precision = D / (D + C) \tag{4}$$

The *f-measure* is the harmonic mean of precision and recall. It has the property that if *either* precision or recall is low, then the *f-measure* is decreased. The *f-measure* is useful for dual assessments that include *both* precision and recall.

$$f\text{-measure} = \frac{2 \cdot prec \cdot pd}{prec + pd} \tag{5}$$

<i>precision</i>	<i>pd = recall</i>	<i>f-measure</i>	severity
0.893	0.833	0.862	1
0.586	0.907	0.712	2
0.719	0.198	0.311	3
0.667	0.077	0.138	4

Fig. 14 Some precision, recall, f-measures from Figure 13.

Note that all these measures fall in the range

$$0 \leq \{pd, prec, f\} \leq 1$$

Also, the *larger* these values, the better the model. Figure 14 shows the *precision*, *recall*, and *f-measure* values for Figure 13.

4 Results

4.1 Data

The above methods where applied to {a,b,c,d,e}, five anonymous PITS projects supplied by Ken Costello (see Figure 15). All these systems were robotic. Note that this data has no severity one issues (these are quite rare and few severity five issues (these often not reported since they have such a low priority).

data set	severity	number
a	1	0
	2	311
	3	356
	4	208
	5	26
b	1	0
	2	23
	3	523
	4	382
	5	59
c	1	0
	2	0
	3	132
	4	180
	5	7
d	1	0
	2	1
	3	167
	4	13
	5	1
e	1	0
	2	24
	3	517
	4	243
	5	41

Fig. 15 Data sets in the this study.

4.2 Stopping and Stemming

Figure 16 shows some disappointing results for stopping and stemming. In these data sets, stopping and stemming methods barely reduced the number of tokens.

4.3 Tf*Idf

Tf*Idf proved to be more powerful than stopping or stemming. Figure 17 shows that in all data sets, there exist a very small number of words with high Tf*Idf scores. These top 100 terms are shown in Figure 18 and Figure 19. We have shown these lists to domain experts but, to date, we have not found any particular domain insights from these words. However, as shown below, even if we don't understand *why* these terms were chosen, they can be used very effectively for the task of predicting issue severity.

4.4 Learning

The issue reports for each data set were then rewritten as frequency counts for those top 100 tokens (with the severity value for each record written to the end of line- see Figure 20). As shown in Figure 21 the resulting data sets are quite sparse. This figure shows the distributions of the frequency counts of the cells in the data sets. Note that most cells have a zero frequency count; 10% of the cells have a frequency count of one, and frequency counts higher than 10 occur in only $\frac{1}{100}$ % of cells, or less.

Figure 22 shows the rules and confusion matrix see when learning from the top 100 tokens of data set "a". This rule

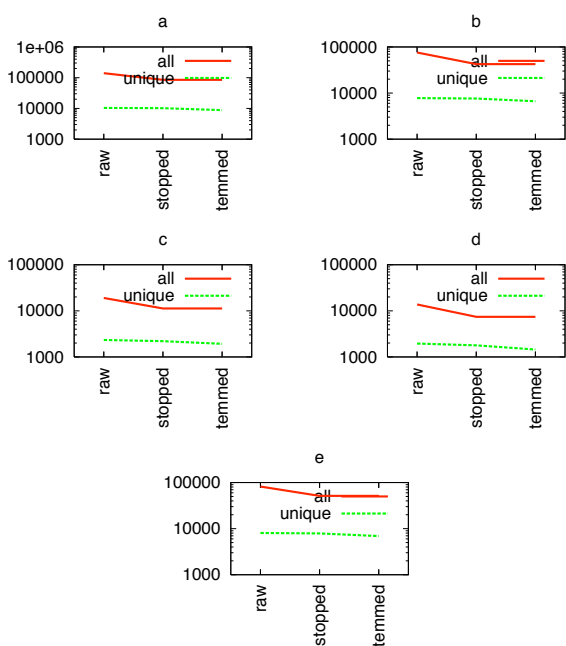


Fig. 16 Effects of stopping and stemming.

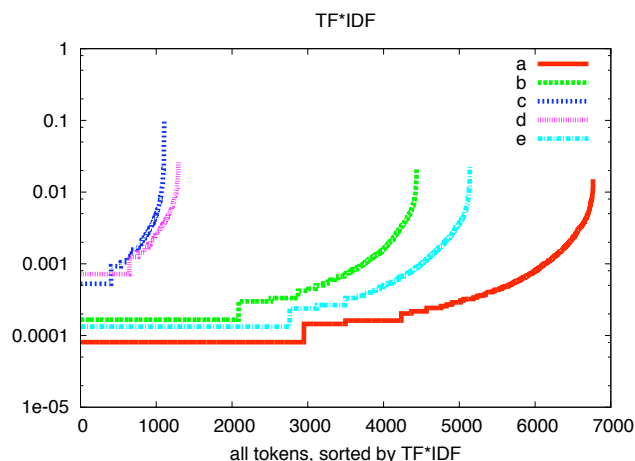


Fig. 17 Tf*Idf scoring for the stopped, stemmed tokens. Note that most tokens can be ignored since they have very low Tf*Idf scores.

rank	data set				
	a	b	c	d	e
1	rvm	fsw	softwar	switch	convert
2	sr	declar	fsw	default	the
3	script	requir	specifi	statement	capabl
4	engentr	arra	command	contain	state
5	set	sr	parent	case	interfac
6	differ	parent	sc	trace	control
7	cdh	comment	trace	code	word
8	l4	us	ground	line	declar
9	indic	verifi	perform	violat	variabl
10	verifi	step	section	comment	line
11	section	gce	spec	detail	fsw
12	link	ac	matrix	avion	inst5
13	flight	scenario	cdh	spacecraft	document
14	paramet	defin	initi	appear	conduct
15	state	valid	who/what	would	septemb
16	obc	base	pip	data	set
17	system	command	child	downward	hardwar
18	onli	valu	tim	fsw	artifact
19	spacecraft	els	s919-er2342	defin	stp
20	all	test	icd	presum	condit
21	trace	includ	mu	pixel	compon
22	check	onli	spacecraft	mask	icd
23	vm	complet	verif	spec	sr
24	softwar	state	glori	process	version
25	bootload	control	comm	fpa	check
26	capabl	all	traceabl	packet	releas
27	number	page	data	on	flight
28	vml	alloc	configur	collater	statu
29	sequenc	caus	card	scienc	implement
30	l3	verif	channel	sc	current
31	specif	flag	artifact	oper	number
32	issu	inform	downlink	logic	specifi
33	oper	fail	valid	mode	power
34	messag	trace	mechan	tabl	l4
35	support	detail	satellit	document	rqmt
36	defin	read	system	initi	list
37	fp	pse	oper	ffi	macro
38	address	ivv	includ	collect	unsign
39	code	function	instrument	onli	messag
40	uplink	condit	launch	column	assign
41	document	interfac	possibl	black	fault
42	command	on	adac	within	short
43	task	tabl	scienc	zero	test
44	rt	thruster	electr	and/or	rev
45	note	document	tp	point	time
46	monitor	initi	mode	apertur	design
47	ground	true	safehold	support	mode
48	accept	the	note	fault	jpl
49	load	in	_vbuf	exist	clear
50	initi	fprintfsetup	common2/includ/vbufh	/line	data

Fig. 18 Top 1 to 50 terms found by TF*IDF, sorted by infogain.

rank	data set				
	a	b	c	d	e
51	calcul	correct	'common2 hdlclite hdlclitec'	cadenc	oper
52	attitud	rate	struct	dure	int
53	telemetri	two	refer	all	paramet
54	fsw	end	list	store	tefsw
55	within	reset	store	csc	rafsw
56	point	req	packet	momentum	read
57	dure	telemetri	all	fine	all
58	log	packet	telemetri	target	inst6
59	packet	address	point	kav	refer
60	receiv	buffer	ap	protect	inst9
61	rate	counter	syspciinbyt	second	function
62	fault	list	interfac	autonom	reset
63	event	uint32	'common2 gener tlm.cmdc'	capabl	inst4
64	reset	rvtm	'glori cdh comm genporte'	level	command
65	process	error	'common2 gener com.cmdc'	note	configur
66	mode	level	detail	manag	enabl
67	refer	issu	rate	ground	dl
68	memori	note	em_map-pageunsign	command	us
69	engin	can	common2/includ/emsfunc	ccd	code
70	error	data	int	implic	trace
71	data	plan	collect	bin	long
72	second	sdn	'common2 router routerc'	long	flow
74	enabl	paramet	subsystem	appar	discret
75	perform	number	valu	perform	chart
76	ac	algorithm	'common2 vbuf vbufvc'	short	c
77	transit	specifi	short	flight	b
78	includ	execut	compar	engin	gener
79	design	exampl	control	field	posit
80	time	void	capabl	set	spacecraft
81	execut	time	hlite_freestruct	hardwar	section
82	arrai	line	us	valu	m
83	specifi	current	soh	softwar	defin
84	control	set	float	implement	case
85	respons	review	char	enter	fail
86	current	indic	vbuf_freestruct	respect	call
87	checksum	case	power	could	initi
88	interrupt	variabl	unsign	nim	l5
89	power	specif	accuraci	smear	actuat
90	case	chang	commun	us	descript
91	tabl	section	asec	fg	tabl
92	singl	code	document	signal	subsystem
93	list[statement	'common2 gener mm_utilc'	segment	valu
94	dump	instanc	long	fulli	softwar
95	us	updat	specif	error	issu
96	valu	procedur	compon	safe	error
97	scrub	need	orbit	design	switch
98	safe	check	ignor	requir	level
99	procedur	softwar	.hlite.cntl.blk	check	requir
100	word	charact	common2/includ/hl_protoh	period	temperatur

Fig. 19 Top 51 to 100 terms found by TF*IDF, sorted by infogain.

```

NR ==1 {
# grab the words we want to count
while (getline < "top100") Want[$0] = 1;
# write the header
for(I in Want)
    printf("%s,",I);
print "severity"
}
NR > 1 { # rewrite each record as counts of "Want"
    gsub(/ /,"",$2); counts[$1,Want, $2)
}
function counts(str,want,klass, sum,out,i,j,n,tmp,got) {
    n=split(str,tmp," ");
    for(i=1;i<=n;i++)
        if (tmp[i] in want)
            got[tmp[i]]++;
    for(j in want) {
        sum += got[j]
        out = out got[j]+0 ",";
    }
    if (sum)
        print out "-" klass
}

```

Fig. 20 Re-writing issue reports as frequency counts.

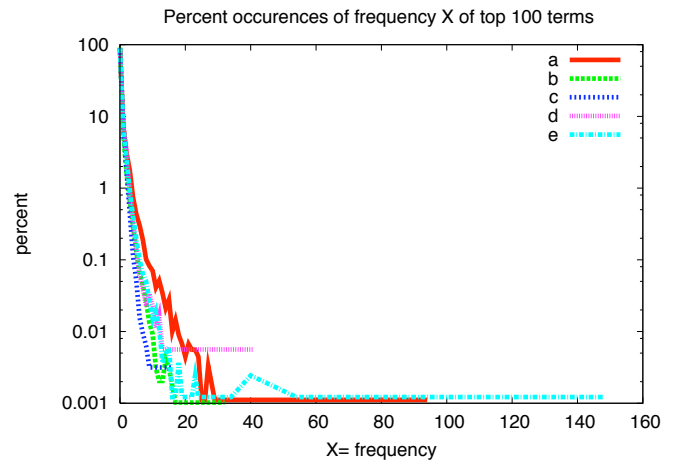


Fig. 21 Frequency counts seen in the cells of our data. Note that our data is mostly sparse: only 10% of the time (or less) were there frequency counts at or over 1. In most data sets, frequency counts of 0 (i.e. empty cell) appeared in 90% of the cells..

set is a little hard to read so Figure 23 shows the same experiment, but only using the top 3 ranked tokens. In those rules `sr` is a stemmed version of `srs`; i.e. systems requirements specification. Note that the rules of Figure 22 use only a subset of the 100 terms in the data set. That is, for data set “a”, there exists a handful of terms that most predict for issue severity. Similar results hold for same results repeat for data sets {b,c,d,e} (see Figure 24 to Figure 31). That is, even when learning from all 100 tokens, most of the rules use a few dozens terms or less. Even though few tokens were used, in many cases, the *f* measures are quite large:

- Data set “a”, for issues of severity=2, *f* = 78. .82%;
- Data set “a”, for issues of severity=3, *f* = 69. .71%;
- Data set “b”, for issues of severity=3, *f* = 70. .71%;
- Data set “c”, for issues of severity=3, *f* = 80. .92%;
- Data set “c”, for issues of severity=4, *f* = 86. .92%;
- Data set “d”, for issues of severity=3, *f* = 96. .98%;
- Data set “d”, for issues of severity=4, *f* = 87. .87%;
- Data set “e”, for issues of severity=3, *f* = 79. .80%;

These results are better than they might first appear:

- These results are listed in the format, e.g. of *f* = 79. .80%. and show the results from using the $N = 3 \dots N = 100$ tokens. Note how using just a vanishingly small number of tokens performed nearly as well as using a much larger number of tokens.
- Recall that these are all results from a 10-way cross-validation which usually over-estimates model error [3], That is, the real performance values are *higher* than the values shown above.

For other severities, the results are not as positive. Recalling Figure 15, none of our data sets had severity=1 errors so the absence of severity=1 results in the above list is not a concern. However, not all datasets resulted in good predictors for severity=2 errors. In all cases where this was observed, the data set had very few examples of such issues:

- Data set “b”, only has 22 records of severity=2;
- Data set “c”, has zero records of severity=2;
- Data set “d”, only has 1 record of severity=2;
- Data set “e”, only has 21 record of severity=2;

5 Discussion

Over the years, the Project Issue Tracking System (PITS) has been extensively and repeatedly modified. Prior attempts at generating generalized conclusions from PITS have required significant levels of manual, hence error-prone, processing.

Here, we show that conclusions can be reached from PITS without heroic effort. Using text mining and machine learning methods, we have shown that it is possible to automatically generate predictors for severity levels from the free text entered into PITS.

Better yet, our rules are self-certifying. Our data mining generation methods builds the rules and prints performance statistics (the confusion matrix). With those statistics, these rules support the following dialogue:

Tim wrote the problem report and he says this is a severity 5 issue. But the agent says that its a severity 3 issue with probability 83%. Hmmm... the agent seems pretty sure of itself- better get someone else to take a look at the issue.

When this work began, we thought that we were conducting a baseline text mining experiment that would serve as a (low) baseline against which we could assess more sophisticated methods. However, for data sets with more than 30 examples of high severity issues, we always found good issue predictors (with high f -measures).

Further, we did so using surprisingly little domain knowledge. In call cases where large f -measures were seen using the top 100 terms, similar f measures were seen when using 3 terms. This is a very exciting result since it speaks to the *usability* of this work. It would be a simple matter to apply these rules. E Given that a few frequency counts are enough to predict for issue severity, even a manual method would suffice.

We end this report with one caution. As seen in Figure 22 to Figure 31, the learned predictors are different for different data sets. We hence recommend adding these text mining tools to PITS and, on a regular basis, generate new rules relevant to just one project.

References

1. R. A. Baeza-Yates and B. Ribeiro-Neto, editors. *Modern Information Retrieval*. Addison-Wesley, 1999.
2. W. Cohen. Fast effective rule induction. In *ICML'95*, pages 115–123, 1995. Available on-line from <http://www.cs.cmu.edu/~wcohen/postscript/ml-95-ripper.ps>.
3. J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. Available from <http://jmlr.csail.mit.edu/papers/v7/demsar06a.html>.
4. M. Porter. An algorithm for suffix stripping. In K. S. Jones and P. Willet, editors, *Readings in Information Retrieval*, San Francisco: Morgan Kaufmann, 1997.
5. G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw Hill, 1983.
6. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.
7. I. H. Witten and E. Frank. *Data mining. 2nd edition*. Morgan Kaufmann, Los Altos, US, 2005.

```

    if CONDITION
    if (script <= 0) and (section >= 2) and (l4 >= 1) and (cdh >= 1) then SEVERITY USED / INCORRECT
else if (sr <= 1) and (issu >= 1) and (code >= 3) then 4 35.0 / 0.0
else if (sr >= 2) and (rvm >= 1) then 4 12.0 / 1.0
else if (sr >= 2) and (l4 >= 1) then 2 183.0 / 9.0
else if (within >= 2) and (state <= 0) and (system <= 0) then 2 55.0 / 1.0
else if (verifi >= 1) and (fsw >= 1) then 2 22.0 / 2.0
else if (control >= 1) and (code >= 1) and (attitud >= 4) then 2 10.0 / 1.0
else if (l3 >= 2) and (obc <= 0) and (perform <= 0) then 2 5.0 / 0.0
else if (script >= 1) and (trace >= 1) then 2 19.0 / 7.0
else if true then 2 3.0 / 0.0
else if true then 3 554.0 / 219.0

a b c d <-- classified as
321 12 21 0 | a = 3
157 41 8 1 | b = 4
49 3 259 0 | c = 2
21 1 2 2 | d = 5

```

Fig. 22 Data set “a”; top 100 tokens; learned rules.

```

    if CONDITION
    if (rvm <= 0) and (sr = 3) then SEVERITY USED / INCORRECT
else if (sr >= 2) then 4 52.0 / 21.0
else if true then 2 289.0 / 54.0
else if true then 3 557.0 / 245.0

a b c d <-- classified as
314 13 27 0 | a = 3
158 25 24 0 | b = 4
69 10 232 0 | c = 2
25 0 1 0 | d = 5

```

Fig. 23 Data set “a”; top 3 tokens; learned rules.

```

    if CONDITION
    if (arrai >= 2) and (line >= 1) and (us <= 1) then SEVERITY USED / INCORRECT
else if (base >= 4) then 2 7.0 / 0.0
else if (fsw <= 0) and (declar >= 1) then 2 3.0 / 0.0
else if (fsw <= 0) and (complet >= 1) and (section <= 0) then 4 86.0 / 26.0
else if (fsw <= 0) and (statement >= 1) and (need <= 0) and (valu <= 0) then 4 27.0 / 4.0
else if true then 4 36.0 / 10.0
else if true then 3 819.0 / 332.0

a b c d <-- classified as
120 253 0 4 | a = 4
69 445 0 7 | b = 3
11 47 0 0 | c = 5
2 9 0 11 | d = 2

```

Fig. 24 Data set “b”; top 100 tokens; learned rules.

```

    if CONDITION
    if (fsw <= 0) and (declar >= 1) then SEVERITY USED / INCORRECT
else if true then 4 94.0 / 34
else if true then 3 884.0 / 383.0

a b c d <-- classified as
60 317 0 0 | a = _4
20 501 0 0 | b = _3
3 55 0 0 | c = _5
11 11 0 0 | d = _2

```

Fig. 25 Data set “b”; top 3 tokens; learned rules.

```

    if CONDITION
    if (section >= 2) and (matrix >= 1) and (icd >= 1) then SEVERITY USED / INCORRECT
else if (softwar >= 1) then 5 7.0 / 1.0
else if (parent <= 0) and (trace <= 0) then 3 95.0 / 3.0
else if true then 3 48.0 / 14.0
else if true then 4 167.0 / 4.0

a b c <-- classified as
162 14 4 | a = _4
7 123 0 | b = _3
2 1 4 | c = _5

```

Fig. 26 Data set “c”; top 100 tokens; learned rules.

```

    if CONDITION                then SEVERITY USED / INCORRECT
    if (softwar >= 1)           then      3   95.0 /  3.0
else if true                    then      4   222.0 / 44.0

  a  b  c  <-- classified as
169 11  0 | a = _4
 37 93  0 | b = _3
  6  1  0 | c = _5

```

Fig. 27 Data set “c”; top 3 tokens; learned rules.

```

    if CONDITION                then SEVERITY USED / INCORRECT
    if (switch >= 2)           then      4    11.0 /  1.0
else if true                    then      3   167.0 / 4.0

  a  b  c  d  <-- classified as
  0  0  1  0 | a = _2
  0 10  2  0 | b = _4
  0  1 163  0 | c = _3
  0  0  1  0 | d = _5

```

Fig. 28 Data set “d”; top 100 tokens; learned rules.

```

    if CONDITION                then SEVERITY USED / INCORRECT
    if (switch >= 2)           then      4    11.0 /  1.0
else if true                    then      3   167.0 / 4.0

  a  b  c  d  <-- classified as
  0  0  1  0 | a = _2
  0 10  2  0 | b = _4
  0  1 163  0 | c = _3
  0  0  1  0 | d = _5

```

Fig. 29 Data set “d”; top 3 tokens; learned rules.

```

    if CONDITION                then SEVERITY USED / INCORRECT
    if (trace >= 1) and (test >= 1) and (case >= 3) then      2     3.0 /  0.0
else if (error >= 1) and (line >= 2)             then      2     3.0 /  0.0
else if (convent >= 2) and (declar <= 0) and (function <= 0) then      5     6.0 /  0.0
else if (case >= 2) and (sr >= 3)                 then      5     6.0 /  1.0
else if (refer >= 1) and (section >= 3) and (refer <= 1) then      5     7.0 /  2.0
else if (the >= 1) and (fsw >= 2)                 then      4    47.0 /  5.0
else if (control <= 0) and (convent >= 3)         then      4    25.0 /  3.0
else if true                                       then      3   723.0 / 214.0

  a  b  c  d  <-- classified as
  1 20  0  0 | a = _2
  3 490 3 20 | b = _3
  0 26  9  6 | c = _5
  0 167 1 74 | d = _4

```

Fig. 30 Data set “e”; top 100 tokens; learned rules.

```

    if CONDITION                then SEVERITY USED / INCORRECT
    if (the >= 1) and (convent >= 3) then      4    30.0 /  8.0
else if true                    then      3   790.0 / 275.0

  a  b  c  d  <-- classified as
  0 21  0  0 | a = _2
  0 515 0  1 | b = _3
  0 34  0  7 | c = _5
  0 222 0 20 | d = _4

```

Fig. 31 Data set “e”; top 3 tokens; learned rules.

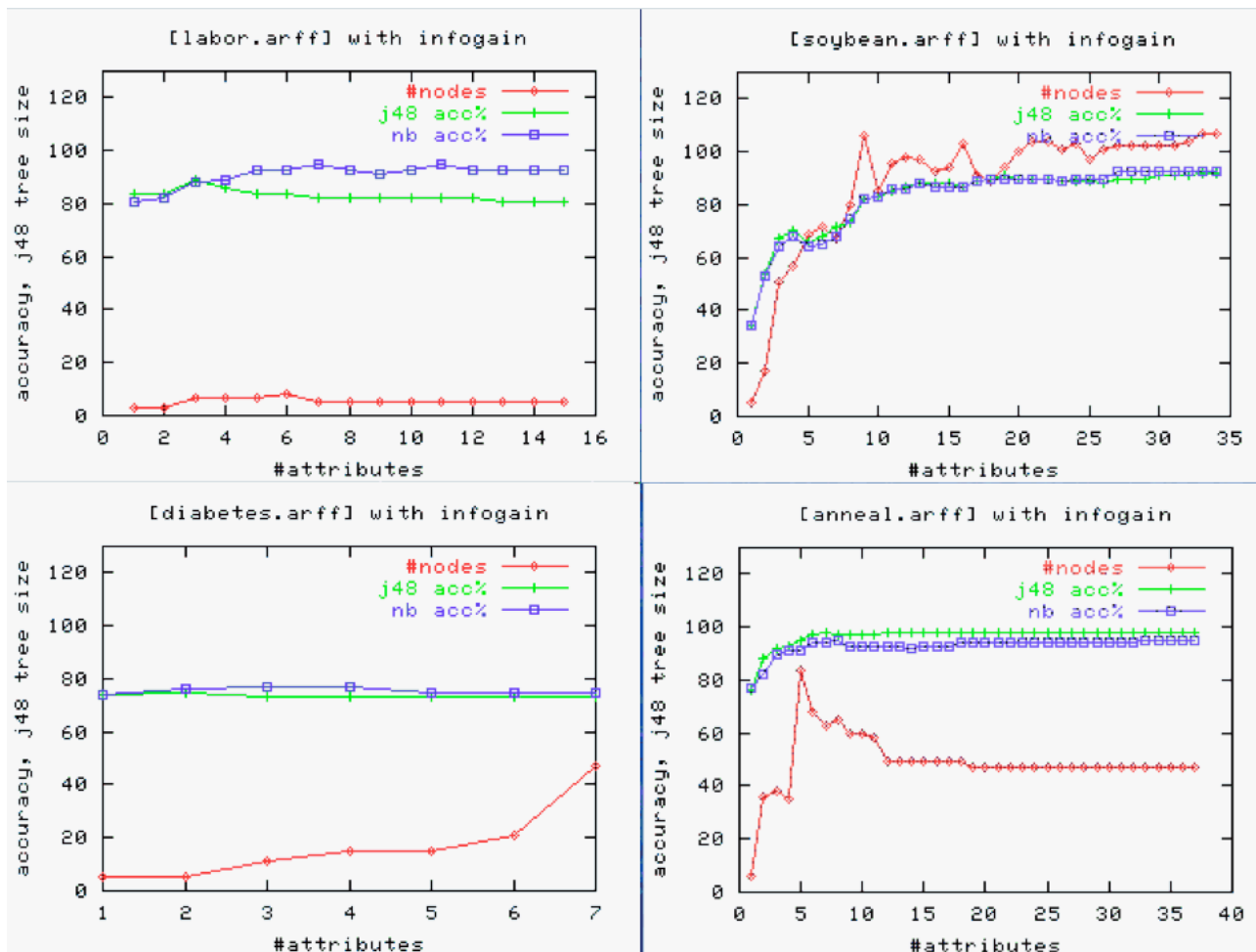
» [fss](#)

Feature Subset Selection (FSS)

- Occam's Razor - The English philosopher, William of Occam (1300-1349) propounded Occam's Razor:
 - Entia non sunt multiplicanda praeter necessitatem.
 - (Latin for "Entities should not be multiplied more than necessary"). That is, the fewer assumptions an explanation of a phenomenon depends on, the better it is.
- (BTW, Occam's razor did not survive into the 21st century.)
 - The data mining community modified it to the *Minimum Description Length* (MDL) principle.
 - MDL: the best theory is the smallest BOTH in size AND number of errors).

The case for FSS

Repeated result: throwing out features rarely damages a theory



And, sometimes, feature removal is very useful:

- E.g. linear regression on [bn.arff](#) yielded:

```
Defects =
  82.2602 * S1=L,M,VH +
  158.6082 * S1=M,VH +
  249.407 * S1=VH +
  41.0281 * S2=L,H +
```

```

68.9153 * S2=H +
151.9207 * S3=M,H +
125.4786 * S3=H +
257.8698 * S4=H,M,VL +
108.1679 * S4=VL +
134.9064 * S5=L,M +
-385.7142 * S6=H,M,VH +
115.5933 * S6=VH +
-178.9595 * S7=H,L,M,VL +
...
[ 50 lines deleted ]

```

- On a 10-way cross-validation, this correlates 0.45 from predicted to actuals.
- 10 times, take 90% of the data and run a *WRAPPER*- a best first search through combinations of attributes. At each step, linear regression was called to assess a particular combination of attributes. In those ten experiments, WRAPPER found that adding feature X to features A,B,C,... improved correlation the following number of times:

```

number of folds (%)  attribute
                2( 20 %)   1 S1
                0(  0 %)   2 S2
                2( 20 %)   3 S3
                1( 10 %)   4 S4
                0(  0 %)   5 S5
                1( 10 %)   6 S6
                6( 60 %)   7 S7      <==
                1( 10 %)   8 F1
                1( 10 %)   9 F2
                2( 20 %)  10 F3
                2( 20 %)  11 D1
                0(  0 %)  12 D2
                5( 50 %)  13 D3      <==
                0(  0 %)  14 D4
                0(  0 %)  15 T1
                1( 10 %)  16 T2
                1( 10 %)  17 T3
                1( 10 %)  18 T4
                0(  0 %)  19 P1
                1( 10 %)  20 P2
                0(  0 %)  21 P3
                1( 10 %)  22 P4
                6( 60 %)  23 P5      <==
                1( 10 %)  24 P6
                2( 20 %)  25 P7
                1( 10 %)  26 P8
                0(  0 %)  27 P9
                2( 20 %)  28 Hours
                8( 80 %)  29 KLoC   <==
                4( 40 %)  30 Language
                3( 30 %)  32 log(hours)

```

- Four variables appeared in the majority of folds. A second run did a 10-way using just those variables to yield a smaller model with (much) larger correlation (98\%):

```

Defects =
  876.3379 * S7=VL +
-292.9474 * D3=L,M +
  483.6206 * P5=M +
   5.5113 * KLoC +
   95.4278

```

Excess attributes

- Confuse decision tree learners
 - Too much early splitting of data
 - Less data available for each sub-tree
- Too many things correlated to class?
 - Dump some of them!

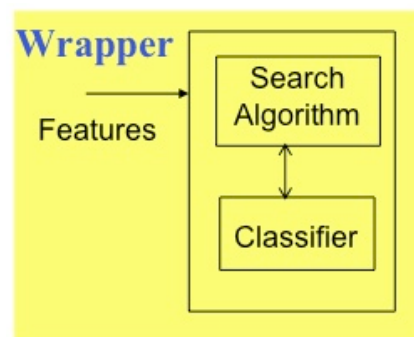
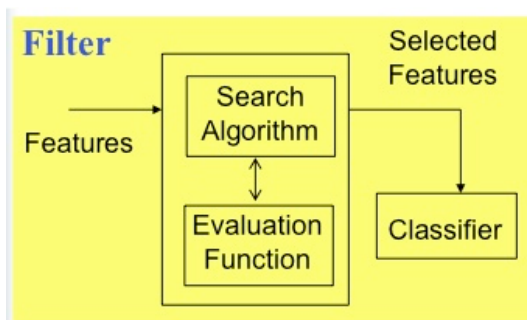
Why FSS?

- throw away noisy attributes
- throw away redundant attributes
- smaller model= better accuracies (often)
- smaller model= simpler explanation
- smaller model= less variance
- smaller model= any downstream processing will thank you

Problem

- Exploring all subsets exponential
- Need heuristic methods to cull search;
 - e.g. forward/back select
- Forward select:
 - start with empty set
 - grow via hill climbing:
 - repeat
 - try adding one thing and if that improves things
 - try again using the remaining attributes
 - until no improvement after N additions OR nothing to add
- Back select
 - as above but start with all attributes and discard, don't add
- Usually, we throw away most attributes:
 - so forward select often better
 - exception: J48 exploits interactions more than, say, NB.
 - so, possibly, back select is better when wrapping j48
 - so, possibly, forward select is as good as it gets for NB

FSS types:



- filters vs wrappers:
 - wrappers: use an actual target learners e.g. WRAPPER
 - filters: study aspects of the data e.g. the rest
 - filters are faster!
 - wrappers exploit bias of target learner so often perform better, when they terminate
 - don't terminate on large data sets
- solo vs combinations:
 - evaluate solo attributes: e.g. INFO GAIN, RELIEF
 - evaluate combinations: e.g. PCA, SVD, CFS, CBS, WRAPPER
 - solos can be faster than combinations
- supervised vs unsupervised:
 - use/ignores class values e.g. PCA/SVD is unsupervised, reset supervised
- numeric vs discrete search methods
 - ranker: for schemes that numerically score attributes e.g. RELIEF, INFO GAIN,
 - best first: for schemes that do heuristic search e.g. CBS, CFS, WRAPPER

Hall and Holmes:

This paper: pre-discretize numerics using entropy.

[Hall & Holmes.](#)

INFO GAIN

- often useful in high-dimensional problems
 - real simple to calculate
- attributes scored based on info gain: $H(C) - H(C|A)$
- Sort of like doing decision tree learning, just to one level.

RELIEF

- [Kononenko97](#)
- useful attributes differentiate between instances from other class
- randomly pick some instances (here, 250)
- find something similar, in an another class
- compute distance this one to the other one
- Stochastic sampler: scales to large data sets.
- Binary RELIEF (two class system) for "n" instances for weights on features "F"

```

set all weights W[f]=0
for i = 1 to n; do
  randomly select instance R with class C
  find nearest hit H      // closest thing of same class
  find nearest miss M    // closest thing of difference class
  for f = 1 to #features; do
    W[f] = W[f] - diff(f,R,H)/n + diff(f,R,M)/n
  done
done

```

- diff:
 - discrete differences: 0 if same 1 if not.
 - continuous: differences absolute differences
 - normalized to 0:1
 - When values are missing, see [Kononenko97](#), p4.
- N-class RELIEF: not 1 near hit/miss, but k nearest misses for each class C

$$W[f] = W[f] - \frac{\sum_{i=1..k} \text{diff}(f, R, H_i)}{(n*k)} + \sum_{C \neq \text{class}(R)} \sum_{i=1..k} \left(\frac{P(C)}{(1 - P(\text{class}(R)))} * \text{diff}(f, R, M_i(C)) / (n*k) \right)$$

The $P(C) / (1 - P(\text{class}(R)))$ expression is a normalization function that

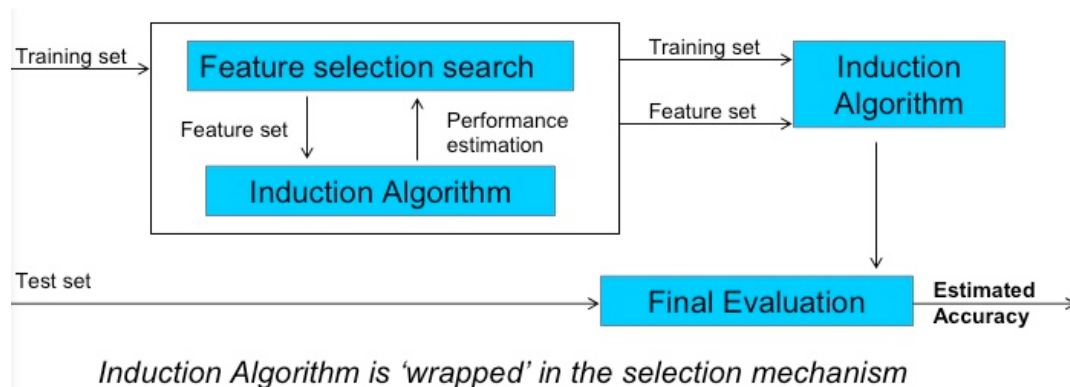
- demotes the effect of R from rare classes
- and rewards the effect of near hits from common classes.

CBS (consistency-based evaluation)

- Seek combinations of attributes that divide data containing a strong single class majority.
 - Kind of like info gain, but emphasis of single winner
- Discrete attributes
- Forward select to find subsets of attributes

WRAPPER

- Forward select attributes
 - score each combination using a 5-way cross val
- When wrapping, best to try different target learners
 - Check that we aren't over exploiting the learner's bias
 - e.g. J48 and NB

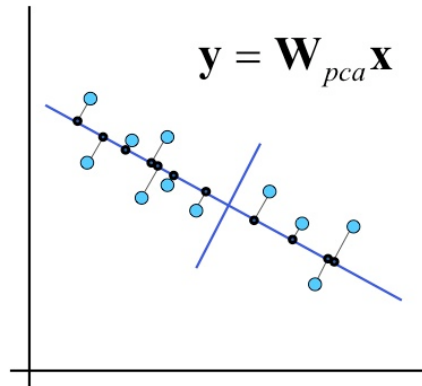


PRINCIPAL COMPONENTS ANALYSIS (PCA)

(The traditional way to do FSS.)

- Only unsupervised method studied here
- Transform dimensions
- Find covariance matrix $C[i,j]$ is the correlation i to j;
 - $C[i,i]=1$;
 - $C[i,j]=C[j,i]$
- Find eigenvectors

- Transform the original space to the eigenvectors
- Rank them by the variance in their predictions
- Report the top ranked vectors



- Makes things easier, right? Well...

```

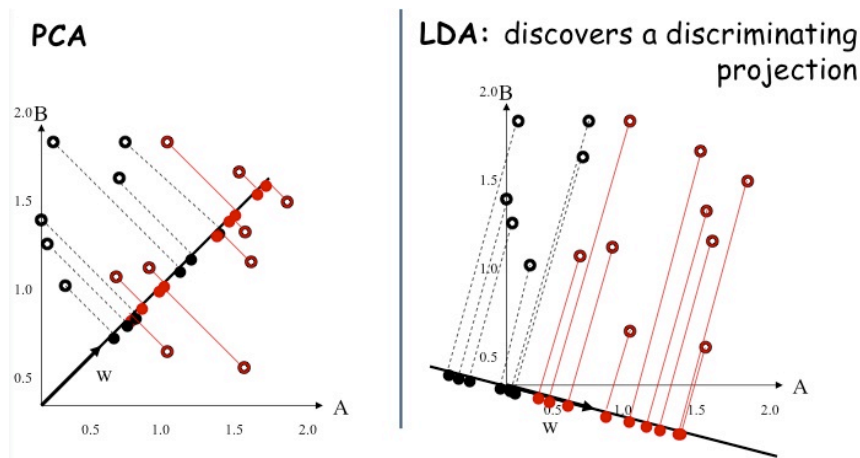
if domain1 <= 0.180
then NoDefects
else if domain1 > 0.180
  then if domain1 <= 0.371 then NoDefects
  else if domain1 > 0.371 then Defects

domain1 = 0.241 * loc      + 0.236 * v(g)
          + 0.222 * ev(g)  + 0.236 * iv(g)      + 0.241 * n
          + 0.238 * v      - 0.086 * l          + 0.199 * d
          + 0.216 * i      + 0.225 * e + 0.236 * b + 0.221 * t
          + 0.241 * lOCode + 0.179 * lOComment
          + 0.221 * lOBlank + 0.158 * lOCodeAndComment
          + 0.163 * uniqOp + 0.234 * uniqOpnd
          + 0.241 * totalOp + 0.241 * totalOpnd
          + 0.236 * branchCount
  
```

PCA vs LDA (linear discriminant analysis)

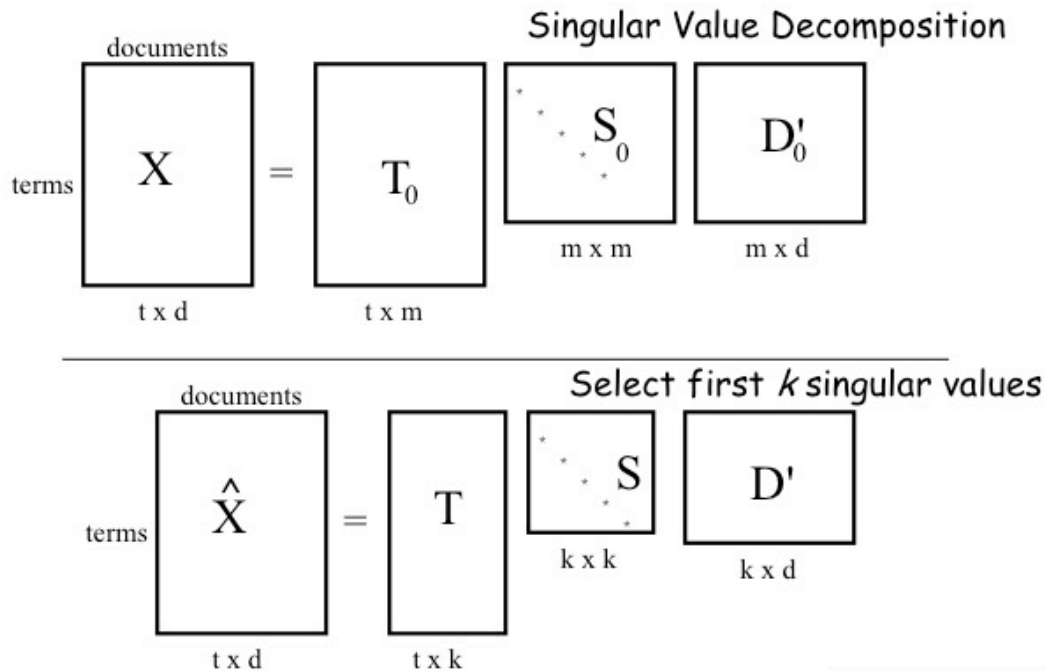
LDA = PCA + class knowledge

(Note: LDA should not be confused with [LDA \(latent Dirichlet allocation\)](#) which currently all the rage in text mining. And that LDA is not covered in this subject.)



Latent Semantic Indexing

- Performing PCA is the equivalent of performing *Singular Value Decomposition* (SVD) on the data.
- Any $n * m$ matrix X (of terms n in documents m) can be rewritten as:
 - $X = T_0 * S_0 * D_0'$
 - S_0 is a diagonal matrix scoring attributes, top to bottom, most interesting to least interesting
 - We can shrink X by dumping the duller (lower) rows of S_0



- Latent Semantic Indexing is a method for selecting informative subspaces of feature spaces.
- It was developed for information retrieval to reveal semantic information from document co-occurrences.
- Terms that did not appear in a document may still associate with a document.
- LSI derives uncorrelated index factors that might be considered artificial concepts.
- SVD easy to perform in Matlab
 - Also, there is some [C-code](#).
 - Also [Java Classes available](#)
 - class SingularValueDecomposition
 - Constructor: SingularValueDecomposition(Matrix Arg)
 - Methods: GetS(); GetU(); GetV(); (U,V correspond to T,D)
- Be careful about using these tools blindly
 - It is no harm to understand what is going on!
- The [Matrix Cookbook](#)
- Note: major win for SVD/LSI: scales very well.
 - Research possibility: text mining for software engineering
 - typically very small corpuses
 - so might we find better FSS for text mining than SVD/LSI

CFS (correlation-based feature selection)

- Scores high subsets with strong correlation to class and weak correlation to each other.

- Numerator: how predictive
- Denominator: how redundant
- FIRST ranks correlation of solo attributes
- THEN heuristic search to explore subsets

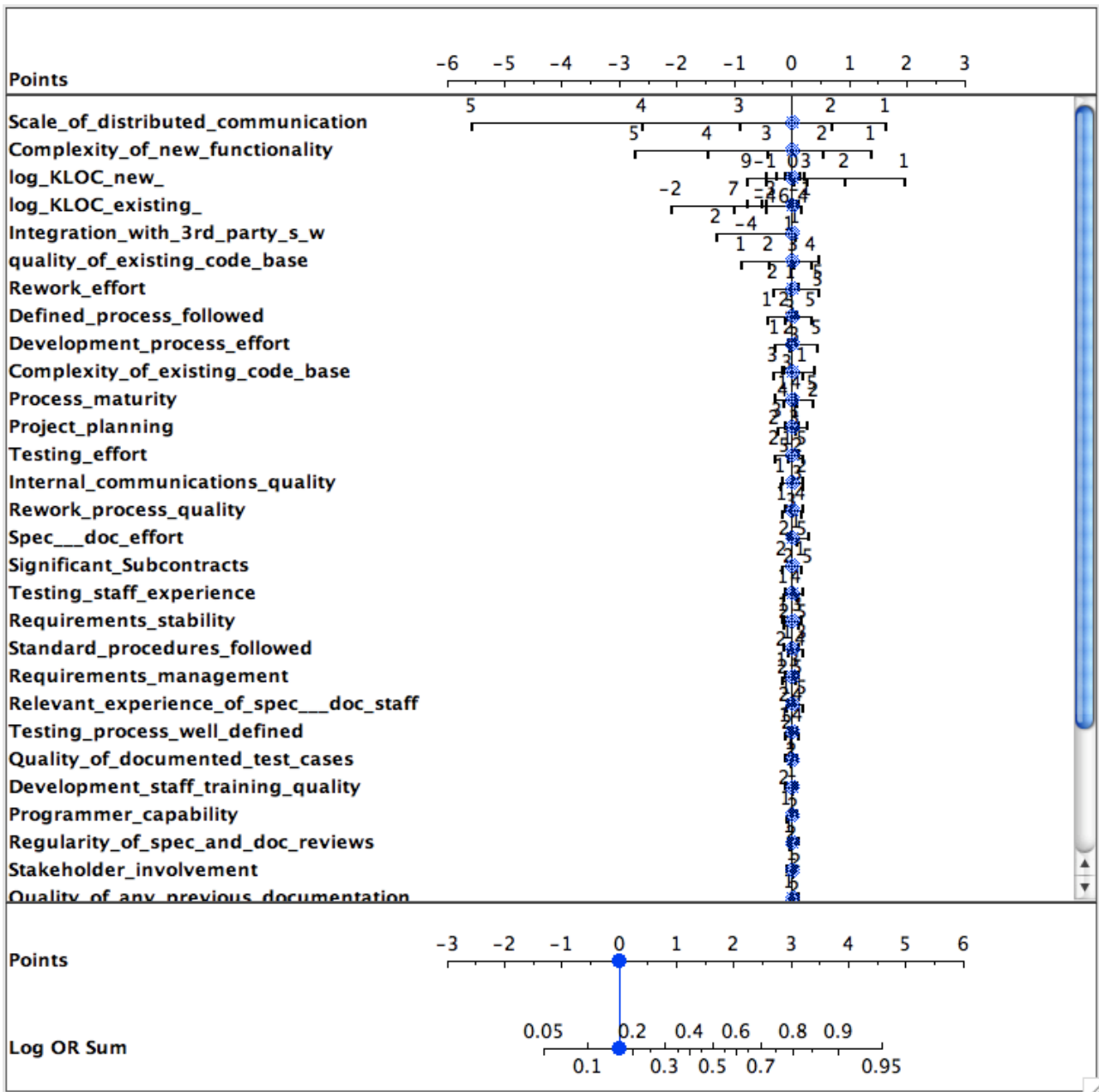
And the winner is:

- Wrapper! and it that is too slow...
- CFS, Relief are best all round performers
 - CFS selects fewer features
- Phew. Hall invented CFS

Other Methods

Other methods not explored by Hall and Holmes...

- Note: the text mining literature has yet to make such an assessment. Usually, SVD rules. But see [An Approach to Classify Software Maintenance Requests](#), from ICSM 2002, for a nice comparison of nearest neighbor, CART, Bayes classifiers, and some other information retrieval methods).
- Using random forests for feature selection of the mth variable:
 - randomly permute all values of the mth variable in the oob data
 - Put these altered oob x-values down the tree and get classifications.
 - Proceed as though computing a new internal error rate (i.e. run the classifier).
 - The amount by which this new error exceeds the original test set error is defined as the importance of the mth variable.
- Use the Nomogram scores



Exploiting the Essential Assumptions of Analogy-based Effort Estimation

Ekrem Kocaguneli, *Student Member, IEEE*, Tim Menzies, *Member, IEEE*,
Ayse Bener, *Member, IEEE*, and Jacky W. Keung, *Member, IEEE*

Abstract

Background: There are too many design options for software effort estimators. How can we best explore them all?

Aim: We seek aspects on general principles of effort estimation that can guide the design of effort estimators.

Method: We identified the essential assumption of analogy-based effort estimation: i.e. the immediate neighbors of a project offer stable conclusions about that project. We test that assumption by generating a binary tree of clusters of effort data and comparing the variance of super-trees vs smaller sub-trees.

Results: For ten data sets (from Coc81, Nasa93, Desharnais, Albrecht, ISBSG, and data from Turkish companies), we found: (a) the estimation variance of cluster sub-trees is usually *larger* than that of cluster super-trees; (b) if analogy is restricted to the cluster trees with lower variance then effort estimates have a significantly lower error (measured using MRE and a Wilcoxon test, 95% confidence, compared to nearest-neighbor methods that use neighborhoods of a fixed size).

Conclusion: Estimation by analogy can be significantly improved by a dynamic selection of nearest neighbors, using only the project data from regions with small variance.

Index Terms

Software Cost Estimation, Analogy, k -NN



1 INTRODUCTION

Software effort estimates are often wrong by a factor of four [1] or even more [2]. As a result, the allocated funds may be inadequate to develop the required project. In the worst case, over-

- Ekrem Kocaguneli is with the Department of Computer Engineering, Bogazici University. E-mail: ekrem.kocaguneli@boun.edu.tr
- Tim Menzies is with the Lane Department of Computer Science and Electrical Engineering, West Virginia University. E-mail: tim@menzies.us
- Ayse Bener is with the Department of Computer Engineering, Bogazici University. E-mail: bener@boun.edu.tr
- Jacky W. Keung is with the School of Computer Science and Engineering, University of New South Wales. E-mail: jacky.keung@nicta.com.au

This research is funded in part by Tubitak EEEAG108E014.

running projects are canceled and the entire development effort is wasted. For example:

- NASA canceled its incomplete Check-out Launch Control System project after the initial \$200M estimate was exceeded by another \$200M [3].
- The ballooning software costs of JPL's Mission Science Laboratory recently forced a two-year delay.

It is clear that we need better ways to generate project effort estimates. However, it is not clear how to do that. For example, later in this paper we document thousands of variations for analogy-based effort estimation (ABE). Effort estimation is an active area of research [4]–[7] and more variations are constantly being developed. We expect many more variations of ABE, and other effort estimation methods, to appear in the very near future.

Recent publications propose data mining toolkits for automatically exploring this very large (and growing) space of options for generating effort estimates. For example, in 2006, Auer et al. [8] propose an extensive search to learn the best weights to assign different project features. Also in that year, Menzies et al. [9]'s COSEEKMO tool explored thousands of combinations of discretizers, data pre-processors, feature subset selectors, and inductive learners. In 2007, Baker proposed an exhaustive search of all possible project features, learners, etc. He concluded that such an exhaustive search was impractical [10].

The premise of this paper is that we can do better than a COSEEKMO-style brute-force search through the space of all variants of effort estimators. Such studies are computationally intensive (the COSEEKMO experiments took two days to terminate). With the ready availability of cheap CPU farms and cloud computing, such CPU-investigations are becoming more feasible. On the other hand, datasets containing historical examples of project effort are typically small¹. In our view, it seems misdirected to spend days of CPU time just to analyze a few dozen examples. These CPU-intensive searches can generate gigabytes of data. Important general properties of the estimation process might be missed, buried in all that data. As shown below, if we exploit these aspects, we can significantly improve effort estimates.

This paper proposes an alternative to brute-force and heuristic search. According to our *easy path* principle for designing an effort predictor:

Find the situations that confuse estimation. Remove those situations.

(Later in this paper, in §3.2, we will offer a precision definition of “confuse estimation”. For now, we need only say that confused estimates are highly inaccurate).

The easy path is not standard practice. Usually, prediction systems are matured by adding mechanisms to handle the harder cases. For example, the AdaBoost algorithm generates a list

1. For example, the effort estimation datasets used in Mendes et al. [11], Auer et al. [8], Baker [10], this study, and Li et al. [12] have median size (13,15,31,33,52), respectively.

of learners, and each learner focuses on the examples that were poorly handled by the one before [13].

Focusing on just the easy cases could be problematic. If we only explore the easy cases, we could perform badly on the hard test cases. On the other hand, if the easy path works, it finds short-cuts that simplifies future effort estimation work. Also, it avoids COSEEKMO's brute-force search since, according to this principle, we only explore the options that challenge the essential assumptions of the predictor.

The rest of this paper uses the easy path to build and evaluate an effort estimator called TEAK (short for "Test Essential Assumption Knowledge"). In keeping with the easy path, we only explored design options that commented on TEAK's essential assumptions; specifically: (a) case subset selection and (b) how many training examples should be used for estimation.

TEAK's design applied the easy path in five steps:

- 1) Select a prediction system.
- 2) Identify the predictor's essential assumption(s).
- 3) Recognize when those assumption(s) are violated.
- 4) Remove those situations.
- 5) Execute the modified prediction system.

On evaluation, we found that for the data sets studied here, TEAK generated significantly *better* estimates than comparable methods.

More generally, the success of the easy path principle recommends it for future research. When designing a predictor, it is useful to *first* try optimizing for the situations where prediction is easy, *before* struggling with arcane and complex mechanisms to handle the harder situations. For example, in future work, we will apply steps 1,2,3,4,5 to other aspects of effort estimation like feature weighting, and similarity measures.

The rest of this paper is structured as follows. After a review of the general field of effort estimation, we will focus on ABE (analogy-based estimation). For ABE, we will work through the above five steps to design TEAK. TEAK's performance will then be compared against six other ABE systems. Our conclusion will be to recommend TEAK for effort estimation.

The paper uses the notation of Figure 1.

2 BACKGROUND

2.1 Scope

This paper is *not* a detailed comparison of analogy-based estimation to other estimation methods (for such a large scale comparison of many different methods, the reader is referred to [9], [14]). While we compare our proposed new technique to a limited number of other estimation methods

Symbol	Explanation
ABE	Analogy Based Estimation.
ABE0	A baseline ABE method.
NNet	A neural net prediction system with one hidden layer.
LR	Linear regression.
GAC	Greedy Agglomerative Clustering.
TEAK	Test Essential Assumption Knowledge.
GAC1, GAC2	First and second GAC trees within TEAK.
x, y	Depending on the context, x and y can refer to two instances/projects in a dataset or alternatively to two vertices in a GAC tree.
x_i, y_i	i^{th} features of projects x and y respectively.
w_i	Feature weight for the difference of features x_i and y_i in Euclidean distance function.
L_x, L_y, L_z	Leaves of the sub-trees whose roots are x, y and z respectively.
k_x, k_y, k_z	The number of leaves in L_x, L_y, L_z respectively.
k -NN	k Nearest Neighbors.
k	An italic k alone refers to analogies, i.e. selected similar projects.
b_i, b_j, c_i, c_j	All these symbols are related to discretization of continuous columns. b_i and b_j refer to break-points i and j , which in return produce discrete bins that have counts of c_i and c_j instances within themselves.
Best(K)	A procedure that heuristically finds the best k value for a dataset.
σ_x^2, σ_y^2	Assuming that x, y and z are vertices in a GAC tree and x is the parent of y and z , σ_x^2 refers to the variance of instances in x and σ_{yz}^2 refers to the weighted sum of the variances of y and z .
$\alpha, \beta, \gamma, R, \max(\sigma^2)$	These symbols are associated with different pruning policies. α, β and γ keep user-defined values to fine-tune pruning. R is a random variable that can have values from 0 to 1. $\max(\sigma^2)$ refers to maximum variance of all sub-trees in a GAC tree.
T, N	T refers to a given dataset and N refers to a test set out of this dataset.
$predicted_i$	The effort of test instance $N_i \in N$ predicted by some induced prediction system.
$actual_i$	The actual effort seen in test instance N_i in N
AR	Absolute residual. $ actual_i - predicted_i $
MRE	magnitude of relative error. $\frac{AR}{actual_i}$
PRED(X)	The percentage of estimates that are within X% of the actual value.
$win_i, tie_i, loss_i$	The total number of wins, ties and losses of a variant in comparison to other variants according to Wilcoxon signed rank test.

Fig. 1: The explanations of symbols that are used in our research are summarized here. Symbols that are related to each other are grouped together.

(specifically, neural networks and regression), that comparison is only to ensure that analogy-based estimation is not noticeably worse than other methods in widespread use.

The main point of this paper is as follows. Analogy-based effort estimation is a widely-used and widely-studied technique [8], [12], [15]–[26]. This paper reports a novel method to improve that technique. Specifically, when estimating via analogy, it is *best to first prune all subsets of the data with high variance*. We will argue that a new *variance heuristic* is a better way to select analogies:

- Without this heuristic, analogies are selected by their distance to the test instance.
- With this heuristic, a pre-processor prunes the space of possible analogies, removing the subsets of the data with high variance (in high variance subsets, training data offers highly variable conclusions).

This heuristic works, we believe, since if a test instance falls into such subsets then (by definition) minor changes to the test will lead to large changes in the prediction (due to the variability in that region). We show below that, by removing those problematic subsets, effort estimation by analogy can be improved.

2.2 Analogous Approaches

While the variance heuristic is novel and unique in the effort estimation literature, analogous proposals can be found in the requirements engineering literature, dating back to the 1990s. In the seminal paper “To Be *and* Not To Be”, Nuseibeh [27] discusses a spectrum of methods for handling inconsistent specifications. One method is *circumvent*; i.e. instead of expending effort resolving regions of contradiction, add “pollution markers” that screen the problematic regions away from the rest of the system. Note that our variance heuristic (that prunes the data subsets with high variance) is something like a pollution marker since it guides the reasoning away from problematic training data.

Another use for such marks is to mark any segments that human agents need to explore. Turning back from requirements engineering (which was the focus of Nuseibeh’s discussion) back to effort estimation (which is the focus of this paper), we could utilize pollution marks to highlight regions where more data collection might be beneficial. Note that this approach is not explored here since our premise is that we must make the best use possible of fixed data. However, this might be a promising area of future research.

2.3 Effort Estimation

Having set the context for this paper, we now turn to the details.

After Shepperd [6], we say that software project effort estimation usually uses one of three methods:

- Human-centric techniques (a.k.a. expert judgment);
- Model-based techniques including:
 - Algorithmic/parametric models such COCOMO [1], [28];
 - Induced prediction systems.

Human centric techniques are the most widely-used estimation method [29], but are problematic. If an estimate is disputed, it can be difficult to reconcile competing human intuitions (e.g.) when one estimate is generated by a manager who is senior to the other estimator. Also, Jorgensen [30] reports that humans are surprisingly poor at reflecting and improving on their expert judgments.

One alternative to expert judgment is a model-based estimate. Models are a reproducible methods for generating an estimate. This is needed for (e.g.) U.S. government software contracts that require a model-based estimate at each project milestone [9]. Such models are used to generate and audit an estimate, or to double-check a human-centric estimate.

Model-based estimates can be generated using an algorithmic/parametric approach or via induced prediction systems. In the former, an expert proposes a general model, then domain data is used to tune that model to specific projects. For example, Boehm's 1981 COCOMO model [1] hypothesized that development effort was exponential on LOC and linear on 15 *effort multipliers* such as analyst capability, product complexity, etc. Boehm defined a *local calibration* procedure to tune the COCOMO model to local data.

Induced prediction systems are useful if the available local training data does not conform to the requirements of a pre-defined algorithmic/parametric model such as COCOMO. There are many induction methods including linear regression, neural nets, and analogy, just to name a few [9], [31]. Analogy-based estimation is discussed in detail in the next section. In order to give the reader some context, we offer here some notes on none-analogy methods. Regression assumes that the data fits some function. The parameters of that function are then adjusted to minimize the difference between the values predicted by the model and the actual values in the training data. For example, in linear regression, the model is assumed to be of the form:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots$$

where x_i are model inputs, y are the model outputs and β_i are the coefficients adjusted by a linear regression induction system.

Neural nets are useful when the data distributions are not simple linear functions [32]–[34]. An input layer of project details is connected to zero or more “hidden” layers which, in turn, connect to an output node (the effort prediction). The connections are weighted directed edges. If the signal arriving to a node sums to greater than some threshold value, the node is said to “fire” and a weight is propagated across the network. Learning in a neural net compares the output value to the expected value, then applies some correction method to improve the edge weights (e.g. the “back propagation” algorithm first invented by Bryson and Ho in 1969 [35], and made popular by Rumelhart et al. in the 1980s [36]).

All induction systems require a bias in order to decide what details can be safely ignored. For example, linear regression assumes that the effort data fits a straight line. When data does not match the bias of the induction system, various patches have been proposed. Boehm [1, p526-529] and Kitchenham & Mendes [37] advocate taking the logarithms of exponential distributions before applying linear regression. Selecting the right patch is typically a manual process requiring an analyst experienced in effort estimation.

2.4 Analogy-based Estimation (ABE)

In ABE, effort estimates are generated for a *test* project by finding similar completed software projects (a.k.a. the *training projects*). Following Kadoda & Shepperd [19], Mendes et al. [11], and Li et al. [12] we define a baseline ABE called ABE0, as follows.

ABE0 executes over a table of data where:

- Each row contains one project;
- Columns contain *independent* variables (features) in the projects and *dependent* variables (features) that stores, for example, effort and duration required to complete one project.

After processing the training projects, ABE0 inputs one test project then outputs an estimate for that project. To generate that estimate, a *scaling measure* is used to ensure all independent features have the same degree of influence on the distance measure between test and training projects. Also, a *feature weighting* scheme is applied to remove the influence of the less informative independent features. For example, in feature subset selection [38], some features are multiplied by zero to remove redundant or noisy features.

The similarity between the target project case and each case in the case-based repository is determined by a similarity measure. There are different methods of measuring similarity have been proposed for different measurement contexts. A similarity measure is measuring the closeness or the distance between two data objects in an n -dimensional Euclidean space, the result is usually presented in a distance matrix (or similarity matrix) identifying the similarity among all cases in the dataset. The Euclidean distance metric is the most commonly used in ABE for its distance measures, and it is suitable for continuous values such as software size, effort and duration of a project. It is based on the principle of Pythagorean Theorem to derive a straight line distance between two points in n -dimensional space.

In general, the unweighted Euclidean distance between two points $P = (p_1, p_2, \dots, p_n)$ and $Q = (q_1, q_2, \dots, q_n)$, and can be defined and calculated as:

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2} \quad (1)$$

An alternative is to apply different weights to each individual project feature to reflect the its relative importance in the prediction system. The weight Euclidean distance can be calculated as:

$$\sqrt{w_1(p_1 - q_1)^2 + w_2(p_2 - q_2)^2 + \dots + w_n(p_n - q_n)^2} = \sqrt{\sum_{i=1}^n w_i(p_i - q_i)^2} \quad (2)$$

where w_1 and w_n are the weights of 1st and n^{th} project features. Note that in the special case of $w_i = 1$ (i.e. equal weighting) then the equations are identical.

The above Euclidean distance functions is suitable for general problems, particularly when values are of continuous nature. There are other different distance metrics for non-continuous variable, these include, but are not limited to Jaccard distance for binary distance [39] and Gower distance described by Gower & Legendre [40]. In this paper, we only consider the Euclidean distance measure which is most relevant to the context of software cost estimation.

Irrespective of the similarity measure used, the objective is to rank similar cases from the dataset to the target case and utilize the known solution of the nearest k -cases. The value of k in this case has been the subject of debate in the ABE research community [19] [17]. Shepperd & Schofield [17] suggested the ideal value for k is 3, that is, only three closest neighboring cases will be considered. These k cases will be adjusted or adapted to better fit the target problem by predefined rules, a human expert or more commonly, using a simple mean or median of the selected k cases.

2.5 Alternatives to ABE0

Within the space of ABE methods, ABE0 is just one approach. Based on our reading of the literature we see other variants that take different approaches to:

- The selection of relevant features;
- The similarity function;
- The weighting method used in similarity function;
- The case subset selection method (a.k.a selected analogies or k value);
- And the adaption strategy (a.k.a solution function)

Not every paper explores every option. For example:

- In [19], the focus of Kadoda et. al. is the impact of the selected number of analogies;
- In [12] Li et. al. study the effects of relevant subset selection in training set (i.e. historical data) as well as feature weighting in the similarity function;
- Auer et. al. propose an optimal weight finding mechanism by means of extensive search in [8];
- In [15], Walkerden et. al. investigate selected analogies and compare the performance of human experts to that of tools;
- Finally in [11], Mendes et. al. limit historical data to a single domain and compare different ABE configurations to non-ABE methods.

Generalizing from the above, the following notes tries to map the space of options within ABE research. Since researchers are developing new technologies for effort estimation all the time, this map is incomplete. However, it does illustrate our general point that there are thousands of possible variants to ABE.

2.5.1 Three Case Subset Selectors

A *case subset selection* is sometimes applied to improve the set of training projects. These selection mechanisms are characterized by how many cases they remove:

- *Remove nothing*: Usually, effort estimators use all training projects [17]. ABE0 is using this variant.
- *Outlier* methods prune training projects with (say) suspiciously large values [25]. Typically, this removes a small percentage of the training data.
- *Prototype* methods find, or generate, a set of representative examples that replace the training cases. Typically, prototype generation removes most of the training data. For example, Chang's prototype generators [41] explored three data sets A, B, C of size 514, 150, 66 instances, respectively. He converted these into new data sets A', B', C' containing 34, 14, 6 prototypes, respectively. Note that the new data sets were very small, containing only 7%, 9%, 9% of the original data.

2.5.2 Ten Feature Weighting Methods

In other work Li et al. [12], and Hall & Holmes [38] review eight different feature weighting schemes. Li et al. uses a genetic algorithm to learn useful feature weights. Hall & Holmes review a variety of methods ranging from WRAPPER (a $O(2^F)$ search through all subsets of F features) to various filters methods (that run much faster than WRAPPER) including their preferred correlation-based method.

In our own work, we have developed yet another feature weighted scheme. The fundamental assumption underlying ABE0 is that projects that are similar with respect to project features will be similar with respect to project effort. To formally evaluate this hypothesis, Keung et al. [25] developed a more comprehensive solution towards ABE0, called Analogy-X (a.k.a AX). For example, given two distance matrices constructed from the selected predictor variables and the response variable, we can correlate the two matrices and show their distance correlation function. However different ordering of the matrix elements may result different matrix correlations, AX applies Mantel's technique that randomly permute the distance matrix elements 1,000 times to produce randomisation statistic distribution. Based on the Mantel correlation, AX selects the project features that improves overall Mantel correlation and uses a set of procedures similar to that of stepwise regression to select the project features that are statistically relevant to the solution space, effectively removes the need for brute force feature selection in the classical ABE0 proposed in [17]. More importantly AX provides a statistical justification as to whether ABE should be used for the dataset under investigation. Keung et al. [25]'s study also concludes that dataset quality and variance within the dataset are influential factors, removing data points with large variance will improve prediction performance.

2.5.3 Five Discretization Methods

Some feature weighting schemes require an initial *discretization* of continuous columns. Discretization divides a continuous range at break points b_1, b_2, \dots , each containing a count c_1, c_2, \dots of numbers [42]. There are many discretization policies in the literature including:

- Equal-frequency, where $c_i = c_j$;
- Equal-width, where $b_{i+1} - b_i$ is a constant;
- Entropy [43];
- PKID [44];
- Do nothing at all.

2.5.4 Six Similarity Measures

Mendes et al. [11] discuss three similarity measures including the weighted Euclidean measure described above, an unweighted variant (where $w_i = 1$) and a “maximum distance” measure that focuses on the single feature that maximizes inter-project distance. Frank et al. [45] offer a fourth similarity measure that uses a triangular distribution that sets to the weight to zero, after the distance is more than “k” neighbors away from the test instance. A fifth and sixth similarity measures are the Minowski distance measure used in [46] and the mean value of the ranking of each project feature used in [15].

2.5.5 Six Adaption Mechanisms

With regards to adaptation, the literature reports many approaches including:

- Report the median effort value of the analogies;
- Report the mean dependent value;
- Summarize the adaptations via a second learner; e.g. regression [10], model trees [9], [47] or neural network [48].
- Report a weighted mean where the nearer analogies are weighted higher than those further away [11];

2.5.6 Six Ways to Select Analogies

Li et al. [12] comment that there is much discussion in the literature regarding the number of analogies to be used for estimation. Numerous methods are proposed, which we divide into *fixed* and *dynamic*.

Fixed methods use the same number of analogies for all items in the test set. For example, Li et al. [12] report that a standard fixed method is to always use $1 \leq k \leq 5$ nearest projects:

- $k = 1$ is used by Lipowezky et al. [49] and Walkerden & Jeffery [15];
- $k = 2$ is used by Kirsopp & Shepperd [50]

- $k = 1, 2, 3$ is used by Mendes et al. [11]

Dynamic methods adjust the number of analogies, according to the task at hand. For example, following advice from Martin Shepperd², Baker [10] tuned k to a particular training set using the following “Best(K)” procedure:

- 1) Select $N \subseteq T$ training projects at random;
- 2) For each $k \in 1..T - N$, compute estimates for $n \in N$;
- 3) Find the k value with least error in step 2.
- 4) When estimating, use the k -nearest neighbors, where k is set by step 3.

Our results show that Best(K) out-performs the standard fixed methods (i.e. $k \leq 5$). More interestingly, as shown in Figure 2, Best(K) recommends k values that are very different to those seen in the standard fixed methods. These results come from three commonly used data sets (Desharnais, NASA93, and the original COCOMO data set from [1]: for notes on these data sets, see the appendix).

While ABE systems differ on many aspects, they all use analogy selection. The Figure 2 results suggest that there may be something sub-optimal about standard, widely-used, fixed selection methods. Hence, the rest of this paper takes a closer look at this aspect of ABE.

3 DESIGNING TEAK

The above sample of the literature describes

$$3 * 10 * 5 * 6 * 6 * 6 > 30,000$$

ways to implement similarity, adaptation, weighting, etc. Some of these ways can be ruled out, straight away. For example, at $k = 1$, then all the adaptation mechanisms return the same result. Also, not all the feature weighting techniques require discretization, decreasing the space of options by a factor of five. However, even after discarding some combinations, there are still thousands of possibilities to explore. How might we explore all these variations?

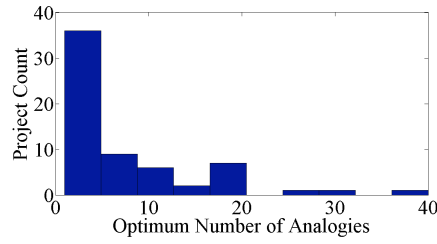
The rest of this paper applies the easy path to design and evaluate an ABE system called TEAK (Test Essential and Assumption Knowledge). TEAK is an ABE0, with the variations described below.

3.1 Select a Prediction System

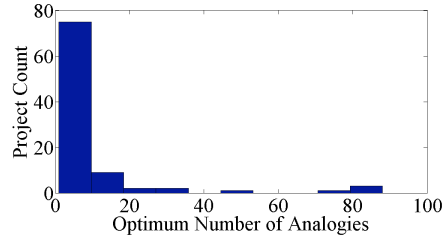
Firstly, we *select a prediction system*. We use ABE since:

- It is a widely studied [8], [12], [15]–[26].
- It works even if the domain data is sparse [51].

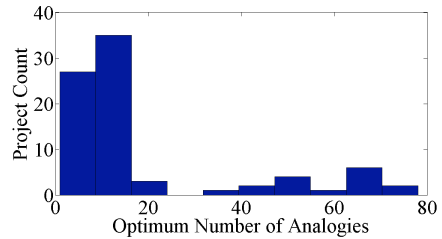
2. Personal communication.



(a) Cocomo81



(b) Nasa93



(c) Desharnais

Fig. 2: Distribution of k after removing each project instance, then applying Best(K) on the remaining data. The y-axis counts the number of times a particular k value was found by Best(K).

- Unlike other predictors, it makes no assumptions about data distributions or an underlying model.
- When the local data does not support standard algorithmic/parametric models like CO-COMO, ABE can still be applied.

The easy path limits the space of design options to just *those that directly address the essential assumptions of the predictor*. As shown below, for ABE, this directs us to issues of case subset selection and the number of analogies used for estimation.

3.2 Identify Essential Assumption(s)

The second step is to *identify the essential assumptions of that prediction system*. Although it is usually unstated, the basic hypothesis underlying the use of analogy-based estimation is that projects that are similar with respect to project and product factors will be similar with respect to project effort [26]. In other words:

Assumption One: Locality implies uniformity.

This assumption holds for project training data with the following property:

- The k -nearest training projects with effort values E_1, E_2, \dots, E_k have a mean value $\mu = \left(\sum_i^k E_i\right)/k$ and a variance $\sigma^2 = \left(\sum_i^k (E_i - \mu)^2\right)/(k - 1)$.
- By *Assumption One*, decreasing k also decreases σ^2 .

Note that if all estimates always have the same distance to the mean μ , then *Assumption One* always fails since in $\left(\sum_i^k (E_i - \mu)^2\right)/(k - 1)$, if the numerator is constant then decrease k will always *increase* the variance. The core of ABE is the premise that in the neighborhood of training instances, the reductions seen in $(E_i - \mu)^2$ dominates over the increases due to $1/(k - 1)$. As we shall see below, this sometimes holds (and sometimes it does not).

3.3 Identify Assumption Violation

The third step is to *recognize situations that violate the essential assumption*. Implementing this step requires some way to compare the variance of larger- k estimates to smaller- k estimates. One way to achieve this is to use some clustering methods that generates a tree of clusters, where each sub-tree contains training data that is closer together than the super-tree.

There are many algorithms for generating trees of clusters. The basic method, called greedy agglomerative clustering (GAC) is used in various fields (data mining [52], databases [53] bioinformatics [54]). GAC executes bottom-up by grouping together at a higher level ($i + 1$) the closest pairs found at level i . The algorithm terminates when some level i is found with only one node. GAC is “greedy” in that it does not pause to consider optimal pairings for vertices with very similar distances. It never backtracks looking for (say) better pairings a level i to reduce the distance between nodes at level $i + 1$.

The result of GAC is a tree like Figure 3. Note that, in this tree, the original training data are found at the leaves of the tree. All other nodes are nodes artificially generated by GAC to represent the median of pairs of the leaves, the median of the medians, and so on (recursively).

A GAC tree can be viewed as a tree of clusters where each node at height i is the centroid of the sub-clusters at height $i - 1$. Given T initial instances, GAC builds a trees of maximum height $\log_2(T)$. Since the number of vertices is halved at each next level, building a GAC tree requires the following number of distance calculations:

$$\left(\sum_i^{\log_2(T)} \left(\frac{T}{2^{i-1}}\right)^2\right) = \frac{4}{3}(T^2 - 1)$$

This $O(T^2)$ computation is deprecated for large T . However, for this study, GAC construction takes less than a second³. Our runtimes were fast because effort estimation data sets are usually very small: Figure 4 shows all our training projects contain less than 100 examples.

3. Using Matlab on a standard Intel x86 dual core notebook running LINUX with 4GB of ram.

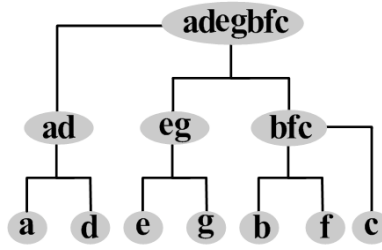


Fig. 3: A sample GAC tree built from 7 instances: a, b, c, d, e, f and g . In each level the closest two nodes are coupled-up to produce the nodes in one higher level. However, odd numbered instances result in unbalanced trees. In this figure, when all leaf nodes are paired up (pairing of letters here is random and just for illustration), c remains alone. The alone remaining node at any level, will find the closest node to itself in one higher level and will merge to that node. For example c merges to the node containing b and f .

There are many other, faster, algorithms for generating trees of clusters including bisecting k-means [55] which, at each level, calls k-means⁴ with $K=2$ several times to split one cluster into two. The division leading to the clusters with the best intra-cluster similarities are then stored and used in subsequent splits. Other approaches, like MESO [56], recursively divide the data into smaller and smaller spheres containing *close* instances. MESO uses an incremental method to learn and update what *close* means for a particular data set.

We use GAC (with the distance measure of Equation ??) rather than other methods like bisecting k-means or MESO for two reasons. Firstly, these other methods use various heuristics to improve their runtimes. Our data sets are so small that such heuristic methods are not necessary. Secondly, our results (described below) with GAC are so promising that we are not motivated to experiment beyond GAC.

Using a GAC tree, finding the k -nearest neighbors in project data can be implemented using the following TRAVERSE procedure:

- 1) Place the test project at the root of the GAC tree.
- 2) Move the test project to the nearest child (where “nearest” is defined by Equation ??).
- 3) Go to step 2

Clearly, a $k = 1$ nearest-neighbor estimate comes from TRAVERSE-ing to a leaf, then reporting the effort of that leaf. More generally, a $k = N$ nearest-neighbor estimate comes from TRAVERSE-ing to a sub-tree with N leaves, then reporting the median efforts of those leaves.

TRAVERSE can test *Assumption One*. Let some current vertex x have children y and z . We say

4. The k-means clustering algorithm selects centroids at random, labels each instance by its nearest centroid, then updates the centroid position to the central position of all instances with the same label. The algorithm repeats till the centroid position stabilizes.

Dataset	Features	$T = Projects $	Content	Historical Effort Data					
				Units	Min	Median	Mean	Max	Skewness
Cocomo81	17	63	NASA projects	months	6	98	683	11400	4.4
Cocomo81e	17	28	Cocomo81 embedded projects	months	9	354	1153	11400	3.4
Cocomo81o	17	24	Cocomo81 organic projects	months	6	46	60	240	1.7
Nasa93	17	93	NASA projects	months	8	252	624	8211	4.2
Nasa93c2	17	37	Nasa93 projects from center 2	months	8	82	223	1350	2.4
Nasa93c5	17	40	Nasa93 projects from center 5	months	72	571	1011	8211	3.4
Desharnais	12	81	Canadian software projects	hours	546	3647	5046	23940	2.0
SDR	22	24	Turkish software projects	months	2	12	32	342	3.9
Albrecht	7	24	Projects from IBM	months	1	12	22	105	2.2
ISBSG-Banking	14	29	Banking projects of ISBSG	hours	662	2355	5357	36046	2.6
Total: 448									

Fig. 4: The 448 projects used in this study come from 10 data sets. Indentation in column one denotes a dataset that is a subset of another dataset. For notes on this data, see the appendix.

that:

- The sub-trees starting at x, y, z have leaves L_x, L_y, L_z (and $L_x = L_y \cup L_z$).
- The number of sub-tree leaves is $k_x = k_y + k_z$.
- The variance of the leaves' efforts are $\sigma_x^2, \sigma_y^2, \sigma_z^2$.
- After C4.5 [57], we say the variance of the trees below x (denoted σ_{yz}^2) is the weighted sum:

$$\sigma_{yz}^2 = \frac{k_y}{k_x} \sigma_y^2 + \frac{k_z}{k_x} \sigma_z^2$$

Parent trees have the nodes of their children (plus one). If we TRAVERSE from a parent x to a child, then the sub-tree size k decreases. That is, TRAVERSE-ing moves into progressively smaller sub-trees.

Assumption One holds if, when TRAVERSE-ing from all vertices x with children y and z , the sub-tree variance *decreases*. That is:

$$\forall x \in T : \sigma_x^2 > \sigma_{yz}^2 \tag{3}$$

Note one special case of the above: in the case of $k = 1$, variance is zero since, by definition, all members of a population of one thing are all the same. Hence, earlier versions of TEAK always just returned just the leaf nodes. The bug, it was realized, was the direction of the search. A *bottom-up traversal* that tries to grow a smaller region to a larger region will rarely grow more than one level (since, at level two, it usually encounters training instances that are somewhat different to the leaf nodes on level one). Accordingly, TEAK was changed to the following *top-down* procedure that works from a larger space to a smaller space. In the usual case, this search stops before the $k=1$ region.

3.4 Remove Violations

The fourth step in TEAK’s design is to *remove the situations that violate the essential assumption*. We instrumented TRAVERSE to report examples where Equation 3 was violated; i.e. where it recursed into sub-trees with a larger variance than the parent tree. We found that this usually occurs if a super-tree contains mostly similar effort values, but one sub-tree has a minority of outliers. For example:

- Suppose some vertex x has children y, z .
- Let each child start sub-trees whose leaves contain the effort values $leaves(y) \in \{1253, 1440\}$ staff hours and $leaves(z) \in \{1562, 5727\}$ staff hours.

In this example:

- The leaves of the parent tree x have similar effort values: 1,253 and 1,562 and 1,440 staff hours.
- But the leaves of the subtree z has outlier values; i.e. 5,727.
- TRAVERSE-ing from the super-tree x to the sub-tree z increases the variance by two orders of magnitude.

A *sub-tree pruning policy* is used to prune sub-trees with a variance that violates the essential assumption. We experimented with various policies that removed subtrees if they had:

- 1) more than α times the parent variance;
- 2) more than $\beta * \max(\sigma^2)$;
- 3) more than $R^\gamma * \max(\sigma^2)$, where R is a random number $0 \leq R \leq 1$.

In order to avoid over-fitting, our pruning policy experiments were restricted to one data set (Boehm’s COCOMO embedded projects [1]) then applied, without modification, to the others. The randomized policy (#3) produced lowest errors, with smallest variance. The success of this randomized policy suggests two properties of effort estimation training data:

- The boundary of “good” training projects is not precise. Hence, it is useful to sometimes permit random selection of projects either side of the boundary.
- The policy tuning experiments recommended $\gamma = 9$. This selects for subtrees with less than 10% of the maximum variance⁵. This, in turn, suggests that the above example is typical of effort estimates; i.e. sub-tree outliers are usually a few large effort values.

In theory, stochastic methods like policy #3 introduce a degree of instability in the performance of the induction system. In practice, this is not an issue with TEAK. In the evaluation section, described below, we repeat our analysis of TEAK 20 times using various performance measures and experimental rigs. When we compare the results of those repeated trials against just running TEAK once, we can see no major performance differences.

5. The mean of $rand()^9 \approx 0.1$.

3.5 Execute the Modified System

The final step in the design of TEAK is to build a new prediction system. TEAK executes as follows:

- Apply GAC to the training projects to build a tree called GAC1;
- Prune GAC1 using the sub-tree pruning policy described above. The remaining leaves are the *prototypes* to be used in effort estimation.
- Apply GAC to the prototypes to build a second tree called GAC2.
- Place the test project at the root of GAC2. Compute an estimate from the median value of the GAC2 projects found by TRAVERSE2. TRAVERSE2 is a variant of TRAVERSE that ensures the essential assumption is never violated. It stops recursing into GAC2 sub-trees when Equation 3 is violated.

4 COMPARISONS

Recall the pre-experimental concern expressed above: Taking the easy path might ignore important design issues, to the detriment of the predictions. To address that concern, this section compares TEAK to a range of other ABE0 variants as well the other induced prediction systems described in §2; i.e. neural nets and linear regression (we selected this particular range of algorithms at the suggestion of reviewers of this paper).

As discussed above in §2.1, the reader should look elsewhere for a detailed comparison of analogy-based estimation to other estimation methods (e.g. [14]). We compare TEAK to a limited number of other estimation methods (specifically, neural networks and regression), merely to ensure that analogy-based estimation is not significantly *worse* than other methods in widespread use.

In the following comparisons, TEAK will be assessed using:

- Two different experimental rigs (leave-one-out and n-way cross-validation: see §4.1);
- Three different performance measures (AR, MRE, PRED(25): see §4.2);

A pre-experimental concern with this approach was that if different performance measures/rigs endorsed different induction methods, how might we judge any overall winner? On experimentation, this theoretical concern was not realized: TEAK performed best over all rigs and performance measures.

4.1 Randomized Trials

Recall that TEAK uses a randomized method for sub-tree pruning. Any evaluation of such a randomized method must be repeated multiple times. Appealing to the central limit theorem, we used twenty repeats.

- Twenty times, for each data set, we randomize the order of the rows in that data set.
- Next, we conducted *both* a Leave-One-Out study and a 3-way cross-validation study.

In Leave-One-Out, given T projects, then $\forall t \in T$, use t as the test project and the remaining $T - 1$ projects for training. In 3-way cross-validation, the data set of T projects is divided into three bins, bin_i is used for testing while the remaining $T - bin_i$ projects are used for testing.

Since some of our data sets are very small (e.g. the 24 instances of Cocomo81o), we used a 3-way cross validation (and not the 10-way used by, say, Quinlan [57]). Some thought was given to using 3-way for small data sets and 10-way for larger data sets. However, this would introduce a complication into the analysis that is neither recommended by the literature, nor handled by any statistical technique that we are aware of.

We use both Leave-one-out and N-way cross-validation since the effort estimation literature is ambiguous on which is most appropriate. In the Kitchenham et al. survey [58], all the projects reviewed in their Table 3 used N-way cross-validation. However, other prominent studies prefer leave-one-out [23].

For these studies, we used all independent features when computing similarities. We applied twenty randomized trials using the 448 projects from the 10 data sets of Figure 4 (for notes on this data, see the appendix). In all, these randomized trials generated $20 \times 2 \times 448 = 17,920$ training/test set pairs.

4.2 Details

For each of these 17,920 pairs of training/test, estimates were generated by TEAK, neural networks, regression, and six other ABE0 variants:

- Five variants returned the median effort seen in the k -th nearest neighbors for $k \in \{1, 2, 4, 8, 16\}$.
- The other variant returned the median effort seen in k neighbors found using Baker's Best(K) procedure. From §2.5, recall that Best(K) adjusts k to each data set by reflecting over all the training projects.

Since this is paired data (same train and test data passed through multiple treatments), we applied a Wilcoxon signed rank test (95% confidence) to rank the resulting estimates. Ranked⁶ statistical tests like the Wilcoxon are useful if it is not clear that the underlying distributions are Gaussian [59]. Also, ranked statistics mitigate the problem of effort estimation results that sometimes contain a small number of very large errors [60].

6. In a ranking analysis, the raw results (10.2,21.3,22.1,24,25,30,100) are replaced with their ranks in the sort order; i.e. (1,2,3,4,5,6,7).


```

wini = 0, tiei = 0, lossi = 0
winj = 0, tiej = 0, lossj = 0
if WILCOXON( $P_i, P_j$ ) says they are the same then
    tiei = tiei + 1;
    tiej = tiej + 1;
else
    if median( $P_i$ ) < median( $P_j$ ) then
        wini = wini + 1
        lossj = lossj + 1
    else
        winj = winj + 1
        lossi = lossi + 1
    end if
end if

```

Fig. 5: Pseudocode for Win-Tie-Loss Calculation Between Variant i and j with performance measures P_i and P_j .

We collected information on three performance metrics: AR, MRE, PRED(25). The magnitude of the absolute residual (AR) is computed from the difference between predicted and actual:

$$AR = |actual_i - predicted_i| \quad (4)$$

We prefer AR to other performance measures since we share Shepperd's concern [61] that anything other than the simplest evaluation statistic can introduce analysis issues. Nevertheless, other measures are more common in the effort estimation literature (e.g. see Table 3 of [58]) such as MRE and PRED(25). MRE is the magnitude of the relative error. It is calculated by expressing AR as a ratio of the actual prediction:

$$MRE = \frac{|actual_i - predicted_i|}{actual_i} \quad (5)$$

AR and MRE are calculated for every item in a test set. PRED(X), on the other hand, is a summary statistic that reports behavior over an entire test suite. PRED(X) reports the average percentage of the N estimates in the test set that were within $X\%$ of the actual values:

$$PRED(X) = \frac{100}{T} \sum_i^T \begin{cases} 1 & \text{if } MRE_i \leq \frac{X}{100} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

For example, PRED(30)=50% means that half the estimates are within 30% of the actual. Chulani & Boehm assesses his models using PRED(30) [62]. We use the stricter criteria of PRED(25) since that is more common in the literature; e.g. [17], [34], [63].

In order to summarize the results of the Wilcoxon comparisons of the MRE, AR, PRED(25) measures, we use the following win-tie-loss procedure. For each iteration of the randomized trials, each data set generated 20*7=140 (MRE, PRED(25), AR) distributions for each induced prediction system (neural nets, regression, the ABE0 variants). To calculate the win-tie-loss values, we first checked if two distributions i, j are statistically different according to the Wilcoxon test. If not, then we incremented tie_i and tie_j . On the other hand, if they turned out to be

different, we updated win_i, win_j and $loss_i, loss_j$ after a numerical comparison of their median values. The pseudocode for win-tie-loss calculation is given in Figure 5.

Note that Figure 5 is different to the “median(MRE)” and “mean(MRE)” performance measures used in other papers (see Table 3 of [58]). In median and mean performance measures, the entire distribution is summarized by its central tendency (measured in terms of median or mean), then two methods are compared solely in terms of those two central points. Unlike the Wilcoxon test used in Figure 5, such single-point assessment does not consider the variance around centrality. Such single point assessments are depreciated in literature; e.g. see Foss et al.’s scathing critique of mean MRE [64].

4.3 Results

Initially, our intention was to report results using all the data sets of Figure 4. However, we found that the Albrecht data set was producing a very large number of ties (over 98%). On closer inspection, we found that in our rig, Albrecht was a data set in which all our treatments generated very similar results (the plots of the MREs generated by our eight methods was indistinguishable). Since Albrecht was mostly unable to distinguish between the different treatments, we excluded it from the rest of our analysis.

The resulting Win/Loss/Ties values from a Leave-One-Out study that measured MRE are shown in Figure 6. When ranked in terms of $win - loss$, in $\frac{6}{9}$ data sets, TEAK is the *top ranked* method; i.e it always ranked first on that performance score. The next best method was linear regression that is found in the top rank in only $\frac{3}{9}$ data sets.

These scores are summarized top left of Figure 7 (see the tables for “MRE”). There is not enough space in this article to repeat Figure 4 for every combination of (Leave-One-Out, Cross-Val)*(MRE, AR, PRED(25)); i.e. six times in all. Hence, we present a summary of those results in Figure 7. In all cases:

- 1) Best(K) and $K \in \{1, 2, 4, 8, 16\}$ rarely appeared in the top ranked methods. That is, standard analogy selection mechanism performed comparatively worse than applying TEAK’s variance heuristic.
- 2) While non-analogy methods sometimes did better on certain data sets, overall, TEAK’s extension to analogy-based reasoning was competitive with non-analogy methods.

From result 1), we recommend variance pruning for analogy estimation since unequivocally, of all the analogy variants studied here, TEAK is the superior system.

As to result 2), we hesitate to conclude, just from this sample, that TEAK is *always* the best effort estimation method. However, its results are encouraging and should motivate continued research into analogy-based methods. In our review of effort estimation [9], we commented that

Data set	Variant	Win	Tie	Loss	Win - Loss
Cocomo81	TEAK	87	73	0	87
	Best(K)	49	110	1	48
	k=16	42	107	11	31
	k=8	41	100	19	22
	k=4	28	96	36	-8
	NNet	37	76	47	-10
	k=1	28	88	44	-16
	k=2	26	82	52	-26
	LR	7	18	135	-128
Cocomo81e	TEAK	55	105	0	55
	NNet	43	117	0	43
	k=8	32	126	2	30
	k=16	32	126	2	30
	Best(K)	32	126	2	30
	k=4	18	113	29	-11
	k=1	8	97	55	-47
	k=2	4	101	55	-51
	LR	11	59	90	-79
Cocomo81o	TEAK		136	0	24
	k=16	9	151	0	9
	k=8	8	152	0	8
	Best(K)	8	152	0	8
	NNet	9	150	1	8
	k=4	7	151	2	5
	LR	7	145	8	-1
	k=2	2	128	30	-28
	k=1	1	125	34	-33
Nasa93c5	TEAK	40	120	0	40
	LR	25	135	0	25
	k=16	17	141	2	15
	Best(K)	17	139	4	13
	k=8	16	134	10	6
	NNet	10	144	6	4
	k=4	10	127	23	-13
	k=2	7	110	43	-36
	k=1	3	100	57	-54
SDR	TEAK	67	93	0	67
	k=1	43	97	20	23
	NNet	25	123	12	13
	k=4	26	118	16	10
	k=8	18	132	10	8
	k=2	20	126	14	6
	Best(K)	16	126	18	-2
	k=16	13	120	27	-14
	LR	0	49	111	-111
ISBSG-Banking	TEAK	30	130	0	30
	NNet	24	136	0	24
	LR	23	137	0	23
	k=16	22	138	0	22
	k=8	19	141	0	19
	Best(K)	21	137	2	19
	k=4	14	112	34	-20
	k=1	8	106	46	-38
	k=2	4	73	83	-79
Nasa93	LR	72	88	0	72
	TEAK	26	134	0	26
	NNet	16	143	1	15
	k=16	13	133	14	-1
	k=8	15	128	17	-2
	Best(K)	14	128	18	-4
	k=4	6	122	32	-26
	k=2	4	113	43	-39
	k=1	6	107	47	-41
Nasa93c2	LR	158	2	0	158
	TEAK	36	106	18	18
	k=16	25	115	20	5
	NNet	17	123	20	-3
	k=8	15	116	29	-14
	Best(K)	15	116	29	-14
	k=4	11	101	48	-37
	k=2	5	95	60	-55
	k=1	6	90	64	-58
Desharnais	LR	63	97	0	63
	NNet	51	109	0	51
	TEAK	37	121	2	35
	k=16	25	129	6	19
	k=8	22	124	14	8
	Best(K)	16	120	24	-8
	k=4	14	116	30	-16
	k=2	6	80	74	-68
	k=1	1	74	85	-84

Fig. 6: MRE based win-loss-tie results from the 20*Leave-One-Out experiments. For each data set results are sorted by win minus loss values. Gray cells indicate variants with zero losses. The performance of the various induced prediction systems is summarized in top-left corner of Figure 7

20 * LEAVE-ONE-OUT									20 * 3-WAY CROSS-VALIDATION											
	TEAK	LR	NNet	Best(K)	k=1	k=16	k=2	k=4	k=8		TEAK	LR	NNet	Best(K)	k=1	k=16	k=2	k=4	k=8	
MRE									MRE											
Cocomo81	▲									Cocomo81	▲									
Cocomo81e	▲									Cocomo81e	▲									
Cocomo81o	▲									Cocomo81o	▲									
Nasa93		▲								Nasa93		▲								
Nasa93c2		▲								Nasa93c2		▲								
Nasa93c5	▲									Nasa93c5	▲									
Desharnais		▲								Desharnais		▲								
Sdr	▲									Sdr	▲									
ISBSG-Banking	▲									ISBSG-Banking	▲									
Count	6	3	0	0	0	0	0	0	0	Count	6	3	0	0	0	0	0	0	0	0
Pred(25)									Pred(25)											
Cocomo81	▲									Cocomo81	▲									
Cocomo81e			▲							Cocomo81e			▲							
Cocomo81o	▲									Cocomo81o	▲									
Nasa93		▲								Nasa93		▲								
Nasa93c2		▲								Nasa93c2		▲								
Nasa93c5	▲									Nasa93c5	▲									
Desharnais		▲								Desharnais		▲								
Sdr	▲									Sdr	▲									
ISBSG-Banking	▲									ISBSG-Banking	▲									
Count	5	3	1	0	0	0	0	0	0	Count	5	3	1	0	0	0	0	0	0	0
AR									AR											
Cocomo81	▲									Cocomo81				▲						
Cocomo81e	▲									Cocomo81e	▲									
Cocomo81o	▲									Cocomo81o	▲									
Nasa93		▲								Nasa93		▲								
Nasa93c2		▲								Nasa93c2		▲								
Nasa93c5	▲									Nasa93c5	▲									
Desharnais		▲								Desharnais		▲								
Sdr	▲									Sdr	▲									
ISBSG-Banking	▲									ISBSG-Banking	▲									
Count	6	3	0	0	0	0	0	0	0	Count	6	3	0	1	0	0	0	0	0	0

Fig. 7: Summary of the random trials; e.g. Figure 6 is summarized top-left in “MRE”. This figure displays the top performing inducted predictive system, measured via $(win - loss)$. This is repeated for all the performance measures (MRE, PRED(25), AR) and both experimental rigs (leave-one-out on the left and 3-way cross-validation on the right). The last row of each table shows the sum of times a method appeared as the top performing variant. In the majority of cases, TEAK appears as the top-ranked predictive system.

best practices include generating estimates from multiple sources. Certainly, these results offer no reason to *exclude* analogy as one of those sources.

4.4 Discussion

Two more features of these results deserve special attention. Firstly, there is a large number of ties seen in all datasets: in some data sets like Nasa93c2 and Cocomo81o in Figure 4, most of the methods tied 80% of the time or more. This point is interesting in that it suggests that as well as assessing effort estimation algorithms, it might be productive to also *assess effort estimation data sets*. If it is too easy or too hard to generate accurate estimates in a particular dataset, then that data set will exhibit *ceiling* or *floor* effects where all learners have the same high or low (respectively) performance scores. We might be able to rank data sets according to their ceiling and floor effects. With that ranking in hand, researchers might care to avoid data sets that do not differentiate between different effort estimation methods (e.g. since all methods perform equally as well, or equally as poorly).

Secondly, there is a marked *conclusion stability* over all our results: the same pattern of results in Figure 7 repeats nearly exactly across all data sets, multiple evaluation criteria, and multiple experimental rigs. When this conclusion stability was first seen, all the scripts generating these results were inspected. Also, sample results from this paper were compared with known results we had generated previously in past publications using different code. No errors in the generation scripts were detected so we assert that this conclusion stability is a true result, and not due to some programming error.

At first glance, these repeated results contradicted *simulation studies* that reported *conclusion instability*. Those studies reported marked instability in the ranking of different effort estimation methods according to the choice of random number seeds for train/test set selection; the choice of data set; and the performance measure used to assess the results [31], [65].

On reflection, it was realized that Figure 7 was generated under different conditions than those simulation studies. Those simulation studies generated artificial data sets using distributions taken from a single data set. Further, those studies used very few project features (in one of them, only a size measure was used⁷). There is hence no contradiction between:

- *Conclusion instability* results found in simulated data from a few features (e.g. [31], [65]);
- And *conclusion stability* results found in naturally occurring data sets, when all features are used (e.g. our results in Figure 7).

Nevertheless, this conclusion stability should be explored further. Perhaps those prior simulation studies need to be revisited using the large space of data sets and induced prediction systems that is now available.

4.5 But Why Does it Work?

In discussions over TEAK, we are sometimes asked if it is wise to use variance to assess the suitability of neighborhood for providing donor cases. The argument goes as follows: While a high variance for a given neighborhood of k suggests that this is a *bad* neighborhood, a low variance does not necessarily imply that the neighborhood is *good*.

In reply, we note that TEAK does not *only* use variance to select the donor cases. TRAVERSE2 pushes *away* from regions with high variance while pushing *towards* regions with similar features to the test instance. TRAVERSE2 pushes away from high variance regions since:

- It executes over a space of training data which high variance regions are pruned away;
- Its recursive descent terminates if it enters a region of increasing variance.

At the same time, TRAVERSE2 pushes towards regions with similar features as follows:

- It descends a binary tree of clusters.

7. EKREM: in both??? No, only in one, Kadoda2001 uses 4 indep. features.

- At each step, the test instance is moved towards the sub-tree whose median is closest to the test instance.

The above experiments show, we argue, this *twin* policy does better than just pushing towards regions with higher similarity. That is, *augmenting* nearest neighbor algorithms with variance avoidance does better than just applying nearest neighbor.

4.6 Threats to Validity

Internal validity questions to what extent the cause-effect relationship between dependent and independent variables hold [66].

The general internal validity issue is that data mining experiments (like those discussed above) do not collect new data, but only generates theories from historical data. Ideally, we should take a learned theory and apply it to some new situation, then observe if the predicted effect occurs in practice. Note that if no explicit theory is generated, then it cannot be applied outside of the learning system. That is, all ABE systems suffer from issues of internal validity since they do not generate an explicit theory. However, it is possible to mitigate this problem by simulating how an ABE system might be applied to a new situation. Note that the Leave-One-Out approach used in this paper generates estimates using test data that is not used in training.

Construct validity (i.e. face validity) assures that we are measuring what we actually intended to measure [67]. In our research we are using a variety of performance measures (AR, MRE, PRED(25) and a pair of evaluation experiments (leave-one-out and cross-val). This was done to increase the construct validity of this study. MRE is widely used for assessing the performance of competing software effort estimation models [68]–[70]. Foss et al. [69] have provided an extensive discussion demonstrating that by using only MRE itself may be leading to incorrect evaluation. Hence, we take care to apply multiple performance measures and a pair of evaluation experiments.

External validity is the ability to generalize results outside the specifications of that study [71]. To ensure the generalizability of our results, we studied a large number of projects. Our datasets contain a wide diversity of projects in terms of their sources, their domains and the time period they were developed in. For example, we used datasets composed of software development projects from different organizations around the world to generalize our results [72]. Our reading of the literature is that this study uses more project data, from more sources, than numerous other papers. All the papers we have read, as well as, Table 4 of [7] list the total number of projects in all data sets used by other studies. The median value of that sample is 186, which is less than half the 448 projects used in our study.

4.7 Future Work

In this paper, we have applied the easy path principle to design a new method for case & analogy selection. In future work, we will apply the easy path to similarity measures, feature weighting, and adaption. For example:

- After grouping together rows with similar estimates, we might weight features by their variance within each group (and higher variance means lower weight).
- Alternatively, Lipowezky [49] observes that feature and case selection are similar tasks (both remove cells in the hypercube of all cases times all columns). Under this view, it should be possible to convert our case selector to a feature selector.

Our investigations in this area are very preliminary and, at this time, we have no conclusive results to report.

Another promising avenue to explore is variations on the GAC clustering used in this paper. Since our results have so far been quite promising, we have not explored alternatives to GAC. For example, we form links between quite distinct clusters merely because their medians are proximal. Perhaps another, more sophisticated, clustering algorithm would be a better way to group the training data.

5 CONCLUSION

In response to the growing number of options for designing software project effort estimators, various researchers (e.g. [9], [10], [12]) have proposed elaborate and CPU-intensive search tools for selecting the best set of design options for some local data. While useful, these tools offer no insight into the effort estimation task: They report *what* the design is in simplifying future effort estimation tasks, but not *why* they were useful. Such insights are useful for reducing the complexity of future effort estimations.

In order to avoid the computation cost of these tools, and to find the insights that simplify effort estimation, we design TEAK using an *easy path* principle. The easy path has five steps.

1. *Select a prediction system:* Analogy-based effort estimation, or ABE, is a widely-studied method that works on sparse data sets. Hence, we selected ABE as our prediction system.

2. *Identify the predictor's essential assumption(s):* The essential assumption of ABE is that *locality implies uniformity*; i.e. the closer the test project approaches the training projects, the smaller the variance in that neighborhood.

3. *Recognize when those assumption(s) are violated:* Mathematically, this can be tested by recursively clustering project data into a tree whose leaves contain historical effort data and whose internal nodes are medians of pairs of child nodes. When descending this tree, the essential ABE assumption is violated when sub-trees have a larger variance than the parents.

4. *Remove those situations:* This assumptions can be removed by pruning sub-trees with the larger variances.

5. *Execute the modified prediction system:* TEAK builds a second tree of clusters using just the projects not found in high variance sub-trees. Estimates are generated from this second tree by a recursive descent algorithm that stops before the sub-tree variance is higher than the super-tree variance. The leaves of terminating sub-tree are then accessed and the estimate is calculated from the median of the effort values in those leaves.

A pre-experimental concern with the easy path was that, in ignoring the hard training cases, we would miss important aspects of the data. Our experiments do not support that concern. TEAK never lost against other ABE methods and always won the most. Also, TEAK performed at least as well (if not better) than certain other non-analogy-based estimation methods.

Our conclusions are two-fold:

- For those use analogy-based estimation, we strongly recommend pruning instances from regions of high variance prior to generating estimates.
- For those designing new data algorithms, we conclude that it may be detrimental to obsess on the hard cases. Rather, it may be better to enhance what a predictor does best, rather than try to patch what it does worst. For example, in the case of ABE, case selection via variance significantly improved the estimates.

REFERENCES

- [1] B. W. Boehm, *Software Engineering Economics*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.
- [2] C. Kemerer, "An empirical validation of software cost estimation models," *Communications of the ACM*, vol. 30, no. 5, pp. 416–429, May 1987.
- [3] Spareref.com, "Nasa to shut down checkout & launch control system," August 26, 2002, <http://www.spaceref.com/news/viewnews.html?id=475>.
- [4] B. Boehm, C. Abts, and S. Chulani, "Software development cost estimation approaches - a survey," *Annals of Software Engineering*, vol. 10, pp. 177–205, 2000.
- [5] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 33–53, 2007.
- [6] M. Shepperd, "Software project economics: a roadmap," in *FOSE '07: 2007 Future of Software Engineering*, 2007, pp. 304–315.
- [7] B. Kitchenham, E. Mendes, and G. H. Travassos, "Cross versus within-company cost estimation studies: A systematic review," *IEEE Trans. Softw. Eng.*, vol. 33, no. 5, pp. 316–329, 2007, member-Kitchenham, Barbara A.
- [8] M. Auer, A. Trendowicz, B. Graser, E. Haunschmid, and S. Biffl, "Optimal project feature weights in analogy-based cost estimation: Improvement and limitations," *IEEE Transactions on Software Engineering*, vol. 32, pp. 83–92, 2006.
- [9] T. Menzies, Z. Chen, J. Hihn, and K. Lum, "Selecting best practices for effort estimation," *IEEE Transactions on Software Engineering*, vol. 32, pp. 883–895, 2006.
- [10] D. Baker, "A hybrid approach to expert and model-based effort estimation," Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007, available from <https://eidr.wvu.edu/etd/documentdata.eTD?documentid=5443>.

- [11] E. Mendes, I. D. Watson, C. Triggs, N. Mosley, and S. Counsell, "A comparative study of cost estimation models for web hypermedia applications," *Empirical Software Engineering*, vol. 8, no. 2, pp. 163–196, 2003.
- [12] Y. Li, M. Xie, and T. Goh, "A study of project selection and feature weighting for analogy based software cost estimation," *Journal of Systems and Software*, vol. 82, pp. 241–252, 2009.
- [13] J. R. Quinlan, "Boosting first-order learning," in *Proceedings of the 7th International Workshop on Algorithmic Learning Theory*, ser. LNAI, S. Arikawa and A. K. Sharma, Eds., vol. 1160. Berlin: Springer, 1996, pp. 143–155.
- [14] T. Menzies, O. Jalali, J. Hihn, D. Baker, and K. Lum, "Stable rankings for different effort models," *Automated Software Engineering*, no. 4, December 2010.
- [15] F. Walkerden and R. Jeffery, "An empirical study of analogy-based software effort estimation," *Empirical Softw. Engg.*, vol. 4, no. 2, pp. 135–158, 1999.
- [16] C. Kirsopp, M. Shepperd, and R. Premrag, "Case and feature subset selection in case-based software project effort prediction," *Research and development in intelligent systems XIX: proceedings of ES2002, the twenty-second SGAI International Conference on Knowledge Based Systems and Applied Artificial Intelligence*, p. 61, 2003.
- [17] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Trans. Softw. Eng.*, vol. 23, no. 11, pp. 736–743, 1997.
- [18] M. Shepperd, C. Schofield, and B. Kitchenham, "Effort estimation using analogy," in *ICSE '96: Proceedings of the 18th international conference on Software engineering*. Washington, DC, USA: IEEE Computer Society, 1996, pp. 170–178.
- [19] G. Kadoda, M. Cartwright, and M. Shepperd, "On configuring a case-based reasoning software project prediction system," *UK CBR Workshop, Cambridge, UK*, pp. 1–10, 2000.
- [20] J. Li and G. Ruhe, "Analysis of attribute weighting heuristics for analogy-based software effort estimation method aqua +," *Learning*, pp. 63–96, 2008.
- [21] —, "A comparative study of attribute weighting heuristics for effort estimation by analogy," *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, p. 74, 2006. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1159733.1159746>
- [22] —, "A comparative study of attribute weighting heuristics for effort estimation by analogy," *Comparative and General Pharmacology*, pp. 66–74, 2006.
- [23] —, "Decision support analysis for software effort estimation by analogy," in *PROMISE '07: Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, 2007, p. 6.
- [24] J. Keung, "Empirical evaluation of analogy-x for software cost estimation," in *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. New York, NY, USA: ACM, 2008, pp. 294–296.
- [25] J. W. Keung, B. A. Kitchenham, and D. R. Jeffery, "Analogy-x: Providing statistical inference to analogy-based software cost estimation," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 471–484, 2008.
- [26] J. Keung and B. Kitchenham, "Experiments with analogy-x for software cost estimation," in *ASWEC '08: Proceedings of the 19th Australian Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 229–238.
- [27] B. Nuseibeh, "To be and not to be: On managing inconsistency in software development," in *Proc. of 8th IEEE International Workshop on Software Specification & Design (IWSSD-8)*, 1996, pp. 164–169.
- [28] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece, *Software Cost Estimation with Cocomo II*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.
- [29] M. Jorgensen, "A review of studies on expert estimation of software development effort," *Journal of Systems and Software*, vol. 70, pp. 37–60, 2004.
- [30] M. Jorgensen and T. Gruschke, "The impact of lessons-learned sessions on effort estimation and uncertainty assessments," *IEEE Transactions on Software Engineering*, vol. 35, no. 3, pp. 368–383, May-June 2009.
- [31] M. Shepperd and G. Kadoda, "Comparing software prediction models using simulation," *IEEE Transactions on Software Engineering*, pp. 1014–1022, 2001.
- [32] H. Park and S. Baek, "An empirical validation of a neural network model for software effort estimation," *Expert Syst. Appl.*, vol. 35, no. 3, pp. 929–937, 2008.

- [33] A. Venkatachalam, "Software cost estimation using artificial neural networks," in *Proceedings of international joint conference on neural networks*, 1993, pp. 987–990.
- [34] G. Wittig and G. Finnie, "Estimating software development effort with connectionist models," *Information and Software Technology*, vol. 39, no. 7, pp. 469–476, 1997.
- [35] Y.-C. H. Arthur Earl Bryson, *Applied optimal control: Optimization, estimation, and control*. New York: Hemisphere Pub. Corp., 1969.
- [36] D. E. Rumelhart, L. McClelland, James, and the PDP Research Group, *Parallel distributed processing: explorations in the microstructure vol. 2*. MIT Press, 1986.
- [37] B. Kitchenham and E. Mendes, "Why comparative effort prediction studies may be invalid," in *PROMISE '09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. New York, NY, USA: ACM, 2009, pp. 1–5.
- [38] M. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions On Knowledge And Data Engineering*, vol. 15, no. 6, pp. 1437–1447, 2003.
- [39] P. N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison Wesley, 2005.
- [40] J. C. Gower and P. Legendre, "Metric and euclidean properties of dissimilarity coefficients," *Journal of Classification*, vol. 3, no. 48, 1986.
- [41] C. Chang, "Finding prototypes for nearest neighbor classifiers," *IEEE Trans. on Computers*, pp. 1179–1185, 1974.
- [42] J. Gama and C. Pinto, "Discretization from data streams: applications to histograms and data mining," in *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*. New York, NY, USA: ACM Press, 2006, pp. 662–667, available from <http://www.liacc.up.pt/~jgama/IWKDDS/Papers/p6.pdf>.
- [43] U. M. Fayyad and I. H. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1993, pp. 1022–1027.
- [44] Y. Yang and G. I. Webb, "A comparative study of discretization methods for naive-bayes classifiers," in *Proceedings of PKAW 2002: The 2002 Pacific Rim Knowledge Acquisition Workshop*, 2002, pp. 159–173.
- [45] E. Frank, M. Hall, and B. Pfahringer, "Locally weighted naive bayes," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 2003, pp. 249–256.
- [46] L. Angelis and I. Stamelos, "A simulation tool for efficient analogy based cost estimation," *Empirical Softw. Engg.*, vol. 5, no. 1, pp. 35–68, 2000.
- [47] J. R. Quinlan, "Learning with Continuous Classes," in *5th Australian Joint Conference on Artificial Intelligence*, 1992, pp. 343–348, available from <http://citeseer.nj.nec.com/quinlan92learning.html>.
- [48] Y. Li, M. Xie, and G. T., "A study of the non-linear adjustment for analogy based software cost estimation," *Empirical Software Engineering*, pp. 603–643, 2009.
- [49] U. Lipowezky, "Selection of the optimal prototype subset for 1-nn classification," *Pattern Recognition Letters*, vol. 19, pp. 907–918, 1998.
- [50] C. Kirsopp and M. Shepperd, "Making inferences with small numbers of training sets," *IEEE Proc.*, vol. 149, 2002.
- [51] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and validity in comparative studies of software prediction models," *IEEE Transactions on Software Engineering*, vol. vol, pp. 31no5pp380–391, May 2005.
- [52] D. Beeferman and A. Berger, "Agglomerative clustering of a search engine query log," in *Knowledge Discovery and Data Mining*, pp. 407–416, 2000.
- [53] S. Guha, R. Rastogi, and K. S. Cure, "An efficient clustering algorithm for large databases," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, vol. pages, pp. 73–84, 1998.
- [54] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein, "Cluster analysis and display of genome-wide expression patterns," *Proc. of the National Academy of Science*, vol. 95, pp. 14863–14868, 1998.
- [55] M. Steinbach, G. Karypis, and V. Kumar, "A comparison of document clustering techniques," in *KDD Workshop on Text Mining*, 2000.
- [56] E. P. Kasten and P. K. McKinley, "Meso: Supporting online decision making in autonomic computing systems," *IEEE Trans. on Knowl. and Data Eng.*, vol. 19, no. 4, pp. 485–499, 2007.
- [57] R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1992, ISBN: 1558602380.

- [58] B. A. Kitchenham, E. Mendes, and G. H. Travassos, "Cross- vs. within-company cost estimation studies: A systematic review," *IEEE Transactions on Software Engineering*, pp. 316–329, May 2007.
- [59] J. Klijnen, "Sensitivity analysis and related analyses: a survey of statistical techniques," *Journal Statistical Computation and Simulation*, vol. 57, no. 1–4, pp. 111–142, 1997.
- [60] T. Menzies and S. Goss, "Vague models and their implications for the kbs design cycle," in *Proceedings PKAW '96: Pacific Knowledge Acquisition Workshop and Monash University Department of Software Development Technical Report TR96-15*, 1996, available from <http://menzies.us/pdf/96abmod.pdf>.
- [61] M. Shepperd, "Personnel communication on the value of different evaluation criteria," 2007.
- [62] S. Chulani, B. Boehm, and B. Steece, "Bayesian analysis of empirical software engineering cost models," *IEEE Trans. Softw. Eng.*, vol. 25, no. 4, pp. 573–583, 1999.
- [63] G. Boetticher, "When will it be done? the 300 billion dollar question, machine learner answers," *IEEE Intelligent Systems*, June 2003.
- [64] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrvtveit, "A simulation study of the model evaluation criterion mmre," *IEEE Transactions on Software Engineering*, vol. 29, no. 11, pp. 985 – 995, November 2003.
- [65] I. Myrvtveit, E. Stensrud, and M. Shepperd, "Reliability and validity in comparative studies of software prediction models," *IEEE Transactions on Software Engineering*, vol. 31, no. 5, pp. 380–391, May 2005.
- [66] E. Alpaydin, *Introduction to Machine Learning*. MIT Press, 2004.
- [67] C. Robson, "Real world research: a resource for social scientists and practitioner-researchers," *Blackwell Publisher Ltd*, 2002.
- [68] Y. Wang, Q. Song, S. MacDonell, M. Shepperd, and J. Shen, "Integrate the gm(1,1) and verhulst models to predict software stage-effort," *IEEE Transactions on Systems*, vol. 39, pp. 647 – 658, 2009.
- [69] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrvtveit, "A simulation study of the model evaluation criterion mmre," *IEEE Transactions on Software Engineering*, 2003.
- [70] L. C. Briand, K. El Emam, D. Surmann, I. Wiczorek, and K. D. Maxwell, "An assessment and comparison of common software cost estimation modeling techniques," in *ICSE '99: Proceedings of the 21st international conference on Software engineering*. New York, NY, USA: ACM, 1999, pp. 313–322.
- [71] D. Milic and C. Wohlin, "Distribution patterns of effort estimations," in *Euromicro*, 2004.
- [72] A. Bakir, B. Turhan, and A. Bener, "A new perspective on data homogeneity in software cost estimation: A study in the embedded systems domain," *Software Quality Journal*, 2009. [Online]. Available: <http://dx.doi.org/10.1007/s11219-009-9081-z>

APPENDIX

With the exception of ISBSG-Banking and SDR, all the data used in this study is available at <http://promisedata.org/data> or from the authors. As shown in Figure 4, our data includes:

- Data from the International Software Benchmarking Standards Group (ISBSG);
- The Desharnais and Albrecht data sets;
- SDR, which is data from projects of various software companies from Turkey. SDR is collected from Softlab, the Bogazici University Software Engineering Research Laboratory repository [72];
- And the standard COCOMO data sets (Cocoma*, Nasa*).

Projects in ISBSG dataset can be grouped according to their business domains. In previous studies, breakdown of ISBSG according to business domain has also been used [72]. Among different business domains we selected banking due to:

1. Banking domain includes many projects whose data quality is reported to be high (ISBSG contains projects with missing attribute values).
2. ISBSG Banking domain is the dataset we have analyzed and worked for a long time due to our hands on experience in building effort estimation models in banking industry.

We denote the banking domain subset of ISBSG as “ISBSG-Banking”.

Note that two of these data sets (Nasa93c2, Nasa93c5) come from different development centers around the United States. Another two of these data sets (Cocomo81e, Cocomo81o) represent different kinds of projects:

- The Cocomo81e “embedded projects” are those developed within tight constraints (hardware, software, operational, ...);
- The Cocomo81o “organic projects” come from small teams with good experience of working with less than rigid requirements.

Note also in Figure 4, the skewness of our effort values (2.0 to 4.4): our datasets are extremely heterogeneous with as much as 40-fold variation. There is also some divergence in the features used to describe our data:

- While our data includes some effort value (measured in terms of months or hours), no other feature is shared by all data sets.
- The Cocomo* and NASA* data sets all use the features defined by Boehm [1]; e.g. analyst capability, required software reliability, memory constraints, and use of software tools.
- The other data sets use a wide variety of features including, number of entities in the data model, number of basic logical transactions, query count and number of distinct business units serviced.

Next Generation “Treatment Learning”

(finding the diamonds in the dust)

Tim Menzies

Lane Dept CS & EE, WVU

`tim@menzies.us`

The strangest thing...

“In any field, find the strangest thing, and explore it” – John Wheeler

■ Q: How have dummies (like me) managed to gain (some) control over a (seemingly) complex world?

■ A: The world is simpler than we think.

- ◆ Models contain clumps
- ◆ A few collar variables decide which clumps to use.

■ TAR2, TAR3, TAR4:

- ◆ Data miners that assume clumps/collars
- ◆ Reports effects never seen before
- ◆ Finds solutions faster than other methods
- ◆ Returns tiniest theories
- ◆ Scales to infinite data streams (⇐ new result)



Introduction

● The strangest thing...

- Complex Models?
- Exploiting Simplicity
- Different learners
- Why Learn Small Theories?
- Definition

In practice...

Scaling Up

Related Work

And so...

Questions? Comments?

How Complex are our Models?

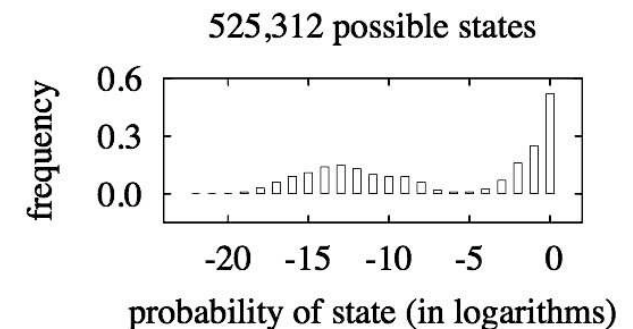
■ COLLARS-

A small number few variables controls the rest:

- ◆ DeKleer [1986]: “Minimal environments” in the ATMS;
- ◆ Menzies and Singh [2003]: “Tiny minimal environments”;
- ◆ Crawford and Baker [1994]: “Master variables” in scheduling;
- ◆ Williams et al. [2003]: “Backdoors” in satisfiability.

■ CLUMPS-

- ◆ Druzdzel [1994]. Commonly, a few states; very rarely, most states;
- ◆ Pelanek [2004]. “Straight jackets” in formal models: state spaces usually sparse, small diameter, many diamonds.



25,000 states in IEEE1394

Introduction

● The strangest thing...

● **Complex Models?**

● Exploiting Simplicity

● Different learners

● Why Learn Small Theories?

● Definition

In practice...

Scaling Up

Related Work

And so...

Questions? Comments?

Exploiting Simplicity

Introduction

- The strangest thing...
- Complex Models?
- Exploiting Simplicity
- Different learners
- Why Learn Small Theories?
- Definition

In practice...

Scaling Up

Related Work

And so...

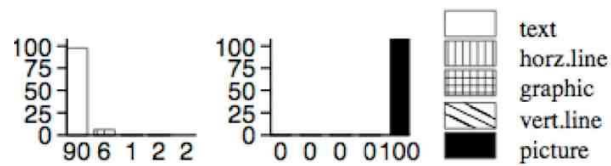
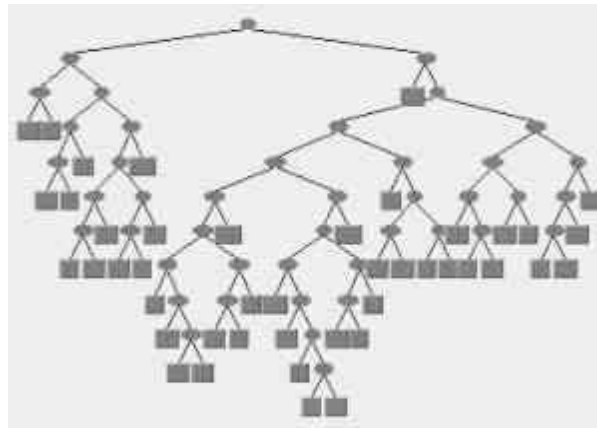
Questions? Comments?

- If clumps
 - ◆ most of the action in a small number of states
 - ◆ effective search space = small
- If collars:
 - ◆ A few variables that switch you between states
- Treatment learning
 - ◆ If a few variables control the rest..
 - All paths *inputs* \rightarrow *outputs* use the collars (by definition).
 - ◆ So don't search for the collars:
 - They'll find you.
 - Just sample, and count frequencies F .
 - ◆ Divide output *good* and *bad*
 - Focus on ranges R_i with large $\frac{F(R_i|good)}{F(R_i|bad)}$
- Great way to learn tiny theories.



Learns Smaller Theories

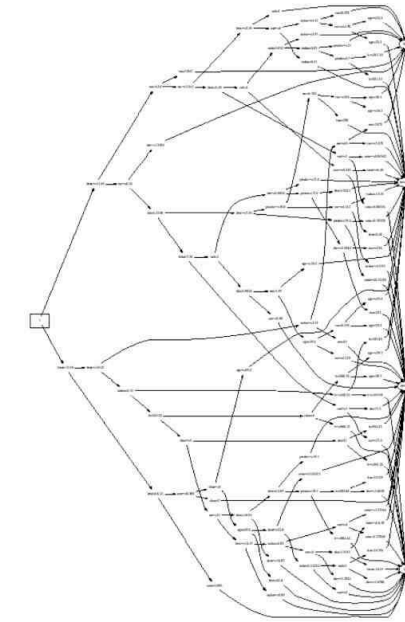
find graphics on a page from 11 features



$$34 \leq \text{height} < 86 \wedge$$

$$3.9 \leq \text{mean_tr} < 9.5$$

find good housing in Boston



$$6.7 \leq RM < 9.8 \wedge$$

$$12.6 \leq PTRATION < 15.9$$

Introduction

- The strangest thing...
- Complex Models?
- Exploiting Simplicity
- Different learners
- Why Learn Small Theories?
- Definition

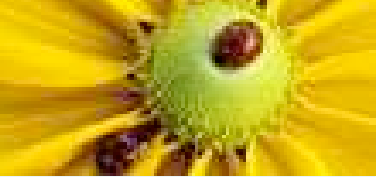
In practice...

Scaling Up

Related Work

And so...

Questions? Comments?



Why Learn Small Theories?

Introduction

- The strangest thing...
- Complex Models?
- Exploiting Simplicity
- Different learners
- Why Learn Small Theories?
- Definition

In practice...

Scaling Up

Related Work

And so...

Questions? Comments?

Reduce Uncertainty:

Linear regression: $\sigma^2 \propto |variables|$ (Miller [2002]);

“Pluralitas non est ponenda sine neccesitate”:

MDL (Wallace and Boulton [1968]); FSS (Hall and Holmes [2003])

Explanation:

Smaller theories are easier to explain (or audit).

Performance:

The simpler the target concept, the faster the learning.

Construction cost:

Need fewer sensors and actuators.

Operations cost:

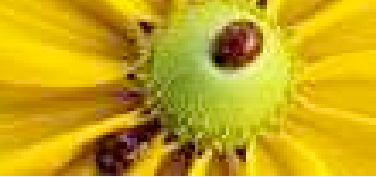
Less to do: important for manual procedures;

Less to watch: important for data-intensive tasks like security monitoring.

Pruning is good modeling:

Real world data often has noisy, irrelevant, redundant variables.

So What is Treatment Learning?



Introduction

- The strangest thing...
- Complex Models?
- Exploiting Simplicity
- Different learners
- Why Learn Small Theories?

● Definition

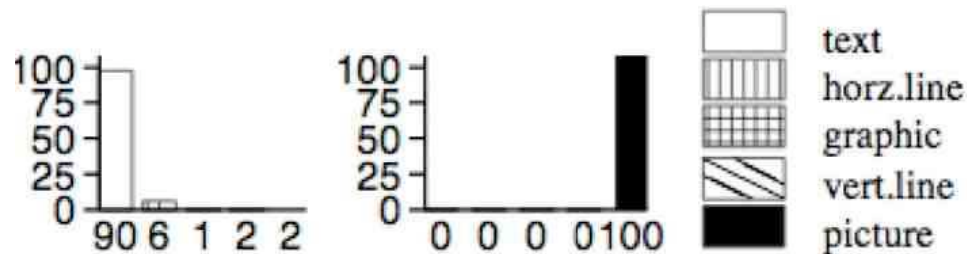
In practice...

Scaling Up

Related Work

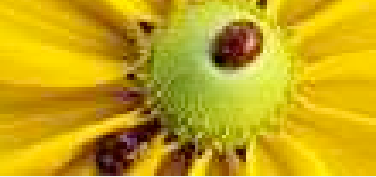
And so...

Questions? Comments?



$$34 \leq height < 86 \wedge 3.9 \leq mean_tr < 9.5$$

- E : training data with examples of $R_i \rightarrow C$
 - ◆ R_i : attribute ranges
 - ◆ C : classes with utilities $\{U_1 < U_2 < .. < U_C\}$
 - ◆ $F_1\%, F_2\%, \dots, F_C\%$: frequencies of C in E
- T treatment of size X : $\{R_1 \wedge R_2 \dots \wedge R_X\}$;
 - ◆ $T \cap E \rightarrow e \subseteq E$ with frequencies $f_1\%, f_2\%, \dots, f_C\%$
 - ◆ seek smallest T with largest $lift = (\sum_C U_C f_C) / (\sum_C U_C F_C)$
- This talk:
 - ◆ Implementation, examples, a new scale-up method



[Introduction](#)

[In practice...](#)

- [Algorithm](#)
- [Saving the World](#)
- [Compare](#)

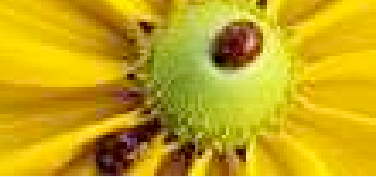
[Scaling Up](#)

[Related Work](#)

[And so...](#)

[Questions? Comments?](#)

In practice...



The TAR3 Treatment Learner

Introduction

In practice...

● Algorithm

● Saving the World

● Compare

Scaling Up

Related Work

And so...

Questions? Comments?

- Assume clumps and collars
 - ◆ Just thrash around some.
- Build treatments
 - $\{R_1 \wedge R_2 \dots \wedge R_X\}$ of size X
 - ◆ FIRST try $X = 1$
 - ◆ THEN use the $X = 1$ results to guide the $X > 1$ search.
- Hu [2002] :: grow *treatments* via a stochastic search.
 - ◆ Discretization: equal frequency binning
- Empirically:
 - ◆ Run times linear on treatment SIZE, number of examples
 - ◆ Works as well as TAR2's complete search

```
function ONE(x = random(SIZE) )
  x timesDo
    treatment = treatment + ANYTHING()
  return treatment

function ANYTHING()
  return a random range from CDF(lift1)

function SOME()
  REPEATS timesDo
    treatments = treatments + ONE()
  sort treatments on lift
  return ENOUGH top items

function TAR3(lives = LIVES )
  for every range r do lift1[r]= lift(r)
  repeat
    before = size(temp)
    temp = union(temp, SOME())
    if (before==size(temp))
      then lives--
    else lives = LIVES
  until lives == 0
  sort temp on lift;
  return ENOUGH top items
```

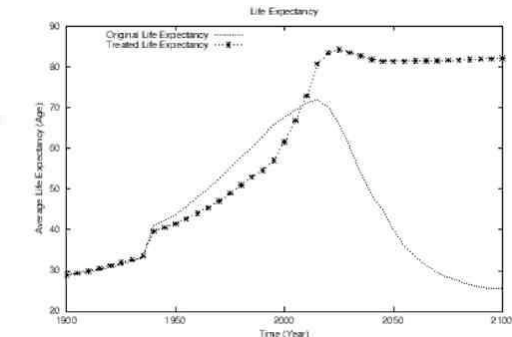
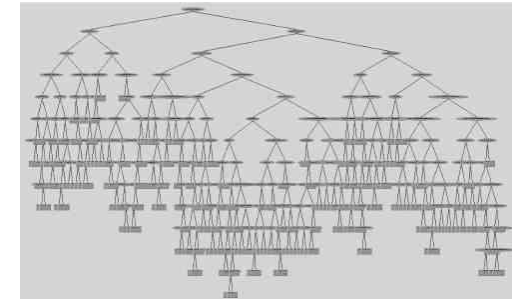
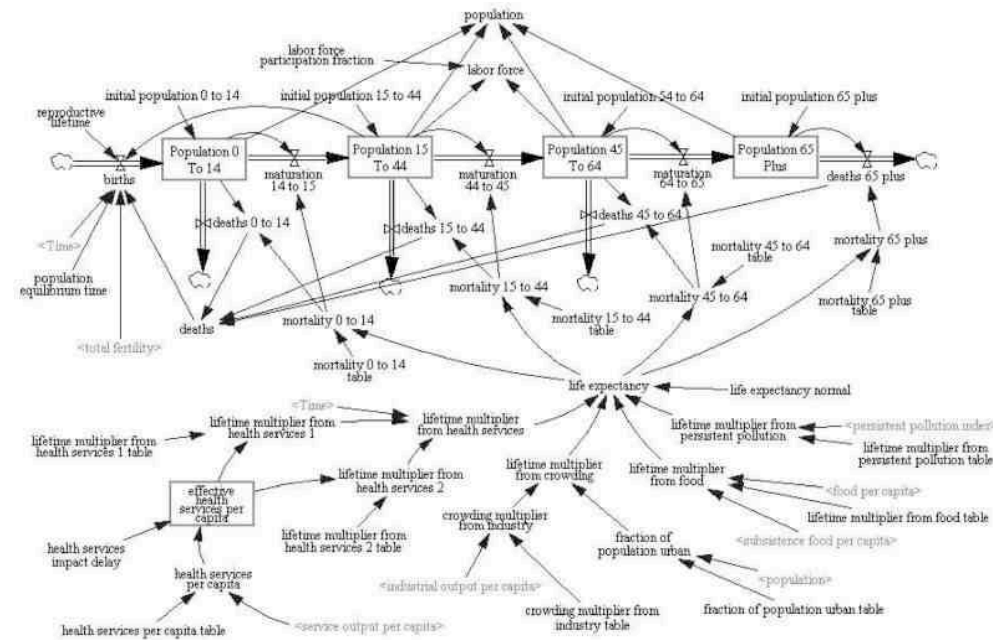
Useful defaults: <SIZE=10, REPEATS=100, ENOUGH=20, LIVES=5>

Saving the World

“Limits to Growth” :: Meadows et al. [1972]

A second look at “Limits to Growth”: Geletko and Menzies [2003]

Vensim’s World-3 (1991): 295 variables



Happily ever after if

- family size ≤ 2 , menstruation onset > 18 , industrial capital output = [3..5).
- This happy ending is *not* mentioned in Meadows et al. [1972].

Introduction

In practice...

- Algorithm
- Saving the World
- Compare

Scaling Up

Related Work

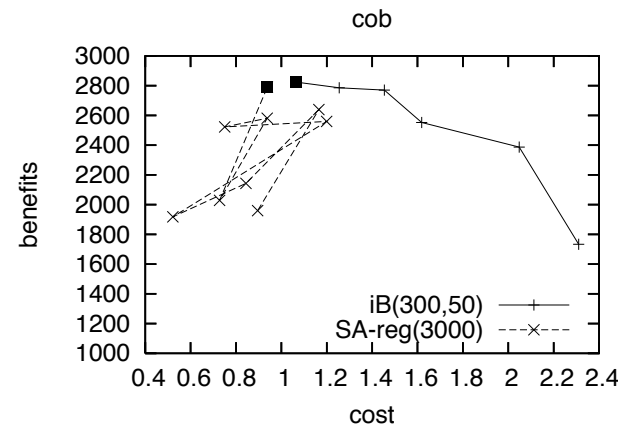
And so...

Questions? Comments?

Compared with More Complete Search

- DDP requirements models from deep-space missions (from JPL).
- Iterative learning: $simulation_i \rightarrow learn \rightarrow constrain \rightarrow simulation_{i+1}$

$$SA = \frac{\frac{benefit}{maxBenefit} + \left(1 - \frac{cost}{maxCost}\right)}{\left(2 * \begin{array}{c} \text{number of} \\ \text{selected mitigations} \end{array}\right) + 1}$$



TAR3: 7*300 samples

SA: 9*3000 samples

Introduction

In practice...

● Algorithm

● Saving the World

● Compare

Scaling Up

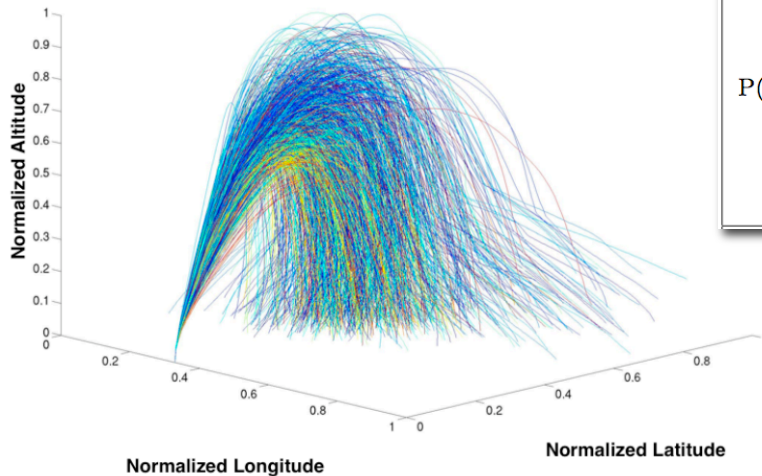
Related Work

And so...

Questions? Comments?

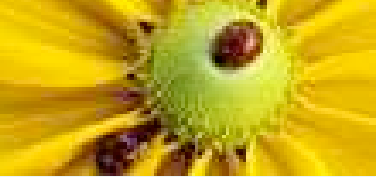
9 years later: TARZAN → TAR2, TAR3, TAR4.1

- TARZAN is no longer a post-processor
 - Branch queries performed directly on discretized data
 - thanks David Poole
 - Stochastic sampling for rule generation
- Benchmarked against state-of-the-art numerical optimizers for GNC control



Metric	Project 1			
	Rank	Program	50%	Quartiles
Runtime	1	TAR4.1	0.13	
	2	TAR3	0.31	
	3	QN	6	
	4	SA-T4	15	
	4	SA-T3	16	
Recall	Rank	Program	50%	Quartiles
	1	TAR4.1	59	
	1	QN	36	
	2	SA-T4	25	
	3	TAR3	22	
P(False Alarm)	Rank	Program	50%	Quartiles
	1	TAR3	1	
	2	SA-T3	9	
	3	TAR4.1	25	
	4	QN	34	
4	SA-T4	71		

Still generating tiny rules
(very easy to read, explain, audit, implement)



[Introduction](#)

[In practice...](#)

Scaling Up

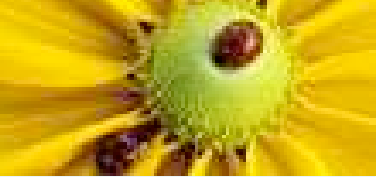
- TAR3 is not a Data Miner
- SAWTOOTH
- NaïveBayes classifiers
- CUBE & TAR4
- Why did TAR4.0 fail?
- TAR4.1
- Pre-condition
- Typical values
- TAR4.1 Works
- So What?
- But Why Big Treatments?

[Related Work](#)

[And so...](#)

[Questions? Comments?](#)

Scaling Up



TAR3 is not a Data Miner

Introduction

In practice...

Scaling Up

● TAR3 is not a Data Miner

- SAWTOOTH
- NaïveBayes classifiers
- CUBE & TAR4
- Why did TAR4.0 fail?
- TAR4.1
- Pre-condition
- Typical values
- TAR4.1 Works
- So What?
- But Why Big Treatments?

Related Work

And so...

Questions? Comments?

The data mining desiderata :: Bradley et al. [1998]:

- Requires one scan, or less of the data
- On-line, anytime algorithm
- Suspend-able, stoppable, resumable
- Efficiently and incrementally add new data to existing models
- Works within the available RAM

TAR3 is *not* a data miner

- Stores all examples in RAM
- Requires at three scans
 1. discretization
 2. collect statistics, build treatments
 3. rank generated theories

SAWTOOTH is a data miner

Introduction

In practice...

Scaling Up

● TAR3 is not a Data Miner

● SAWTOOTH

● NaïveBayes classifiers

● CUBE & TAR4

● Why did TAR4.0 fail?

● TAR4.1

● Pre-condition

● Typical values

● TAR4.1 Works

● So What?

● But Why Big Treatments?

Related Work

And so...

Questions? Comments?

SAWTOOTH= incremental NaïveBayes classifier Menzies and Orrego [2005]

■ Exploits the “saturation effect”:

- ◆ Learners performance improves and plateaus, after 100s of examples
- ◆ Processes data in chunks (window = 250)
- ◆ Disables learning while performance stable

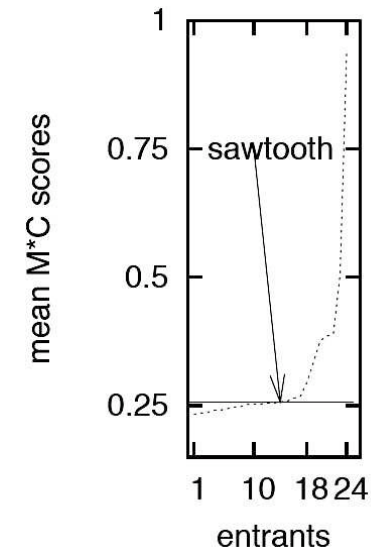
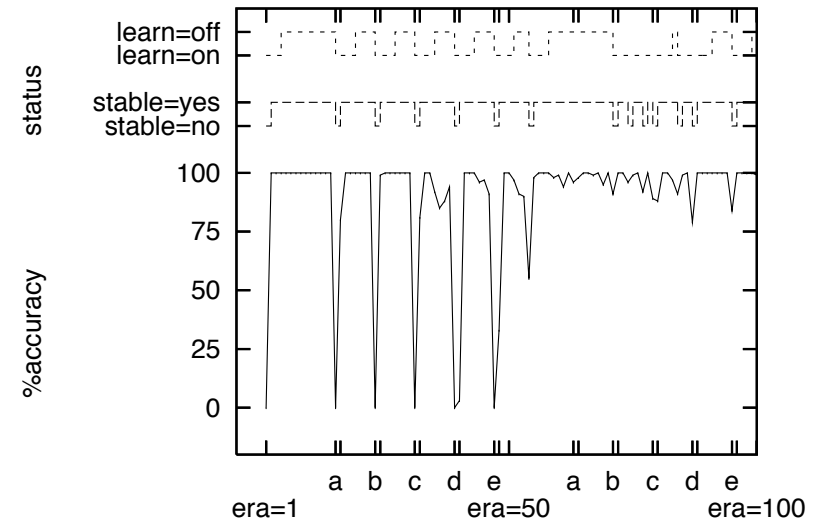
■ One-pass through the data

- ◆ Incremental discretization of numeric data (SPADE)
- ◆ Input each example, converted to frequency counts, then deletes

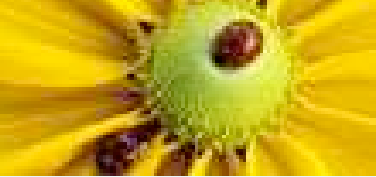
■ Results

- ◆ Small memory; scales.
- ◆ Recognizes and reacts to concept drift

■ Can we model treatment learning as a NaïveBayes classifier?



NaïveBayes classifiers



evidence E , hypothesis H

$$P(H|E) = \underbrace{\left(\prod_i P(E_i|H) \right)}_{\text{now*}} * \underbrace{\frac{P(H)}{P(E)}}_{\text{past}}$$

	E_1	E_2	E_3
$H = car$	job	suburb	wealthy?
ford	tailor	NW	y
ford	tailor	SE	n
ford	tinker	SE	n
bmw	tinker	NW	y
bmw	tinker	NW	y
bmw	tailor	NW	y

P(H)	$P(E_i H)$		
	job	suburb	wealthy?
ford:3=0.5	tinker:1=0.33 tailor:2=0.67	NW:1=0.33 SE:2=0.67	y:1=0.33 n:2=0.67
bmw:3=0.5	tinker:2=0.67 tailor:1=0.33	NW:3=1.00 SE:0=0.00	y:3=1.00 n:0=0.00

- $E = \text{job=tailor \& suburb=NW}$
- likelihood = $L(bmw|E) = \prod_i P(E|bmw) * P(bmw) = 0.33*1.00*0.5 = 0.16500$
- $L(ford|E) = \prod_i P(E|ford) * P(ford) = 0.67*0.33*0.5 = 0.11055$
- $Prob(bmw|E) = \frac{L(bmw|E)}{L(bmw|E)+L(ford|E)} = 59.9\%$
- $Prob(ford|E) = \frac{L(ford|E)}{L(bmw|E)+L(ford|E)} = 40.1\%$
- So our tailor drives a *bmw*
- Naïve: assumes independence; counts single attribute ranges (not combinations)
 - ◆ But optimal under the one-zero assumption Domingos and Pazzani [1997].
 - ◆ Incremental simple, fast learning/classification speed, low storage space.

Introduction

In practice...

Scaling Up

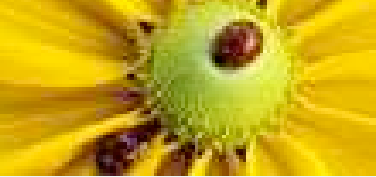
- TAR3 is not a Data Miner
- SAWTOOTH
- NaïveBayes classifiers
- CUBE & TAR4
- Why did TAR4.0 fail?
- TAR4.1
- Pre-condition
- Typical values
- TAR4.1 Works
- So What?
- But Why Big Treatments?

Related Work

And so...

Questions? Comments?

CUBE & TAR4



Introduction

In practice...

Scaling Up

- TAR3 is not a Data Miner
- SAWTOOTH
- NaïveBayes classifiers
- CUBE & TAR4
- Why did TAR4.0 fail?
- TAR4.1
- Pre-condition
- Typical values
- TAR4.1 Works
- So What?
- But Why Big Treatments?

Related Work

And so...

Questions? Comments?

outlook	U_1 : minimize temperature	humidity	windy	U_2 : maximize play	up_i	$down_i$
overcast	64	65	TRUE	yes=1	1.00	0
rainy	68	80	FALSE	yes=1	0.87	0.13
...
sunny	80	90	TRUE	no=0	0.11	0.89
sunny	85	85	FALSE	no=0	0.00	1

- Examples are placed in a U -dimensional hypercube (one dimension for each utility):

- ◆ apex = best = $\{1,1,1,1,\dots\}$;
- ◆ base = worst = $\{0,0,0,0,\dots\}$

- $example_i$ has distance $0 \leq D_i \leq 1$ from apex (normalized by $U^{0.5}$)

- Each range $R_j \in example_i$ adds

$down_i = D_i$ and $up_i = 1 - D_i$ to $F(R_j|base)$ and $F(R_j|apex)$.

$$P(apex) = \frac{\sum_i up_i}{(\sum_i up_i + \sum_i down_i)}$$

$$P(base) = \frac{\sum_i down_i}{(\sum_i up_i + \sum_i down_i)}$$

$$P(R_j|apex) = F(R_j|apex) / \sum_i up_i$$

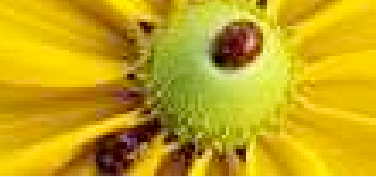
$$P(R_j|base) = F(R_j|base) / \sum_i down_i$$

$$L(apex|R_k \wedge R_l \wedge \dots) = \prod_x P(R_x|apex) * P(apex)$$

$$L(base|R_k \wedge R_l \wedge \dots) = \prod_x P(R_x|base) * P(base)$$

TAR4.0: Bayesian treatment learner = find the *smallest* treatment T that *maximizes*:

$$P(apex|T) = \frac{L(apex|T)}{L(apex|T) + L(base|T)} \quad ; \text{ didn't work: out-performed by TAR3}$$



Why did TAR4.0 fail?

Introduction

In practice...

Scaling Up

- TAR3 is not a Data Miner
- SAWTOOTH
- NaïveBayes classifiers
- CUBE & TAR4
- Why did TAR4.0 fail?
- TAR4.1
- Pre-condition
- Typical values
- TAR4.1 Works
- So What?
- But Why Big Treatments?

Related Work

And so...

Questions? Comments?

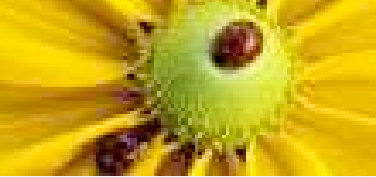
- Hypothesis: muddled-up by dependent attributes;
- “Naïve” Bayes: assume independence, keeps *singleton* counts.

	E_1	E_2	E_3
$H = car$	job	suburb	wealthy?
ford	tailor	NW	y
ford	tailor	SE	n
ford	tinker	SE	n
bmw	tinker	NW	y
bmw	tinker	NW	y
bmw	tailor	NW	y

E	P(bmw E)	P(ford E)
$job = tailor \& suburb = NW$	59.9%	40.1%
$job = tailor \& suburb = NW \& wealthy = y$	81%	19.0%

- Adding redundant information radically changes probabilities? Bad!
- Note: gets class probabilities WRONG, but RANKS classes correctly Domingos and Pazzani [1997]
- We asked TAR4.0 to do what you must never do:
 - ◆ compare numeric of probabilities of the same class in NaïveBayes.

TAR4.1



Introduction

In practice...

Scaling Up

● TAR3 is not a Data Miner

● SAWTOOTH

● NaïveBayes classifiers

● CUBE & TAR4

● Why did TAR4.0 fail?

● TAR4.1

● Pre-condition

● Typical values

● TAR4.1 Works

● So What?

● But Why Big Treatments?

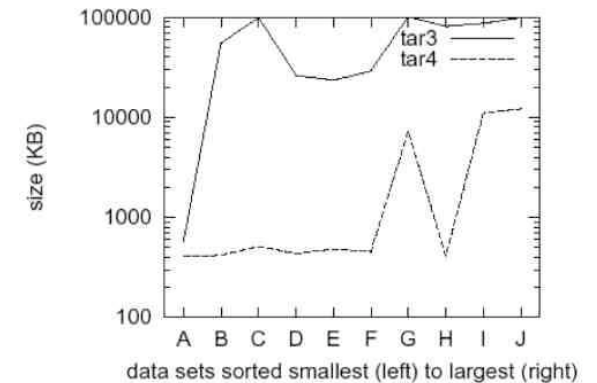
Related Work

And so...

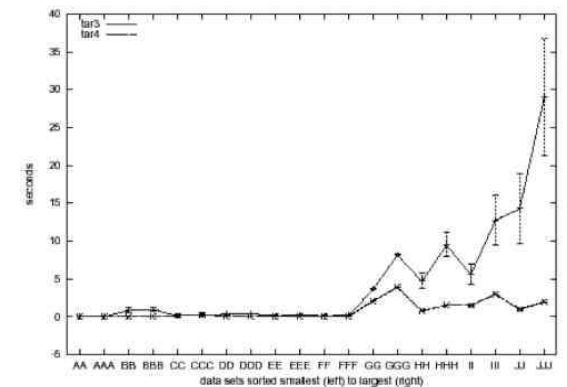
Questions? Comments?

- Prune treatments with low support in the data.
- What does “support” mean?
 - ◆ Maximal when includes all examples from a class
 - ◆ $0 \leq support \leq 1$
 - ◆ $support = likelihood = \prod_x P(R_x | H) * P(H)$
- $probability * support = \frac{L(apex|E)^2}{L(apex|E) + L(base|E)}$
- Worked!
 - ◆ Much faster, less memory than TAR3:
 - No need for a second scan
 - No need to hold examples in RAM
 - ◆ Bayesian guess-timate for *support* of best class (almost) the same as TAR3
 - ◆ No connection treatment size to guess-timate error.
- But why did it work so well?

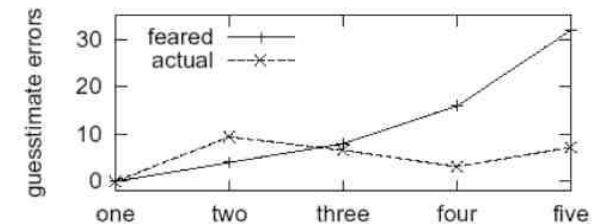
less memory

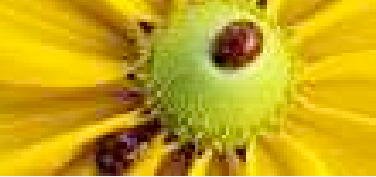


faster, less variance



lift errors small





When Won't Dependencies Confuse TAR4?

Introduction

In practice...

Scaling Up

- TAR3 is not a Data Miner
- SAWTOOTH
- NaïveBayes classifiers
- CUBE & TAR4
- Why did TAR4.0 fail?
- TAR4.1
- **Pre-condition**
- Typical values
- TAR4.1 Works
- So What?
- But Why Big Treatments?

Related Work

And so...

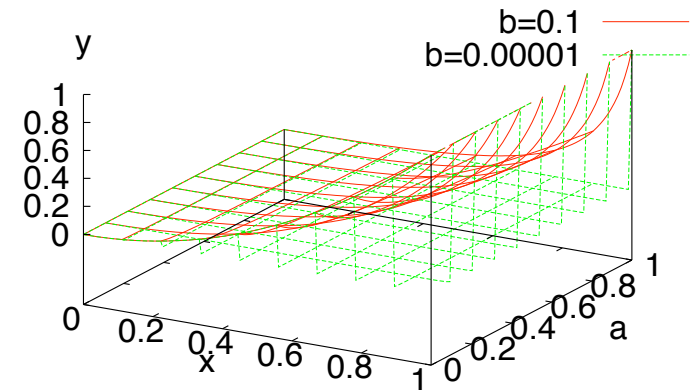
Questions? Comments?

- $T' = T + t$ where t is an attribute dependent on members of T ;
- TAR4.1 *not* confused by t when it ignores treatments that use it.

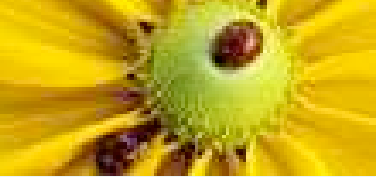
$$\begin{aligned}
 a &= L(\text{apex}|T') = \overbrace{P(t|\text{apex})}^x * \prod_i P(T_i|\text{apex}) * P(\text{apex}) \\
 b &= L(\text{base}|T') = \underbrace{P(t|\text{base})}_y * \prod_i P(T_i|\text{base}) * P(\text{base})
 \end{aligned}$$

- Then when is *support * probability* increased by ignoring x and y ?

$$\left(\begin{array}{c} \text{ignoring } x \text{ and } y \\ \frac{(a/x)^2}{a/x + b/y} \end{array} > \begin{array}{c} \text{using } x \text{ and } y \\ \frac{a^2}{a + b} \end{array} \right) \implies y > \frac{bx^2}{b + a - xa}$$



- And for TAR4.0:s pre-condition for no confusion: $\frac{(a/x)}{a/x + b/y} > \frac{a}{a + b}$



Typical Values and Constraints:: $\frac{(a/x)^2}{a/x+b/y}$

$0 < i \leq 20$; treatment size

$b < a$; apex is better than base

$10^{-10} < x \leq y \leq 0.25$; see graphs

$0 < a \leq x^i \leq x \leq 0.25$; a combines many x-like numbers

$0 < b \leq y^i \leq y \leq 0.25$; b combines many y-like numbers

Introduction

In practice...

Scaling Up

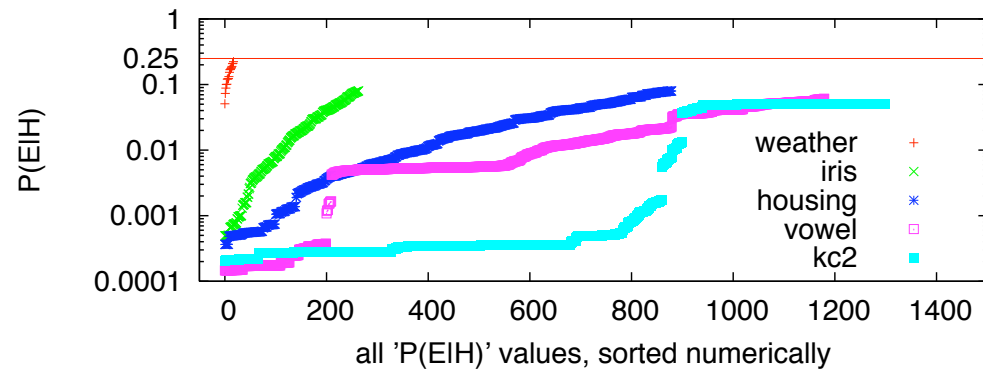
- TAR3 is not a Data Miner
- SAWTOOTH
- NaïveBayes classifiers
- CUBE & TAR4
- Why did TAR4.0 fail?
- TAR4.1
- Pre-condition
- Typical values
- TAR4.1 Works
- So What?
- But Why Big Treatments?

Related Work

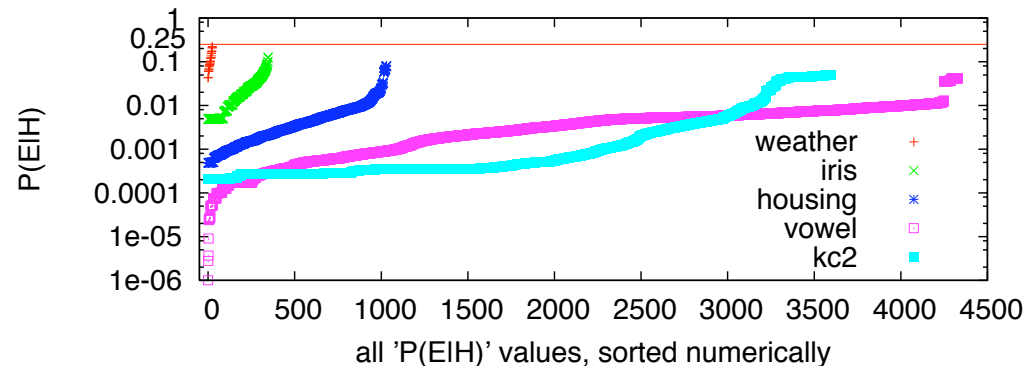
And so...

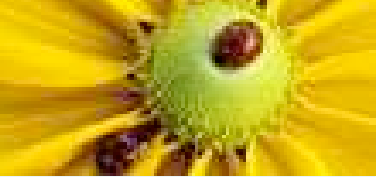
Questions? Comments?

three bins



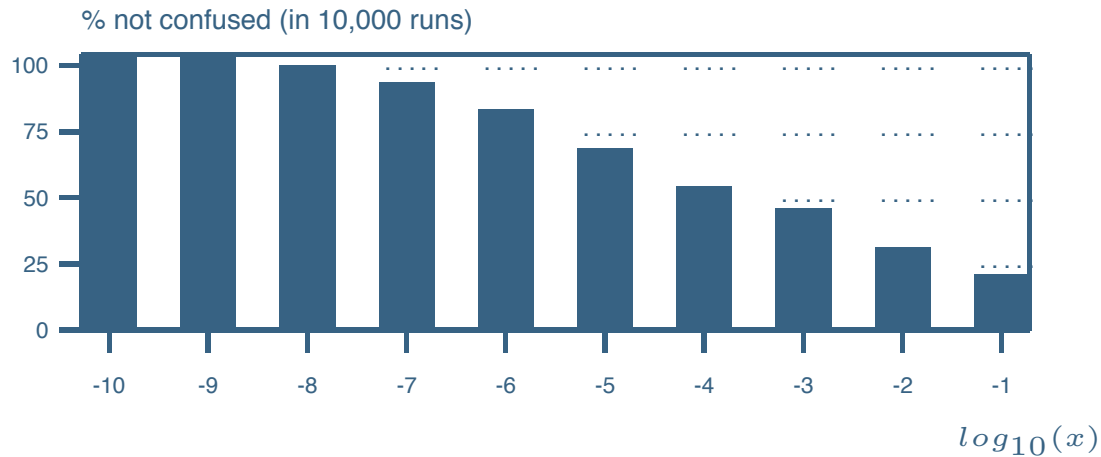
bin logging





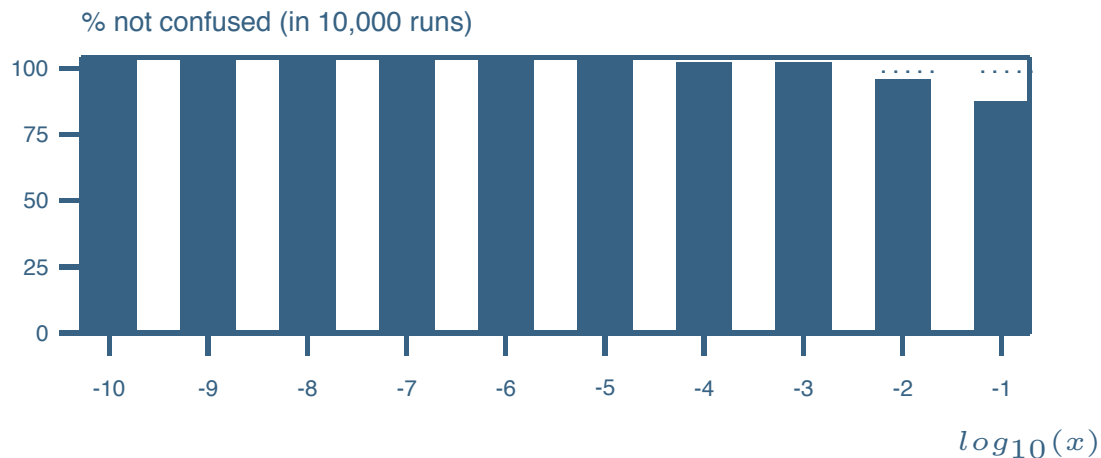
TAR4.1 Works

- Pick $\{a,b,x,y,i\}$ at random within typical values; reject those violate our constraints;
- Check pre-conditions; report rounded \log_{10} values;
- TAR4.0: not confused when $\left(\frac{(a/x)}{a/x+b/y} > \frac{a}{a+b} \right)$



Often confused.

- TAR4.1: not confused when $\left(\frac{(a/x)^2}{a/x+b/y} > \frac{a^2}{a+b} \right)$



Rarely confused.

Introduction

In practice...

Scaling Up

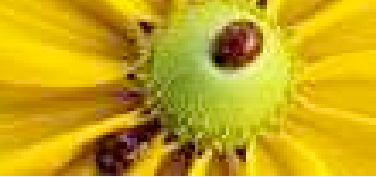
- TAR3 is not a Data Miner
- SAWTOOTH
- NaïveBayes classifiers
- CUBE & TAR4
- Why did TAR4.0 fail?
- TAR4.1
- Pre-condition
- Typical values
- **TAR4.1 Works**
- So What?
- But Why Big Treatments?

Related Work

And so...

Questions? Comments?

So What?



Introduction

In practice...

Scaling Up

- TAR3 is not a Data Miner
- SAWTOOTH
- NaïveBayes classifiers
- CUBE & TAR4
- Why did TAR4.0 fail?
- TAR4.1
- Pre-condition
- Typical values
- TAR4.1 Works
- So What?
- But Why Big Treatments?

Related Work

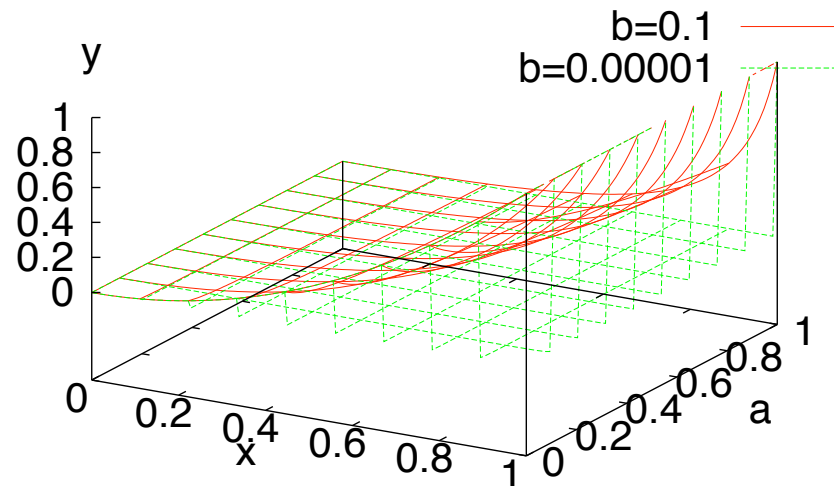
And so...

Questions? Comments?

- Mathematically, TAR4.0 will always fails (except for $x \ll 1$);
- TAR4.1 succeeds since pre-condition is usually satisfied
 - ◆ In 96.52% of our simulations
- So, theoretically and empirically:
 - ◆ Bayesian treatment learning with CUBE can guess effect of treatments using frequency counts,
 - ◆ Does not need a second scan of the data (providing you use *support * probability*)
 - ◆ Now we have a data miner TAR4.1.
- By the way,
 - ◆ No need for Bayes nets in this domain
 - ◆ Why doesn't this mean that treatments will *never* grow beyond size=1?

But Why Big Treatments?

- When are larger treatments acceptable; i.e. $\left(\frac{(a/x)^2}{a/x+b/y} < \frac{a^2}{a+b}\right)$?
- When is $y < \frac{bx^2}{b+a-xa}$.



- When x is large and y is much smaller than x
- i.e. when some attribute ranges has a high frequency in the apex *and* a much lower frequency in the base.
- If collars then such ranges are not common; i.e. dependencies unlikely.

Introduction

In practice...

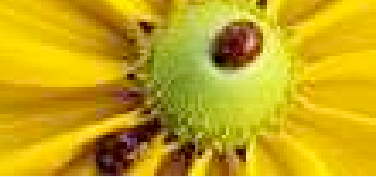
Scaling Up

- TAR3 is not a Data Miner
- SAWTOOTH
- NaïveBayes classifiers
- CUBE & TAR4
- Why did TAR4.0 fail?
- TAR4.1
- Pre-condition
- Typical values
- TAR4.1 Works
- So What?
- But Why Big Treatments?

Related Work

And so...

Questions? Comments?



[Introduction](#)

[In practice...](#)

[Scaling Up](#)

Related Work

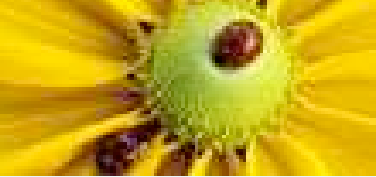
- [References](#)
- [References \(2\)](#)
- [References \(3\)](#)
- [References \(4\)](#)
- [References \(5\)](#)

[And so...](#)

[Questions? Comments?](#)

Related Work

References



Introduction

In practice...

Scaling Up

Related Work

● References

● References (2)

● References (3)

● References (4)

● References (5)

And so...

Questions? Comments?

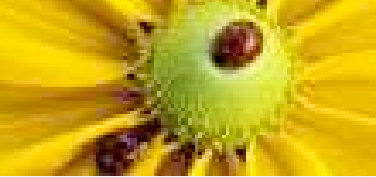
■ SAWTOOTH

- ◆ Tim Menzies and Andres Orrego. Incremental discretization and bayes classifiers handles concept drift and scaled very well. 2005. Submitted, IEEE TKDE, Available from <http://menzies.us/pdf/05sawtooth.pdf>

■ Treatment learning

- ◆ R. Clark. Faster treatment learning, 2005
- ◆ D. Geletko and T. Menzies. Model-based software testing via treatment learning. In *IEEE NASE SEW 2003*, 2003. Available from <http://menzies.us/pdf/03radar.pdf>
- ◆ Y. Hu. Treatment learning, 2002. Masters thesis, University of British Columbia, Department of Electrical and Computer Engineering. In preparation
- ◆ T. Menzies, R. Gunnalan, K. Appukutty, Srinivasan A, and Y. Hu. Learning tiny theories. In *International Journal on Artificial Intelligence Tools (IJAIT)*, to appear, 2005. Available from <http://menzies.us/pdf/03select.pdf>
- ◆ T. Menzies and Y. Hu. Just enough learning (of association rules): The TAR2 treatment learner. In *Artificial Intelligence Review (to appear)*, 2006. Available from <http://menzies.us/pdf/02tar2.pdf>
- ◆ T. Menzies and Y. Hu. Data mining for very busy people. In *IEEE Computer*, November 2003. Available from <http://menzies.us/pdf/03tar2.pdf>

References (2)



Introduction

In practice...

Scaling Up

Related Work

● References

● References (2)

● References (3)

● References (4)

● References (5)

And so...

Questions? Comments?

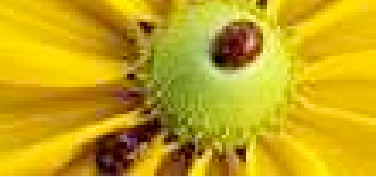
■ Phase transition

- ◆ H.H. Hoos and T. Stutzle. Evaluating las vegas algorithms - pitfalls and remedies. In *Proc. of UAI-98*, 1998. Available from <http://www.cs.ubc.ca/~hoos/Publ/uai98.ps>
- ◆ David G. Mitchell, Bart Selman, and Hector J. Levesque. Hard and easy distributions for SAT problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 459–465, Menlo Park, California, 1992. AAAI Press. Available from <http://www.citeseer.ist.psu.edu/mitchell92hard.html>

■ Contrast set learners

- ◆ S.B. Bay and M.J. Pazzani. Detecting change in categorical data: Mining contrast sets. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, 1999. Available from <http://www.ics.uci.edu/~pazzani/Publications/stucco.pdf>
- ◆ C.H. Cai, A.W.C. Fu, C.H. Cheng, and W.W. Kwong. Mining association rules with weighted items. In *Proceedings of International Database Engineering and Applications Symposium (IDEAS 98)*, August 1998. Available from http://www.cse.cuhk.edu.hk/~kdd/assoc_rule/paper.pdf

References (3)



Introduction

In practice...

Scaling Up

Related Work

● References

● References (2)

● References (3)

● References (4)

● References (5)

And so...

Questions? Comments?

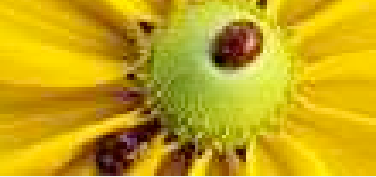
■ Collars and clumps

- ◆ J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI '94*, 1994
- ◆ J. DeKleer. An Assumption-Based TMS. *Artificial Intelligence*, 28:163–196, 1986
- ◆ M.J. Druzdzel. Some properties of joint probability distributions. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 187–194, 1994
- ◆ R. Pelanek. Typical structural properties of state spaces. In *Proceedings SPIN'04 Workshop*, 2004. Available from http://www.fi.muni.cz/~xpelanek/publications/state_spaces.ps
- ◆ R. Williams, C.P. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of IJCAI 2003*, 2003. <http://www.cs.cornell.edu/gomes/FILES/backdoors.pdf>

■ Data mining

- ◆ Paul S. Bradley, Usama M. Fayyad, and Cory Reina. Scaling clustering algorithms to large databases. In *Knowledge Discovery and Data Mining*, pages 9–15, 1998. Available from <http://citeseer.ist.psu.edu/bradley98scaling.html>
- ◆ P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000. URL citeseer.ist.psu.edu/domingos00mining.html

References (4)



Introduction

In practice...

Scaling Up

Related Work

● References

● References (2)

● References (3)

● References (4)

● References (5)

And so...

Questions? Comments?

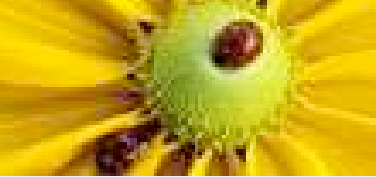
■ Feature subset selection

- ◆ M.A. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions On Knowledge And Data Engineering*, 15(6): 1437– 1447, 2003
- ◆ Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997. URL citeseer.nj.nec.com/kohavi96wrappers.html
- ◆ A. Miller. *Subset Selection in Regression (second edition)*. Chapman & Hall, 2002. ISBN 1-58488-171-2

■ Why Does NaïveBayes Work?

- ◆ P. Langley and S. Sage. Induction of selective bayesian classifiers. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 399–406, 1994. Available from <http://lyonesse.stanford.edu/~langley/papers/select.uai94.ps>
- ◆ Pedro Domingos and Michael J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997. URL citeseer.ist.psu.edu/domingos97optimality.html

References (5)



Introduction

In practice...

Scaling Up

Related Work

● References

● References (2)

● References (3)

● References (4)

● References (5)

And so...

Questions? Comments?

■ Machine learning

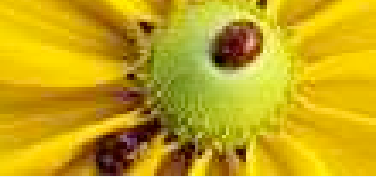
- ◆ R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1992. ISBN: 1558602380

■ Misc

- ◆ D.H. Meadows, D.L. Meadows, J. Randers, and W.W. Behrens. *The Limits to Growth*. Potomac Associates, 1972

■ MDL & MML

- ◆ C.S. Wallace and D.M. Boulton. An information measure for classification. *Computer Journal*, vol 11.2:185–194, 1968
- ◆ R.A. Baxter and J.J. Oliver. MDL and MML: Similarities and differences. Technical report, Computer Science, Monash University, Melbourne, Australia, March 1995. Available from <http://citeseer.ist.psu.edu/baxter95mdl.html>



[Introduction](#)

[In practice...](#)

[Scaling Up](#)

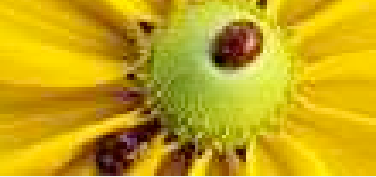
[Related Work](#)

[And so...](#)

- Success Despite Complexity
- A Final Word

[Questions? Comments?](#)

And so...



Success Despite Complexity

Introduction

In practice...

Scaling Up

Related Work

And so...

● Success Despite Complexity

● A Final Word

Questions? Comments?

- Maybe....
 - ◆ The world is not as complex as we think
 - ◆ Real world models clump, have collars.
 - ◆ Possible to quickly search, find ways to select for preferred states.
- Ultimately, this is an empirical study.
 - ◆ Q: When does a clumping/collaring-inspired search engine succeed?
 - ◆ A: Often
 - Reports effects never seen before (limits to growth)
 - Finds solutions faster than other methods (JPL).
 - Returns tiniest theories (fss)
 - Scales to infinite data streams (TAR4.1)
- Many applications. May I try this on your problems?

A Final Word

Introduction

In practice...

Scaling Up

Related Work

And so...

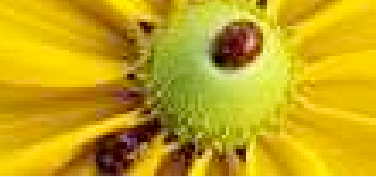
● Success Despite Complexity

● A Final Word

Questions? Comments?

- Sometimes the world is complex:
 - ◆ 2% optimizing air-flow over leading wing in trans-sonic range
 - ◆ synthesis of optimized code for complex engineering problems
- And sometimes it ain't.
 - ◆ Try the simple solution before the more complex.
 - ◆ Benchmark the complex against the seemingly less sophisticated.
 - ◆ Warning: your straw man may not burn





Questions? Comments?

[Introduction](#)

[In practice...](#)

[Scaling Up](#)

[Related Work](#)

[And so...](#)

[Questions? Comments?](#)

to partition an N -dimensional continuous space into a *Voronoi Tessellation* and then represent the set of points in each region by the region into which it falls. This discretization method creates local regions and is thus a local discretization method. Alternatively, it can be thought of as a complete *instance space* discretization as opposed to the *feature space* discretizations discussed here.

Table 1 shows a summary of these discretization methods, identified by the global/local and supervised/unsupervised dimensions. All the methods presented are static discretizers.

3 Methods

In our study, we consider three methods of discretization in depth: equal width intervals, 1RD, the method proposed by Holte for the 1R algorithm, and the entropy minimization heuristic (Fayyad & Irani 1993, Catlett 1991b).

3.1 Equal Width Interval Binning

Equal width interval binning is perhaps the simplest method to discretize data and has often been applied as a means for producing nominal values from continuous ones. It involves sorting the observed values of a continuous feature and dividing the range of observed values for the variable into k equally sized bins, where k is a parameter supplied by the user. If a variable x is observed to have values bounded by x_{min} and x_{max} then this method computes the bin width

$$\delta = \frac{x_{max} - x_{min}}{k}$$

and constructs bin boundaries, or *thresholds*, at $x_{min} + i\delta$ where $i = 1, \dots, k-1$. The method is applied to each continuous feature independently. It makes no use of instance class information whatsoever and is thus an unsupervised discretization method.

3.2 Holte’s 1R Discretizer

Holte (1993) describes a simple classifier that induces one-level decision trees, sometimes called decision stumps (Iba & Langley 1992). In order to properly deal with domains that contain continuous valued features, a simple supervised discretization method is given. This method, referred to here as 1RD (One-Rule Discretizer), sorts the observed values of a continuous feature and attempts to greedily divide the domain of the feature into bins that each contain only instances of one particular class. Since such a scheme could possibly lead to one bin for each observed real value, the algorithm is constrained to forms bins of at

least some minimum size (except the rightmost bin). Holte suggests a minimum bin size of 6 based on an empirical analysis of 1R on a number of classification tasks, so our experiments used this value as well. Given the minimum bin size, each discretization interval is made as “pure” as possible by selecting cut-points such that moving a partition boundary to add an observed value to a particular bin cannot make the count of the dominant class in that bin greater.

3.3 Recursive Minimal Entropy Partitioning

A method for discretizing continuous attributes based on a minimal entropy heuristic, presented in Catlett (1991b) and Fayyad & Irani (1993), is also used in our experimental study. This supervised algorithm uses the class information entropy of candidate partitions to select bin boundaries for discretization. Our notation closely follows the notation of Fayyad and Irani. If we are given a set of instances S , a feature A , and a partition boundary T , the class information entropy of the partition induced by T , denoted $E(A, T; S)$ is given by:

$$E(A, T; S) = \frac{|S_1|}{|S|} Ent(S_1) + \frac{|S_2|}{|S|} Ent(S_2) .$$

For a given feature A , the boundary T_{min} which minimizes the entropy function over all possible partition boundaries is selected as a binary discretization boundary. This method can then be applied recursively to both of the partitions induced by T_{min} until some stopping condition is achieved, thus creating multiple intervals on the feature A .

Fayyad and Irani make use of the *Minimal Description Length Principle* to determine a stopping criteria for their recursive discretization strategy. Recursive partitioning within a set of values S stops iff

$$Gain(A, T; S) < \frac{\log_2(N-1)}{N} + \frac{\Delta(A, T; S)}{N},$$

where N is the number of instances in the set S ,

$$Gain(A, T; S) = Ent(S) - E(A, T; S),$$

$$\Delta(A, T; S) = \log_2(3^k - 2) - [k \cdot Ent(S) - k_1 \cdot Ent(S_1) - k_2 \cdot Ent(S_2)],$$

and k_i is the number of class labels represented in the set S_i . Since the partitions along each branch of the recursive discretization are evaluated independently using this criteria, some areas in the continuous spaces will be partitioned very finely whereas others (which have relatively low entropy) will be partitioned coarsely.

Defect prediction from static code features: current results, limitations, new approaches

Tim Menzies · Zach Milton · Burak Turhan ·
Bojan Cukic · Yue Jiang · Ayşe Bener

Received: 17 November 2009 / Accepted: 5 May 2010
© Springer Science+Business Media, LLC 2010

Abstract Building quality software is expensive and software quality assurance (QA) budgets are limited. Data miners can learn defect predictors from static code features which can be used to control QA resources; e.g. to focus on the parts of the code predicted to be more defective.

Recent results show that better data mining technology is not leading to better defect predictors. We hypothesize that we have reached the limits of the standard learning goal of maximizing area under the curve (AUC) of the probability of false alarms and probability of detection “AUC(pd, pf)”; i.e. the area under the curve of a probability of false alarm versus probability of detection.

Accordingly, we explore changing the standard goal. Learners that maximize “AUC(effort, pd)” find the *smallest* set of modules that contain the *most* errors.

This research was supported by NSF grant CCF-0810879 and the Turkish Scientific Research Council (Tubitak EEEAG 108E014). For an earlier draft, see <http://menzies.us/pdf/08bias.pdf>.

T. Menzies (✉) · Z. Milton · B. Cukic · Y. Jiang
West Virginia University, Morgantown, USA
e-mail: tim@menzies.us

Z. Milton
e-mail: zmilton@mix.wvu.edu

B. Cukic
e-mail: bojan.cukic@mail.csee.wvu.edu

Y. Jiang
e-mail: yjiang1@mix.wvu.edu

B. Turhan
University of Oulu, Oulu, Finland
e-mail: burak.turhan@oulu.fi

A. Bener
Boğaziçi University, Istanbul, Turkey
e-mail: bener@boun.edu.tr

WHICH is a meta-learner framework that can be quickly customized to different goals. When customized to AUC(effort, pd), WHICH out-performs all the data mining methods studied here. More importantly, measured in terms of this new goal, certain widely used learners perform *much worse* than simple manual methods.

Hence, we advise against the indiscriminate use of learners. Learners must be chosen and customized to the goal at hand. With the right architecture (e.g. WHICH), tuning a learner to specific local business goals can be a simple task.

Keywords Defect prediction · Static code features · WHICH

1 Introduction

A repeated result is that static code features such as lines of code per module, number of symbols in the module, etc., can be used by a data miner to predict which modules are more likely to contain defects.¹ Such defect predictors can be used to allocate the appropriate verification and validation budget assigned to different code modules.

The current high water mark in this field has been curiously static for several years. For example, for three years we have been unable to improve on our 2006 results (Menzies et al. 2007b). Other studies report the same *ceiling effect*: many methods learn defect predictors that perform statistically insignificantly different to the best results. For example, after a careful study of 19 data miners for learning defect predictors seeking to maximize the area under the curve of detection-vs-false alarm curve, Lessmann et al. (2008) conclude

...the importance of the classification model is less than generally assumed
...practitioners are free to choose from a broad set of models when building defect predictors.

This article argues for a very different conclusion. The results of Lessmann et al. are certainly correct for the goal of maximizing detection and minimizing false alarm rates. However, this is not the only possible goal of a defect predictor. WHICH (Milton 2008) is a meta-learning scheme where domain specific goals can be inserted into the core of the learner. When those goals are set to one particular business goal (e.g. “finding the *fewest* modules that contain the *most* errors”) then the ceiling effect disappears:

- WHICH significantly out-performs other learning schemes.
- More importantly, certain widely used learners perform *worse than simple manual methods*.

That is, contrary to the views of Lessmann et al., the selection of a learning method appropriate to a particular goal is very critical. Learners that appear useful when

¹ See e.g. Weyuker et al. (2008), Halstead (1977), McCabe (1976), Chapman and Solomon (2002), Nagappan and Ball (2005a, 2005b), Hall and Munson (2000), Nikora and Munson (2003), Khoshgoftaar (2001), Tang and Khoshgoftaar (2004), Khoshgoftaar and Seliya (2003), Porter and Selby (1990), Tian and Zelkowitz (1995), Khoshgoftaar and Allen (2001), Srinivasan and Fisher (1995).

pursuing certain goals, can be demonstrably inferior when pursuing others. We recommend WHICH as a simple method to create such customizations.

The rest of this paper is structured as follows. Section 2 describes the use of static code features for learning defect predictors. Section 3 documents the ceiling effect that has stalled progress in this field. After that, Sects. 4 and 5 discuss a novel method to break through the ceiling effect.

2 Background

This section motivates the use of data mining for static code features and reviews recent results. The rest of the paper will discuss limits with this approach, and how to overcome them.

2.1 Blind spots

Our premise is that building high quality software is expensive. Hence, during development, developers *skew* their limited quality assurance (QA) budgets towards artifacts they believe most require extra QA. For example, it is common at NASA to focus QA more on the on-board guidance system than the ground-based database which stores scientific data collected from a satellite.

This skewing process can introduce an inappropriate bias to quality assurance (QA). If the QA activities concentrate on project artifacts, say A, B, C, D , then that leaves *blind spots* in E, F, G, H, I, \dots . Blind spots can compromise high assurance software. Leveson remarks that in modern complex systems, unsafe operations often result from an unstudied interaction between components (Leveson 1995). For example, Lutz and Mikulski (2003) found a blind spot in NASA deep-space missions: most of the mission critical *in-flight* anomalies resulted from errors in *ground software* that fails to correctly collect in-flight data.

To avoid blind spots, one option is to rigorously assess all aspects of all software modules, however, this is impractical. Software project budgets are finite and QA effectiveness increases with QA effort. A *linear* increase in the confidence C that we have found all faults can take *exponentially* more effort. For example, to detect one-in-a-thousand module faults, moving C from 90% to 94% to 98% takes 2301, 2812, and 3910 black box tests (respectively).² Lowry et al. (1998) and Menzies and Cukic (2000) offer numerous other examples where assessment effectiveness is exponential on effort.

Exponential cost increase quickly exhausts finite QA resources. Hence, blind spots can't be avoided and must be managed. Standard practice is to apply the best available assessment methods on the sections of the program that the best available domain knowledge declares is the most critical. We endorse this approach. Clearly, the most

²A randomly selected input to a program will find a fault with probability x . Voas observes (Voas and Miller 1995) that after N random black-box tests, the chance of the inputs not revealing any fault is $(1-x)^N$. Hence, the chance C of seeing the fault is $1 - (1-x)^N$ which can be rearranged to $N(C, x) = \frac{\log(1-C)}{\log(1-x)}$. For example, $N(0.90, 10^{-3}) = 2301$.

critical sections require the best known assessment methods, in hope of minimizing the risk of safety or mission critical failure occurring post deployment. However, this focus on certain sections can blind us to defects in other areas which, through interactions, may cause similarly critical failures. Therefore, the standard practice should be augmented with a *lightweight sampling policy* that (a) explores the rest of the software and (b) raises an alert on parts of the software that appear problematic. This sampling approach is incomplete by definition. Nevertheless, it is the only option when resource limits block complete assessment.

2.2 Lightweight sampling

2.2.1 Data mining

One method for building a lightweight sampling policy is *data mining* over *static code features*. For this paper, we define *data mining* as the process of summarizing tables of data where rows are *examples* and columns are the *features* collected for each example.³ One special feature is called the *class*. The [Appendix](#) to this paper describes various kinds of data miners including:

- Naïve Bayes classifiers use statistical combinations of features to predict for class value. Such classifiers are called “naive” since they assume all the features are statistically independent. Nevertheless, a repeated empirical result is that, on average, seemingly naïve Bayes classifiers perform as well as other seemingly more sophisticated schemes (e.g. see Table 1 in Domingos and Pazzani 1997).
- Rule learners like RIPPER (Cohen 1995a) generate lists of rules. When classifying a new code module, we take features extracted from that module and iterate over the rule list. The output classification is the first rule in the list whose condition is satisfied.
- Decision tree learners like C4.5 (Quinlan 1992b) build one single-parent tree whose internal nodes test for feature values and whose leaves refer to class ranges. The output of a decision tree is a branch of satisfied tests leading to a single leaf classification.

There are many alternatives and extensions to these learners. Much recent work has explored the value of building *forests* of decision trees using randomly selected subsets of the features (Breimann 2001; Jiang et al. 2008b). Regardless of the learning method, the output is the same: combinations of standard features that predict for different class values.

2.2.2 Static code features

Defect predictors can be learned from tables of data containing static code features, whose class label is *defective* and whose values are *true* or *false*. In those tables:

³Technically, this is *supervised learning* in the *absence of a background theory*. For notes on *unsupervised learning*, see papers discussing *clustering* such as Bradley et al. (1998). For notes on *using a background theory*, see (e.g.) papers discussing the learning or tuning of Bayes nets (Fenton and Neil 1999).

Fig. 1 Static code features

m = McCabe		$v(g)$	cyclomatic_complexity	
		$iv(G)$	design_complexity	
		$ev(G)$	essential_complexity	
locs	loc	loc_total (one line = one count)		
	loc(other)	loc_blank		
		loc_code_and_comment		
		loc_comments		
		loc_executable		
		number_of_lines		
		(opening to closing brackets)		
Halstead	h	N_1	num_operators	
		N_2	num_operands	
		μ_1	num_unique_operators	
		μ_2	num_unique_operands	
	H		N	length: $N = N_1 + N_2$
			V	volume: $V = N * \log_2 \mu$
			L	level: $L = V^* / V$ where $V^* = (2 + \mu_2^*) \log_2(2 + \mu_2^*)$
			D	difficulty: $D = 1/L$
			I	content: $I = \hat{L} * V$ where $\hat{L} = \frac{2}{\mu_1} * \frac{\mu_2}{N_2}$
			E	effort: $E = V/\hat{L}$
			B	error_est
			T	prog_time: $T = E/18$ seconds

- Rows describe data from one *module*. Depending on the language, modules may be called “functions”, “methods”, “procedures” or “files”.
- Columns describe one of the static code features of Fig. 1. The [Appendix](#) of this paper offers further details on these features.

These static code features are collected from prior development work. The *defective* class summarizes the results of a whole host of QA methods that were applied to that historical data. If any manual or automatic technique registered a problem with this module, then it was marked “defective = true”. For these data sets, the data mining goal is to learn a binary prediction for *defective* from past projects that can be applied to future projects.

This paper argues that such defect predictors are useful and describes a novel method for improving their performance. Just in case we overstate our case, it is important to note that defect predictors learned from static code features can only *augment*, but never *replace*, standard QA methods. Given a limited budget for QA, the manager’s task is to decide which set of QA methods M_1, M_2, \dots that cost C_1, C_2, \dots should be applied. Sometimes, domain knowledge is available that can indicate that certain modules deserve the most costly QA methods. If so, then some subset of the

Fig. 2 Sample of percentage of defects seen in different modules. Note that only a very small percentage of modules have more than one defect. For more details on these data sets, see Fig. 3

N defects	Percentage of modules with N defects				
	cm1	kc1	kc3	mw1	pc1
1	10.67	6.50	1.96	5.69	4.15
2	02.17	3.04	1.53	0.74	1.53
3	01.19	2.18	2.83		0.45
4	00.99	0.76		0.25	0.09
5	00.40	0.33			0.09
6	00.20	0.43			0.09
7	00.40	0.28			0.09
8		0.24			
9		0.05			0.09
10		0.05			
11					
12		0.05			
Totals	16.01	13.90	6.32	6.68	6.58

system may receive more attention by the QA team. We propose defect predictors as a rapid and cost effective lightweight sampling policy for checking if the rest of the system deserves additional attention. As argued above, such a sampling method is essential for generating high-quality systems under the constraint of limited budgets.

2.3 Frequently asked questions

2.3.1 Why binary classifications?

The reader may wonder why we pursue such a simple binary classification scheme ($defective \in \{true, false\}$) and not, say, *number of defects* or *severity of defects*. In reply, we say:

- We do not use *severity of defects* since in large scale data collections, such as those used below, it is hard to distinguish defect “severity” from defect “priority”. All too often, we have found that developers will declare a defect “severe” when they are really only stating a preference on what bugs they wish to fix next. Other authors have the same reservations:
 - Nikora cautions that “without a widely agreed upon definition of severity, we can not reason about it” (Nikora 2004).
 - Ostrand et al. make a similar conclusion: “(severity) ratings were highly subjective and also sometimes inaccurate because of political considerations not related to the importance of the change to be made. We also learned that they could be inaccurate in inconsistent ways” (Ostrand et al. 2004).
- We do not use *number of defects* as our target variable since, as shown in Fig. 2, only a vanishingly small percent of our modules have more than one issue report. That is, our data has insufficient examples to utilize (say) one method in the kc1 data set with a dozen defects.

2.3.2 Why not use regression?

Other researchers (e.g. Mockus et al. 2005; Zimmermann and Nagappan 2009), use a logistic regression model to predict software quality features. Such models have the general form

$$\text{Probability}(Y) = \frac{e^{(c+a_1X_1+a_2X_2+\dots)}}{1 + e^{(c+a_1X_1+a_2X_2+\dots)}}$$

where a_i are the logistic regression predicted constants and the X_i are the independent variables used for building the logistic regression model. For example, in the case of Zimmermann et al.'s work, those variables are measures of code change, complexity, and pre-release bugs. These are used to predict number of defects.

Another regression variant is the negative binomial regression (NBM) model used by Ostrand et al. (2004) to predict defects in AT&T software. Let y_i equal the number of faults observed in file i and x_i be a vector of characteristics for that file. NBM assumes that y_i given x_i has a Poisson distribution with mean λ_i computed from $\lambda_i = \gamma_i e^{\beta x_i}$ where γ_i is the gamma distribution with mean 1 and variance $\sigma^2 \geq 0$ (Ostrand et al. compute σ^2 and β using a maximum likelihood procedure).

Logistic regression and NBM fit one model to the data. When data is multi-modal, it is useful to fit multiple models. A common method for handling arbitrary distributions to approximate complex distributions is via a set of piecewise linear models. *Model tree learners*, such as Quinlan's M5' algorithm (Quinlan 1992a), can learn such piecewise linear models. M5' also generates a decision tree describing when to use which linear model.

We do not use regression for several reasons:

- Regression assumes a continuous target variable and, as discussed above, our target variable is binary and discrete.
- There is no definitive result showing that regression methods are better/worse than the data miners used in this study. In one of the more elaborate recent studies, Lessmann et al. found no statistically significant advantage of logistic regression over a large range of other algorithms (Lessmann et al. 2008) (the Lessmann et al. result is discussed, at length, below).
- In previous work, we have assessed various learning methods (including regression methods and model trees) in terms of their ability to be guided by various business considerations. Specifically, we sought learners that could tune their conclusions to user-supplied utility weights about false alarms, probability of detection, etc. Of the fifteen defect prediction methods used in that study, regression and model trees were remarkably *worst* at being able to be guided in this way (Menzies and Stefano 2003). The last section of this paper discusses a new learner, called WHICH, that was specially designed to support simple tuning to user-specific criteria.

2.3.3 Why static code features?

Another common question is why just use static code features? Fenton (1994) divides software metrics into process, product, and personnel and uses these to collect information on how the software was built, what was built, and who built it. Static code

measures are just product metrics and, hence, do not reflect process and personnel details. For this reason, other researchers use more than just static code measures. For example:

- Reliability engineers use knowledge of how the frequency of faults seen in a running system changes over time (Musa et al. 1987; Littlewood and Wright 1997).
- Other researchers explore *churn*; i.e. the rate at which the code base changes (Hall and Munson 2000).
- Other researchers reason about the development team. For example, Nagappan et al. comment on how organizational structure affects software quality (Nagappan and Murphy 2008) while Weyuker et al. document how large team sizes change defect rates (Weyuker et al. 2008).

When replying to this question, we say that static code features are one of the few measures we can collect in a consistent manner across many projects. Ideally, data mining occurs in some CMM level 5 company where processes and data collection is precisely defined. In that ideal case, there exists extensive data sets collected over many projects and many years. These data sets are in a consistent format and there is no ambiguity in the terminology of the data (e.g. no confusion between “severity” and “priority”).

We do not work in that ideal situation. Since 1998, two of the authors (Menzies and Cukic) have been research consultants to NASA. Working with NASA’s Independent Software Verification and Validation Facility (IV&V), we have tried various methods to add value to the QA methods of that organization. As we have come to learn, NASA is a very dynamic organization. The NASA enterprise has undergone major upheavals following the 2003 loss of the Columbia shuttle, then President Bush’s new vision for interplanetary space exploration in 2004, and now (2010) the cancellation of that program. As research consultants, we cannot precisely define data collection in such a dynamic environment. Hence, we do not ask “what are the right features to collect?”. Instead, we can only ask “what features can we access, right now?” This question is relevant to NASA as well as any organization where data collection is not controlled by a centralized authority such as:

- agile software projects;
- out-sourced projects;
- open-sourced projects;
- and organizations that make extensive use of sub-contractors and sub-sub contractors.

In our experience, the one artifact that can be accessed in a consistent manner across multiple different projects is the source code (this is particularly true in large projects staffed by contractors, sub-contractors, and sub-sub contractors). Static code features can be automatically and cheaply extracted from source code, even for very large systems (Nagappan and Ball 2005a). By contrast, other methods such as manual code reviews are labor-intensive. Depending on the review methods, 8 to 20 LOC/minute can be inspected and this effort repeats for all members of the review team, which can be as large as four or six (Menzies et al. 2002).

For all the above reasons, many industrial practitioners and researchers (including ourselves) use static attributes to guide software quality predictions (see the list

shown in the introduction). Verification and validation (V&V) textbooks (Rakitin 2001) advise using static code complexity attributes to decide which modules are worthy of manual inspections. At the NASA IV&V facility, we know of several large government software contractors that will not review software modules *unless* tools like McCabe predict that some of them might be fault prone.

2.3.4 What can be learned from static code features?

The previous section argued that, for pragmatic reasons, all we can often collect are static code measures. This is not to say that if we *use* those features, then they yield useful or interesting results. Hence, a very common question we hear about is “what evidence is there that anything useful can be learned from static code measures?”.

There is a large body of literature arguing that static code features are an inadequate characterization of the internals of a function:

- Fenton offers an insightful example where *the same* functionality is achieved via *different* language constructs resulting in *different* static measurements for that module (Fenton and Pfleeger 1997). Using this example, Fenton argues against the use of static code features.
- Shepperd and Ince present empirical evidence that the McCabe static attributes offer nothing more than uninformative attributes like lines of code. They comment “for a large class of software it (cyclomatic complexity) is no more than a proxy for, and in many cases outperformed by, lines of code” (Shepperd and Ince 1994).
- In a similar result, Fenton and Pfleeger note that the main McCabe attributes (cyclomatic complexity, or $v(g)$) are highly correlated with lines of code (Fenton and Pfleeger 1997).

If static code features were truly useless, then the defect predictors learned from them would satisfy two predictions:

Prediction 1: They would perform badly (not predict for defects);

Prediction 2: They would have no generality (predictors learned from one data set would not be insightful on another).

At least in our experiences, these predictions do not hold. This evidence falls into two groups: *field studies* and a *controlled laboratory study*. In the *field studies*:

- Our prediction technology was commercialized in the *Predictive* tool and sold across the United States to customers in the telecom, energy, technology, and government markets, including organizations such as Compagnie Financière Alcatel (Alcatel); Chevron Corporation; LogLogic, Inc.; and Northrop Grumman Corporation. As an example of the use of Predictive, one company (GB Tech, Inc.) used it to manage safety critical software for a United States manned strike fighter. This code had to be tested extensively to ensure safety (the software controlled a lithium ion battery, which can overcharge and possibly explode). First, a more expensive tool for structural code coverage was applied. Later, the company ran that tool and Predictive on the same code. Predictive produced consistent results with the more expensive tools while being able to faster process a larger code base than the more expensive tool (Turner 2006).

- We took the defect prediction technology of this paper (which was developed at NASA in the USA) and applied it to a software development company from another country (a Turkish software company). The results were very encouraging: when inspection teams focused on the modules that trigger our defect predictors, they found up to 70% of the defects using 40% of the effort (measured in staff hours). Based on those results, we were subsequently invited by two companies to build tools to incorporate our defect prediction methods into their routine daily processes (Tosun et al. 2009).
- A subsequent, more detailed, study on the Turkish software compared how much code needs to be inspected using a random selection process vs selection via our defect predictors. Using the random testing strategy, 87% of the files would have to be inspected in order to detect 87% of the defects. However, if the inspection process was restricted to the 25% of the files that trigger our defect predictors, then 88% of the defects could be found. That is, the same level of defect detection (after inspection) can be achieved using $\frac{87-25}{87} = 71\%$ less effort (Tosun and Bener 2010).

The results of these field studies run counter to Prediction 1. However, they are not reproducible results. In order to make a claim that other researchers can verify, we designed a *controlled experiment* to assess Predictions 1 and 2 in a reproducible manner (Turhan et al. 2009). That experiment was based on the public domain data sets of Fig. 3. These data sets are quite diverse and are written in different languages (C, C++, JAVA); written in different countries (United States and Turkey); and written for different purposes (control and monitoring of white goods, NASA flight systems, ground-based software).

Before we can show that experiment, we must first digress to define performance measures for defect prediction. When such a predictor fires then $\{A, B, C, D\}$ denotes the true negatives, false negatives, false positives, and true positives (respectively). From these measures we can compute:

$$pd = recall = \frac{D}{B + D}$$

$$pf = \frac{C}{A + C}$$

In the above, pd is the *probability of detecting* a faulty module while pf is the *probability of false alarm*. Other performance measures are $accuracy = \frac{A+D}{A+B+C+D}$ and $precision = \frac{D}{C+D}$. Figure 4 shows an example of the calculation of these measures.

Elsewhere (Menzies et al. 2007a), we show that accuracy and precision are highly unstable performance indicators for data sets like Fig. 3 where the target concept occurs with relative infrequency: in Fig. 3, only $\frac{1}{7}$ th (median value) of the modules are marked as defective. Therefore, for the rest of this paper, we will not refer to accuracy or precision.

Having defined performance measures, we can now check Predictions 1 & 2; i.e. static defect features lead to poor fault predictors and defect predictors have no generality between data sets. If D denotes all the data in Fig. 3, and D_i denote one particular data set $D_i \in D$, then we can conduct two kinds of experiments:

Project	Source	Language	Description	# modules	Features	%defective
pc1	NASA	C++	Flight software for earth orbiting satellites	1,109	21	6.94
kc1	NASA	C++	Storage management for ground data	845	21	15.45
kc2	NASA	C++	Storage management for ground data	522	21	20.49
cm1	NASA	C++	Spacecraft instrument	498	21	9.83
kc3	NASA	JAVA	Storage management for ground data	458	39	9.38
mw1	NASA	C++	A zero gravity experiment related to combustion	403	37	7.69
ar4	Turkish white goods manufacturer	C	Refrigerator	107	30	18.69
ar3	Turkish white goods manufacturer	C	Dishwasher	63	30	12.70
mc2	NASA	C++	Video guidance system	61	39	32.29
ar5	Turkish white goods manufacturer	C	Washing machine	36	30	22.22
Total: 4,102						

Fig. 3 Tables of data, sorted in order of number of examples, from <http://promisedata.org/data>. The rows labeled “NASA” come from NASA aerospace projects while the other rows come from a Turkish software company writing applications for domestic appliances. All this data conforms to the format of Sect. 2.2.2

Fig. 4 Performance measures

		Module found in defect logs?	
		no	yes
Signal detected?	no	A = 395	B = 67
	yes	C = 19	D = 39
$pf = Prob.falseAlarm = 5\%$			
$pd = Prop.detected = 37\%$			
$acc = accuracy = 83\%$			
$prec = precision = 67\%$			

Fig. 5 Results of round-robin and self experiments. From Turhan et al. (2009). All the pd and pf results are statistically different at the 95% level (according to a Mann-Whitney test)

Experiment	Notes	Median	
		pd%	pf%
RR	round-robin	94	68
RR2	round-robin + relevancy filtering	69	27
SELF	self test	75	29

SELF: *Self-learning* experiments where we *train* on 90% of D_i then test on the remaining 10%. Note that such self-learning experiments will let us comment on Prediction 1.

RR: *Round-robin* experiments where we *test* on 10% (randomly selected) of data set D_i after *training* on the remaining nine data sets $D - D_i$. Note that such round-robin experiments will let us comment on Prediction 2.

It turns out that the round-robin results are unimpressive due to an *irrelevancy effect*, discussed below. Hence, it is also useful to conduct:

RR2: Round-robin experiments where a *relevancy filter* is used to filter away irrelevant parts of the training data.

After repeating experiments RR, SELF, RR2 twenty times for each data set $D_i \in D$, the median results are shown in Fig. 5. At first glance, the round-robin results of RR seem quite impressive: a 98% probability of detection. Sadly, these high detection probabilities are associated with an unacceptably high false alarm rate of 68%.

In retrospect, this high false alarm rate might have been anticipated. A median sized data set from Fig. 3 (e.g. *mw1*) has around 450 modules. In a round-robin experiment, the median size of the training set is over 3600 modules taken from nine other projects. In such an experiment, it is highly likely that the defect predictor will be learned from numerous irrelevant details from other projects.

To counter the problem of irrelevant training data, the second set of round-robin experiments constructed training sets for D_i from the union of the 10 nearest neighbors within $D - D_i$. The RR2 results of Fig. 5 show the beneficial effects of relevancy filtering: false alarm rates reduced by $\frac{68}{27} = 252\%$ with only a much smaller reduction in pd of $\frac{94}{69} = 136\%$.

Returning now to Prediction 1, the SELF and RR2 $pd \geq 69\%$ results are much larger than those seen in industrial practice:

- A panel at *IEEE Metrics 2002* (Shull et al. 2002) concluded that manual software reviews can find $\approx 60\%$ of defects.⁴
- Raffo found that the defect detection capability of industrial review methods can vary from $pd = TR(35, 50, 65)\%$ ⁵ for full Fagan inspections (Fagan 1976) to $pd = TR(13, 21, 30)\%$ for less-structured inspections (Raffo 2005).

⁴That panel supported neither Fagan claim (Fagan 1986) that inspections can find 95% of defects before testing or Shull's claim that specialized directed inspection methods can catch 35% more defects than other methods (Shull et al. 2000).

⁵ $TR(a, b, c)$ is a triangular distribution with min/mode/max of a, b, c .

That is, contrary to Prediction 1, defect predictors learned from static code features perform well, relative to standard industrial methods.

Turning now to Prediction 2, note that the RR2 round-robin results (with relevancy filtering) are close to the SELF:

- The *pd* results are only $1 - \frac{75}{69} = 8\%$ different;
- The *pf* results are only $\frac{29}{27} - 1 = 7\%$ different.

That is, contrary to Prediction 2, there is generality in the defect predictions learned from static code features. Learning from local data is clearly best (SELF's *pd* results are better than RR2), however, nearly the same performance results as seen in SELF can be achieved by applying defect data from one site (e.g. NASA flight systems) to another (e.g. Turkish white good software).

2.4 Summary

For all the above reasons, we research defect predictors based on static code features. Such predictors are:

- *Useful*: they out-perform standard industrial methods. Also, just from our own experience, we can report that they have been successfully applied in software companies in the United States and Turkey.
- *Generalizable*: as the RR2 results show, the predictions of these models generalize across data sets taken from different organizations working in different countries.
- *Easy to use*: they can automatically process thousands of modules in a matter of seconds. Alternative methods such as manual inspections are much slower (8 to 20 LOC per minute).
- *Widely-used*: We can trace their use as far back as 1990 (Porter and Selby 1990). We are also aware of hundreds of publications that explore this method (for a partial sample, see the list shown in the introduction).

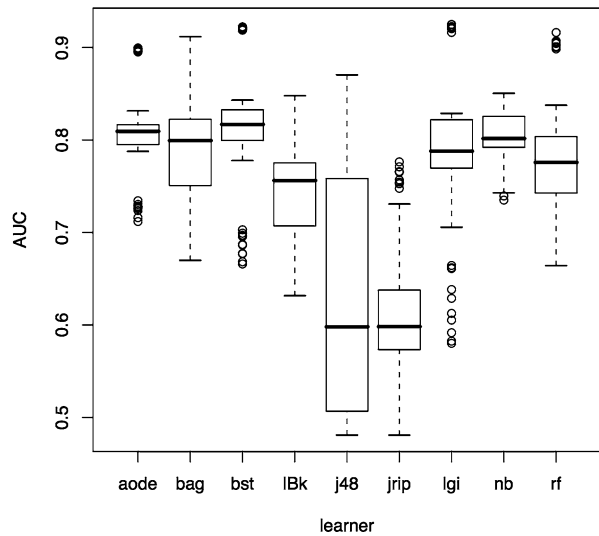
3 Ceiling effects in defect predictors

Despite several years of exploring different learners and data pre-processing methods, the performance of our learners has not improved. This section documents that ceiling effect and the rest of this paper explores methods to break through the ceiling effect.

In 2006 (Menzies et al. 2007b), we defined a repeatable defect prediction experiment which, we hoped, others could improve upon. That experiment used public domain data sets and open source data miners. Surprisingly, a simple naïve Bayes classifier (with some basic pre-processor for the numerics) out-performed the other studied methods. For details on naïve Bayes classifiers, see the [Appendix](#).

We made the experiment repeatable in the hope that other researchers could improve or refute our results. So far, to the best of our knowledge, no study using just static code features has out-performed our 2006 result. Our own experiments (Jiang et al. 2008b) found little or no improvement from the application of numerous data mining methods. Figure 6 shows some of those results using (in order, left to right) *aode* average one-dependence estimators (Yang et al. 2006); *bag* bagging (Brieman 1996); *bst* boosting (Freund and Schapire 1997); *IBk* instance-based learning (Cover

Fig. 6 Box plot for AUC(pf, pd) seen with 9 learners when, 100 times, a random 90% selection of the data is used for training and the remaining data is used for testing. The rectangles show the inter-quartile range (the 25% to 75% quartile range). The line shows the minimum to maximum range, unless that range extends beyond 1.5 times the inter-quartile range (in which case dots are used to mark these extreme outliers). From Jiang et al. (2008b)



and Hart 1967); *C4.5* *C4.5* (Quinlan 1992b); *jrjp* RIPPER (Cohen 1995b); *lgi* logistic regression (Breiman et al. 1984); *nb* naïve Bayes (second from the right); and *rf* random forests (Breimann 2001). These histograms show area under the curve (AUC) of a *pf*-vs-*pd* curve. To generate such a “AUC(pf, pd)” curve:

- A learner is executed multiple times on different subsets of data;
- The *pd*, *pf* results are collected from each execution;
- The results are sorted on increasing order of *pf*;
- The results are plotted on a 2-D graph using *pf* for the *x*-axis and *pd* for the *y*-axis.

A statistical analysis of Fig. 6 results showed that only boosting on discretized data offers a statistically better result than naïve Bayes. However, we cannot recommend boosting: boosting is orders of magnitudes slower than naïve Bayes; and the median improvement over naïve Bayes is negligible.

Other researchers have also failed to improve our results. For example, Fig. 7 shows results from a study by Lessmann et al. on statistical differences between 19 learners used for defect prediction (Lessmann et al. 2008). At first glance, our preferred naïve Bayes method (shown as “NB” on the sixth line of Fig. 7) seems to perform poorly: it is ranked in the lower third of all 19 methods. However, as with all statistical analysis, it is important to examine not only central tendencies but also the variance in the performance measure. The vertical dotted lines in Fig. 7 show Lessmann et al.’s statistical analysis that divided the results into regions where all the results are significantly different: the performance of the top 16 methods are statistically *insignificantly different* from each other (including our preferred “NB” method). Lessmann et al. comment:

Only four competitors are significantly inferior to the overall winner (k-NN, K-start, BBF net, VP). The empirical data does not provide sufficient evidence to judge whether RndFor (Random Forest), performs significantly better than

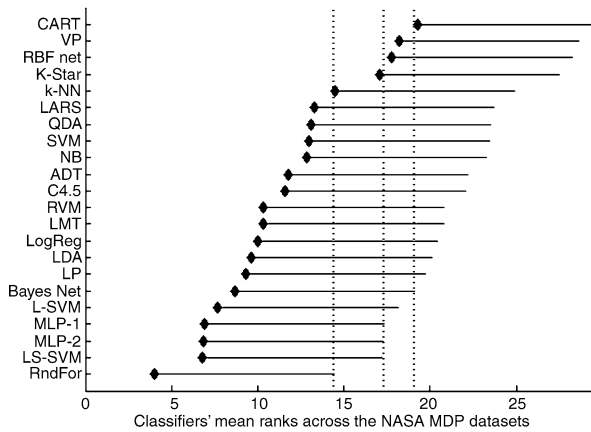


Fig. 7 Range of AUC(pf, pd) ranks seen in 19 learners building defect predictors when, 10 times, a random 66% selection of the data is used for training and the remaining data is used for testing. In ranked data, values from one method are replaced by their rank in space of all sorted values (so *smaller* ranks means *better* performance). In this case, the performance value was *area under the false positive vs true-positive curve* (and larger values are better). *Vertical lines* divide the results into regions where the results are statistically similar. For example, all the methods whose top ranks are 4 to 12 are statistically *insignificantly different*. From Lessmann et al. (2008)

QDA (Quadratic Discriminant Analysis) or any classifier with better average rank.

In other words, Lessmann et al. are reporting a ceiling effect where a large number of learners exhibit performance results that are indistinguishable.

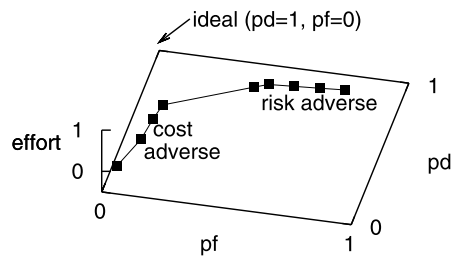
4 Breaking through the ceiling

This section discusses methods for breaking through the ceiling effects documented above.

One constant in the results of Figs. 6 and 7 is the performance *goal* used in those studies: both those results assumed the goal of the learning was to maximize AUC(pf, pd), i.e. the area under a *pf-vs-pd* curve. As shown below, if we change the goal of the learning, then we can break the ceiling effect and find better (and worse) methods for learning defect predictors from static code measures.

Depending on the business case that funded the data mining study, different goals may be most appropriate. To see this, consider the typical *pf-vs-pd-vs-effort* curve of Fig. 8:

- The *pf*, *pd* performance measures were defined above.
- *Effort* is the percentage of the code base found in the modules predicted to be faulty (so if all modules are predicted to be faulty, the 100% of the code base must be processed by some other, slower, more expensive QA method).

Fig. 8 *Pf-vs-pd-vs-effort*

For the moment, we will just focus on the pf , pd plane of Fig. 8. A perfect detector has no false alarm rates and finds all fault modules; i.e. $pf, pd = 0, 1$. As shown in Fig. 8, the $AUC(pf, pd)$ can bend towards this ideal point but may never reach there:

- Detectors learned from past experience have to make some inductive leaps and, in doing so, make some mistakes. That is, the only way to achieve high pds is to accept some level of pfs .
- The only way to avoid false alarms is to decrease the probability that the detector will trigger. That is, the only way to achieve low pfs is to decrease pd .

Different businesses prefer different regions of Fig. 8 curve:

- Mission-critical systems are *risk averse* and may accept very high false alarm rates, just as long as they catch any life-threatening possibility.
- For less critical software, *cost averse* managers may accept lower probabilities of detection, just as long as they do not waste budgets on false alarms.

That is, different businesses have different *goals*:

Goal 1: Risk averse developments prefer high pd ;

Goal 2: Cost averse developments accept mid-range pd , provided they get low pf .

Arisholm and Briand (2006) propose yet another goal:

Goal 3: A budget-conscious team wants to know that if $X\%$ of the modules are predicted to be defective, then modules contain more than $X\%$ of the defects. Otherwise, they argue, the cost of generating the defect predictor is not worth the effort.

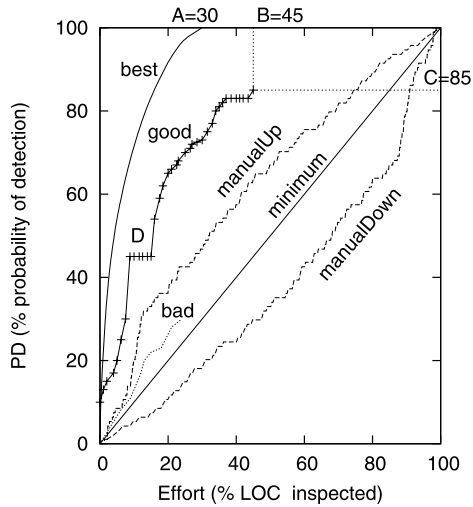
The *effort*-based evaluation of Goal 3 uses a dimension not explored by the prior work that reported ceiling effects (Lessmann et al. or our work Jiang et al. 2008a; Menzies et al. 2007b). Hence, for the rest of this paper, we will assess the impacts of the Arisholm and Briand goal of maximizing the “ $AUC(\text{effort}, pd)$ ”.

4.1 Experimental set up

4.1.1 Operationalizing $AUC(\text{effort}, pd)$

To operationalize Goal 3 from Arisholm and Briand evaluation, we assume that:

- After a data miner predicts a module is defective, it is inspected by a team of human experts.
- This team correctly recognize some subset Δ of the truly defective modules, (and $\Delta = 1$ means that the inspection teams are perfect at their task).

Fig. 9 Effort-vs-PD

- Our goal is to find learners that find the *most* number of defective modules in the *smallest* number of modules (measured in terms of LOC).

For Arisholm and Briand to approve of a data miner, it must fall in the region $pd > effort$. The *minimum* curve in Fig. 9 shows the lower boundary of this region and a “good” detector (according to $AUC(effort, pd)$) must fall above this line. Regarding the x -axis and y -axis of this figure:

- The x -axis shows all the modules, sorted on size. For example, if we had 100 modules of 10 LOC, 10 modules of 15 LOC, and 1 module of 20 LOC then the x -axis would be 111 items long with the 10 LOC modules on the left-hand side and the 20LOC module on the right-hand side.
- Note that the y -axis of this figure assumes $\Delta = 1$; i.e. inspection teams correctly recognizes all defective modules. Other values of Δ are discussed below.

4.1.2 Upper and lower bounds on performance

It is good practice to compare the performance of some technique against theoretical *upper* and *lower* bounds (Cohen 1995a). Automatic data mining methods are interesting if they out-perform manual methods. Therefore, for a *lower-bound* on expected performance, we compare them against some manual methods proposed by Koru et al. (2007, 2008, 2009):

- They argue that the relationship between module *size* and *number of defects* is not linear, but *logarithmic*; i.e. smaller modules are proportionally more troublesome.
- The *manualUp* and *manualDown* curves of Fig. 9 show the results expected by Koru et al. from inspecting modules in increasing/decreasing order of size (respectively).
- With *manualUp*, all modules are selected and sorted in increasing order of size, so that curve runs from 0 to 100% of the LOC.

In a result consistent with Koru et al., our experiments show *manualUp* usually defeating *manualDown*. As shown in Fig. 9, *manualUp* scores higher on *effort-vs-PD* than *manualDown*. Hence, we define an *upper bound* on our performance as follows. Consider an optimal oracle that restricts module inspections to just the modules that are truly defective. If *manualUp* is applied to just these modules, then this would show the upper-bound on detector performance. For example, Fig. 9 shows this *best* curve where 30% of the LOC are in defective modules.

In our experiments, we ask our learners to make a binary decision (*defective*, *nonDefective*). All the modules identified as *defective* are then sorted in order of increasing size (LOC). We then assess their performance by $AUC(\text{effort}, \text{pd})$. For example, the *bad* learner in Fig. 9 performs worse than the *good* learner since the latter has a larger area under its curve.

In order to provide an upper-bound on our AUC, we report them as a ratio of the area under the *best* curve. All the performance scores mentioned in the rest of this paper are hence *normalized AUC(effort, pd)* values ranging from 0% to 100% of the *best* curve.

Note that normalization simplifies our assessment criteria. If the effectiveness of the inspection team is independent of the method used to select the modules that they inspect, then Δ is the *same across all data miners*. By expressing the value of a defect predictor as a ratio of the area under the *best* curve, this Δ cancels out so we can assess the relative merits of different defect predictors *independently* of Δ .

4.1.3 Details

Three more details will complete our discussion of Fig. 9. Defect detectors usually do not trigger on all modules. For example, the *good* curve of Fig. 9 triggers on $B = 43\%$ of the code while only detecting 85% of the defective modules. Similarly, the *bad* curve stops after finding 30% of the defective modules in 24% of the code. To complete the *effort-vs-PD* curve, we must fill in the gap between the termination point and $X = 100$. Later in this article, we will assume that test engineers inspect the modules referred to by the data miner. Visually, for the *good* curve, this assumption would correspond to a flat line running to the right from point $C = 85$ (i.e. the 85% of the code triggered by the learner that generated the *good* curve).

Secondly, the following observation will become significant when we tune a learner to $AUC(\text{effort}, \text{pd})$. Even though Fig. 9 shows *effort-vs-PD*, it can also indirectly show false alarms. Consider the plateau in the *good* curve of Fig. 9, marked with “D”, at around $\text{effort} = 10$, $\text{PD} = 45$. Such plateaus mark false alarms where the detectors are selecting modules that have no defects. That is, to maximize the area under an *effort-vs-PD*, we could assign a heavy penalty against false alarms that lead to plateaus.

Thirdly, Fig. 9 assumes that inspection effort is linear on size of module. We make this assumption since a previous literature review reported that current inspection models all report linear effort models (Menzies et al. 2002). Nevertheless, Fig. 9 could be extended to other effort models as follows: stretch the x -axis to handle, say, non-linear effects such as longer modules that take exponentially more time to read and understand.

4.2 Initial results

Figure 9's *bad* and *manualUp* curves show our first attempt at applying this new evaluation bias. These curves were generated by applying *manualUp* and the C4.5 tree learner (Quinlan 1992b) to one of the data sets studied by Lessmann et al. Observe how the automatic method performed far worse than a manual one. To explain this poor performance, we comment that data miners grow their models using a search bias B_1 , then we assess them using a different evaluation bias B_2 . For example:

- During *training*, a decision-tree learner may stop branching if the diversity of the instances in a leaf of a branch⁶ falls below some heuristic threshold.
- During *testing*, the learned decision-tree might be tested on a variety of criteria such as Lessmann et al.'s AUC measure or our operationalization of AUC(effort, pd).

It is hardly surprising that C4.5 performed so poorly in Fig. 9. C4.5 was not designed to optimize AUC(effort, pdf) (since B_1 was so different to B_2). Some learning schemes support biasing the learning according to the overall goal of the system; for example:

- The *cost-sensitive learners* discussed by Elkan (2001).
- The *ROC ensembles* discussed by Fawcett (2001) where the conclusion is a summation of the conclusions of the ensemble of ROC curves,⁷ proportionally weighted, to yield a new learner.
- Our cost curve meta-learning scheme permits an understanding of the performance of a learner across the entire space of pd-vs-pf trade-offs (Jiang et al. 2008a).

At best, such biasing only indirectly controls the search criteria. If the search criteria is orthogonal to the success criteria of, say, maximizing *effort-vs-pd*, then cost-sensitive learning or ensemble combinations or cost curve meta-learning will not be able to generate a learner that supports that business application. Accordingly, we decided to experiment with a new learner, called WHICH, whose internal search criteria can be tuned to a range of goals such as AUC(effort, pd).

5 WHICH

The previous section argued for a change in the goals of data miners. WHICH (Milton 2008) is a meta-learning scheme that uses a configurable search bias to grow its models. This section describes WHICH, how to customize it, and what happened when we applied those customizations to the data of Fig. 3.

⁶For numeric classes, this diversity measure might be the standard deviation of the class feature. For discrete classes, the diversity measure might be the entropy measure used in C4.5.

⁷ROC = receiver-operator characteristic curves such as Lessmann et al.'s plots of *PD-vs-PF* or *PD-vs-precision*.

5.1 Details

WHICH loops over the space of possible feature ranges, evaluating various combinations of features:

- (1) Data from continuous features is discretized into “ N ” equal width bins. We tried various bin sizes and, for this study, best results were seen using $N \in \{2, 4, 8\}$ bins of width $(\max - \min)/N$.
- (2) WHICH maintains a stack of feature combinations, sorted by a customizable search bias B_1 . For this study, WHICH used the AUC(effort, pd) criteria, discussed below.
- (3) Initially, WHICH’s “combinations” are just each range of each feature. Subsequently, they can grow to two or more features.
- (4) Two combinations are picked at random, favoring those combinations that are ranked highly by B_1 .
- (5) The two combinations are themselves combined, scored, then sorted into the stacked population of prior combinations.
- (6) Go to step 4.

For the reader aware of the artificial intelligence (AI) literature, we remark that WHICH is a variant of beam search. Rather than use a fixed beam size, WHICH uses a fuzzy beam where combinations deeper in the stack are exponentially less likely to be selected. Also, while a standard beam search just adds child states to the current frontier, WHICH can add entire sibling branches in the search tree (these sibling branches are represented as other combinations on the stack).

After numerous loops, WHICH returns the highest ranked combination of features. During testing, modules that satisfy this combination are predicted as being “defective”. These modules are sorted on increasing order of size and the statistics of Fig. 9 are collected.

The looping termination criteria was set using our engineering judgment. In studies with UCI data sets (Blake and Merz 1998), Milton showed that the score of top-of-stack condition usually stabilizes in less than 100 picks (Milton 2008) (those results are shown in Fig. 10). Hence, to be cautious, we looped 200 times.

The following expression guides WHICH’s search:

$$B_1 = 1 - \frac{\sqrt{PD^2 * \alpha + (1 - PF)^2 * \beta + (1 - effort)^2 * \gamma}}{\sqrt{\alpha + \beta + \gamma}} \quad (1)$$

The $(PD, PF, effort)$ values are normalized to fall between zero and one. The (α, β, γ) terms in (1) model the relative utility of $PD, PF, effort$ respectively. These values range $0 \leq (\alpha, \beta, \gamma) \leq 1$. Hence:

- $0 \leq B_1 \leq 1$;
- larger values of B_1 are better;
- increasing $(effort, PF, PD)$ leads to $(decreases, decreases, increases)$ in B_1 (respectively).

Initially, we gave PD and $effort$ equal weights and ignored PF ; i.e. $\alpha = 1, \beta = 0, \gamma = 1$. This yielded disappointing results: the performance of the learned detectors

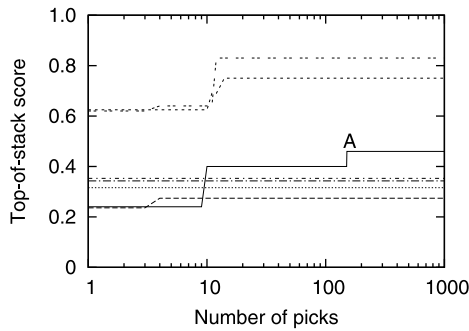


Fig. 10 Top-of-stack scores of the WHICH stack seen after multiple “picks” (selection and scoring of two conditions picked at random, then combined) for seven data sets from the UCI data mining repository (Blake and Merz 1998). Usually, top-of-stack stabilizes after just a dozen pick. However, occasionally, modest improvements are seen after a few hundred “picks” (see the plot marked with an “A”)

varied wildly across our cross-validation experiments. An examination of our data revealed why: there exists a small number of modules with very large LOCs. For example, in one data set with 126 modules, most have under 100 lines of code but a few of them are over 1000 lines of code long. The presence of small numbers of very large modules means that $\gamma = 1$ is not recommended. If the very large modules fall into a particular subset of some cross-validation, then the performance associated with WHICH’s rule can vary unpredictably from one run to another.

Accordingly, we had to use PF as a surrogate measure for $effort$. Recall from the above discussion that we can restrain decreases in PD by assigning a heavy penalty to the false alarms that lead to plateaus in an $effort$ -vs- PD curve. In the following experiments, we used a B_1 equation that disables $effort$ but places a very large penalty on PF ; i.e.

$$\alpha = 1, \quad \beta = 1000, \quad \gamma = 0 \quad (2)$$

We acknowledge that the choice (1) and (2) is somewhat arbitrary. In defense of these decisions, we note that in the following results, these decisions lead to a learner that significantly out-performed standard learning methods.

5.2 Results

Figure 11 shows results from experimental runs with different learners on the data sets of Fig. 3. Each run randomized the order of the data ten times, then performed a $N = 3$ -way cross-val study ($N = 3$ was used since some of our data sets were quite small). For each part of the cross-val study, pd -vs- $effort$ curves were generated using:

- *Manual methods*: manualUp and manualDown;
- *Using standard data miners*: the C4.5 decision tree learner, the RIPPER rule learner, and our previously recommended naïve Bayes method. For more details on these learners, see Appendix. Note that these standard miners included methods that we have advocated in prior publications.

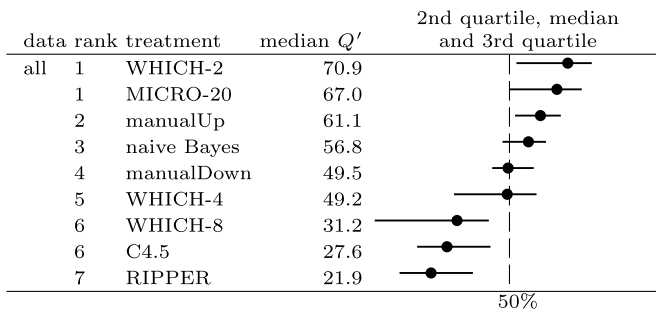


Fig. 11 Results from all data sets of Fig. 3, combined from 10 repeats of a 3-way cross-val, sorted by median Q' . Each row shows 25 to 75% percentile range of the normalized AUC(effort, pdf) results (and the large black dot indicates the median score). Two rows have different ranks (in the left-hand-side column) if their median AUC scores are different and a Mann-Whitney test (95% confidence) indicates that the two rows have a different wins + ties results. Note that we do not recommend WHICH-4 and WHICH-8 since these discretization policies performed much worse than WHICH-2

- *Three versions of WHICH*: This study applied several variants of WHICH. WHICH-2, WHICH-4, and WHICH-8 discretize numeric ranges into 2, 4, and 8 bins (respectively).
- *MICRO-20*: MICRO-20 was another variant motivated by the central limit theorem. According to the central limit theorem, the sum of a large enough sample will be approximately normally distributed (the theorem explains the prevalence of the normal probability distribution). The sample can be quite small, sometimes even as low as 20. Accordingly, MICRO-20 was a variant of WHICH-2 that learns from just 20 + 20 examples of defective and non-defective modules (selected at random).

5.2.1 Overall results

Figure 11 shows the results for *all* the data sets of Fig. 3, combined:

- Each row shows the normalized AUC(effort, pdf) results for a particular learner over 30 experiments (10 repeats of a three-way). These results are shown as a 25% to 75% quartile range (and the large black dot indicates the median score).
- The left-hand-side column of each row shows the results of a Mann-Whitney (95% confidence test) of each row. Row i has a different rank to row $i + 1$ if their median scores are different and the Mann-Whitney test indicates that the two rows have a different wins + ties results. See the appendix for a discussion on why the Mann-Whitney test was used on these results.

In Fig. 11, WHICH performs *relatively* and *absolutely* better than all of the other methods studied in this paper:

- *Relative performance*: WHICH-2 and the MICRO-20 learner have the highest ranks;
- *Absolute performance*: In our discussion of Fig. 9, the *best* curve was presented as the upper bound in performance for any learner tackling AUC(effort, pd).

WHICH's performance rises close to this upper bound, rising to 70.9 and 80% (median and 75% percentile range) of the *best* possible performance.

Several other results from Fig. 11 are noteworthy.

- There is no apparent benefit in detailed discretization: WHICH-2 outperforms WHICH-4 and WHICH-8.
- In a result consistent with our prior publications (Menzies et al. 2007b), our naïve Bayes classifier out-performs other standard data miners (C4.5 and RIPPER).
- In a result consistent with Koru et al.'s logarithmic defect hypothesis, manualUp defeats manualDown.
- In Fig. 11, standard data miners are defeated by manual method (manualUp). The size of the defeat is very large: median values of 61.1% to 27.6% from manualUp to C4.5.

This last result is very sobering. In Fig. 11, two widely used methods (C4.5 and RIPPER) are defeated by manualDown; i.e. by a manual inspection method that Koru et al. would argue is the *worst* possible inspection policy. These results call into question the numerous prior defect prediction results, including several papers written by the authors.

5.2.2 Individual results

Figure 11 combines results from all data sets. Figures 12, 13, 14, and 15 look at each data set in isolation. The results divide into three patterns:

- In the eight data sets of pattern #1 (shown in Figs. 12 and 13), WHICH-2 has *both* the highest median Q' performance *and* is found to be in the top rank by the Mann-Whitney statistical analysis.
- In the two data sets of pattern #2 (shown in Fig. 14), WHICH-2 does not score the highest median performance, but still is found in the top-rank.
- In the one data set that shows pattern #3 (shown in Fig. 15), WHICH-2 is soundly defeated by manual methods (manualUp). However, in this case, the WHICH-2 variant MICRO-20 falls into the second rank.

In summary, when looking at each data set in isolation, WHICH performs very well in $\frac{9}{10}$ of the data sets.

5.3 External validity

We argue that the data sets used in this paper are far broader (and hence, more externally valid) than seen in prior defect prediction papers. All the data sets explored by Lessmann et al. (2008) and our prior work (Menzies et al. 2007b) come from NASA aerospace applications. Here, we use that data, plus three extra data sets from a Turkish company writing software controllers for dishwashers (ar3), washing machines (ar4) and refrigerators (ar5). The development practices from these two organizations are very different:

- The Turkish software was built in a profit- and revenue-driven commercial organization, whereas NASA is a cost-driven government entity.

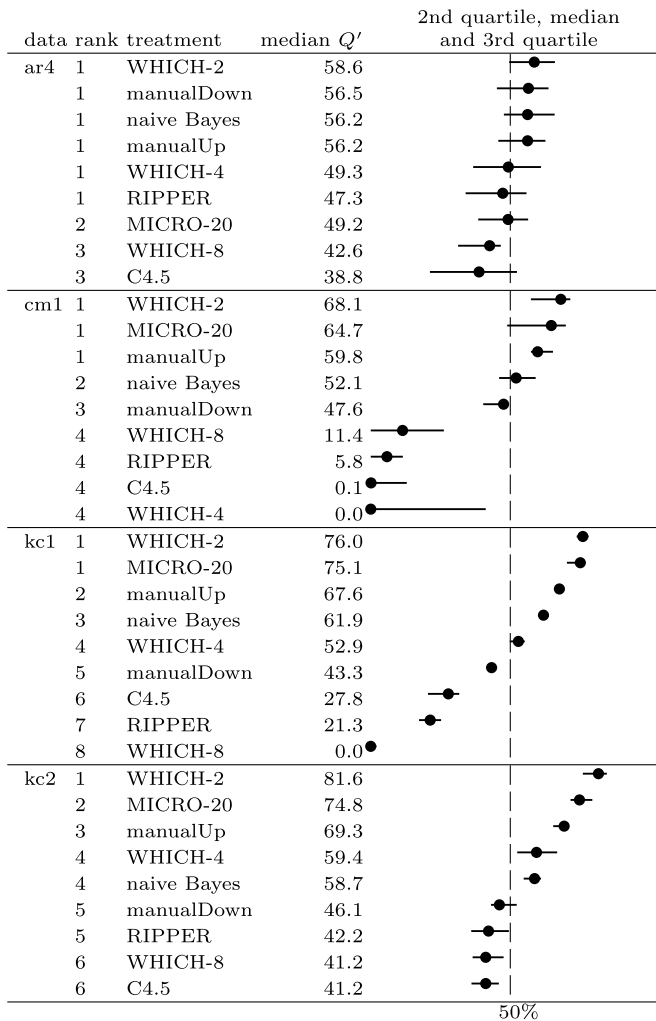


Fig. 12 Four examples of pattern #1: WHICH-2 ranked #1 and has highest median. This figure is reported in the same format as Fig. 11

- The Turkish software was developed by very small teams (2–3 people) working in the same physical location while the NASA software was built by much larger team spread around the United States.
- The Turkish development was carried out in an ad-hoc, informal way rather than the formal, process oriented approach used at NASA.

Our general conclusion, that WHICH is preferred to other methods when optimizing for AUC(effort, pd), holds for $\frac{6}{7}$ of the NASA data sets and $\frac{3}{3}$ of the Turkish sets. The fact that the same result holds for such radically different organizations is a strong argument for the external validity of our results.

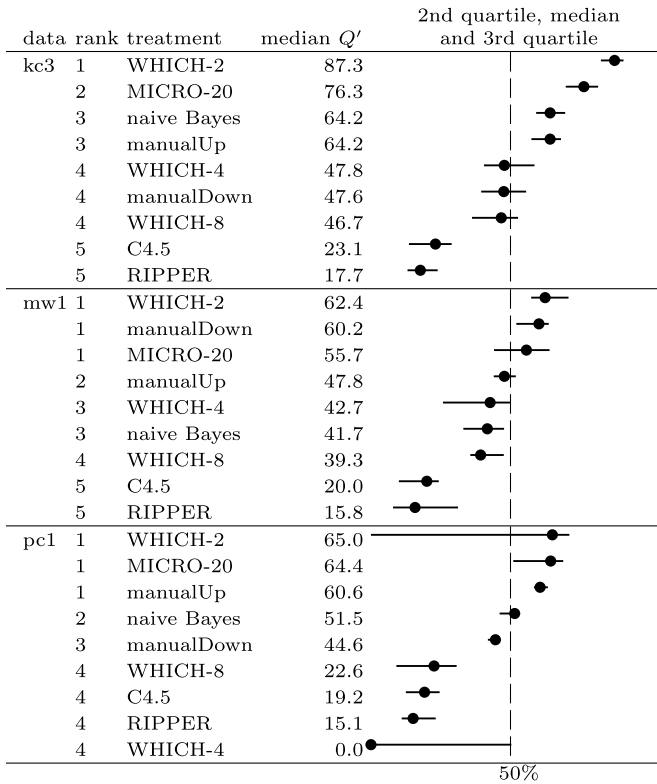


Fig. 13 Three examples of pattern #1: WHICH-2 ranked #1 and has the highest median. This figure is reported in the same format as Fig. 11

While the above results, based on ten data sets, are no promise of the efficacy of WHICH on future data sets, these results are strong evidence that, when a learner is assessed using AUC(effort, pd), then:

- Of all the learners studied here, WHICH or MICRO-20 is preferred over other learners.
- Standard learners such as naïve Bayes, the RIPPER rule learner, and the C4.5 decision tree learner perform much worse than simple manual methods. Hence, we must strongly deprecate their use when optimizing for AUC(effort, pd).

6 Discussion

This goal of this paper was to comment on Lessmann et al.’s results by offering one example where knowledge of the evaluation biases alters which learner “wins” a comparative evaluation study. The current version of WHICH offers that example.

While that goal was reached, there are many open issues that could be fruitfully explored, in future work. Those issues divide into *methodological issues* and *algorithmic issues*.

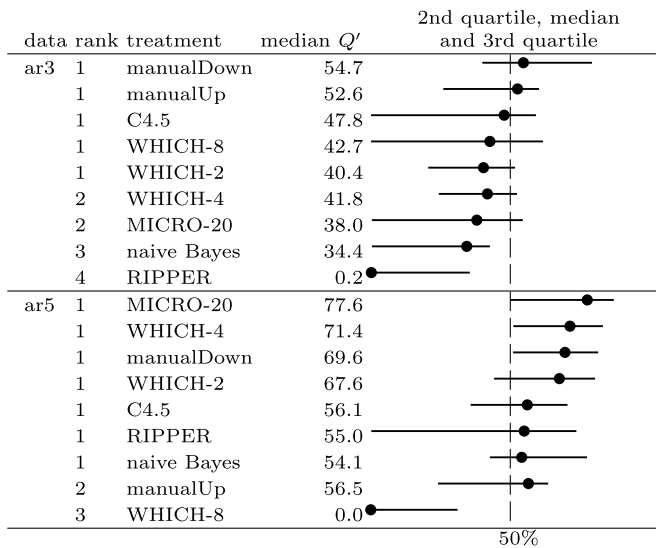


Fig. 14 Two examples of pattern #2: While WHICH-2 did not achieve the highest medians, it was still ranked #1 compared to eight other methods. This *figure* is reported in the same format as Fig. 11

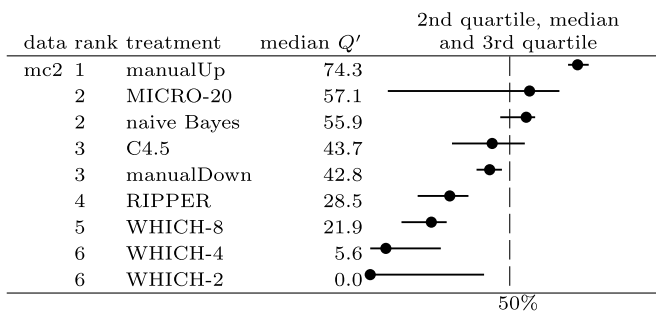


Fig. 15 The only example of pattern #3: WHICH-2 loses (badly) but MICRO-20 still ranks high. This *figure* is reported in the same format as Fig. 11

6.1 Methodological issues

This paper has commented that the use of a new goal (AUC(effort, pd)) resulted in improved performance for certain learners tuned to that new goal. It should be noted that trying different goals for learners randomly is perhaps too expensive. Such an analysis may never terminate since the space of possible goals is very large.

We do not recommend random goal selection. Quite the reverse, in fact. We would propose that:

- *Before* commencing data mining, there must be some domain analysis with the goal of determining the success criteria that most interests the user population. (For a sample of such goals, recall the discussion at the start of Sect. 4 regarding *mission-critical* and *other* systems.)

- Once the business goals have been modeled, then the data miners should be customized to those goals.

That is, rather than conduct studies with randomized business goals, we argue that it is better to *let business considerations guide the goal exploration*. Of course, such an analysis would be pointless unless the learning tool can be adapted to the business goals. WHICH was specially designed to enable the rapid customization of the learner to different goals. For example, while the current version supports AUC(effort, pd), that can be easily changed to other goals.

6.2 Algorithmic issues

The algorithmic issues concerning the inner details of WHICH:

- Are there better values for (α, β, γ) than (2)?
- The above study only explored AUC(effort, pd) and this is only one possible goal of a defect predictor. It could be insightful to explore other goals (e.g. how to skew a learner to maximize precision; or how to choose an evaluation criteria that leads to least variance in the performance of the learner).
- It is possible to restrict the size of the stack to some maximum depth (and new combinations that score less than bottom-of-stack are discarded). For the study shown here, we used an unrestricted stack size.
- Currently, WHICH sorts new items into the stack using a linear time search from top-of-stack. This is simple to implement via a linked list structure but a faster alternative would be a binary-search over skip lists (Pugh 1990).
- Other rule learners employ a greedy back-select to prune conditions. To implement such a search, check if removing any part of the combined condition improves the score. If not, terminate the back select. Otherwise, remove that part and recurse on the shorter condition. Such a back-select is coded in the current version of WHICH, but the above results were obtained with back-select disabled.
- Currently our default value for *MaxLoops* is 200. This may be an overly cautious setting. Given the results of Fig. 10, *MaxLoops* might be safely initialized to 20 and only increased if no dramatic improvement seen in the first loop. For most domains, this would yield a ten-fold speed up of our current implementation.

We encourage further experimentation with WHICH. The current release is released under the GPL3.0 license and can be downloaded from <http://unbox.org/wisp/tags/which>.

7 Conclusion

Given limited QA budgets, it is not possible to apply the most effective QA method to all parts of a system. The manager's job is to decide what needs to be tested most, or tested least. Static code defect predictors are one method for auditing those decisions. Learned from historical data, these detectors can check which parts of the system deserve more QA effort. As discussed in Sect. 2.4, defect predictors learned from

static code measures are *useful* and *easy to use*. Hence, as shown by a list offered in the introduction, they are very *widely-used*.

Based on our own results, and those of Lessmann et al., it seems natural to conclude that many learning methods have equal effectiveness at learning defect predictors from static code features. In this paper, we have shown that this *ceiling effect* does not necessarily hold when studying performance criteria other than AUC(pf, pd). When defect predictors are assessed by other criteria such as “read less, see more defects” (i.e. AUC(effort, pd)), then the selection of the appropriate learner becomes critical:

- A learner tuned to “read less, see more defects” performs best.
- A simple manual analysis out-performs certain standard learners such as NB, C4.5, RIPPER. The use of these learners is therefore deprecated for “read less, see more defects”.

Our conclusion is that knowledge of the *goal* of the learning can and should be used to select a preferred learner for a particular domain. The WHICH meta-learning framework is one method for quickly customizing a learner to different goals.

We hope that this paper prompts a new cycle of defect prediction research focused on selecting the *best* learner(s) for *particular* business goals. In particular, based on this paper, we now caution that it the following is an open and urgent question: “which learners perform better than simple manual method?”

Appendix

Learners used in this study

WHICH, *manualUp*, and *manualDown* was described above. The other learners used in this study come from the WEKA toolkit (Witten and Frank 2005) and are described below.

Naive Bayes classifiers, or NB, offer a relationship between fragments of evidence E_i , a prior probability for a posteriori probability an hypothesis given some evidence $P(H|E)$; and a class hypothesis $P(H)$ probability (in our case, we have two hypotheses: $H \in \{\text{defective}, \text{nonDefective}\}$). The relationship comes from Bayes Theorem: $P(H|E) = \prod_i P(E_i|H) \frac{P(H)}{P(E)}$. For numeric features, a feature’s mean μ and standard deviation σ are used in a Gaussian probability function (Witten and Frank 2005): $f(x) = 1/(\sqrt{2\pi}\sigma)e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. Simple naive Bayes classifiers are called “naive” since they assume independence of each feature. Potentially, this is a significant problem for data sets where the static code measures are highly correlated (e.g. the number of symbols in a module increases linearly with the module’s lines of code). However, Domingos and Pazzini have shown theoretically that the independence assumption is a problem in a vanishingly small percent of cases (Domingos and Pazzani 1997). This result explains (a) the repeated empirical result that, on average, seemingly naïve Bayes classifiers perform as well as other seemingly more sophisticated schemes (e.g. see Table 1 in Domingos and Pazzani (1997)); and (b) our prior

experiments where naive Bayes did not perform worse than other learners that continually re-sample the data for dependent instances (e.g. decision-tree learners like C4.5 that recurse on each “split” of the data (Quinlan 1992b)).

This study used J48 (Witten and Frank 2005), a JAVA port of Quinlan’s C4.5 decision tree learner C4.5, release 8 (Quinlan 1992b). C4.5 is an *iterative dichotomization algorithm* that seeks the best attribute value *splitter* that most simplifies the data that falls into the different splits. Each such splitter becomes a root of a tree. Sub-trees are generated by calling iterative dichotomization recursively on each of the splits. C4.5 is defined for discrete class classification and uses an information-theoretic measure to describe the diversity of classes within a data set. A leaf generated by C4.5 stores the most frequent class seen during training. During test, an example falls into one of the branches in the decision tree and is assigned the class from the leaf of that branch. C4.5 tends to produce big “bushy” trees so the algorithm includes a *pruning* step. Sub-trees are eliminated if their removal does not greatly change the error rate of the tree.

JRip is a JAVA port of the RIPPER (Cohen 1995b) *rule-covering* algorithm. One rule is learned at each pass for one class. All the examples that satisfy the rule condition are marked as *covered* and are removed from the data set. The algorithm then recurses on the remaining data. JRip takes a rather unique stance to rule generation and has operators for *pruning*, *description length* and *rule-set optimization*. For a full description of these techniques, see Dietterich (1997). In summary, after building a *rule*, RIPPER performs a back-select to see what parts of a *condition* can be pruned, without degrading the performance of the rule. Similarly, after building a *set of rules*, RIPPER tries pruning away some of the rules. The learned rules are built while minimizing their *description length*; the size of the learned rules, as well as a measure of the rule errors. Finally, after building rules, RIPPER tries replacing straw-man alternatives (i.e. rules grown very quickly by some naive method).

Details on static code features

This section offers some details on the Halstead and McCabe features.

The Halstead features were derived by Maurice Halstead in 1977. He argued that modules that are hard to read are more likely to be fault prone (Halstead 1977). Halstead estimates reading complexity by counting the number of operators and operands in a module: see the h features of Fig. 1. These three raw h Halstead features were then used to compute the H : the eight derived Halstead features using the equations shown in Fig. 1. In between the raw and derived Halstead features are certain intermediaries:

- $\mu = \mu_1 + \mu_2$;
- minimum operator count: $\mu_1^* = 2$;
- μ_2^* is the minimum operand count (number of module parameters).

An alternative to the Halstead features are the complexity features proposed by Thomas McCabe in 1976. Unlike Halstead, McCabe argued that the complexity of pathways *between* module symbols are more insightful than just a count of the symbols (McCabe 1976). The McCabe measures are defined as follows.

- A module is said to have a *flow graph*; i.e. a directed graph where each node corresponds to a program statement, and each arc indicates the flow of control from one statement to another.
- The *cyclomatic complexity* of a module is $v(G) = e - n + 2$ where G is a program's flow graph, e is the number of arcs in the flow graph, and n is the number of nodes in the flow graph (Fenton and Pfleeger 1995).
- The *essential complexity*, ($ev(G)$) of a module is the extent to which a flow graph can be “reduced” by decomposing all the subflowgraphs of G that are *D-structured primes* (also sometimes referred to as “proper one-entry one-exit subflowgraphs” (Fenton and Pfleeger 1995)). $ev(G) = v(G) - m$ where m is the number of subflowgraphs of G that are D-structured primes (Fenton and Pfleeger 1995).
- Finally, the *design complexity* ($iv(G)$) of a module is the cyclomatic complexity of a module's reduced flow graph.

Choice of statistical test

For several reasons, this study uses the Mann Whitney test. Firstly, many authors, including Demsar (2006), remark that ranked statistical tests such as Mann-Whitney are not susceptible to errors caused by non-Gaussian performance distributions. Accordingly, we do not use t-tests since they make a Gaussian assumption.

Also, recall that Fig. 9 shows the results of a two-stage process: first, select some detectors; second, rank them and watch the *effort-vs-pd* curve grow as we sweep right across Fig. 9 (this two-stage process is necessary to baseline the learners against *manualUp* and *manualDown*, as well as allowing us to express the results as the ratio of a *best* curve). The second stage of this process violates the paired assumptions of, say, the Wilcoxon tests since different test cases may appear depending on which modules are predicted to be defective. Accordingly, we require a non-paired test like Mann Whitney to compare distributions (rather than pairs of treatments applied to the same test case).

Further, while much has been written of the inadequacy of other statistical tests (Demsar 2006; Huang and Ling 2005), to the best of our knowledge, there is no current negative critique of Mann Whitney as a statistical test for data miners.

Lastly, unlike some other tests (e.g. Wilcoxon), Mann-Whitney does not demand that the two compared populations are of the same size. Hence, it is possible to run one test that compares each row of (e.g.) Fig. 12 to every other row in the same division. This simplifies the presentation of the results (e.g. avoids the need for a display of, say, the Bonferroni-Dunn test shown in Fig. 2 of Demsar (2006)).

References

- Arisholm, E., Briand, L.: Predicting fault-prone components in a java legacy system. In: 5th ACM-IEEE International Symposium on Empirical Software Engineering (ISESE), Rio de Janeiro, Brazil, September 21–22 (2006). Available from <http://simula.no/research/engineering/publications/Arisholm.2006.4>
- Blake, C., Merz, C.: UCI repository of machine learning databases (1998). URL: <http://www.ics.uci.edu/~mllearn/MLRepository.html>

- Bradley, P.S., Fayyad, U.M., Reina, C.: Scaling clustering algorithms to large databases. In: Knowledge Discovery and Data Mining, pp. 9–15 (1998). Available from <http://citeseer.ist.psu.edu/bradley98scaling.html>
- Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and regression trees. Tech. rep., Wadsworth International, Monterey, CA (1984)
- Breiman, L.: Random forests. *Mach. Learn.* **45**, 5–32 (2001)
- Breiman, L.: Bagging predictors. *Mach. Learn.* **24**(2), 123–140 (1996)
- Chapman, M., Solomon, D.: The relationship of cyclomatic complexity, essential complexity and error rates. In: Proceedings of the NASA Software Assurance Symposium, Coolfont Resort and Conference Center in Berkley Springs, West Virginia (2002). Available from http://www.ivv.nasa.gov/business/research/osmasas/conclusion2002/Mike_Chapman_The_Relationship_of_Cyclomatic_Complexity_Essential_Complexity_and_Error_Rates.ppt
- Cohen, P.: Empirical Methods for Artificial Intelligence. MIT Press, Cambridge (1995a)
- Cohen, W.: Fast effective rule induction. In: ICML'95, pp. 115–123 (1995b). Available on-line from <http://www.cs.cmu.edu/~wcohen/postscript/ml-95-ripper.ps>
- Cover, T.M., Hart, P.E.: Nearest neighbour pattern classification. *IEEE Trans. Inf. Theory* **IT-13**, 21–27 (1967)
- Demsar, J.: Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.* **7**, 1–30 (2006). Available from <http://jmlr.csail.mit.edu/papers/v7/demsar06a.html>
- Dieterich, T.: Machine learning research: four current directions. *AI Mag.* **18**(4), 97–136 (1997)
- Domingos, P., Pazzani, M.J.: On the optimality of the simple Bayesian classifier under zero-one loss. *Mach. Learn.* **29**(2–3), 103–130 (1997)
- Elkan, C.: The foundations of cost-sensitive learning. In: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01) (2001). Available from <http://www-cse.ucsd.edu/users/elkan/rescale.pdf>
- Fagan, M.: Design and code inspections to reduce errors in program development. *IBM Syst. J.* **15**(3), 182–211 (1976)
- Fagan, M.: Advances in software inspections. *IEEE Trans. Softw. Eng.* **SE-12**, 744–751 (1986)
- Fawcett, T.: Using rule sets to maximize roc performance. In: 2001 IEEE International Conference on Data Mining (ICDM-01) (2001). Available from http://home.comcast.net/~tom.fawcett/public_html/papers/ICDM-final.pdf
- Fenton, N.E., Neil, M.: A critique of software defect prediction models. *IEEE Trans. Softw. Eng.* **25**(5), 675–689 (1999). Available from <http://citeseer.nj.nec.com/fenton99critique.html>
- Fenton, N.E., Pfleeger, S.: *Software Metrics: A Rigorous & Practical Approach*, 2nd edn. International Thompson Press (1995)
- Fenton, N.E., Pfleeger, S.: *Software Metrics: A Rigorous & Practical Approach*. International Thompson Press (1997)
- Fenton, N., Pfleeger, S., Glass, R.: Science and substance: a challenge to software engineers. *IEEE Softw.*, 86–95 (1994)
- Freund, Y., Schapire, R.: A decision-theoretic generalization of on-line learning and an application to boosting. *JCSS: J. Comput. Syst. Sci.* **55** (1997)
- Hall, G., Munson, J.: Software evolution: code delta and code churn. *J. Syst. Softw.* 111–118 (2000)
- Halstead, M.: *Elements of Software Science*. Elsevier, Amsterdam (1977)
- Huang, J., Ling, C.: Using AUC and accuracy in evaluating learning algorithms. *IEEE Trans. Knowledge Data Eng.* **17**(3), 299–310 (2005)
- Jiang, Y., Cukic, B., Ma, Y.: Techniques for evaluating fault prediction models. *Empir. Softw. Eng.*, 561–595 (2008a)
- Jiang, Y., Cukic, B., Menzies, T.: Does transformation help? In: Defects (2008b). Available from <http://menzies.us/pdf/08transform.pdf>
- Khoshgoftaar, T.: An application of zero-inflated Poisson regression for software fault prediction. In: Proceedings of the 12th International Symposium on Software Reliability Engineering, Hong Kong, pp. 66–73 (2001)
- Khoshgoftaar, T., Allen, E.: Model software quality with classification trees. In: Pham, H. (ed.): *Recent Advances in Reliability and Quality Engineering*, pp. 247–270. World Scientific, Singapore (2001)
- Khoshgoftaar, T.M., Seliya, N.: Fault prediction modeling for software quality estimation: comparing commonly used techniques. *Empir. Softw. Eng.* **8**(3), 255–283 (2003)
- Koru, A., Zhang, D., Liu, H.: Modeling the effect of size on defect proneness for open-source software. In: Proceedings PROMISE'07 (ICSE) (2007). Available from <http://promisedata.org/pdf/mps2007KoruZhangLiu.pdf>

- Koru, A., Emam, K.E., Zhang, D., Liu, H., Mathew, D.: Theory of relative defect proneness: replicated studies on the functional form of the size-defect relationship. *Empir. Softw. Eng.*, 473–498 (2008)
- Koru, A., Zhang, D., El Emam, K., Liu, H.: An investigation into the functional form of the size-defect relationship for software modules. *Softw. Eng. IEEE Trans.* **35**(2), 293–304 (2009)
- Lessmann, S., Baesens, B., Mues, C., Pietsch, S.: Benchmarking classification models for software defect prediction: a proposed framework and novel findings. *IEEE Trans. Softw. Eng.* (2008)
- Leveson, N.: *Safeware System Safety and Computers*. Addison-Wesley, Reading (1995)
- Littlewood, B., Wright, D.: Some conservative stopping rules for the operational testing of safety-critical software. *IEEE Trans. Softw. Eng.* **23**(11), 673–683 (1997)
- Lowry, M., Boyd, M., Kulkarni, D.: Towards a theory for integration of mathematical verification and empirical testing. In: *Proceedings, ASE'98: Automated Software Engineering*, pp. 322–331 (1998)
- Lutz, R., Mikulski, C.: Operational anomalies as a cause of safety-critical requirements evolution. *J. Syst. Softw.* (2003). Available from <http://www.cs.iastate.edu/~rlutz/publications/JSS02.ps>
- McCabe, T.: A complexity measure. *IEEE Trans. Softw. Eng.* **2**(4), 308–320 (1976)
- Menzies, T., Cukic, B.: When to test less. *IEEE Softw.* **17**(5), 107–112 (2000). Available from <http://menzies.us/pdf/00iesoft.pdf>
- Menzies, T., Stefano, J.S.D.: How good is your blind spot sampling policy? In: *2004 IEEE Conference on High Assurance Software Engineering* (2003). Available from <http://menzies.us/pdf/03blind.pdf>
- Menzies, T., Raffo, D., Setamanit, S., Hu, Y., Tootoonian, S.: Model-based tests of truisms. In: *Proceedings of IEEE ASE 2002* (2002). Available from <http://menzies.us/pdf/02truisms.pdf>
- Menzies, T., Dekhtyar, A., Distefano, J., Greenwald, J.: Problems with precision. *IEEE Trans. Softw. Eng.* (2007a). <http://menzies.us/pdf/07precision.pdf>
- Menzies, T., Greenwald, J., Frank, A.: Data mining static code attributes to learn defect predictors. *IEEE Trans. Softw. Eng.* (2007b). Available from <http://menzies.us/pdf/06learnPredict.pdf>
- Milton, Z.: Which rules. M.S. thesis (2008)
- Mockus, A., Zhang, P., Li, P.L.: Predictors of customer perceived software quality. In: *ICSE '05: Proceedings of the 27th International Conference on Software Engineering*, pp. 225–233. ACM, New York (2005)
- Musa, J., Iannino, A., Okumoto, K.: *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, New York (1987)
- Nagappan, N., Ball, T.: Static analysis tools as early indicators of pre-release defect density. In: *ICSE 2005, St. Louis* (2005a)
- Nagappan, N., Ball, T.: Static analysis tools as early indicators of pre-release defect density. In: *ICSE*, pp. 580–586 (2005b)
- Nagappan, N., Murphy, B., Basili, V.: The influence of organizational structure on software quality: An empirical case study. In: *ICSE'08* (2008)
- Nikora, A.: Personnel communication on the accuracy of severity determinations in NASA databases (2004)
- Nikora, A., Munson, J.: Developing fault predictors for evolving software systems. In: *Ninth International Software Metrics Symposium (METRICS'03)* (2003)
- Ostrand, T.J., Weyuker, E.J., Bell, R.M.: Where the bugs are. In: *ISSTA'04: Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 86–96. ACM, New York (2004)
- Porter, A., Selby, R.: Empirically guided software development using metric-based classification trees. *IEEE Softw.* 46–54 (1990)
- Pugh, W.: Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM* **33**(6), 668–676 (1990). Available from <ftp://ftp.cs.umd.edu/pub/skipLists/skiplist.pdf>
- Quinlan, J.R.: Learning with continuous classes. In: *5th Australian Joint Conference on Artificial Intelligence*, pp. 343–348 (1992a). Available from <http://citeseer.nj.nec.com/quinlan92learning.html>
- Quinlan, J.R.: *C4.5: Programs for Machine Learning*. Morgan Kaufman, San Mateo (1992b). ISBN: 1558602380
- Raffo, D.: *Personnel communication* (2005)
- Rakitin, S.: *Software Verification and Validation for Practitioners and Managers*, 2nd edn. Artech House, Norwood (2001)
- Shepperd, M., Ince, D.: A critique of three metrics. *J. Syst. Softw.* **26**(3), 197–210 (1994)
- Shull, F., Rus, I., Basili, V.: How perspective-based reading can improve requirements inspections. *IEEE Comput.* **33**(7), 73–79 (2000). Available from <http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/82.77.pdf>

- Shull, F., Boehm, B., B., V., Brown, A., Costa, P., Lindvall, M., Port, D., Rus, I., Tesoriero, R., Zelkowitz, M.: What we have learned about fighting defects. In: Proceedings of 8th International Software Metrics Symposium, Ottawa, Canada, pp. 249–258 (2002). Available from http://fc-md.umd.edu/fcmd/Papers/shull_defects.ps
- Srinivasan, K., Fisher, D.: Machine learning approaches to estimating software development effort. IEEE Trans. Soft. Eng. 126–137 (1995)
- Tang, W., Khoshgoftaar, T.M.: Noise identification with the k -means algorithm. In: ICTAI, pp. 373–378 (2004)
- Tian, J., Zelkowitz, M.: Complexity measure evaluation and selection. IEEE Trans. Softw. Eng. **21**(8), 641–649 (1995)
- Tosun, A., Bener, A.: Ai-based software defect predictors: Applications and benefits in a case study. In: IAAI'10 (2010)
- Tosun, A., Bener, A., Turhan, B.: Practical considerations of deploying ai in defect prediction: a case study within the Turkish telecommunication industry. In: PROMISE'09 (2009)
- Turhan, B., Menzies, T., Bener, A., Distefano, J.: On the relative value of cross-company and within-company data for defect prediction. Empir. Softw. Eng. **68**(2), 278–290 (2009). Available from <http://menzies.us/pdf/08ccwc.pdf>
- Turner, J.: A predictive approach to eliminating errors in software code (2006). Available from http://www.sti.nasa.gov/tto/Spinoff2006/ct_1.html
- Voas, J., Miller, K.: Software testability: the new verification. IEEE Softw. 17–28 (1995). Available from <http://www.cigital.com/papers/download/ieeesoftware95.ps>
- Weyuker, E., Ostrand, T., Bell, R.: Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models. Empir. Softw. Eng. (2008)
- Witten, I.H., Frank, E.: Data Mining, 2nd edn. Morgan Kaufmann, Los Altos (2005)
- Yang, Y., Webb, G.I., Cerquides, J., Korb, K.B., Boughton, J.R., Ting, K.M.: To select or to weigh: a comparative study of model selection and model weighing for spode ensembles. In: ECML, pp. 533–544 (2006)
- Zimmermann, T., Nagappan, N., E.G., H.G., Murphy, B., Cross-project defect prediction. In: ESEC/FSE'09 (2009)

This hour

- Claim:
 - Current SE empirical practice asks for conclusions that are external valid
 - apply to more than one domain
 - So far, such external valid conclusions are illusive
 - Despite decades of research.
- Implications:
 - The goal is wrong
 - Seek not for general theories
 - Only for local lessons.
- “W”
 - a baseline tool for generating local lessons
 - [Case-Based Reasoning vs Parametric Models Software Quality Optimization, Adam Brady, Tim Menzies, PROMISE 2010](#)

What general lessons have we learned from all this data mining?

Only a small minority of PROMISE papers (11/64) discuss results that repeated in data sets from multiple projects

E.g. [Ostrand, Weyuker, Bell PROMISE '08, '09](#)

Same functional form

Predicts defects for generations of AT&T software

E.g. [Turhan, Menzies, Bener PROMISE '08, '09](#)

10 projects

Learn on 9

Apply to the 10th

Defect models learned from NASA projects work for Turkish white goods software

Caveat: need to filter irrelevant training examples. See also

•[When to Use Data from Other Projects for Effort Estimation](#) Ekrem Kocaguneli, Gregory Gay, Tim Menzies, Ye Yang, Jacky W. Keung, ASE 2010

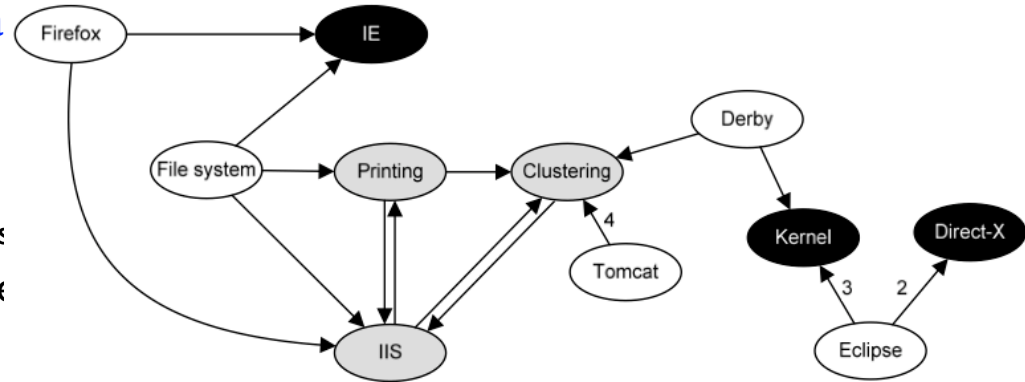
•[B. Turhan, T. Menzies, A. Bener, and J. Distefano. On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering, 68\(2\):278–290, 2009](#)

What general lessons have we learned from all this data mining?

- The usual conclusion is that we learn that we can learn very little

- **FSE'09: Zimmerman et al**

- Defect models not generalizable
 - Learn “there”, apply “here” only works in 4% of their 600+ experiments
- Opposite to Turhan'09 re
 - ?add relevancy filter



- **ASE'09: Green, Menzies et al.**

- AI search for better software project options
- Conclusions highly dependent on local business value proposition

- And others

- **TSE '01, '05: Shepperd et al**

- Any conclusion regarding “best” effort estimator varies by data sets, performance criteria, random selection train/test set

- **TSE'06: Menzies, Greenwald:**

- attributes selected by column selection vary wildly across projects

The gods are angry



- Fenton at PROMISE' 07 (invited talk)
 - "... much of the current software metrics research is inherently irrelevant to the industrial mix ..."
 - "... any software metrics program that depends on some extensive metrics collection is doomed to failure ..."
- Budgen & Kitchenham:
 - "Is Evidence Based Software Engineering mature enough for Practice & Policy?"
 - Need for better reporting: more reviews.
 - Empirical SE results too immature for making policy.
 - [B. Kitchenham D. Budgen, P. Brereton. Is evidence based software engineering mature enough for practice & policy? In 33rd Annual IEEE Software Engineering Workshop 2009 \(SEW-33\), Skvde, Sweden, 2009.](#)
- Basili : still far to go
 - But we should celebrate the progress made over the last 30 years.
 - And we are turning the corner

A new hope (actually, quite old)

- Experience factories
 - Method for find local lessons
- Basili'09 (pers. comm.):
 - “All my papers have the same form.
 - “For the project being studied, we find that changing X improved Y.”
- Translation (mine):
 - Even if we can't find general models (which seem to be quite rare)....
 - ... we can still research general methods for finding local lessons learned





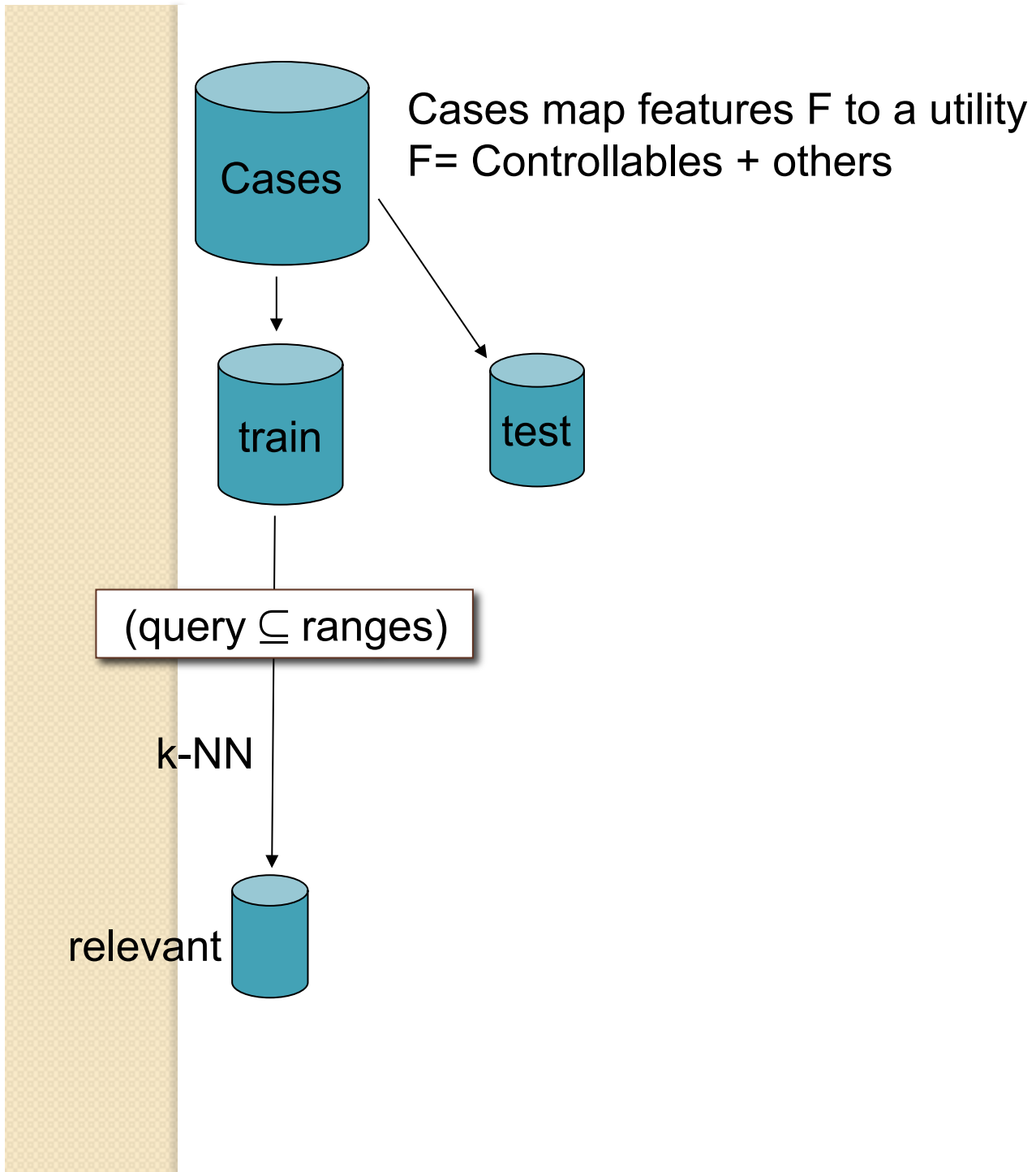
“W” + CBR: Preliminaries

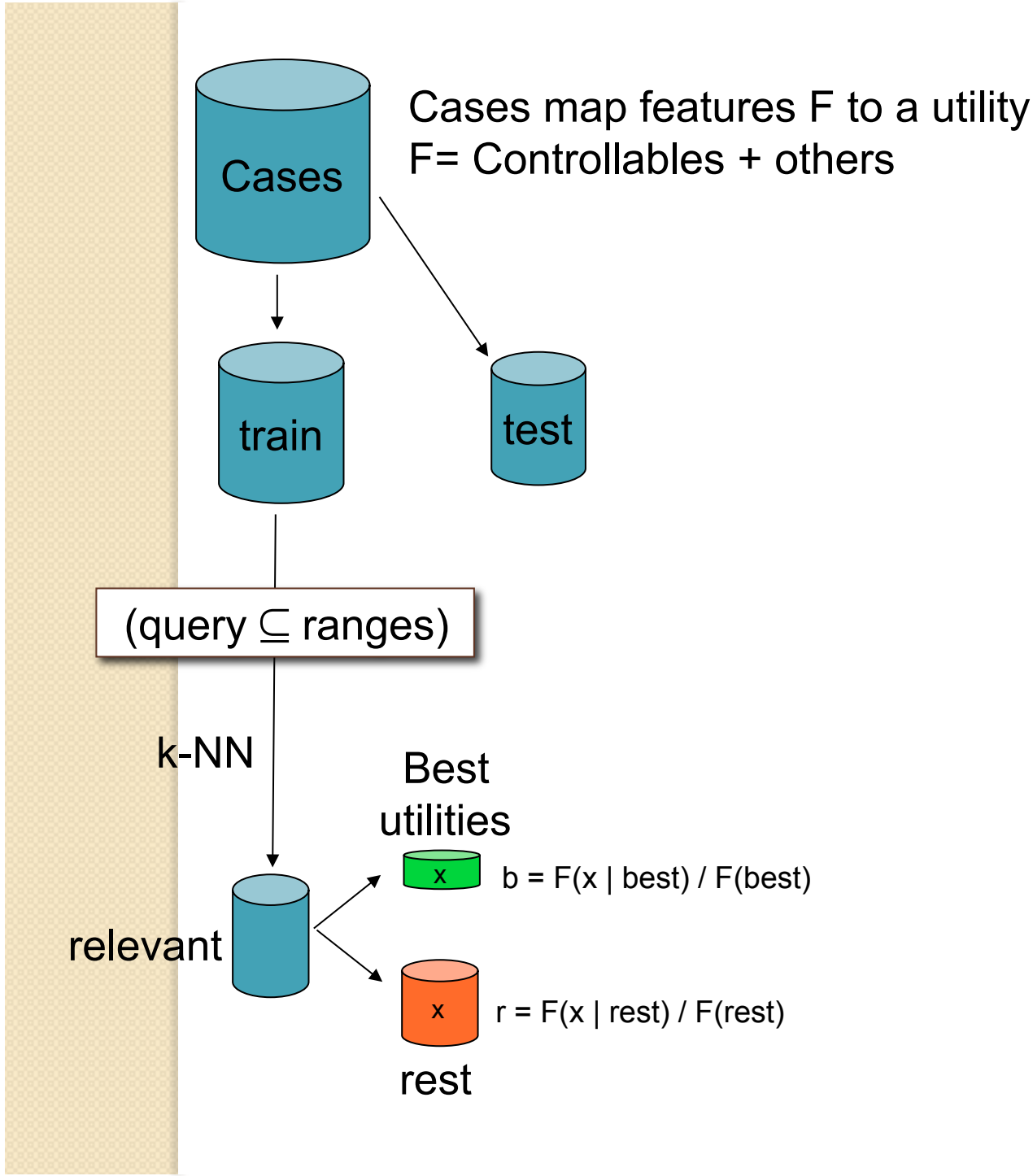
- “Query”
 - What kind of project you want to analyze; e.g.
 - Analysts not so clever,
 - High reliability system
 - Small KLOC
- “Cases”
 - Historical records, with their development effort
- Output:
 - A recommendation on how to change our projects in order to reduce development effort



Cases map features F to a utility
 $F = \text{Controllables} + \text{others}$









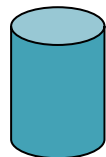
Cases map features F to a utility
 $F = \text{Controllables} + \text{others}$



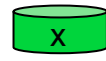
(query \subseteq ranges)

k-NN

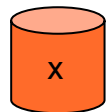
relevant



Best utilities



$$b = F(x \mid \text{best}) / F(\text{best})$$

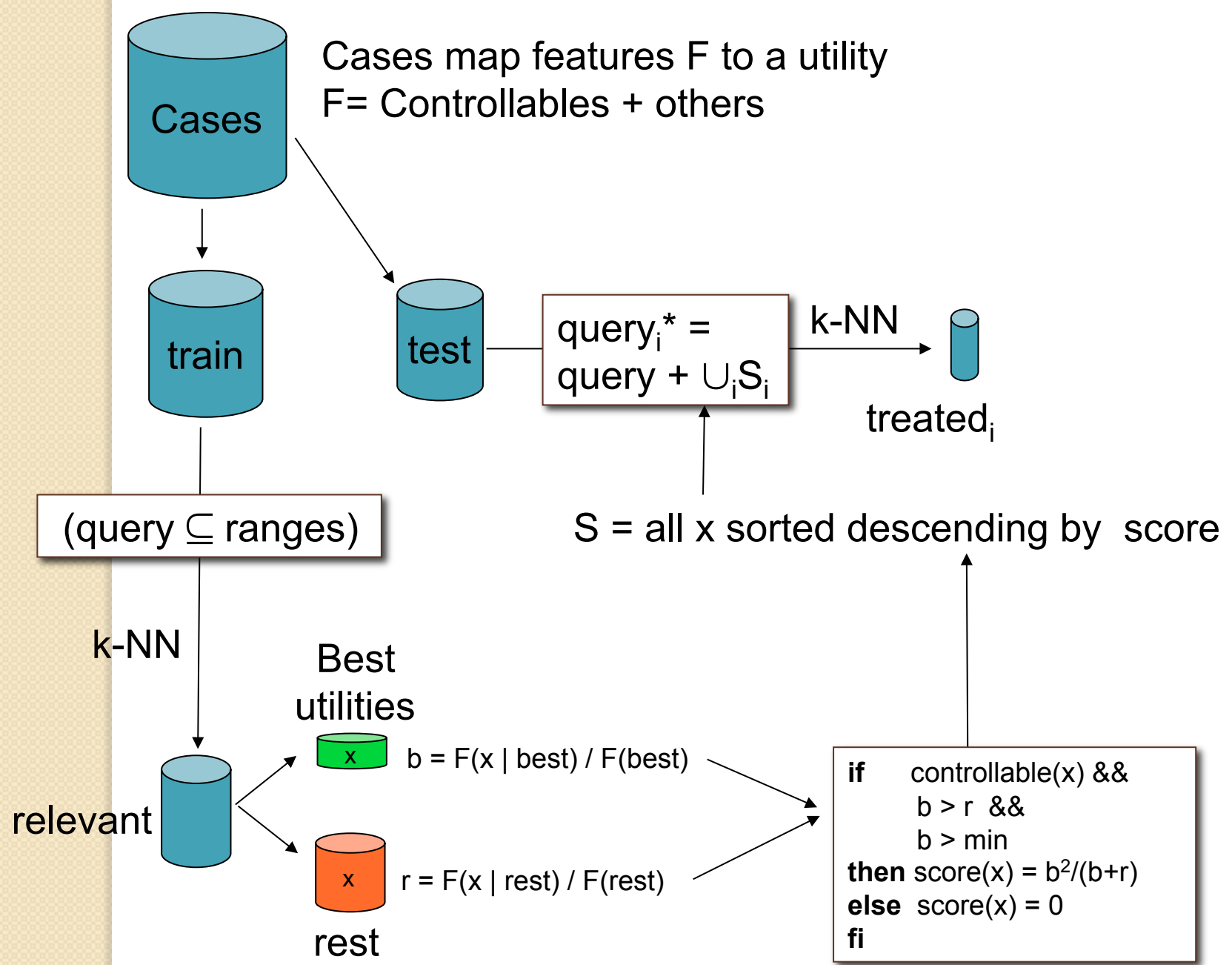


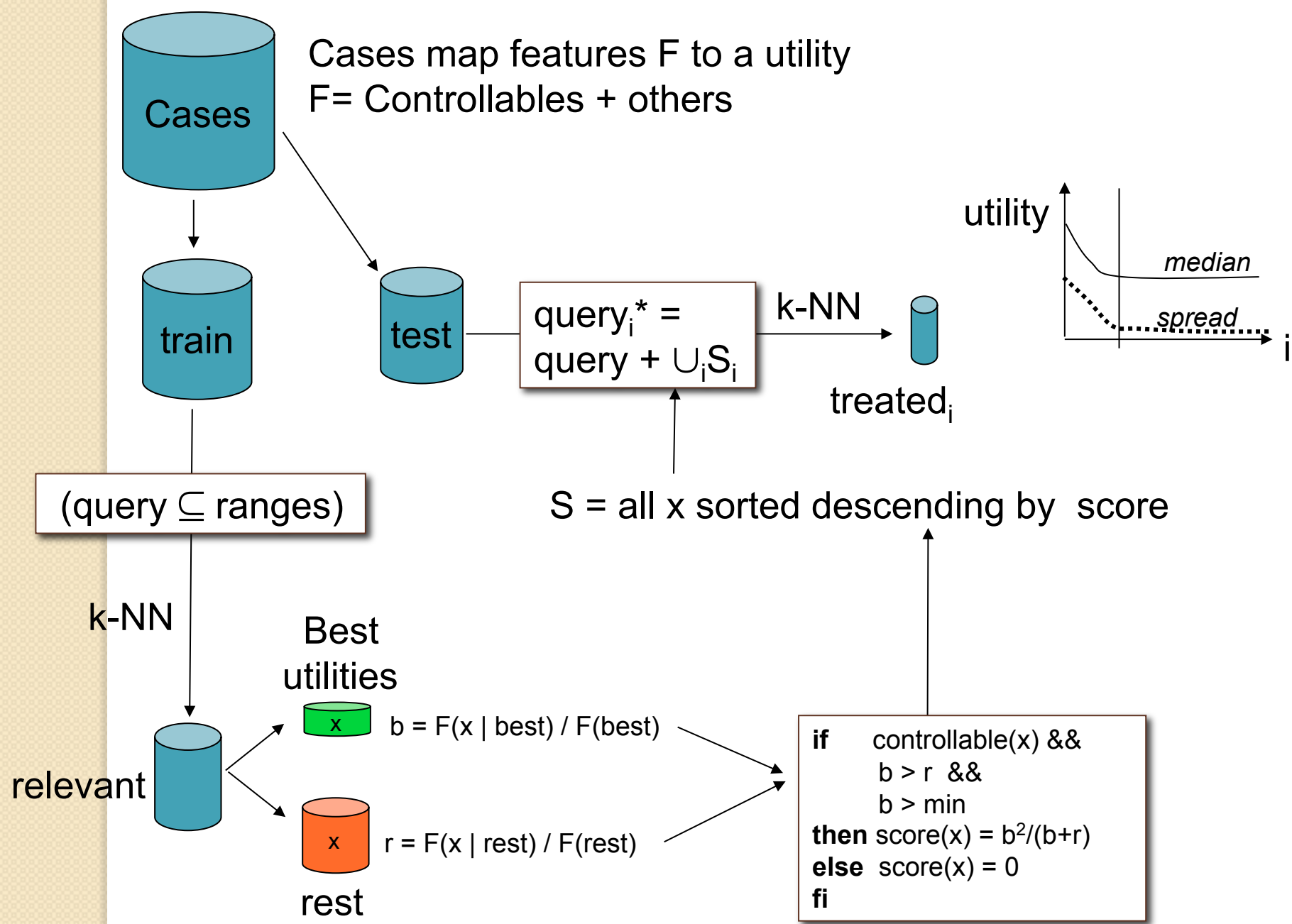
$$r = F(x \mid \text{rest}) / F(\text{rest})$$

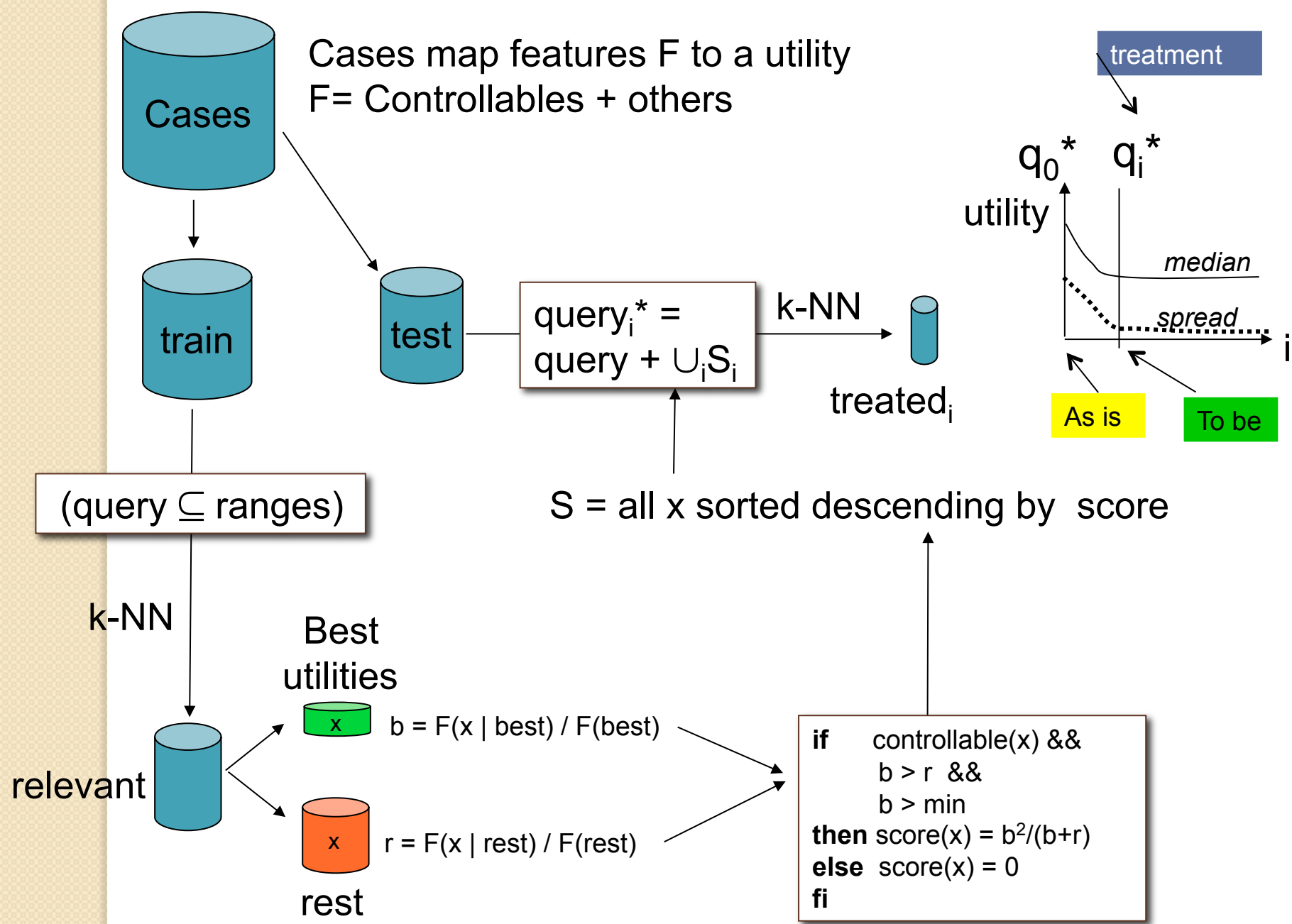
rest

S = all x sorted descending by score

```
if controllable(x) &&
   b > r &&
   b > min
then score(x) = b2/(b+r)
else score(x) = 0
fi
```







Results (distribution of development efforts in q_i^*)

cases	query	X = as is		Y = to be		(X-Y) / X	
		median	spread	median	spread	median	spread
nasa93ground		162	349	99	80	61%	23%
nasa93flight		215	398	131	100	61%	25%
nasa93osp		117.6	396	68	79	58%	20%
nasa93osp2		170	409	95	94	56%	23%
ccc91flight		88	205	24	156	39%	76%

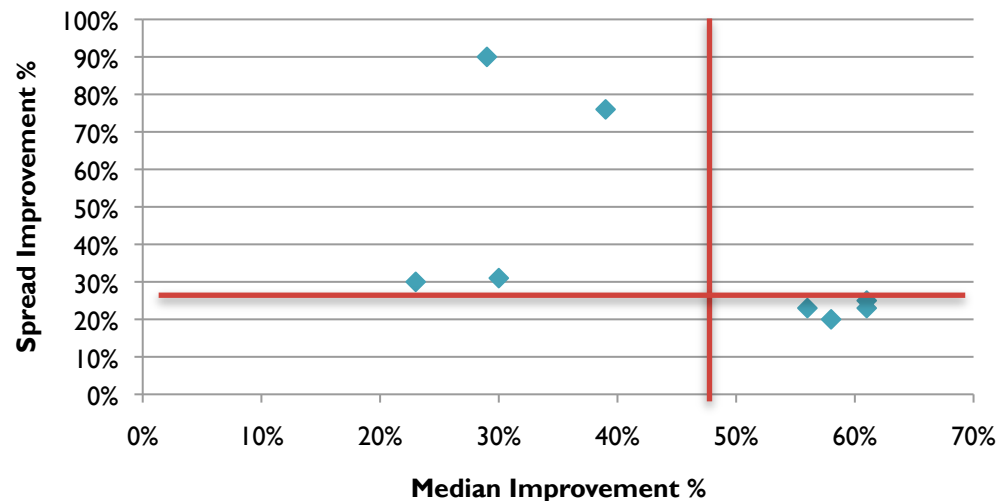
Cases from *promisedata.org/data*

Median = 50% percentile

Spread = 75% - 25% percentile

Improvement = $(X - Y) / X$

- X = as is
- Y = to be
- more is better



Usually:

- spread reduced to 25% of “as is”
- median reduction to 45% of “as is”



But that was so easy

- And that's the whole point
- Yes, finding local lessons learned need not be difficult
- Strange to say...
 - There are no references in the CBR effort estimation literature for anything else than estimate = nearest neighbors
 - No steps beyond into planning , etc
 - Even though that next steps is easy

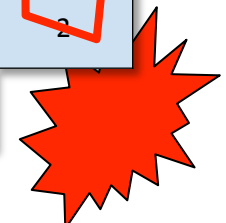
What should change?

$$(q_j^* - q_o^*)$$

<i>cases</i>	<i>query</i>	<i>acap</i>	<i>apex</i>	<i>ltex</i>	<i>ltex</i>	<i>plex</i>	<i>pmat</i>	<i>pmat</i>	<i>sced</i>	<i>sced</i>	<i>stor</i>	<i>time</i>	<i>tool</i>	# of Changes
		3	3	3	4	3	3	4	2	3	3	3	3	
nasa93	ground					100%	55%				85%			3
nasa93	flight					95%	70%				100%			3
nasa93	osp	95%	90%										100%	3
nasa93	osp2				100%			80%	85%					3
coc81	flight						60%					65%		2
coc81	osp2			55%	55%		65%			100%				4
coc81	ground						80%					100%		2
coc81	osp						65%			65%				2
Overall:		12%	11%	7%	19%	24%	49%	10%	11%	21%	23%	21%	13%	

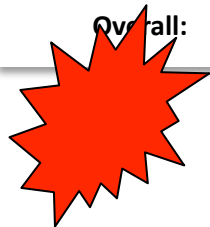
Good news: improving estimates requires very few changes

<i>cases</i>	<i>query</i>	<i>acap</i>	<i>apex</i>	<i>ltex</i>	<i>ltex</i>	<i>plex</i>	<i>pmat</i>	<i>pmat</i>	<i>sced</i>	<i>sced</i>	<i>stor</i>	<i>time</i>	<i>tool</i>	# of Changes
nasa93	ground					100%	55%				85%			3
nasa93	flight					95%	70%				100%			3
nasa93	osp	95%	90%										100%	3
nasa93	osp2				100%			80%	85%					3
coc81	flight						60%					65%		2
coc81	osp2			55%	55%		65%			100%				4
coc81	ground						80%					100%		2
coc81	osp						65%			65%				2
Overall:		12%	11%	7%	19%	24%	49%	10%	11%	21%	23%	21%	13%	



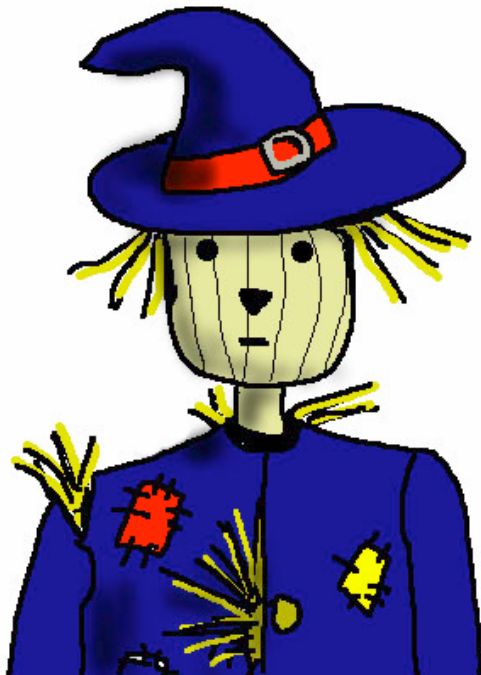
Not-so-good news: local lessons very local

<i>cases</i>	<i>query</i>	<i>acap</i>	<i>apex</i>	<i>ltex</i>	<i>ltex</i>	<i>plex</i>	<i>pmat</i>	<i>pmat</i>	<i>sced</i>	<i>sced</i>	<i>stor</i>	<i>time</i>	<i>tool</i>	<i># of Changes</i>
		3	3	3	4	3	3	4	2	3	3	3	3	
nasa93	ground					100%	55%				85%			3
nasa93	flight					95%	70%				100%			3
nasa93	osp	95%	90%										100%	3
nasa93	osp2				100%			80%	85%					3
coc81	flight						60%					65%		2
coc81	osp2			55%	55%		65%			100%				4
coc81	ground						80%					100%		2
coc81	osp						65%			65%				2
Overall:		12%	11%	7%	19%	24%	49%	10%	11%	21%	23%	21%	13%	



Q: Can we do better than “W”?

A: Most certainly!



- “W” contains at least a dozen arbitrary design decisions
 - Which is best?
- But the algorithm is so simple
 - It should least be a baseline tool
 - Against which we compare supposedly more sophisticated methods.
 - The straw man
- Methodological advice
 - Before getting complex, get simple
 - Warning: often: my straw men don't burn

Certainly, we should always strive for generality

- But don't be alarmed if you can't find it.
- The experience to date is that,
 - with rare exceptions,
 - SE research does not lead to general models
- But that's ok
 - Very few others have found general models (in SE)
 - E.g. Turhan, Menzies, Ayse ESE journal '09
 - B. Turhan, T. Menzies, A. Bener, and J. Distefano. On the relative value of cross-company and within- company data for defect prediction. Empirical Software Engineering, 68(2): 278–290, 2009
 - E.g. Menzies et al ASE conference, 2010
 - When to Use Data from Other Projects for Effort Estimation Ekrem Kocaguneli, Gregory Gay, Tim Menzies, Ye Yang, Jacky W. Keung , ASE 2010
- Anyway
 - If there are few general results, there may be general methods to find local results
 - Seek not “models as products”
 - But general “models to generate products”



Two definitions of “model”

- A hypothetical description of a complex entity or process.
 - Model as output from research machine
 - The “product” of research
- A plan to create, according to a model or models
 - Model of the research machine
 - The “generator” of products
- “W” is a general model generator.

If we can't find general models, is it science?

Popper '60: Everything is a “hypothesis”

- And the good ones have weathered the most attack
- SE “theories” aren't even “hypotheses”
- [Karl Popper, Conjectures and Refutations, London: Routledge and Keagan Paul, 1963](#)

Endres & Rombach '03: Distinguish “observations”, “laws”, “theory”

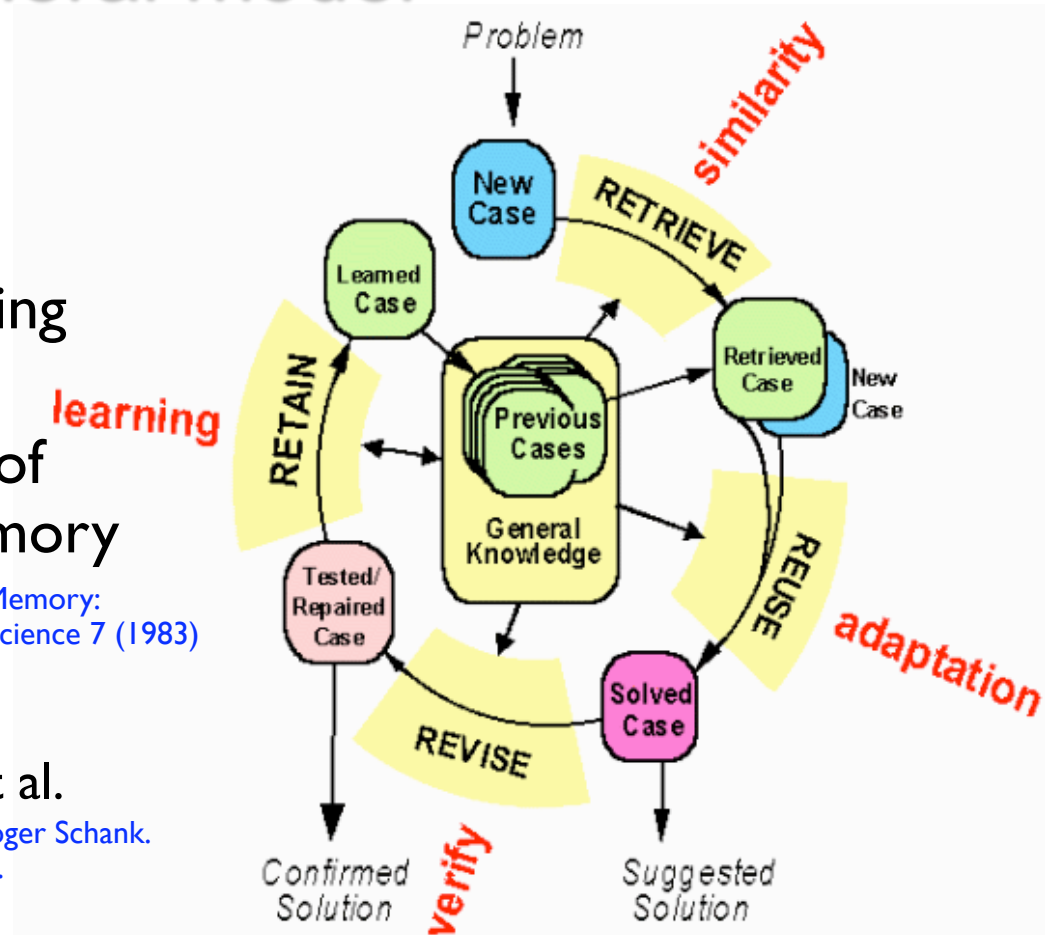
- Laws predict repeatable observations
- Theories explain laws
- Laws are either hypotheses (tentatively accepted) or conjectures (guesses)
- [Rombach A. Endres, H.D. A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories. Addison Wesley, 2003.](#)

Sjoberg '08 : 5 types of “theory”:

- [Building Theories in Software Engineering Dag I. K. Sjøberg, Tore Dyba Bente C. D. Anda and Jo E. Hannay, GUIDE TO ADVANCED EMPIRICAL SOFTWARE ENGINEERING 2008,](#)
- 1. Analysis (e.g. ontologies, taxonomies)
- 2. Explanation (but it is hard to explain “explanation”)
- 3. Prediction (some predictors do not explain)
- 4. Explanation and prediction
- 5. “models” for design + action
 - Don't have to be “right”
 - Just “useful”
 - A.k.a. Endres & Rombach's “laws”?

Btw, constantly (re)building local models is a general model

- Case-based reasoning
- Kolodner's theory of reconstructive memory
 - Janet Kolodner, "Reconstructive Memory: A Computer Model," *Cognitive Science* 7 (1983)
- The Yale group
 - Shank & Riesbeck et al.
 - Riesbeck, Christopher, and Roger Schank. *Inside Case-based Reasoning*. Northvale, NJ: Erlbaum, 1989.
 - Memory, not models
 - Don't "think", remember



Kludges: they work

Ask some good old fashioned AI types

Minsky'86: "Society of Mind"

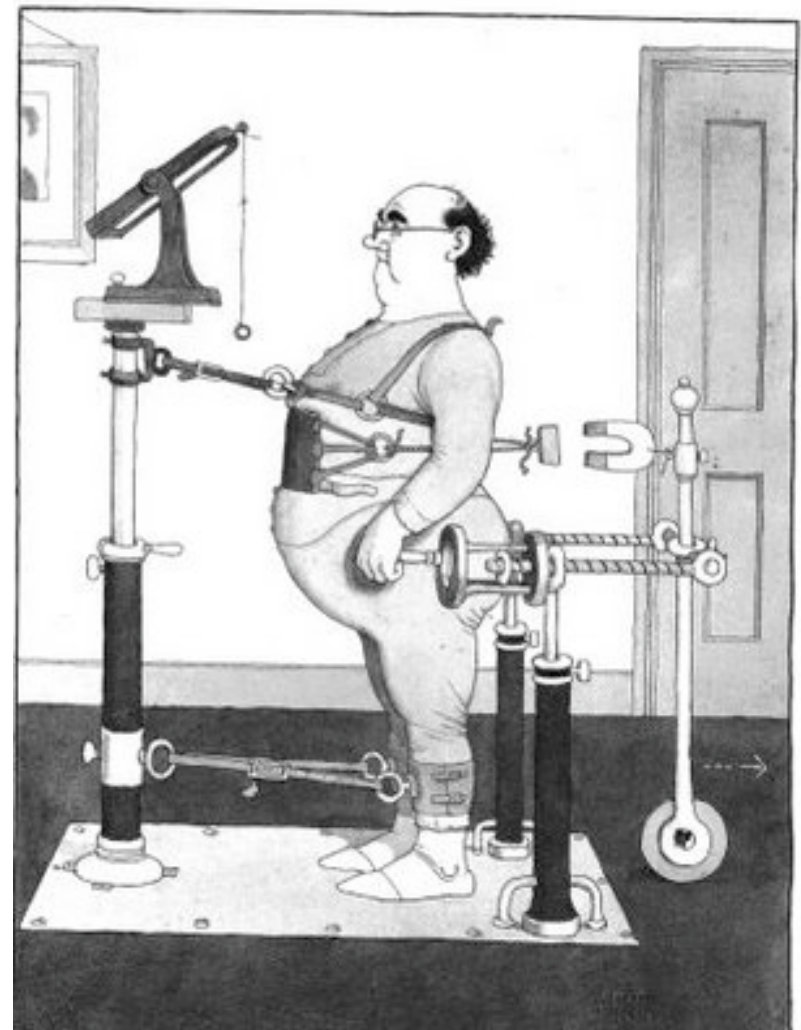
- The brain is a set of 1000+ kludges
 - [Minsky, Marvin The Society of Mind, Simon and Schuster, New York. 1988.](#)

Feigenbaum'83

- Don't take your heart attack to the Maths Dept.
 - Were they will diagnose and treat you using first principles
- Instead, go to the E.R room
 - Staffed by doctors who spent decades learning the quirks of drugs, organs, diseases, people, etc
 - [Edward Feigenbaum and Pamela McCorduck The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World, Addison-Wesley \(1983\)](#)

Seek out those that study kludges.

- You'll be treated faster
- You'll live longer





BIAS (IS YOUR FRIEND)



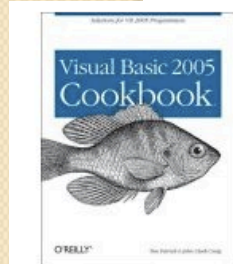
Road map

1. Data mining & SE (overview)
2. Data mining tools (guided tour of “WEKA”)
3. Data “carving” (core operators of DM)
4. Generality (or not)
5. **Bias (is your friend)**
6. Evaluation (does it really work?)

Q: What is the “best” programming language?

A1: Eiffel! (of course)

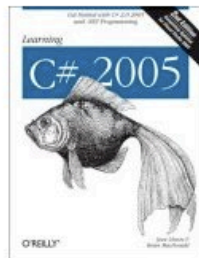
A2: Depends on the bias



Visual Basic 2005 Cookbook by Tim Patrick (Paperback)

Buy new: **\$32.99**
57 used and new from **\$24.43**

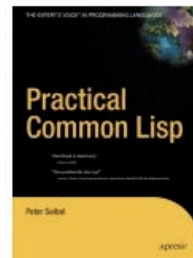
11.



Learning C# 2005 by Jesse Liberty (Paperback)

Buy new: **\$26.39**
56 used and new from **\$21.91**

12.



Practical Common Lisp by Peter Seibel (Hardcover)

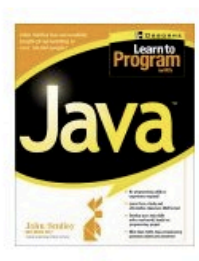
Buy new: **\$34.75**
47 used and new from **\$30.27**

Tag Score: 2



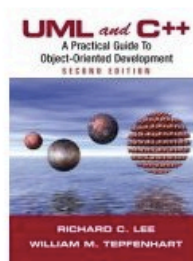
Ruby by Example by Kevin Baird (Paperback)

14.



Learn to Program with Java by John Smiley (Paperback)

15.



UML and C++ by Richard C. Lee (Paperback)



x	language	mean	-
1.0	D Digital Mars	1.58	
1.0	C gcc	1.59	1
1.1	C++ g++	1.71	
1.1	OCaml	1.78	2
1.2	Oberon-2 OO2C	1.84	7
1.2	Clean	1.92	3
1.3	SML MLton	1.98	2
1.3	Lisp SBCL	2.08	3
1.3	BASIC FreeBASIC	2.09	2
1.3	Eiffel SmartEiffel	2.10	2
1.3	Scala	2.11	
1.3	Java 6-server	2.12	
1.4	Nice	2.17	3
1.4	Haskell GHC	2.24	
1.4	Ada 95 GNAT	2.25	2
1.5	C# Mono	2.40	2
1.8	Fortran G95	2.86	6
1.8	Forth bigForth	2.87	1
1.8	CAL	2.88	2
2.8	Lua	4.50	
3.0	Erlang HiPE	4.68	1
3.0	Smalltalk VisualWorks	4.79	1
3.2	Python	5.03	
3.3	Pike	5.28	3
3.5	Scheme MzScheme	5.53	7
3.5	Perl	5.61	2
3.9	Icon	6.09	8
4.1	PHP	6.46	3
4.1	Mozart/Oz	6.51	2
5.3	JavaScript SpiderMonkey	8.37	7
5.4	Tcl	8.55	3
5.5	Ruby	8.63	2
8.8	Prolog SWI	13.96	9

Calculate Reset

multipliers

Full CPU Time

Memory Use

GZip Bytes

benchmark	weight
binary-trees	<input type="text" value="1"/>
chameneos	<input type="text" value="0"/>
cheap-concurrency	<input type="text" value="0"/>
fannkuch	<input type="text" value="1"/>
fasta	<input type="text" value="1"/>
k-nucleotide	<input type="text" value="1"/>
mandelbrot	<input type="text" value="1"/>
meteor-contest	<input type="text" value="0"/>
n-body	<input type="text" value="1"/>
nsieve	<input type="text" value="1"/>
nsieve-bits	<input type="text" value="1"/>
partial-sums	<input type="text" value="1"/>
pidigits	<input type="text" value="1"/>
recursive	<input type="text" value="1"/>
regex-dna	<input type="text" value="1"/>
reverse-complement	<input type="text" value="1"/>
spectral-norm	<input type="text" value="1"/>
startup	<input type="text" value="0"/>
sum-file	<input type="text" value="1"/>

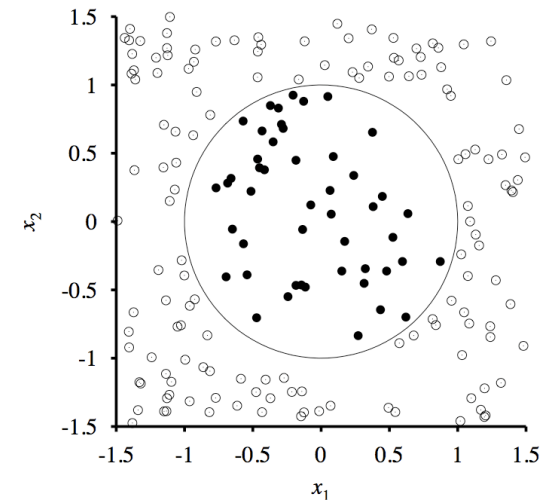
Bias is unavoidable

- Without bias
 - we can't assess relevance / irrelevance
- Without irrelevance,
 - we can't prune the data
- Without pruning,
 - we can't summarize
- Without summarization,
 - we can't generalize
- Without generalizing past experience
 - we can't predict the future
- So bias makes us blind (to some things)
 - But also, it lets us see (the future)

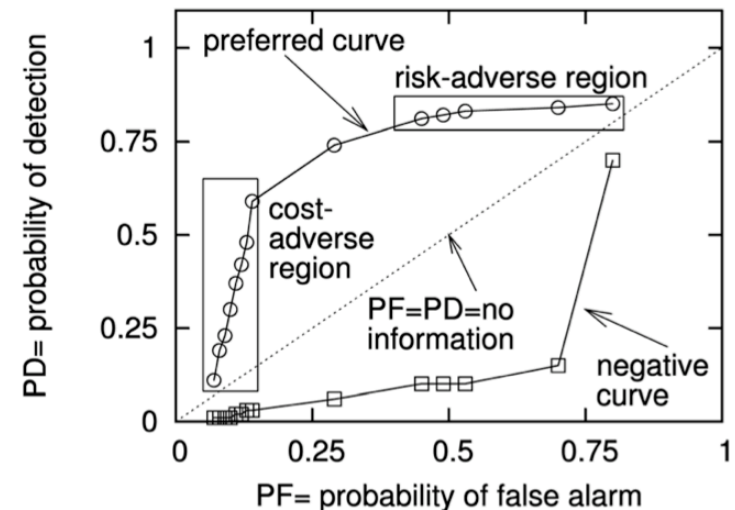


Sources of bias

- Sampling:
 - what data do you select in the pre-process?
- Language
 - E.g. if propositional, can't learn linear equations
- Search
 - When growing a model, what do you look at next?
- Over-fitting avoidance
 - When pruning a model, what is chopped first?
- Evaluation
 - Do you seek high accuracy? high support? What?



e.g. language bias. Hard to describe a circle if your language is restricted to “Z op Value”



Different learners use different biases

- 48 learners, 320 combinations of biases
 - $48/320 = 15\%$
- Separate-and-conquer rule learning]. Furnkranz Artificial Intelligence Review, 13, pages 3--54, 1999. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.4894>

Algorithm	Language Bias						Search Bias						Overfitting Avoidance			
	Static					Dyn.	Algorithm				Strategy		Pre-Pruning	Post-Pruning	Integrated	
	Selectors	Literals	Synt. Restr.	Rel. Clichés	Rule Models	Lang. Hier.	Constr. Ind.	Hill-Climbing	Beam Search	Best First	Stochastic	Top-Down				Bottom-Up
AQ	x							x	x			x				
AQ15	x							x	x			x				x
AQ17	x					x		x	x			x				
ATRIS	x							x			x			x		
BEXA	x							x	x			x			x	x
CHAMP	x	x	x					x	x			x			x	x
CtPF	x					x		x				x				x
CN2	x							x	x			x				x
CN2-MCI	x					x		x	x			x				x
CLASS	x									x		x				
DLG	x							x	x				x			
FOCL	x	x		x				x				x				x
FOIL	x	x	x					x				x				x
FOSSIL	x	x	x	x				x				x				x
GA-SMART	x	x		x	x						x	x				x
GOLEM		x	x					x					x			
GREEDY3	x							x				x				x
GRENDEL					x			x				x				
GROW	x							x				x				x
HYDRA	x	x						x				x				
IBL-SMART	x	x		x						x				x		
INDUCE	x	x						x	x			x				
I-REP, I ² -REP	x	x	x	x				x				x				x
JoJo	x	x						x						x		
m-FOIL	x	x	x					x	x			x				x
MILP	x	x	x								x	x				x
ML-SMART	x	x		x				x	x	x		x				x
NINA					x	x		x					x			
POSEIDON	x							x	x			x				x
PREPEND	x							x				x				
PRISM	x							x				x				
PROGOL	x	x	x							x		x				
REP	x	x		x				x				x				x
RIPPER	x							x				x				x
RDT					x			x				x				x
SFOIL	x										x	x				x
SIA	x										x		x			x
SMART+	x	x		x	x			x	x	x	x	x				x
SWAP-1	x							x						x		x
TDP	x	x	x	x				x				x				x



Bias can change conclusions

- Every data miner has its own bias
- Same data, different data miners, different conclusions
 - Changing biases changes what we best believe
- So, relativistic soup?
 - No basis to make policies, to plan for the future?
 - Data mining is a pack of lies?
 - No more than any other inductive generalization process



Nothing is “right”, but some things are “useful”

- Sure, one data set supports many theories.
 - But there are many many more theories that are unsupported.
- No model is *right*, but some things are *useful*
 - (perform well on test data)
 - George Box
- And many many many more ideas are *useless*
 - Can't make predictions
 - Not defined enough to support (possible) refutation

Embrace bias

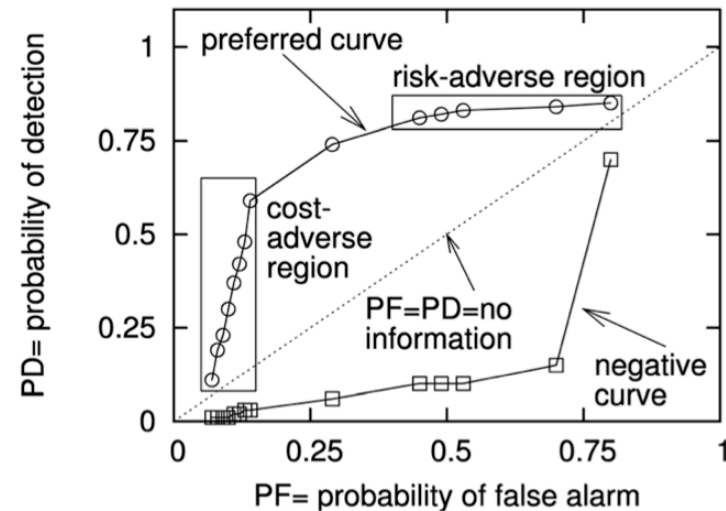
- When reporting a conclusion, report the biases that generated it.
- Make it a first class modeling construct
- Example #1: “W”
 - Recall the sampling bias of “W”
 - Different biases (the query “q”) lead to different conclusions
 - Case-Based Reasoning vs Parametric Models Software Quality Optimization, Adam Brady, Tim Menzies, PROMISE 2010
- Example #2: “WHICH”

Defect prediction from static code features: current results, limitations, new approaches. Tim Menzies, Zach Milton, Burak Turhan, Bojan Cukic, Yue Jiang and Ayşe Bener Automated Software Engineering (2010) 17: 375-407, July 23, 2010. <http://menzies.us/pdf/10which.pdf>



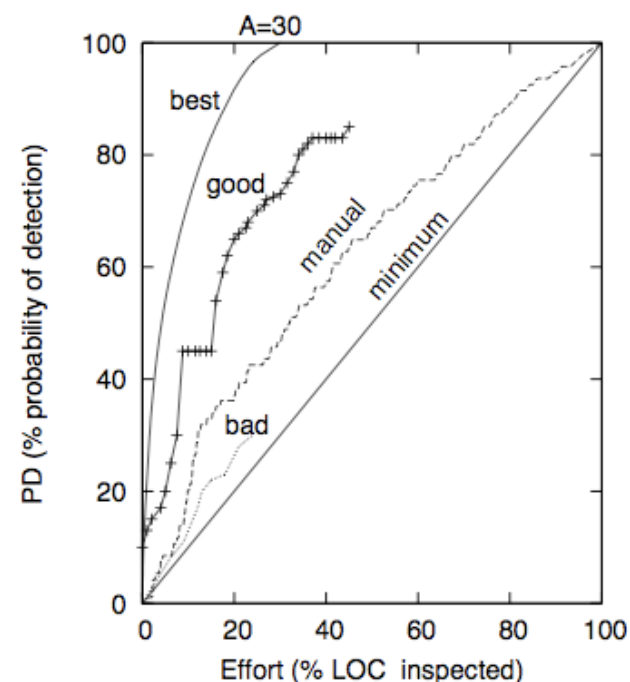
Evaluation Bias #1 : AUC(Pd,Pf)

- Much research
- Little recent improvement:
 - Lessmann, S., Baesens, B., Mues, C., Pietsch, S.: Benchmarking classification models for software defect prediction: a proposed framework and novel findings. IEEE Trans. Softw. Eng. (2008)
- A shallow well?
 - And we've reached the bottom?



Evaluation Bias #2 : AUC(Pd,effort)

- Inspect fewest LOC to find the most bugs.
- Arisholm and Briand[2006]
 - E.Arisholm and L. Briand. Predicting fault-prone components in a java legacy system. In 5th ACM-IEEE International Symposium on Empirical Software Engineering (ISESE), Rio de Janeiro, Brazil, September 21-22, 2006. Available from <http://simula.no/research/engineering/publications/Arisholm.2006.4>.
 - For a budget-conscious team,
 - if X% of modules predicted to be faulty
 - But they contain $\leq X\%$ of the defects,
 - Then that defect predictor is not useful
 - i.e. their bias is $pd > effort$
- Operationalizing their bias:
 - Find modules triggered by the learner
 - Sort them in ascending order of size
 - Assume human inspectors find Δ of the defects in the triggered modules
 - Use ratios of “best” effort-vs-pd curve
 - “best” only triggers on defective modules
 - Note: Δ cancels out



“bad” : worse than manual
“good” : beats manual

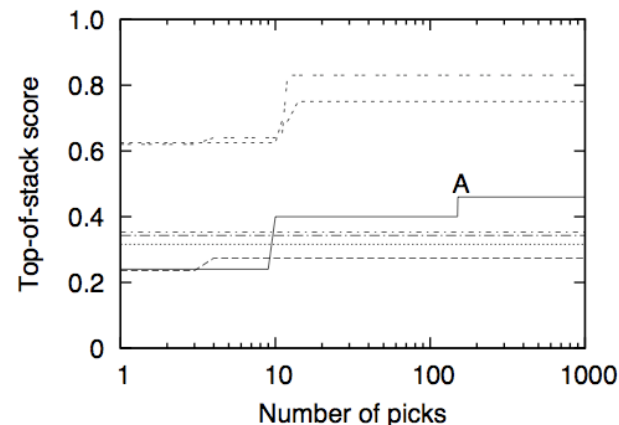
Implementing a bias-specific learner

- All learners have an search bias S and an evaluation bias E . e.g. C4.5:
 - $S = \text{infogain}$
 - $E = \text{pd, pf, accuracy, etc}$
- Note: usually, $\text{not}(S = E)$
- Question: What if we make $S = E$?
 - Answer: “WHICH”



Implementing a bias-specific learner (more)

- Fuzzy beam search
 1. Discretize all numeric features.
 2. Sort all ranges using E on to a stack
 3. Pick any 2 items near top-of-stack
 4. Combine items, score them with E, insert them into the sorted stack.
 5. Goto 3
- Note: no S and E is customizable
- But when to stop? (Use 200 picks)

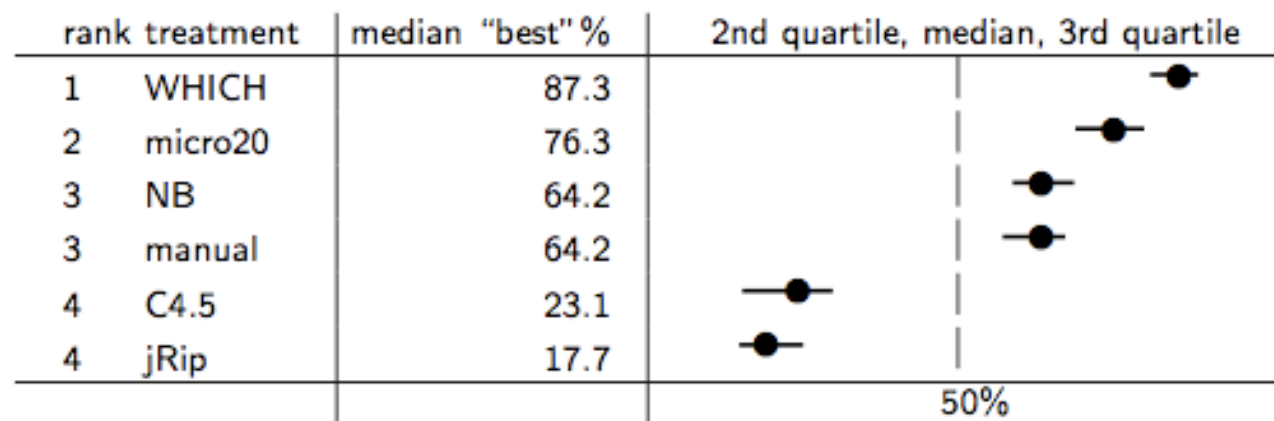


Top of stack stabilizes quickly (UCI data).

Results:

10 random orderings * 3-way cross-val

- 10 sets of static code features from NASA, Turkish whitegoods
- “Rank” computed using Mann-Whitney U test (95%)
- $E = AUC(\text{effort}, \text{pd})$
- Micro20: training on 20 defective + 20 non-defective



WHICH destroys classic learners

- Which were built to optimize accuracy
- So bias changes everything
- BTW, once again a shallow well
 - we do not need much data to do it (40 examples).

Discussion

- Bias changes everything
- But this is not a problem
 - It is a research opportunity
- What biases are current in industrial SE?
 - How do they effect our conclusions?





Coming up...

- Let's focus on one particular bias
 - Evaluation



**EVALUATION
(DOES IT REALLY WORK?)**

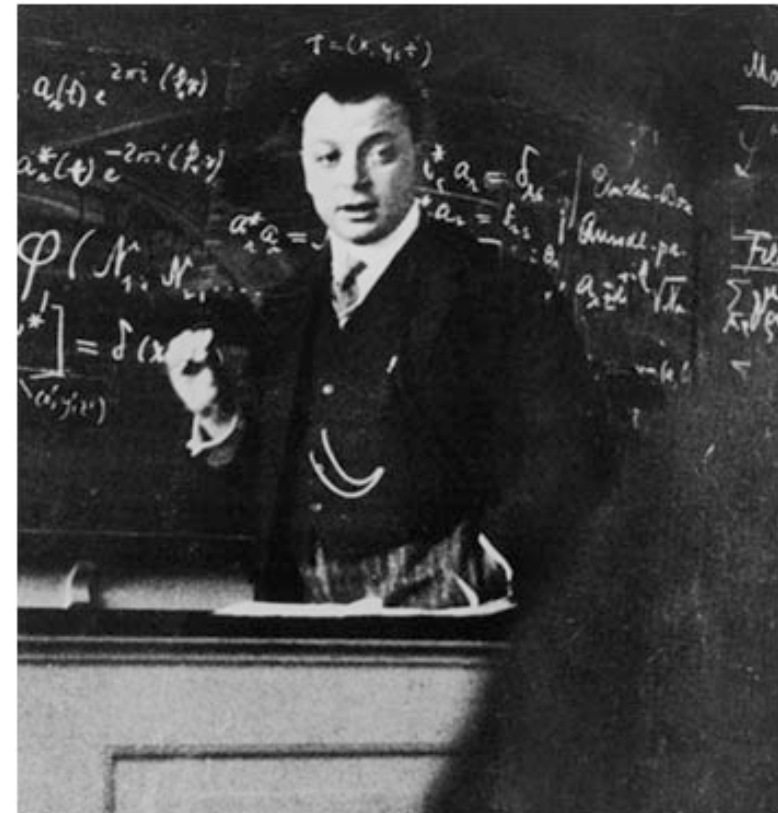


Road map

1. Data mining & SE (overview)
2. Data mining tools (guided tour of “WEKA”)
3. Data “carving” (core operators of DM)
4. Generality (or not)
5. Bias (is your friend)
6. **Evaluation (does it really work?)**

Wolfgang Pauli: the conscience of physics

- The critic to whom his colleagues were accountable.
- Scathing in his dismissal of poor theories
 - often labeling it *ganz falsch*, utterly false.
- But “*ganz falsch*” was not his most severe criticism,
 - He hated theories so unclearly presented as to be
 - untestable
 - unevaluatable,
 - Worse than wrong
 - because they could not be proven wrong.
 - Not properly belonging within the realm of science,
 - even though posing as such.
 - Famously, he wrote of of such unclear paper:
 - “This paper is right. It is not even wrong.”



Lesson: evaluation is important

So evaluation is important

- We saw above how “evaluation” actually became “the learning algorithm”
 - The “WHICH” experiment
- So evaluation is not some post hoc bolt,
 - Only to be explored as an after-thought once the work is done
 - Rather, it is an integral part of the work
 - Best to be get continual feedback from your algorithms as you go along
- BTW: to fail at a data mining Ph.D.
 - Plan to start evaluation in year3

Lesson: build the evaluation rig FIRST

Performance measures for continuous classes

- Absolute residual = $AR = (\text{actual} - \text{predicted})$
- Relative error = $RE = AR/\text{actual}$
- Magnitude of relative error = $MRE = \text{abs}(RE)$
 - Can be surprisingly large (see next slide)
- $MER = AR / \text{predicted}$
- Median MRE, Median MER
- Mean MRE (severely deprecated)
 - Tron Foss, Erik Stensrud, Barbara Kitchenham, Ingunn Myrtveit, "A Simulation Study of the Model Evaluation Criterion MMRE," *IEEE Transactions on Software Engineering*, vol. 29, no. 11, pp. 985-995, Nov. 2003
- $\text{Pred}(X) = \text{percents of RE within } X\% \text{ of actual}$
 - E.g. if 80% of the predictions are within 30% of actual then $\text{Pred}(30) = 80$
 - Note Pred will not notice if a small number of predictions are really bad

Performance measures for discrete classes

a	b	c	<-- classified as
15	0	0	a= Iris-setosa
0	19	0	b=Iris-versicolor
0	2	15	c=Iris-virginica

consider "TRUE"= iris-virginica and FALSE= everything else

	Ground truth	
	FALSE	TRUE
detector silent	A = 34	B = 2
detector loud	C = 0	D = 15

accuracy	$(A+D)/(A+B+C+D)$	$(34+15)/51$	96%
recall (pd)	$D/(B+D)$	$15/(2+15)$	88%
false alarm (pf)	$C/(A+C)$	$0/34$	0%
precision	$D/(C+D)$	$15/(15+0)$	100%
f-measure	$2*prec*pd/(prec+pd)$	$2*1*0.88/(1+0.88)$	94%

Collect separately for each class.

Repeat 10 times (re-ordering data) * 10-way

Repeat for each learner * discretizer * x * y *

Instability and Precision

- Tim Menzies, Alex Dekhtyar, Justin S. Di Stefano, Jeremy Greenwald: Problems with Precision: A Response to "Comments on 'Data Mining Static Code Attributes to Learn Defect Predors'", IEEE Transactions on Software Engineering, Volume 33, Number 9, September 2007

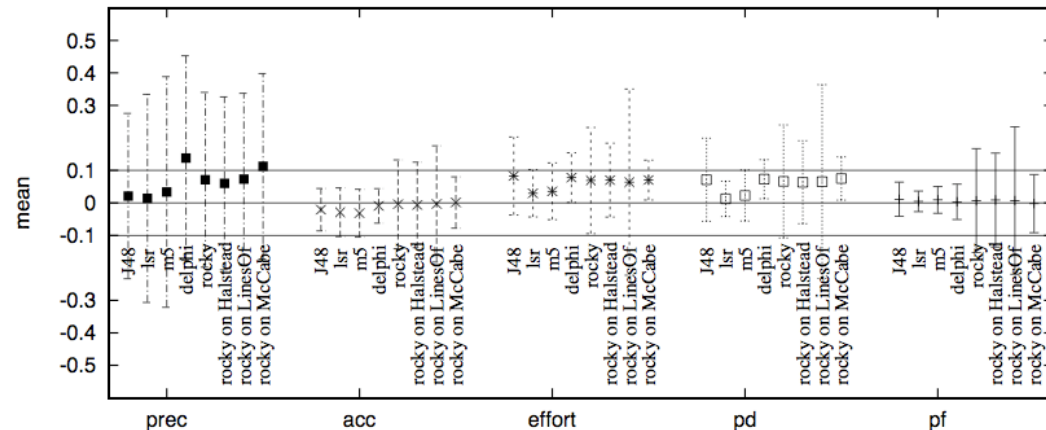
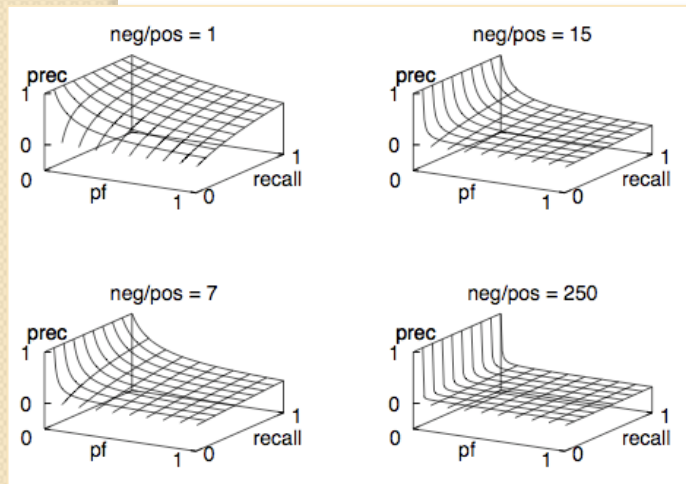
$$\begin{aligned}
 pd = recall &= \frac{D}{B+D} \\
 pf &= \frac{A+C}{D} \\
 prec = precision &= \frac{D+C}{A+D} \\
 acc = accuracy &= \frac{A+B+C+D}{A+B+C+D} \\
 selectivity &= \frac{C+D}{A+B+C+D} \\
 neg/pos &= \frac{A+C}{B+D}
 \end{aligned}$$



$$prec = \frac{D}{D+C} = \frac{1}{1 + \frac{C}{D}} = \frac{1}{1 + neg/pos \cdot pf/recall}$$

which can be rearranged to

$$pf = \frac{pos}{neg} \cdot \frac{(1 - prec)}{prec} \cdot recall$$



Lesson: avoid precision when target class is rare

Strange tales of performance measures

		<u>Truth</u>			
		0	1		
<u>Detector</u>	0	A	B	Prec = $D/(C+D)$	Acc = $(A+D) / (A+B+C+D)$
	1	C	D	PD = $D/(B+D)$	PF = $C / (A+C)$
<hr/>					
<u>Detector</u>	0	0	0	PF = PD = 1 (so detection does not preclude bad false alarm rates)	
	1	10	10		
<hr/>					
<u>Detector</u>	0	80	15	Acc = 85%	(so when target is comparatively rare, Acc does not predict for PD)
	1	0	5	PD = 33%	
<hr/>					
<u>Detector</u>	0	100	0	Acc = 100%	(so highly accurate predictors can miss everything)
	1	0	0	PD = 0	
<hr/>					
<u>Detector</u>	0	0	10	PD = 80%	(so PD does not predict for precision)
	1	50	40	Prec = 44%	

Lesson: avoid Accuracy; consider both PD and Pf

Evaluation is time-consuming

```
analysis1(){
  local origdata=$1
  local outstats=$2
  local nattrs="2 4 6 8 10 12 14 16 18 20"
  local learners="nb10 j4810 zeror10 oner10 adtree10"
  local reducers="infogain chisquared oneR"
  local tmpred=$Tmp/red
  echo "n,reducer,learner,accuracy" > $outstats

  for n in $nattrs; do
    for reducer in $reducers; do
      $reducer $origdata $n $tmpred
      for learner in $learners; do
        accur=`$learner $tmpred.arff | acc
        out="$n,$reducer,$learner,$accur"
        blabln $out
        echo $out >> $outstats
      done
    done
  done
}
```

Learners * data sets * pre-processors

- Repeated 30 – 100 times for statistical validity

Time to run experiments

- Hours to days (first time)

Then comes the “oh dear moment”

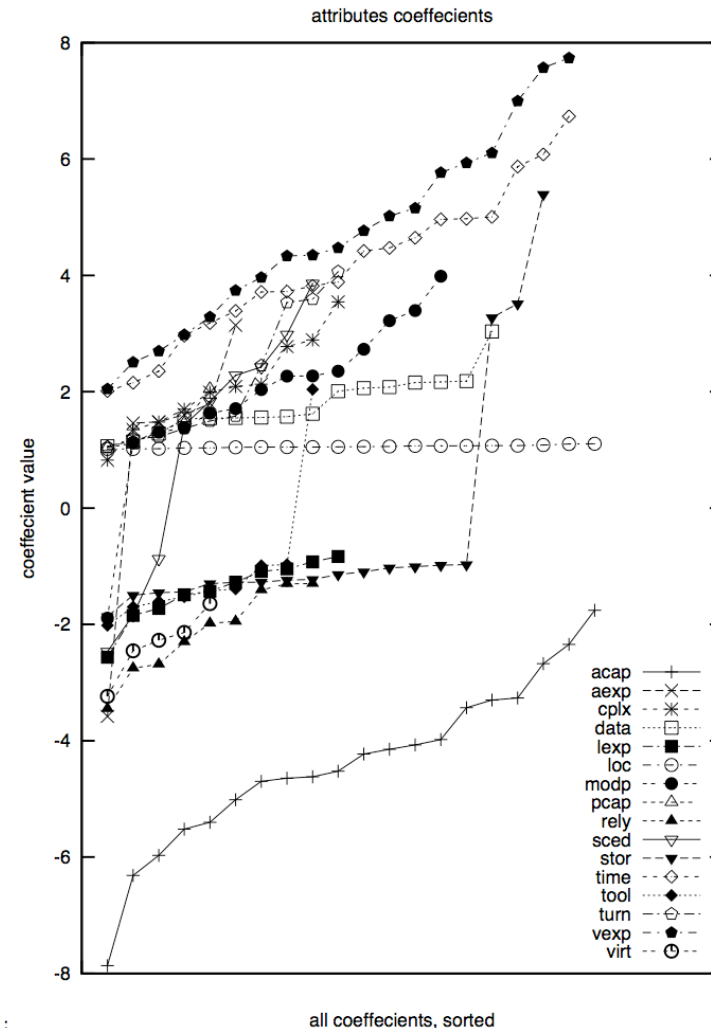
- Do it all again

1 masters = 20 days of CPU (for evaluation)

Lesson: start your evaluations ASAP

Variance problems (more)

- "Simple Software Cost Estimation: Safe or Unsafe?" by Tim Menzies and Zhihao Chen and Dan Port and Jairus Hihn. Proceedings, PROMISE workshop, ICSE 2005 2005 . Available from <http://menzies.us/pdf/05safewhen.pdf> .
- 20 experiments, using 66% of the data (selected at random)
- Linear regression:
 - $\text{Effort} = b_0 + \text{sum of } b_i * x_i$
 - Followed by a greedy back-select to prune dull variables
- Results
 - LOC influence stable
 - Some variables pruned away half the time
 - Large ranges (max – min)
 - Nine attributes even change the sign on their coefficients



Lesson: avoid Accuracy; consider both PD and Pf

Evaluation (using hypothesis testing) is contentious

- Statistical significance tests of the form (H0 vs H1) are a ‘potent but sterile intellectual rake who leaves . . . no viable scientific offspring’.
 - Cohen J. 1988. The earth is round ($p < .05$). *American Psychologist* 49: 997 – 1003.
- Consider one study showing that, using significance testing, estimates from multiple sources are no better than those from a single source.
 - How to explain 31 other studies where multiple sources out-performed single source by 3.4 to 23.4% (average = 12.5%).
 - Odds of that happening at random?
 - $2^{31} <$ less than a billionth
 - Armstrong JS. 2007. Significance tests harm progress in forecasting. *International Journal of Forecasting* 23: 21 – 327.

Table: Error Reductions from Combining Ex Ante Forecasts

Study	Methods	Components	Criterion	Data	Situation	Validation Forecasts	Forecast Horizon	Percent error reduction
Levine (1960)	intentions	2	MAPE	annual	capital expenditures	6	1	18.0
Okun (1960)	"	2	"	"	housing starts	6	1	7.0
Landefeld & Seskin (1986)	"	2	MAE	"	plant & equipment	11	1	20.0
Armstrong et al. (2000)	"	4	RAE	"	consumer products	65	varied	5.5
Winkler & Poses (1993)	expert	4	Brier	cross-section	survival of patients	231	varied	12.2
Thorndike (1938)	"	4 to 6	% wrong	"	knowledge questions	30	varied	6.6
Makridakis et al. (1993)	"	5	MAPE	monthly	economic time series	322	1 thru 14	19.0
Richards & Fraser (1977)	"	5	"	annual	company earnings	213	1	8.1
Batchelor & Dua (1995)	"	10	MSE	"	macroeconomic	40	1	16.4
Kaplan et al. (1950)	"	26	% wrong	cross-section	technology events	16	varied	13.0
Zarnowitz (1984)	"	79	RMSE	quarterly	macroeconomic	288	1	10.0
Sanders & Ritzman (1989)	extrapolation	3	MAPE	daily	public warehouse	260	1	15.1
Makridakis & Winkler (1983)	"	5	"	monthly	economic time series	617	18	24.2
Makridakis et al. (1993)	"	5	"	"	"	322	1 thru 14	4.3
Lobo (1992)	"	5	"	quarterly	company earnings	6,560	1 thru 4	13.6
Schmaers (1986)	"	7	"	annual	consumer products	1,412	1 thru 5	20.0
Landefeld & Seskin (1986)	econometric	2	MAE	annual	plant & equipment	7	1	21.0
Clemen & Winkler (1986)	"	4	MAD	quarterly	GNP (real & nominal)	45	1 thru 4	3.4
Shamseldin et al. (1997)	"	5	MAPE	annual	rainfall runoff	22	1	9.4
Lobo (1992)	expert/extrap	2	MAPE	"	company earnings	6,560	1 thru 4	11.0
Lawrence et al. (1986)	"	3	"	annual monthly	economic time series	1,224	1 thru 18	10.7
Sanders & Ritzman (1989)	"	3	"	daily	public warehouse	260	1	15.5
Lobo & Nair (1990)	"	4	"	annual	company earnings	768	1	6.4
Landefeld & Seskin (1986)	intentions/econ	2	MAE	annual	plant & equipment	11	1	11.5
Vandome (1963)	extrap/econ	2	MAPE	quarterly	macroeconomic	20	1	10.1
Armstrong (1985)	"	2	"	annual	photo sales by country	17	6	4.2
Weinberg (1986)	expert/econ	2	"	cross-section	performing arts	15	varied	12.5
Bessler & Brandt (1981)	exprt/extrap/econ	3	"	quarterly	cattle & chicken prices	48	1	13.6
Fildes (1991)	"	3	MAE	annual	construction	72	1 & 2	8.0
Brandt & Bessler (1983)	"	6	MAPE	quarterly	hog prices	24	1	23.5
Unweighted average								12.5

Lesson: Don't base conclusions on just hypothesis testing

Evaluation is humbling

- All that clever programming, then...
 - Then simpler ideas do as well, or better, than the more sophisticated
- Example
 - E.g. “Bayes”= simple correlation unaware learner
 - C4.5 = more sophisticated method, correlation aware
 - And no evidence here that the added complexity of C4.5 is better than dumb Bayes
 - [Pedro Domingos and Michael J. Pazzani, On the Optimality of the Simple Bayesian Classifier under Zero-One Loss, Machine Learning, Volume 29, number 2-3, pages 103-130, 1997](#)

Table 1. Classification accuracies and sample standard deviations, averaged over 20 random training/test splits. “Bayes” is the Bayesian classifier with discretization and “Gauss” is the Bayesian classifier with Gaussian distributions. Superscripts denote confidence levels for the difference in accuracy between the Bayesian classifier and the corresponding algorithm, using a one-tailed paired *t* test: 1 is 99.5%, 2 is 99%, 3 is 97.5%, 4 is 95%, 5 is 90%, and 6 is below 90%.

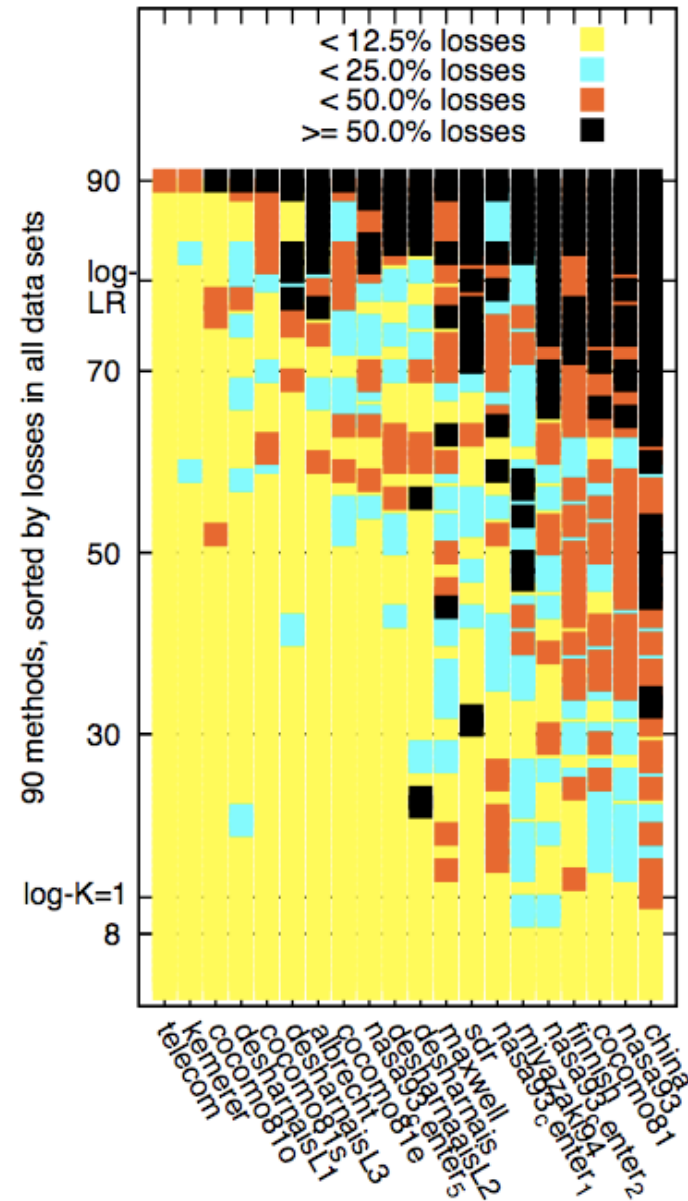
Data Set	Bayes	Gauss	C4.5	PEBLs	CN2	Def.
Audiology	73.0±6.1	73.0±6.1 ⁶	72.5±5.8 ⁶	75.8±5.4 ³	71.0±5.1 ⁵	21.3
Annealing	95.3±1.2	84.3±3.8 ¹	90.5±2.2 ¹	98.8±0.8 ¹	81.2±5.4 ¹	76.4
Breast cancer	71.6±4.7	71.3±4.3 ⁶	70.1±6.8 ⁵	65.6±4.7 ¹	67.9±7.1 ¹	67.6
Credit	84.5±1.8	78.9±2.5 ¹	85.9±2.1 ³	82.2±1.9 ¹	82.0±2.2 ¹	57.4
Chess endgames	88.0±1.4	88.0±1.4 ⁶	99.2±0.1 ¹	96.9±0.7 ¹	98.1±1.0 ¹	52.0
Diabetes	74.5±2.4	75.2±2.1 ⁶	73.5±3.4 ⁵	71.1±2.4 ¹	73.8±2.7 ⁶	66.0
Echocardiogram	69.1±5.4	73.4±4.9 ¹	64.7±6.3 ¹	61.7±6.4 ¹	68.2±7.2 ⁶	67.8
Glass	61.9±6.2	50.6±8.2 ¹	63.9±8.7 ⁶	62.0±7.4 ⁶	63.8±5.5 ⁶	31.7
Heart disease	81.9±3.4	84.1±2.8 ¹	77.5±4.3 ¹	78.9±4.0 ¹	79.7±2.9 ³	55.0
Hepatitis	85.3±3.7	85.2±4.0 ⁶	79.2±4.3 ¹	79.0±5.1 ¹	80.3±4.2 ¹	78.1
Horse colic	80.7±3.7	79.3±3.7 ¹	85.1±3.8 ¹	75.7±5.0 ¹	82.5±4.2 ²	63.6
Hypothyroid	97.5±0.3	97.9±0.4 ¹	99.1±0.2 ¹	95.9±0.7 ¹	98.8±0.4 ¹	95.3
Iris	93.2±3.5	93.9±1.9 ⁶	92.6±2.7 ⁶	93.5±3.0 ⁶	93.3±3.6 ⁶	26.5
Labor	91.3±4.9	88.7±10.6 ⁶	78.1±7.9 ¹	89.7±5.0 ⁶	82.1±6.9 ¹	65.0
Lung cancer	46.8±13.3	46.8±13.3 ⁶	40.9±16.3 ⁵	42.3±17.3 ⁶	38.6±13.5 ³	26.8
Liver disease	63.0±3.3	54.8±5.5 ¹	65.9±4.4 ¹	61.3±4.3 ⁶	65.0±3.8 ³	58.1
LED	62.9±6.5	62.9±6.5 ⁶	61.2±8.4 ⁶	55.3±6.1 ¹	58.6±8.1 ²	8.0
Lymphography	81.6±5.9	81.1±4.8 ⁶	75.0±4.2 ¹	82.9±5.6 ⁶	78.8±4.9 ³	57.3
Post-operative	64.7±6.8	67.2±5.0 ³	70.0±5.2 ¹	59.2±8.0 ²	60.8±8.2 ⁴	71.2
Promoters	87.9±7.0	87.9±7.0 ⁶	74.3±7.8 ¹	91.7±5.9 ³	75.9±8.8 ¹	43.1
Primary tumor	44.2±5.5	44.2±5.5 ⁶	35.9±5.8 ¹	30.9±4.7 ¹	39.8±5.2 ¹	24.6
Solar flare	68.5±3.0	68.2±3.7 ⁶	70.6±2.9 ¹	67.6±3.5 ⁶	70.4±3.0 ²	25.2
Sonar	69.4±7.6	63.0±8.3 ¹	69.1±7.4 ⁶	73.8±7.4 ¹	66.2±7.5 ⁵	50.8
Soybean	100.0±0.0	100.0±0.0 ⁶	95.0±9.0 ³	100.0±0.0 ⁶	96.9±5.9 ³	30.0
Splice junctions	95.4±0.6	95.4±0.6 ⁶	93.4±0.8 ¹	94.3±0.5 ¹	81.5±5.5 ¹	52.4
Voting records	91.2±1.7	91.2±1.7 ⁶	96.3±1.3 ¹	94.9±1.2 ¹	95.8±1.6 ¹	60.5
Wine	96.4±2.2	97.8±1.2 ³	92.4±5.6 ¹	97.2±1.8 ⁶	90.8±4.7 ¹	36.4
Zoology	94.4±4.1	94.1±3.8 ⁶	89.6±4.7 ¹	94.6±4.3 ⁶	90.6±5.0 ¹	39.4

Lesson: baseline your new method against a simpler alternative

Evaluation is humbling (2)

- 90 data miners
 - 9 learners with
 - 10 pre-processors
- 20 datasets
- (Win – Loss) results when one miner is compared to 89 others.
- Sum of five different performance measures
- And most miners perform about the same

Lesson: beware “ceiling effects”



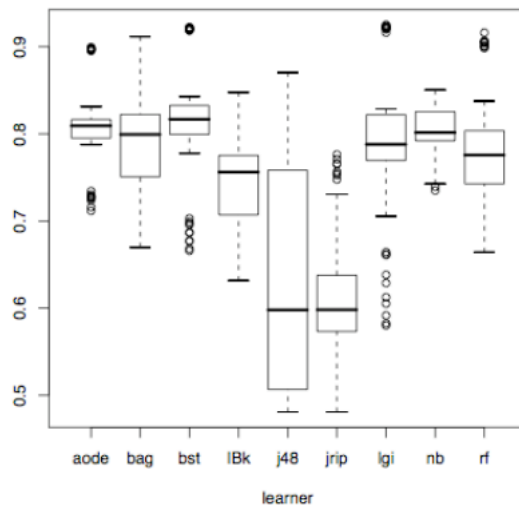
Evaluation is humbling (3)

- Left:

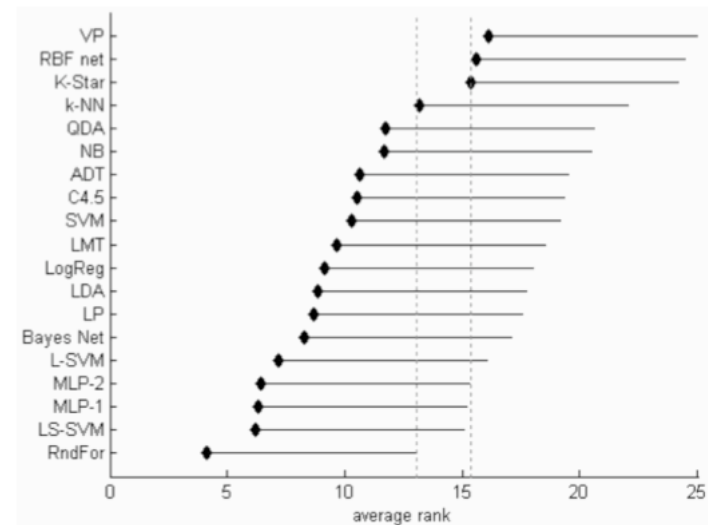
- Y. Jiang, B. Cukic, and T. Menzies. Fault prediction using early lifecycle data. In ISSRE'07, 2007. Available from <http://menzies.us/pdf/07issre.pdf>.

- Right:

- Lessmann, S., Baesens, B., Mues, C., Pietsch, S.: Benchmarking classification models for software defect prediction: a proposed framework and novel findings. IEEE Trans. Softw. Eng. (2008)



6/9 methods are “best”



14/19 methods are “best”

Lesson: most “improvements”, aren’t

No consensus on the “best” evaluation

	1999	2000	2001	2002	2003
Total number of papers	54	152	80	87	118
Relevant papers for our study	19	45	25	31	54
Sampling method [%]					
cross validation, leave-one-out	22	49	44	42	56
random resampling	11	29	44	32	54
separate subset	5	11	0	13	9
Score function [%]					
classification accuracy	74	67	84	84	70
classification accuracy - <i>exclusively</i>	68	60	80	58	67
recall, precision...	21	18	16	25	19
ROC, AUC	0	4	4	13	9
deviations, confidence intervals	32	42	48	42	19
Overall comparison of classifiers [%]	53	44	44	26	45
averages over the data sets	0	4	6	0	10
t-test to compare two algorithms	16	11	4	6	7
pairwise t-test one vs. others	5	11	16	3	7
pairwise t-test each vs. each	16	13	4	6	4
counts of wins/ties/losses	5	4	0	6	9
counts of <i>significant</i> wins/ties/losses	16	4	8	16	6

An overview of the papers accepted to International Conference on Machine Learning in years 1999–2003. The reported percentages (the third line and below) apply to the number of papers relevant for our study.

Janez Demsar: Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research 7: 1-30 (2006)

- No global standard
- Advice:
 1. Study evaluation methods in current state-of-the-art papers
 - Copy them
 2. Avoid t-tests and their simplistic Gaussian assumptions
 3. Don't bother with results that report a (say) 4% improvement
 4. Be prepared to change the evaluation to make the reviewers happy
 5. Favor informative visualizations,
 - Use statistical tests as sanity checks on the conclusions from the visualization

Visualizations need not be elaborate

Rank	Treatment	PERCENTILES								
		0%	50%	100%	10	30	50	70	90	
1	(M 3 K 3)			---	*	81	88	94	100	100
1	(M 3 K 2)			-----	*	76	88	94	100	100
1	(M 3 K 1)			---	*	76	82	94	100	100
1	(M 3 K 0)			---	*	81	88	94	100	100
1	(M 2 K 3)				*	81	82	94	100	100
1	(M 2 K 2)			-----	*	76	88	94	100	100
1	(M 2 K 1)			---	*	76	82	94	100	100
1	(M 1 K 3)			-----	*	76	88	94	100	100
1	(M 1 K 2)			-----	*	76	88	94	100	100
1	(M 1 K 1)			----	*	76	85	94	100	100
1	(M 1 K 0)			-----	*	76	88	94	100	100
1	(M 2 K 0)			----	*	76	85	88	100	100
2	(M 0 K 0)		---		*	41	49	65	100	100
3	(M 0 K 3)		-----		*	35	50	59	100	100
4	(M 0 K 2)		-----		*	38	50	59	100	100
5	(M 0 K 1)		-----		*	35	47	59	100	100

M,K: two magic params inside a NaiveBayes classifier handling low frequency counts

PD measurements in a 10*3 cross-val on IRIS

Rank set by a Mann-Whintey (95%(comparing each row to proceeding rows of the same rank 187