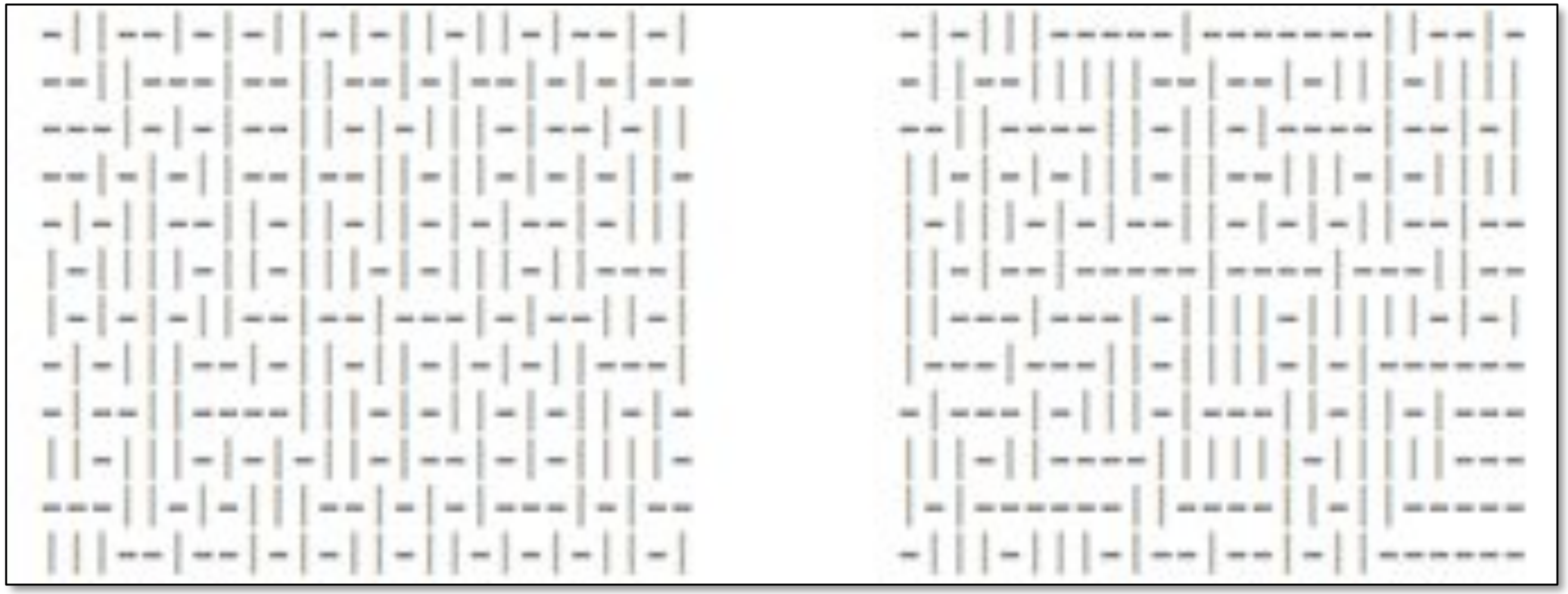
For other view on DM + SE

- 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/asergrp/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 (1)

- 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

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

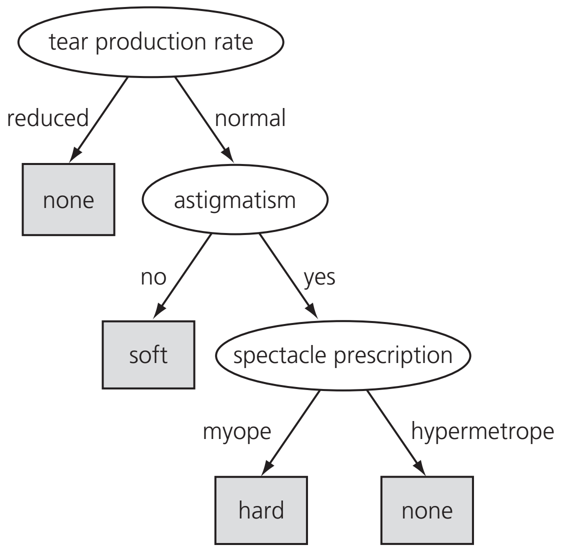


Figure 1.2 Decision tree for the contact lens data.

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

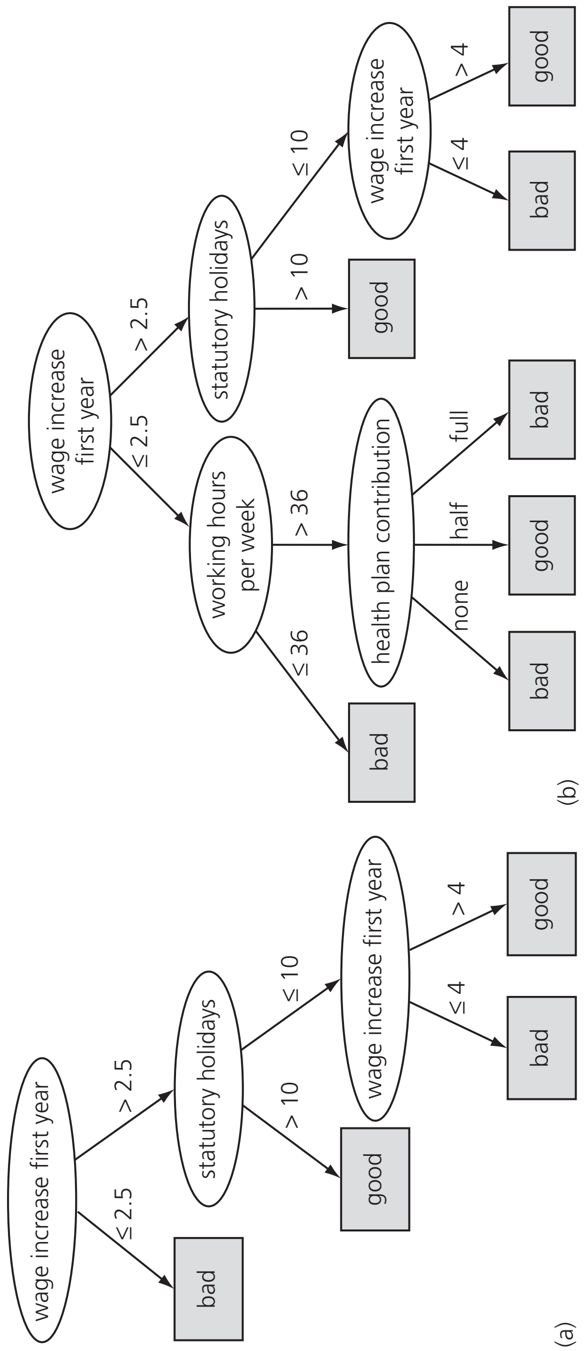


Figure 1.3 Decision trees for the labor negotiations data.

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.*

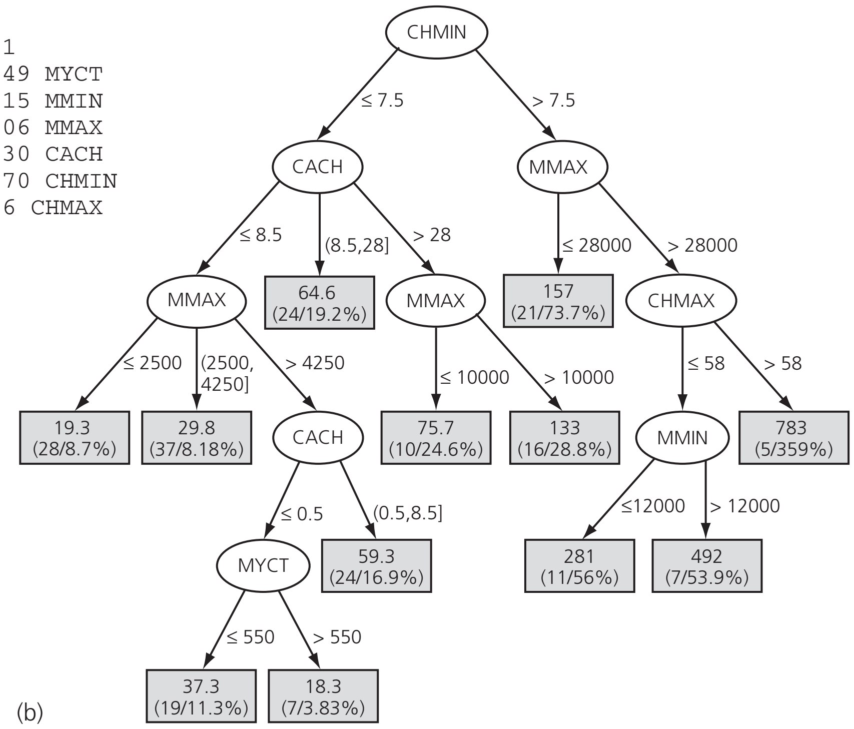
Figure 3.7(a) shows a regression equation for the CPU performance data, and Figure 3.7(b) shows a regression tree. The leaves of the tree are numbers that represent the average outcome for instances that reach the leaf. The tree is much larger and more complex than the regression equation, and if we 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.

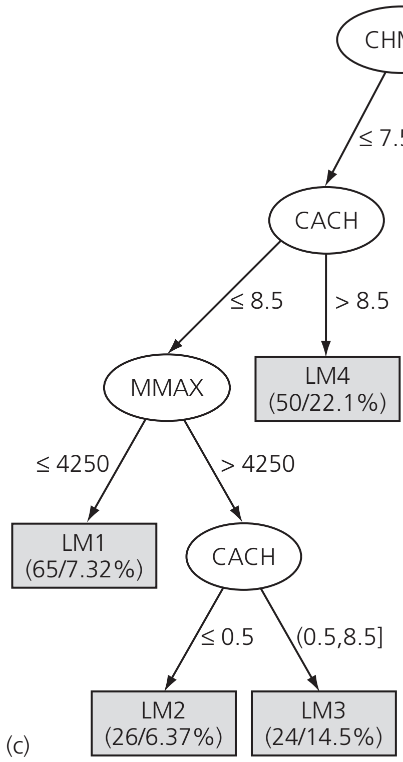
It is possible to combine regression equations with regression trees. Figure 3.7(c) is a tree whose leaves contain linear expressions—that is, regression equations—rather than single predicted values. This is (slightly confusingly) called a *model tree.* Figure 3.7(c) contains the six linear models that 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)





```
LM1 PRP=8.29+0.004 MMAX+2.77 CHMIN
LM2 PRP=20.3+0.004 MMIN-3.99 CHMIN
        +0.946 CHMAX
LM3 PRP=38.1+0.012 MMIN
LM4 PRP=19.5+0.002 MMAX+0.698 CACH
        +0.969 CHMAX
LM5 PRP=285-1.46 MYCT+1.02 CACH
        -9.39 CHMIN
LM6 PRP=-65.8+0.03 MMIN-2.94 CHMIN
        +4.98 CHMAX
```

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

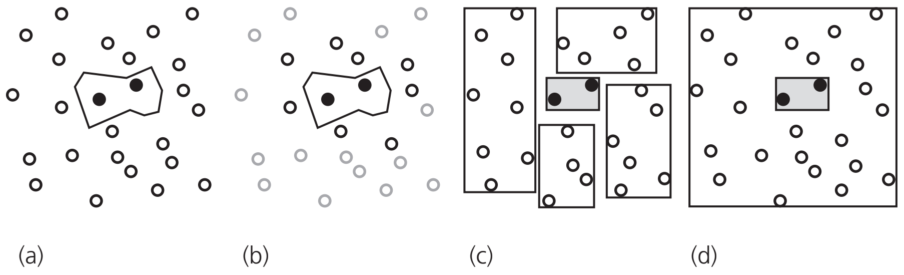
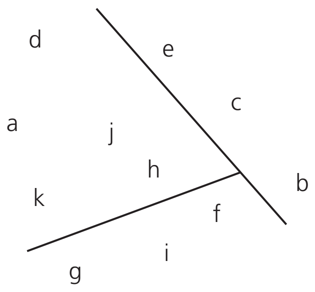


Figure 3.8 Different ways of partitioning the instance space.

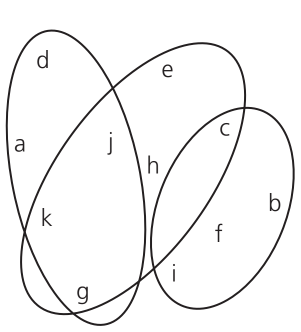
3.9 Clusters

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



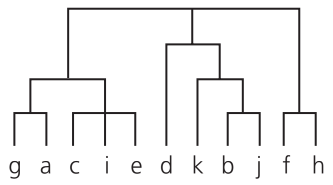
(a)



(b)

	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)

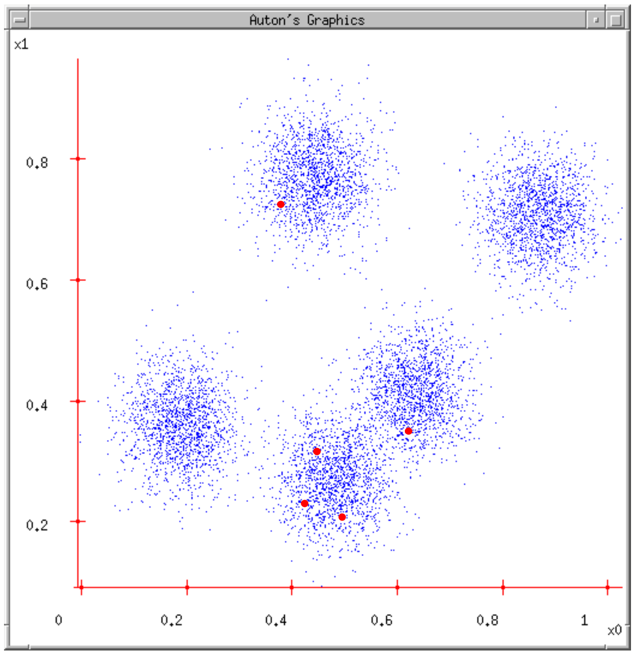


(d)

Figure 3.9 Different ways of representing clusters.

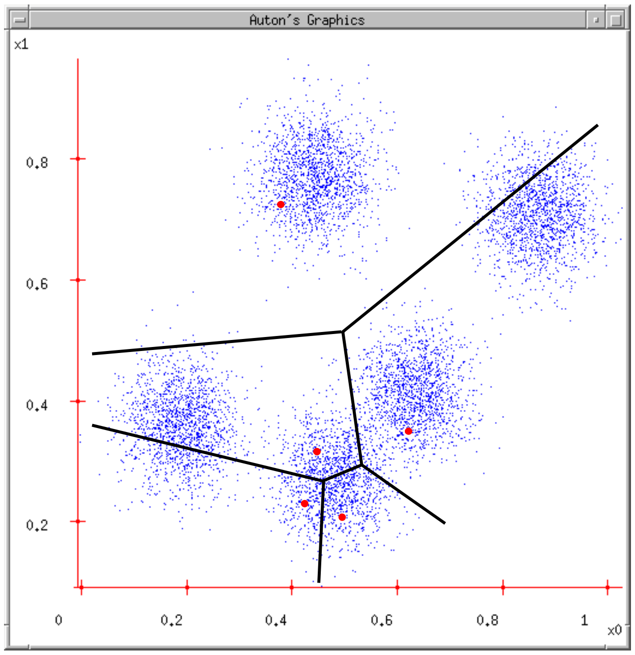
K-means

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



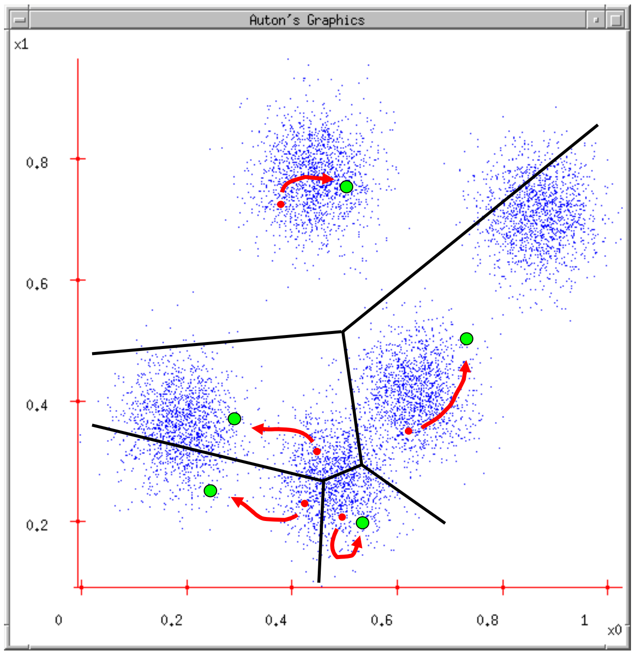
K-means

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)



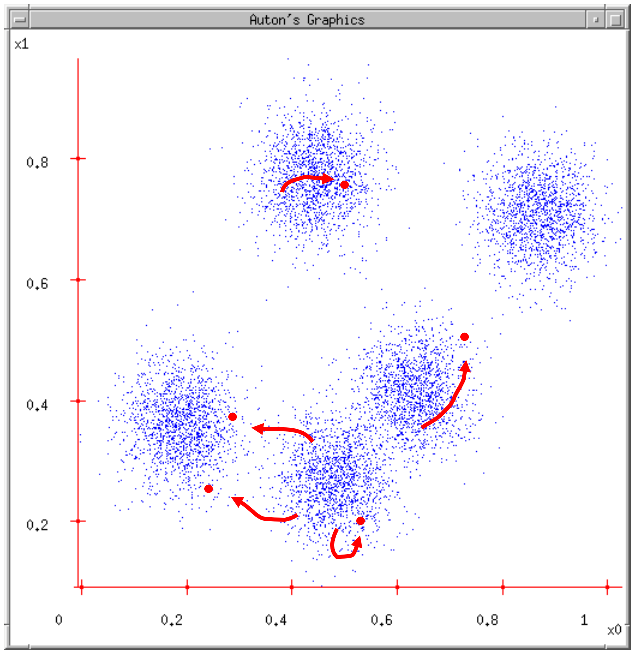
K-means

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



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!



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	2/5	4/14
		overcast → yes	0/4	
		rainy → yes	2/5	
2	temperature	hot → no*	2/4	5/14
		mild → yes	2/6	
		cool → yes	1/4	
3	humidity	high → no	3/7	4/14
		normal → yes	1/7	
4	windy	false → yes	2/8	5/14
		true → no*	3/6	

*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	
	yes	*no*		*yes*	*no*		*yes*	*no*		*yes*	*no*	*yes*	*no*
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:

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

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

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

LectureTrees
Learning Trees

Updated Oct 29, 2009 by menzies.tim

In this lecture, we discuss "iterative dichotomization"

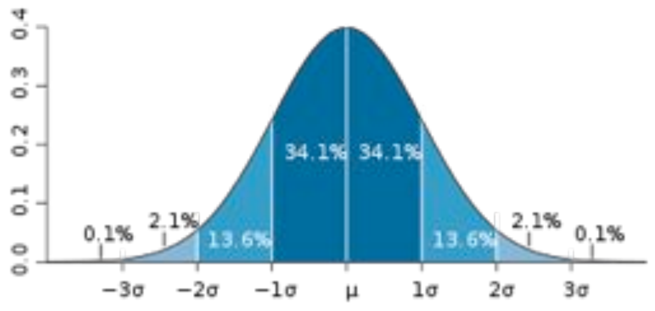
- Split things up
- Recurse on each split

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



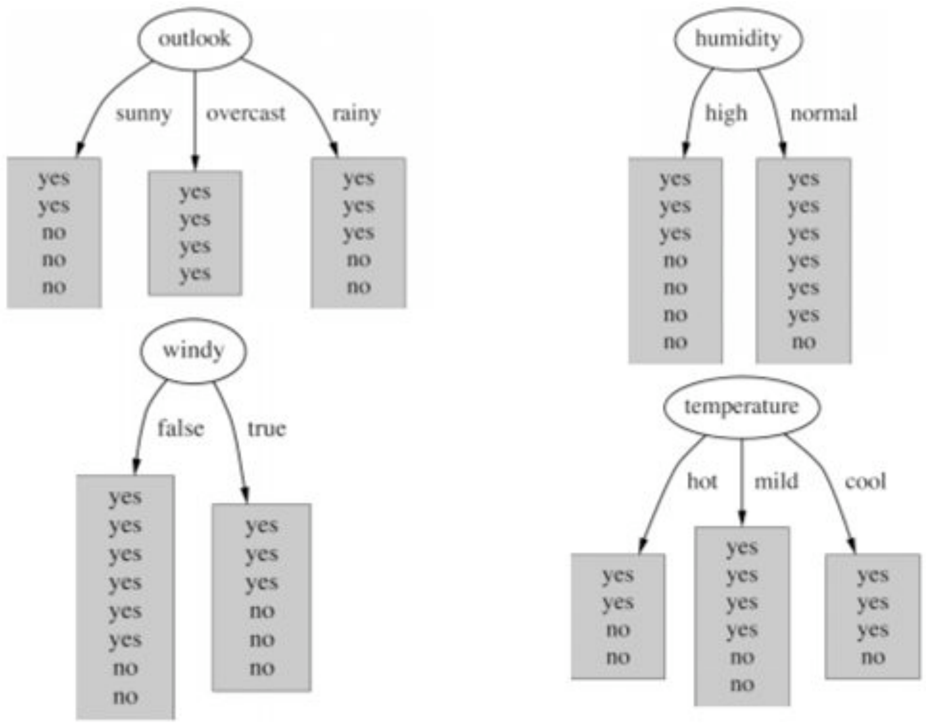
    - condense: report the average of the instances in each bag.

- M5prime: (model trees)
    - measure: standard deviation
    - condense: generate a linear model of the form a+b * x1 +c * x2 + d * x3 +...
- J48: (decision trees)
    - measure: "entropy"

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

- 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

- info((2,3))= entropy(2/5,3/5) = -2/5 * log(2/5) - 3/5 * log(3/5) = 0.971 bits

Outlook = overcast

- info((4,0)) = entropy(1,0) = -1 * log(1) - 0 * log(0) = 0 bits

Outlook = rainy

- info((3,2)) = 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

- 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:
- gain("Outlook") = info(9,5?) - info(2,3?,4,0?,3,2?) = 0.940 - 0.963 = 0.247 bits
- gain("Temperature") = 0.247 bits
- gain("Humidity") = 0.152 bits
- gain("Windy") = 0.048 bits

Problem: Numeric Variables

No problem:

- use cliff learning to split the numerics

LectureNaiveBayes
NaiveBayes Classifiers 101

Updated Oct 27, 2009 by kelcecil

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

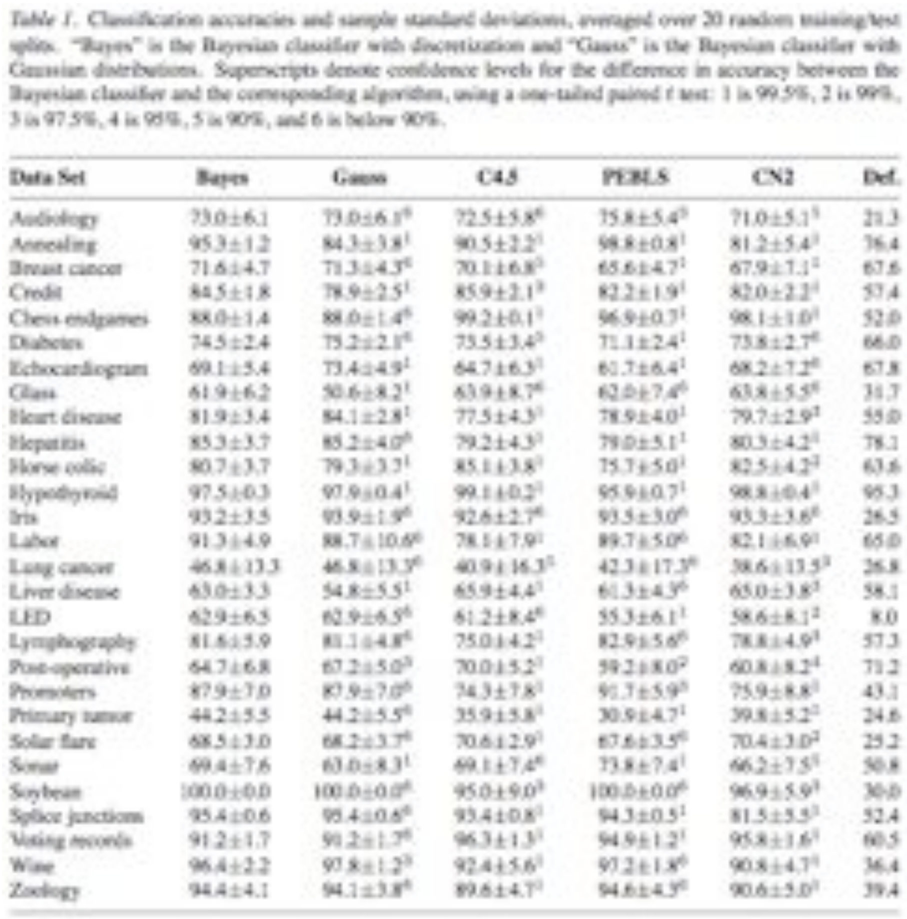


Table 1. Classification accuracies and sample standard deviations, averaged over 20 random training/test splits. "Bayes" is the Bayesian classifier with discretization and "Gauss" is the Bayesian classifier with Gaussian distributions. Superscripts denote confidence levels for the difference in accuracy 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 put 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              Yes   No             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("yes") = 0.0053 / (0.0053 + 0.0206) = 0.205
  - P("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(E | H ) * Pr(H)
      Pr(H | E) =  -------------------
                       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(E1 | H )* Pr(E2 | H ) * ....  *Pr(En | H ) * Pr(H )
      Pr(H | E) = ---------------------------------------------------
                                     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":

```
      Pr( yes | E) = Pr(Outlook     = Sunny | yes) *
                     Pr(Temperature = Cool  | yes) *
                     Pr(Humidity     = High  | yes) * Pr( yes)
                     Pr(Windy       = True  | yes) * Pr(yes) / Pr(E)
                   = (2/9 * 3/9 * 3/9 * 3/9)       * 9/14)   / Pr(E)
```

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("yes") = 0.0238 / (0.0238 + 0.0343) = 41%
- P("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(Humidity = High | yes) = 0
- A posteriori probability will also be zero! Pr(yes | 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,         klass,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=klass}
   }
   return what
}
```

More Complex

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

```
function likelihood(l,         klass,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=klass}
   }
   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            Temperature           Humidity
=====================    =================      =================
           Yes     No            Yes     No             Yes      No
Sunny        2      3             83     85              86      85
Overcast     4      0             70     80              96      90
Rainy        3      2             68     65              80      70
          -----------            ----------           ----------
Sunny      2/9    3/5    mean     73     74.6  mean     79.1    86.2
Overcast   4/9    0/5    std dev   6.2    7.9  std dev  10.2     9.7
Rainy      3/9    2/5

              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
```

Example density value:

- f(temperature=66|yes)= 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("yes") = 0.000036 / (0.000036 + 0. 000136) = 20.9%
    - P("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. Yang03?. Such discretization is good practice since it can dramatically improve the performance of a Naive Bayes classifier (see Dougherty95?.

Simple Extensions

From Naive Bayes to Hyper Pipes

When NaiveBayes? sees a new value, it increments a count.

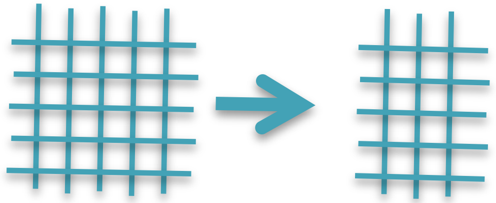
- When HyperPipes sees a value, it just sets the count for that value to "one".

So NaiveBayes? 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:

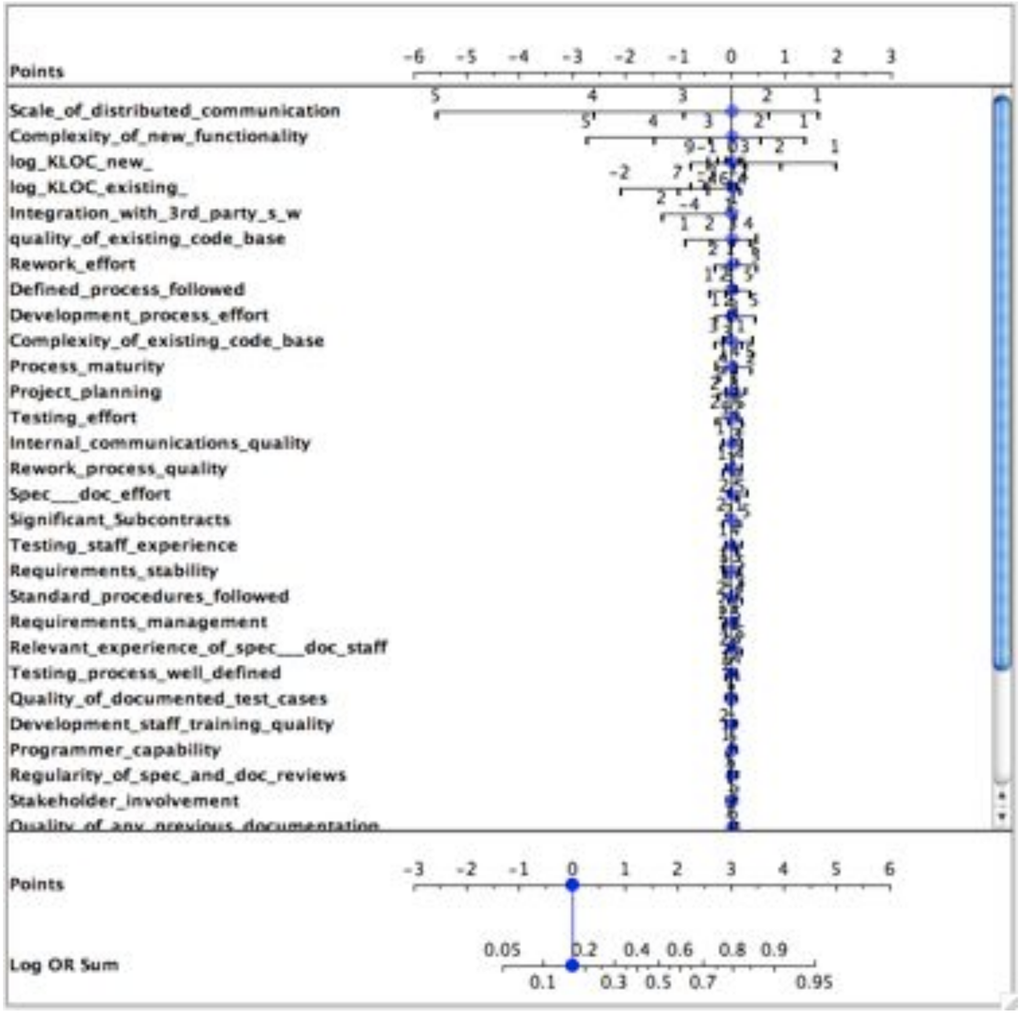
Select columns



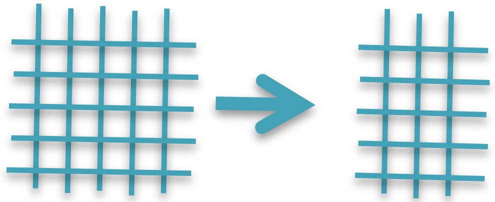
- Occam's Razor - Entia non sunt multiplicanda praeter necessitatem. ("Entities should not be multiplied more than necessary").
  - the fewer features used to explain something, the better
- Log(OR):
  - Discrete every feature. For all pairs of target / other of size C1, C2 count frequency of range N1, N2 in each class
  - Log(odds ratio) = log((N1/C1) / (N2/C2)) > 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: assses 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)

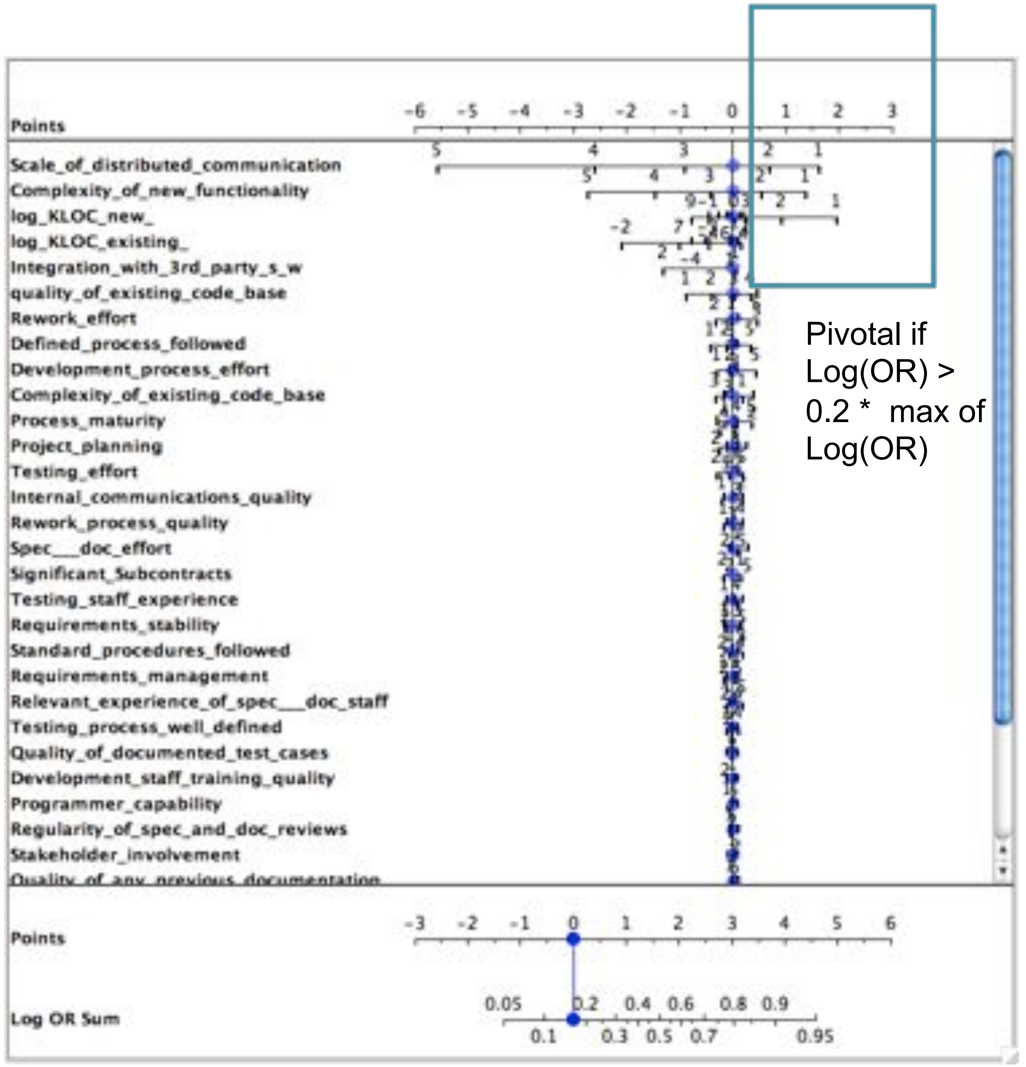
- Data from Norman Fenton's Bayes Net
  - Project Data Incorporating Qualitative Factors for Improved Software Defect Prediction Norman Fenton, Martin Neil, William Marsh, Peter Hearty, Lukasz Radlinski and Paul Krause., PROMISE 2008
- Target class. worse defects
- Only a few features matter
- Only a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



Select columns with log(OR)

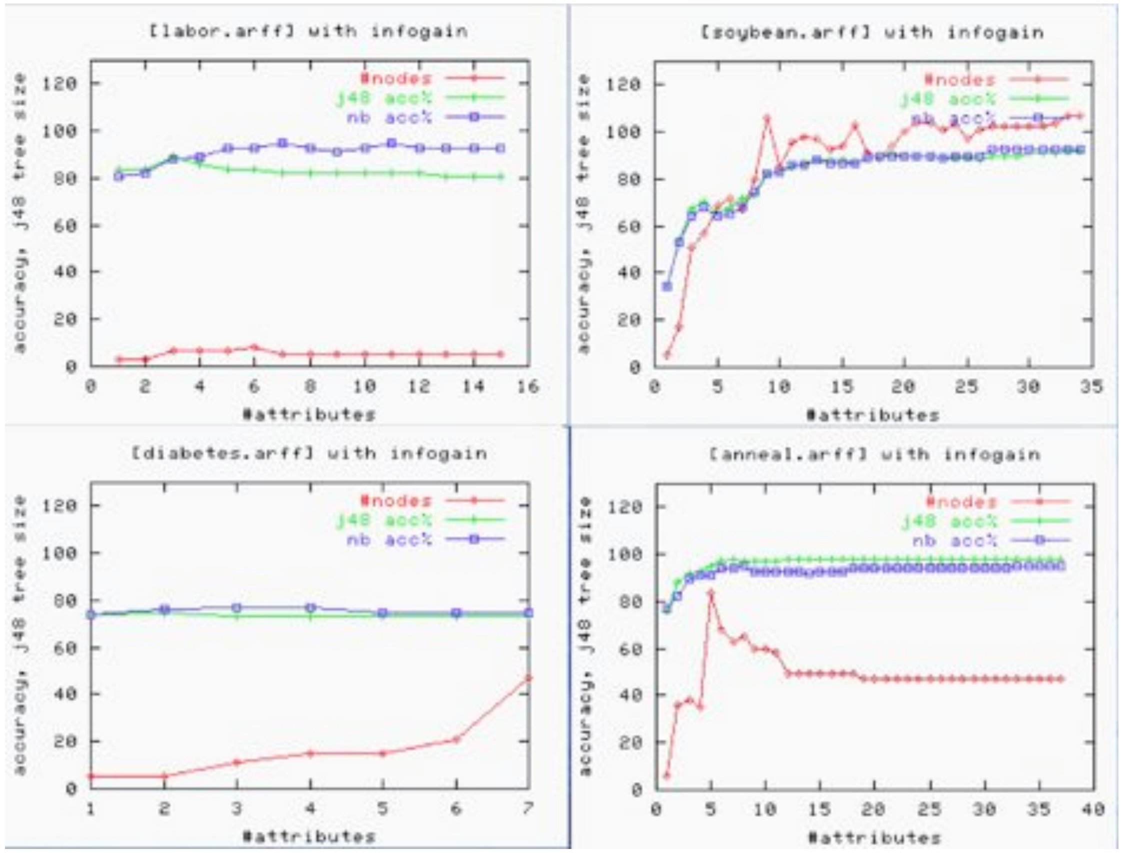


- Data from Norman Fenton's Bayes Net
  - Project Data Incorporating Qualitative Factors for Improved Software Defect Prediction Norman Fenton, Martin Neil, William Marsh, Peter Hearty, Lukasz Radlinski and Paul Krause., PROMISE 2008
- Target class. worse defects
- Only a few features matter
- Only a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



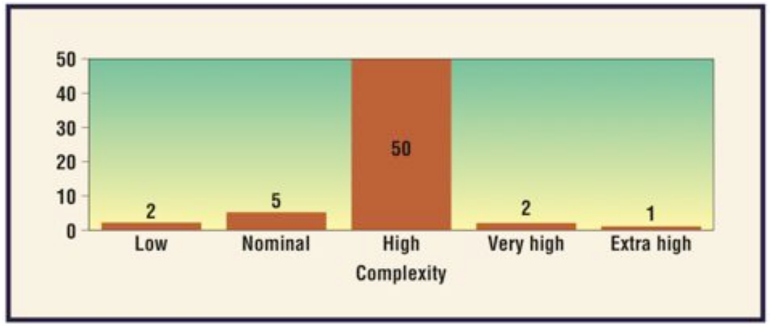
Pivotal if Log(OR) > 0.2 * max of Log(OR)

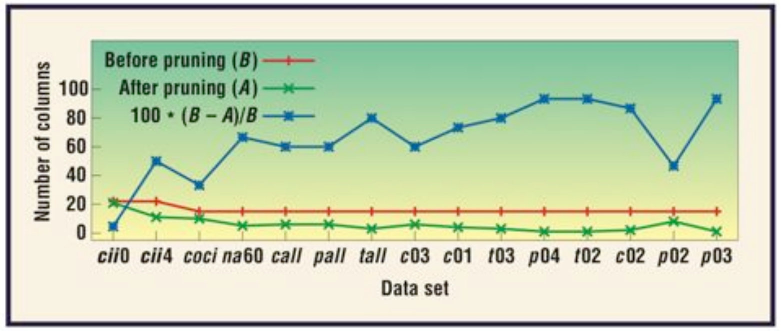
Select columns with InfoGain

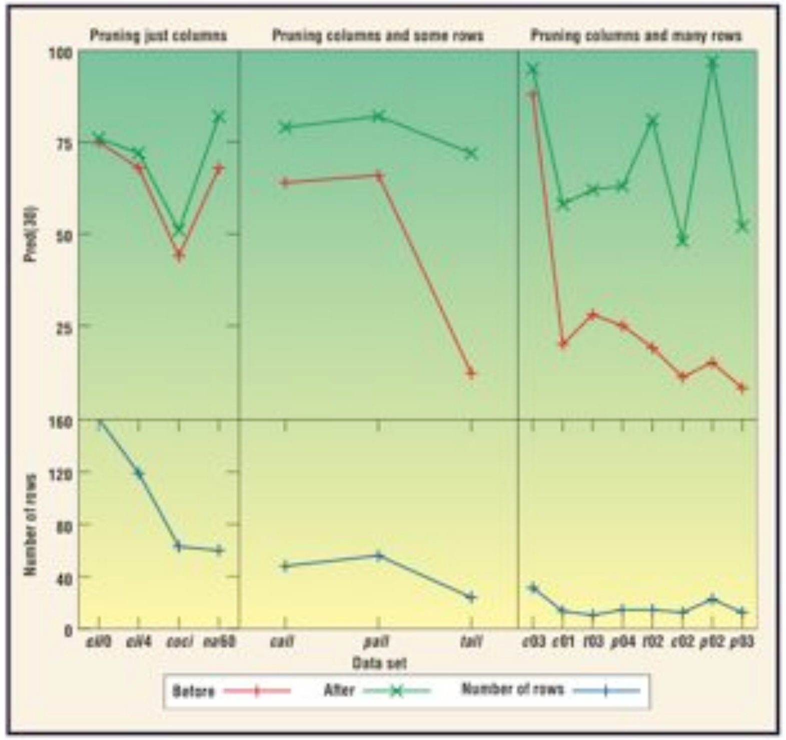


Simpler theories after column selection.
Work just as well as using everything

Select columns with WRAPPER

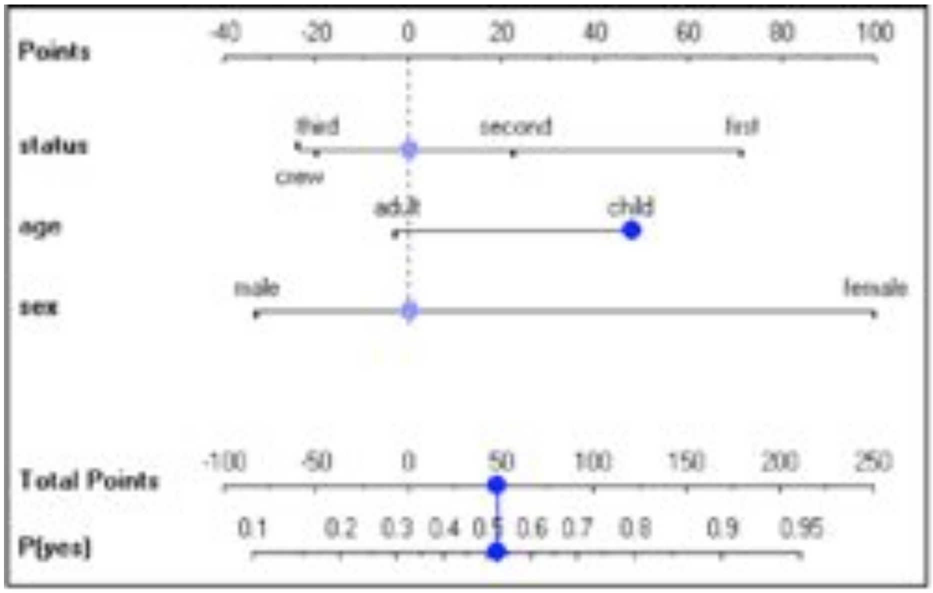






- Finding the Right Data for Software Cost Modeling Chen, Menzies, Port, Boehm, IEEE Software Nov/Dec 2005 the particulars of that data are not what really matter.

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(OR) = log ((N1 / H1) / (N2 / H2))

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 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

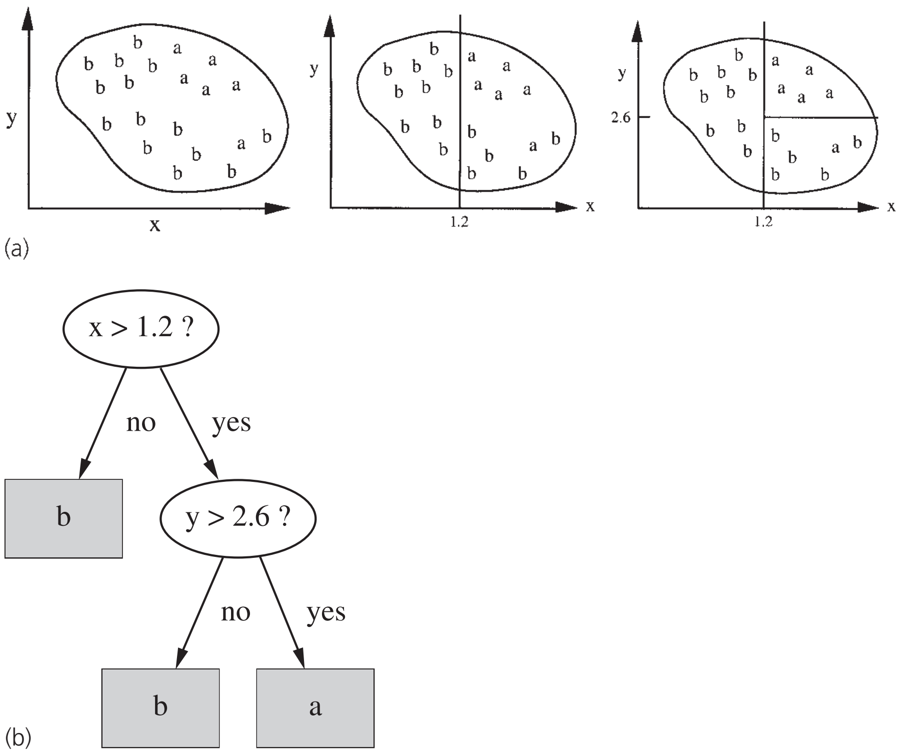


Figure 4.6 Covering algorithm: (a) covering the instances and (b) the decision tree for the same problem.

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 ≤ 1.2 then class = b
If x > 1.2 and y ≤ 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

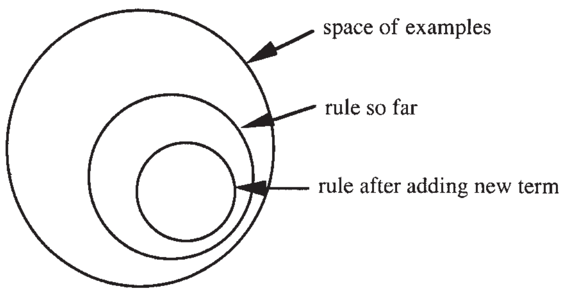


Figure 4.7 The instance space during operation of a covering algorithm.

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.

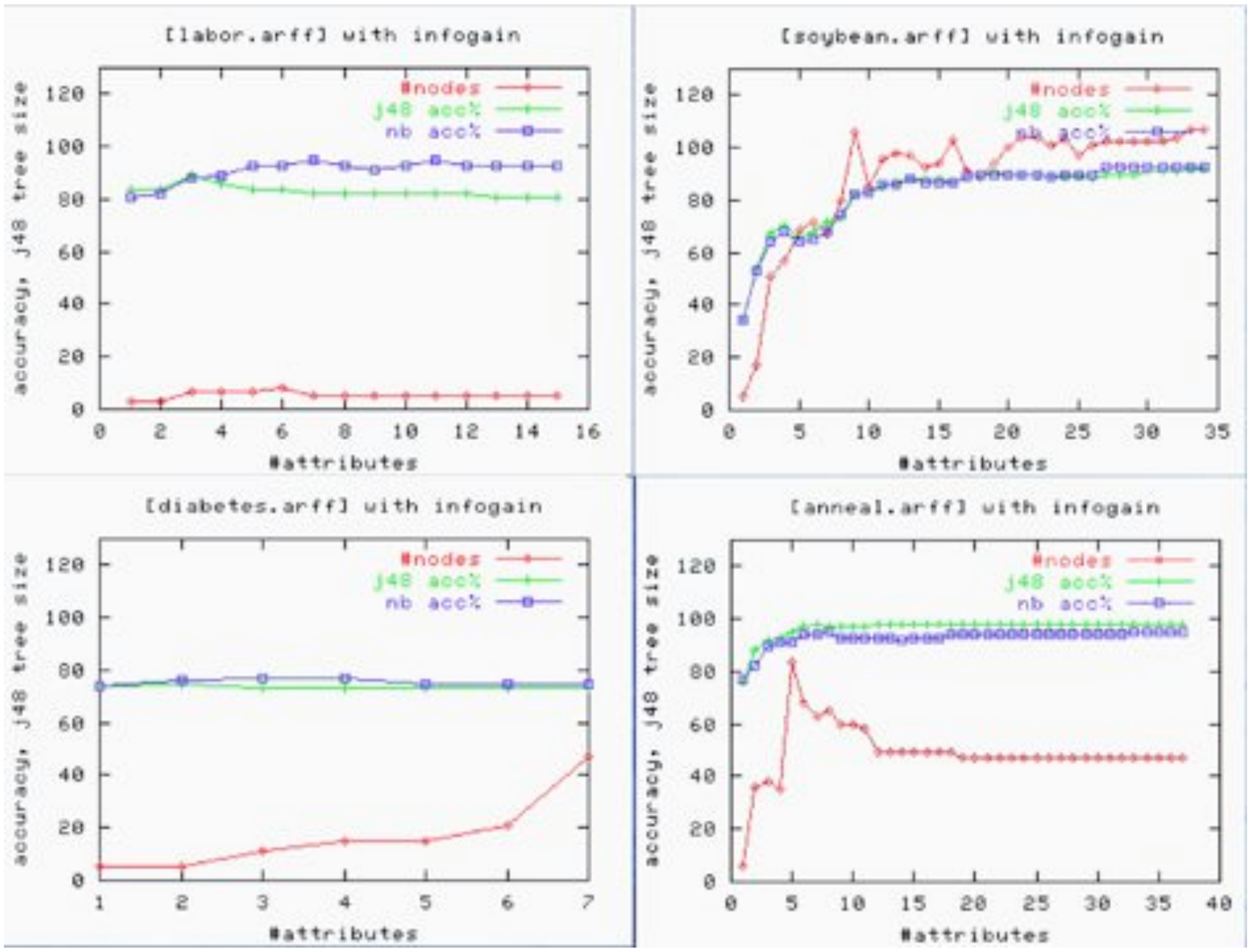
» fss

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 is 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 date and run a *WRAPPER*- a best first search through combinations of attributes. At each step, linear regression was called to asses 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

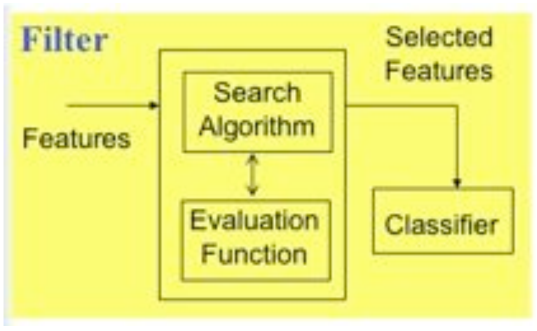
- Back select

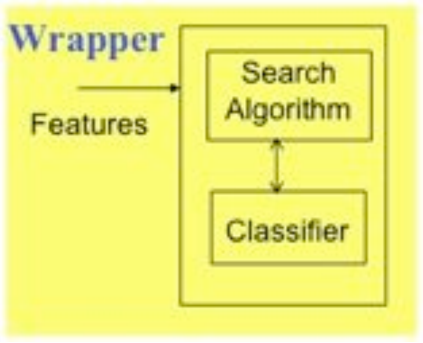
    - as above but start with all attributes and discard, don't add

- 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] - $\sum_{i=1..k}$ diff(f,R, $H_i$) / (n*k)
           + $\sum_{C \neq class(R)}$ $\sum_{i=1..k}$ (
                                  P(C) / ( 1 - P(class(R)))
                                  * diff(f,R, $M_i$(C)) / (n*k)
                                 )
```

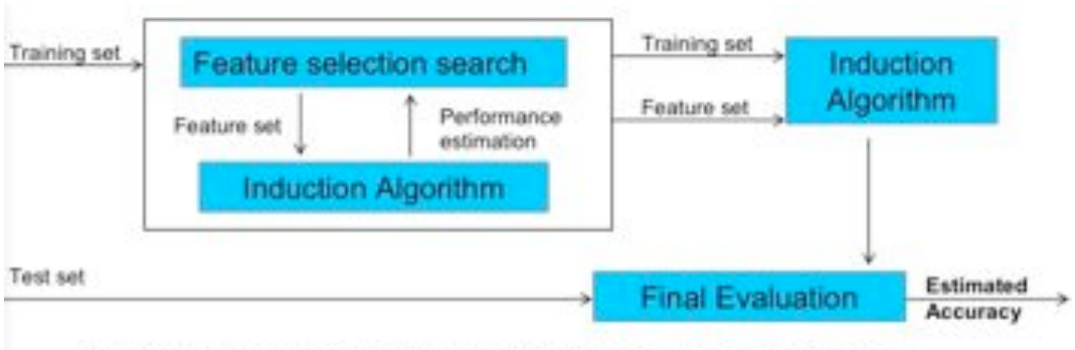
The *P(C) / (1 - P(class(R))* expression is a normalization function that
  - demotes the effect of R from rare classes
  - and rewards the effect of near hits from common classes.

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



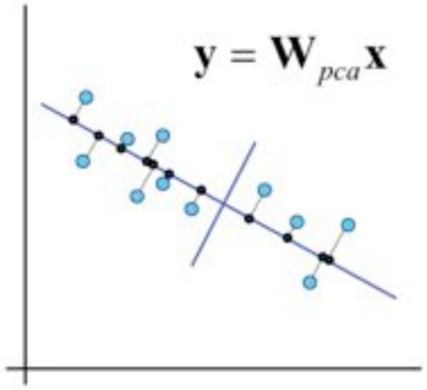
Induction Algorithm is 'wrapped' in the selection mechanism

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...

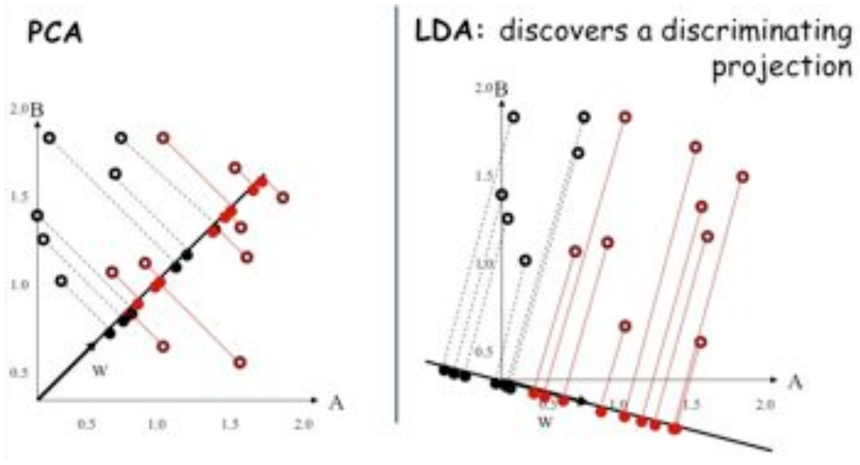
```
if   domain1  <= 0.180
then NoDefects
else if domain1 > 0.180
     then if domain1 <= 0.371 then NoDefects
     else if domain1 > 0.371 then Defects


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 * uniqO p  + 0.234 * uniqOpnd
        + 0.241 * totalOp  + 0.241 * totalOpnd
        + 0.236 * branchCount
```

PCA vs LDA (linear discrminant 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.)

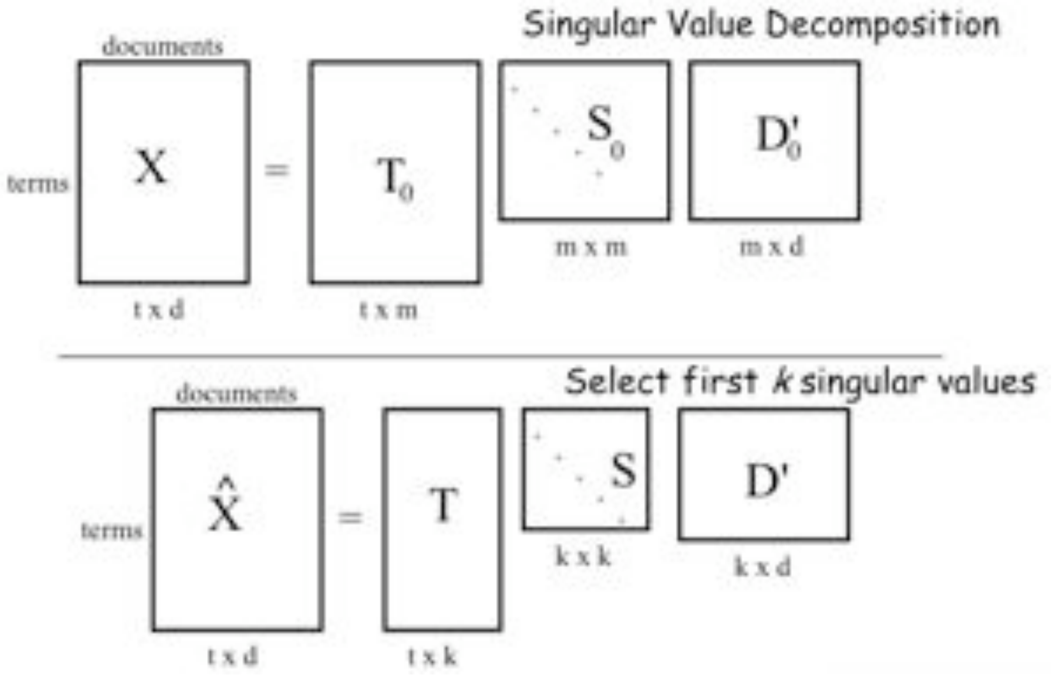


Latent Semantic Indexing

- 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 = To * So * Do'
    - So 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 So



- 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

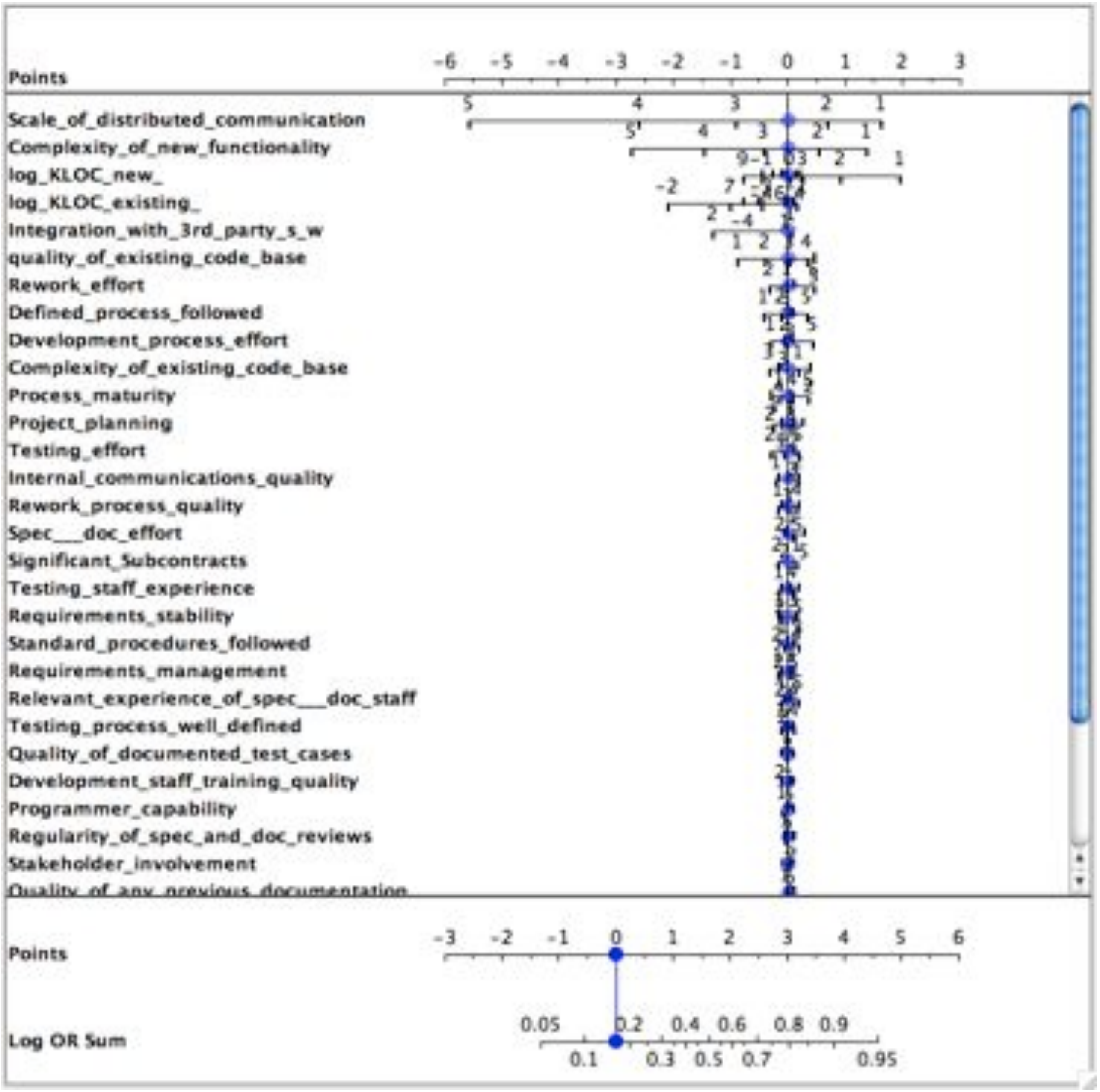
Other Methods

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


Points
-6 -5 -4 -3 -2 -1 0 1 2 3
Scale_of_distributed_communication
Complexity_of_new_functionality
log_KLOC_new_
log_KLOC_existing_
Integration_with_3rd_party_s_w
quality_of_existing_code_base
Rework_effort
Defined_process_followed
Development_process_effort
Complexity_of_existing_code_base
Process_maturity
Project_planning
Testing_effort
Internal_communications_quality
Rework_process_quality
Spec___doc_effort
Significant_Subcontracts
Testing_staff_experience
Requirements_stability
Standard_procedures_followed
Requirements_management
Relevant_experience_of_spec___doc_staff
Testing_process_well_defined
Quality_of_documented_test_cases
Development_staff_training_quality
Programmer_capability
Regularity_of_spec_and_doc_reviews
Stakeholder_involvement
Quality_of_any_previous_documentation
Points
-3 -2 -1 0 1 2 3 4 5 6
Log OR Sum
0.05 0.2 0.4 0.6 0.8 0.9
0.1 0.3 0.5 0.7 0.95

Next Generation “Treatment Learning”

(finding the diamonds in the dust)

Tim Menzies

Lane Dept CS & EE, WVU

`tim@menzies.us`

The strangest thing...

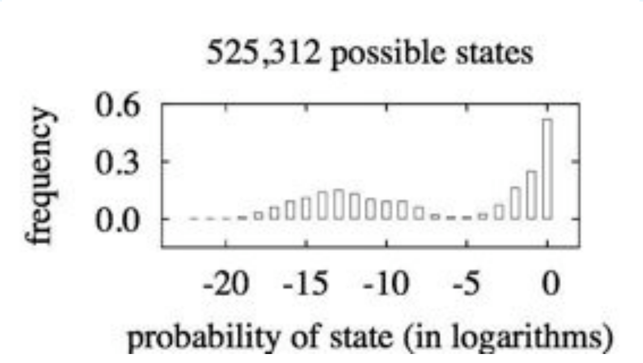
"In any field, find the strangest thing, and explore it" – John Wheeler

- 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 ($\Longleftarrow$ new result)

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.

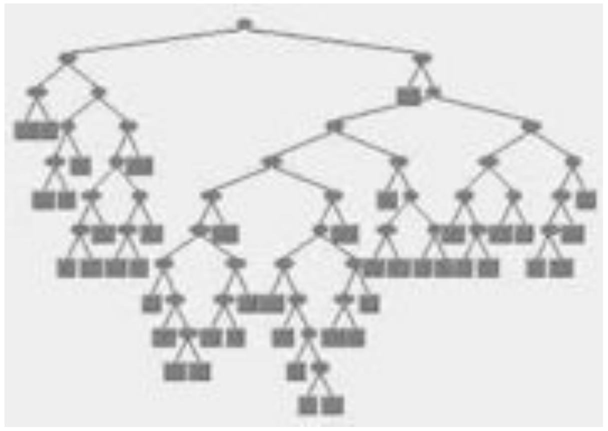
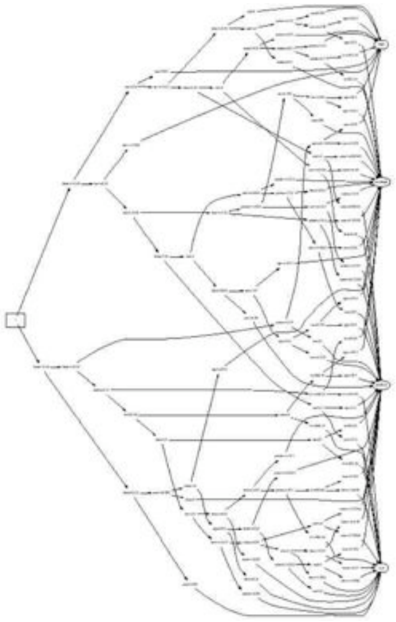
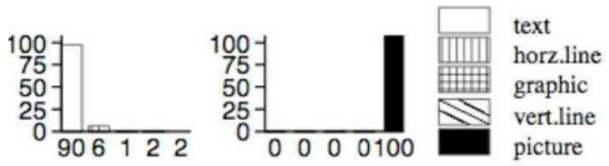
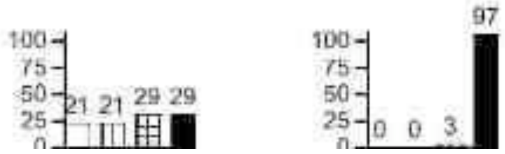


25,000 states in IEEE1394

Exploiting Simplicity

- 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, then..
    - 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	find good housing in Boston
	
	
$34 \leq height < 86 \wedge$ $3.9 \leq mean\_tr < 9.5$	$6.7 \leq RM < 9.8 \wedge$ $12.6 \leq PTRATION < 15.9$

Why Learn Small Theories?

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).

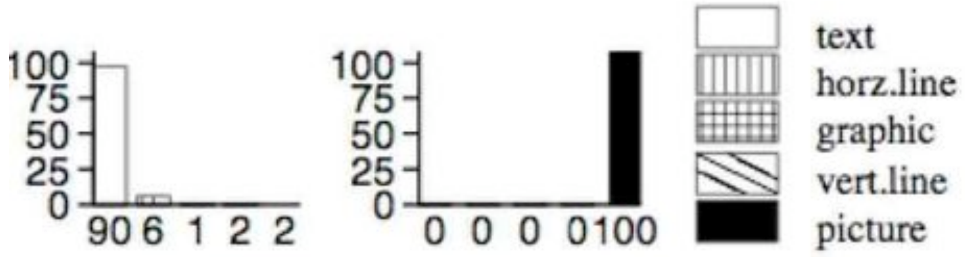
Performance:
The simpler the target concept, the faster the learning.

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?



$34 \leq height < 86 \ \wedge 3.9 \leq mean\_tr < 9.5$

- E: training data with examples of $R_i \rightarrow C$
  - R_i: attribute ranges
  - C: classes with utilities $\{U_1 < U_2 < .. < U_C\}$
  - $F_1\%, F_2\%, ..., F_C\%$: frequencies of C in E
- T treatment of size X: $\{R_1 \wedge R_2 ... \wedge R_X\}$;
  - $T \cap E \rightarrow e \subseteq E$ with frequencies $f_1\%, f_2\%, ... f_C\%$
  - 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

In practice...

The TAR3 Treatment Learner

- Assume clumps and collars
  - Just thrash around some.
- Build treatments $\{R_1 \wedge R_2 \ldots \wedge R_X\}$ 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
```

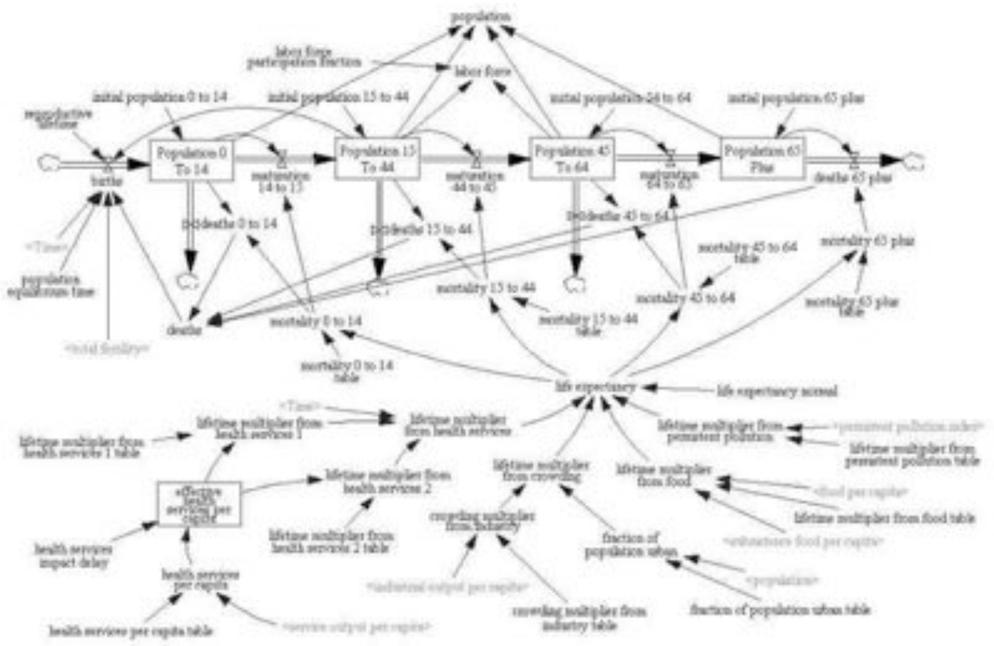
Useful defaults: <SIZE=10, REPEATS=100, ENOUGH=20, LIVES=5>

Saving the World

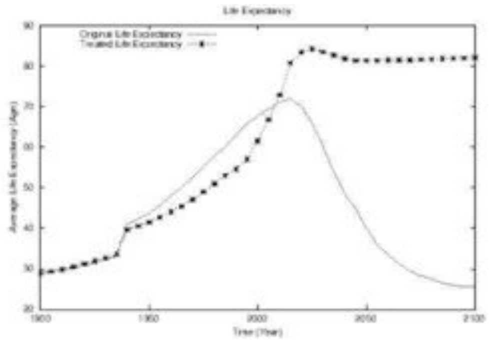
"Limits to Growth" :: Meadows et al. [1972]

A second look at "Limits to Growth": Geletko and Menzies [2003]

Vensim's World-3 (1991): 295 variables



population
labor force participation fraction
labor force
initial population 0 to 14
initial population 15 to 44
initial population 45 to 64
initial population 65 plus
reproductive lifetime
births
Population 0 To 14
maturation 14 to 15
Population 15 To 44
maturation 44 to 45
Population 45 To 64
maturation 64 to 65
Population 65 Plus
deaths 65 plus
deaths 0 to 14
deaths 15 to 44
deaths 45 to 64
<Time>
population equilibrium time
deaths
<initial fertility>
mortality 0 to 14
mortality 0 to 14 table
mortality 15 to 44
mortality 15 to 44 table
mortality 45 to 64
mortality 45 to 64 table
mortality 65 plus
mortality 65 plus table
life expectancy
life expectancy normal
lifetime multiplier from health services 1 table
lifetime multiplier from health services 1
lifetime multiplier from health services
lifetime multiplier from persistent pollution
<persistent pollution index>
lifetime multiplier from persistent pollution table
effective health services per capita
lifetime multiplier from health services 2
lifetime multiplier from health services 2 table
lifetime multiplier from crowding
lifetime multiplier from food
<food per capita>
lifetime multiplier from food table
crowding multiplier from industry
fraction of population urban
<subsistence food per capita>
health services impact delay
health services per capita
<industrial output per capita>
<population>
health services per capita table
<service output per capita>
crowding multiplier from industry table
fraction of population urban table



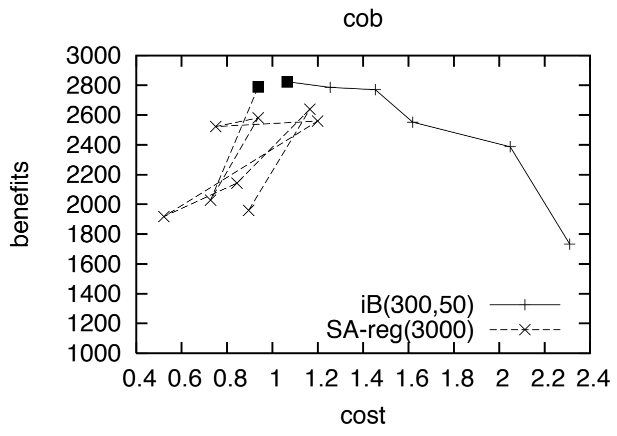
Happily ever after if

- family size $\leq$ 2, menstruation onset $>$ 18, industrial capital output = [3..5).
- This happy ending is *not* mentioned in Meadows et al. [1972].

Compared with More Complete Search

- DDP requirements models from deep-space missions (from JPL).
- Iterative learning: $simulation_i \rightarrow learn \rightarrow constrain \rightarrow simulation_{i+1}$

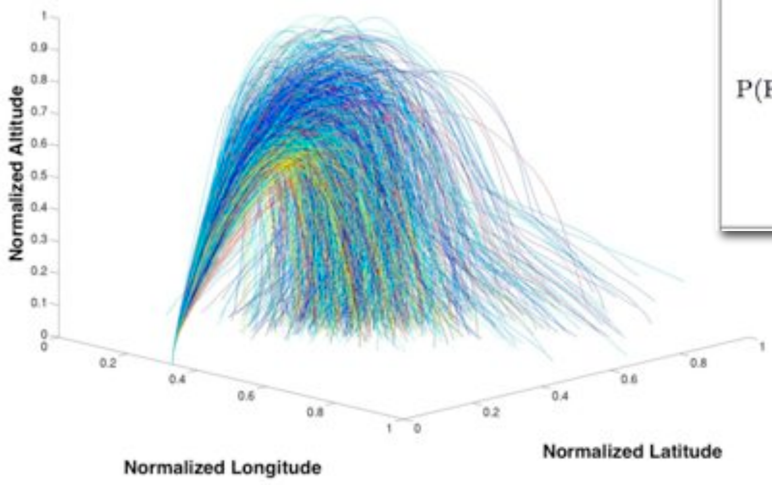
$$SA = \frac{\frac{benefit}{maxBenefit} + (1 - \frac{cost}{maxCost})}{\left(2 * \begin{array}{c}\text{number of}\\ \text{selected mitigations}\end{array}\right) + 1}$$

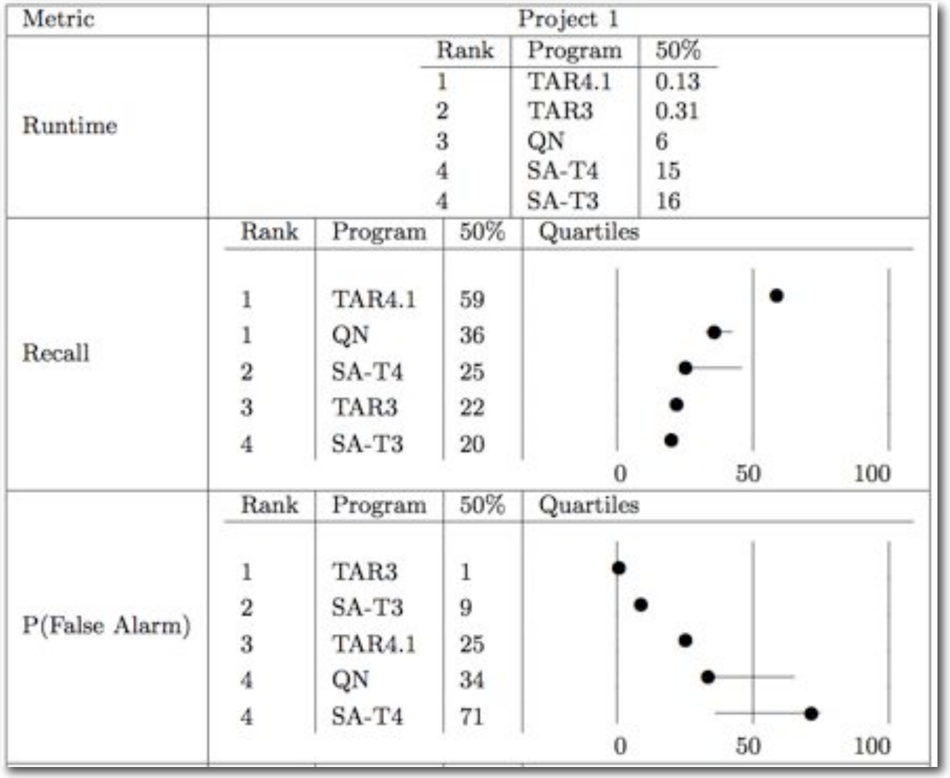


TAR3: 7*300 samples
SA: 9*3000 samples

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	Rank	Program	50%	Quartiles
Runtime (Project 1)	1	TAR4.1	0.13	
	2	TAR3	0.31	
	3	QN	6	
	4	SA-T4	15	
	4	SA-T3	16	
Recall	1	TAR4.1	59	
	1	QN	36	
	2	SA-T4	25	
	3	TAR3	22	
	4	SA-T3	20	
P(False Alarm)	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)

Scaling Up

TAR3 is not a Data Miner

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

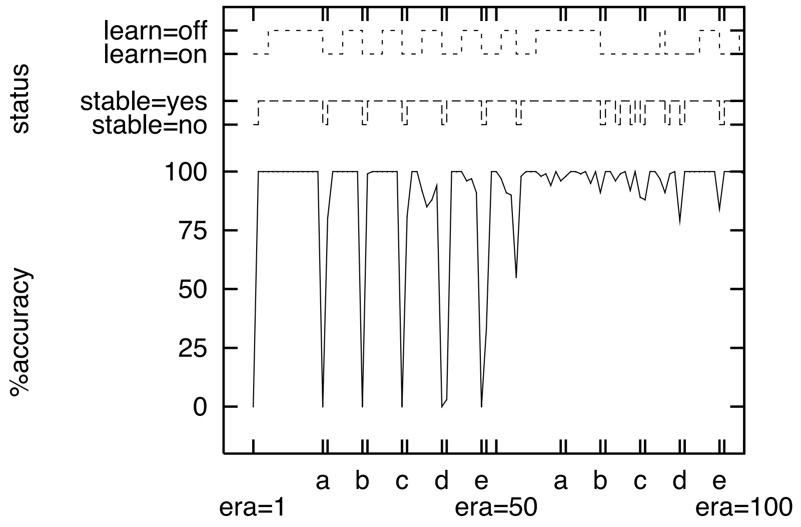
TAR3 is *not* a data miner

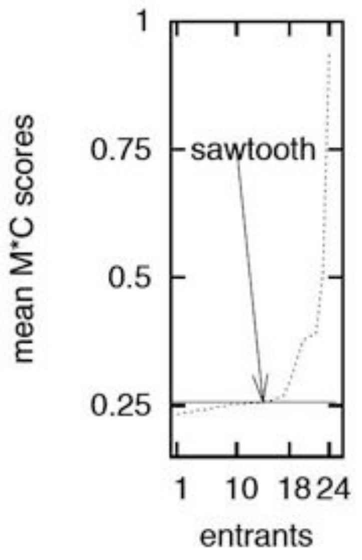
- Stores all examples in RAM
- Requires at three scans
  1. discretization
  2. collect statistics, build treatments
  3. rank generated theories

SAWTOOTH is a data miner

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

$$\overbrace{P(H|E)}^{future=} = \overbrace{\left(\prod_i P(E_i|H)\right)}^{now*} * \overbrace{\frac{P(H)}{P(E)}}^{past}$$

	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

	$P(E_i\|H)$		
P(H)	job	suburb	wealthy?
ford:3=0.5	tinker:1=0.33	NW:1=0.33	y:1=0.33
	tailor:2=0.67	SE:2=0.67	n:2=0.67
bmw:3=0.5	tinker:2=0.67	NW:3=1.00	y:3=1.00
	tailor:1=0.33	SE:0=0.00	n:0=0.00

- E = 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.

CUBE & TAR4

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...};
  - base = worst = {0,0,0,0,...}
- $example_i$ 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 $F(R_j|apex)$.

$$\begin{aligned}
P(apex) &= \textstyle\sum_i up_i / \left(\sum_i up_i + \sum_i down_i\right) \\
P(base) &= \textstyle\sum_i down_i / \left(\sum_i up_i + \sum_i down_i\right) \\
P(R_j|apex) &= F(R_j|apex) / \textstyle\sum_i up_i \\
P(R_j|base) &= F(R_j|base) / \textstyle\sum_i down_i \\
L(apex|R_k \wedge R_l \wedge \ldots) &= \textstyle\prod_x P(R_x|apex) * P(apex) \\
L(base|R_k \wedge R_l \wedge \ldots) &= \textstyle\prod_x P(R_x|base) * P(base)
\end{aligned}$$

TAR4.0: Bayesian treatment learner = find the *smallest* treatment T that *maximizes*:

$$P(apex|T) = \frac{L(apex|T)}{L(apex|T) + L(base|T)}$$

; didn't work: out-performed by TAR3

Why did TAR4.0 fail?

- 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

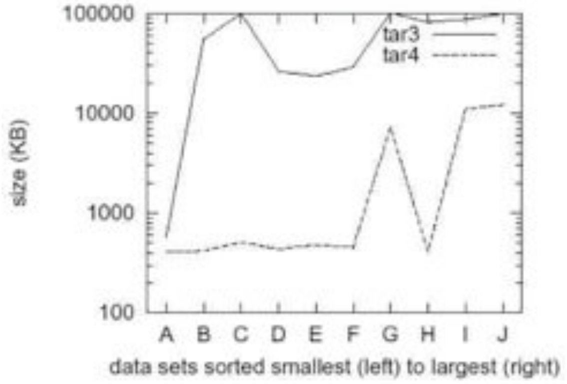
E	P(bmwlE)	P(fordlE)
$job = tailor$ & $suburb = NW$	59.9%	40.1%
$job = tailor$ & $suburb = NW$ & $wealthy = y$	81%	19.0%

- 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.

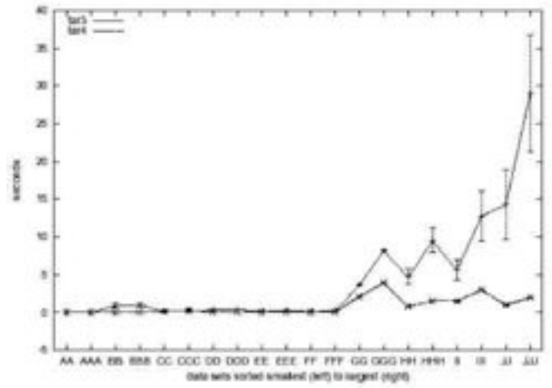
TAR4.1

- 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?

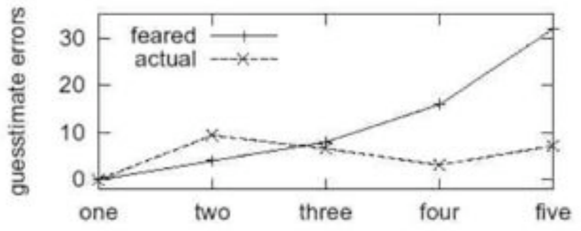
less memory



faster, less variance



lift errors small



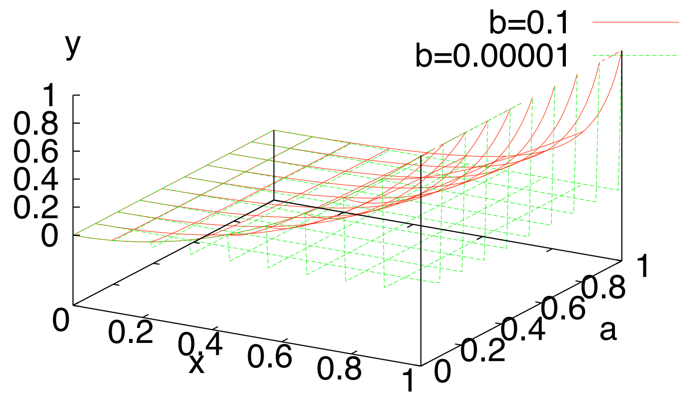
When Won't Dependencies Confuse TAR4?

- $T' = T + t$ where t is an attribute dependent on members of T;
- TAR4.1 *not* confused by t when it ignores treatments that use it.

$$\begin{aligned} a = \quad & L(apex|T') = & \overbrace{P(t|apex)}^{x} * \textstyle\prod_i P(T_i|apex) * P(apex) \\ b = \quad & L(base|T') = & \underbrace{P(t|base)}_{y} * \textstyle\prod_i P(T_i|base) * P(base) \end{aligned}$$

- Then when is $support * probability$ increased by ignoring x and y?

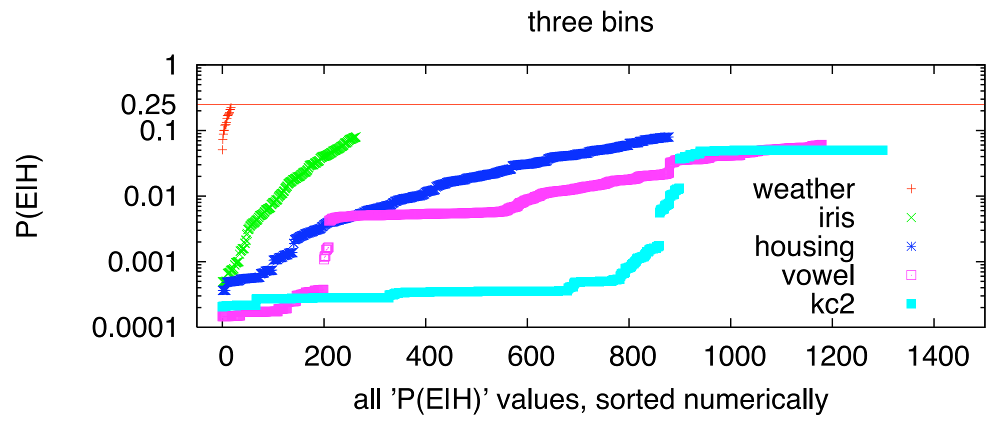
$$\left(\overbrace{\frac{(a/x)^2}{a/x + b/y}}^{\text{ignoring } x \text{ and } y} > \overbrace{\frac{a^2}{a+b}}^{\text{using } x \text{ and } y} \right) \Longrightarrow y > \frac{bx^2}{b + a - xa}$$

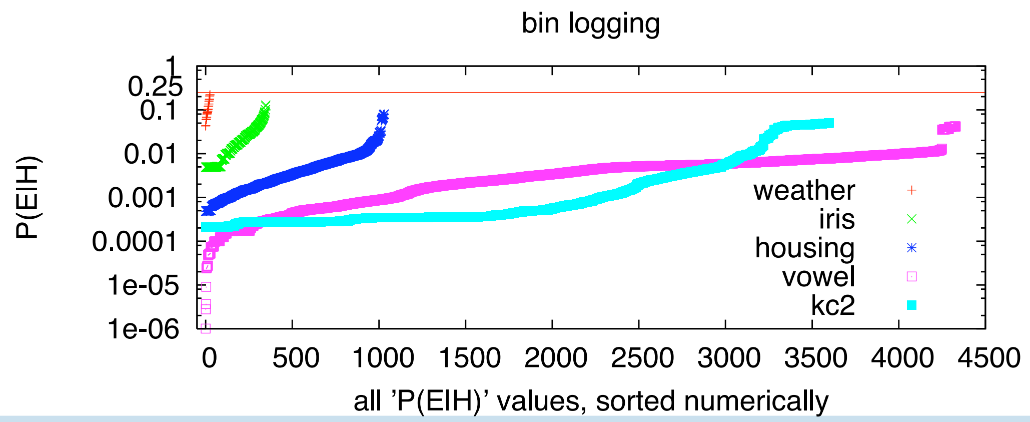


- And for TAR4.0:s pre-condition for no confusion: $\frac{(a/x)}{a/x+b/y} > \frac{a}{a+b}$

Typical Values and Constraints:: $\frac{(a/x)^2}{a/x+b/y}$

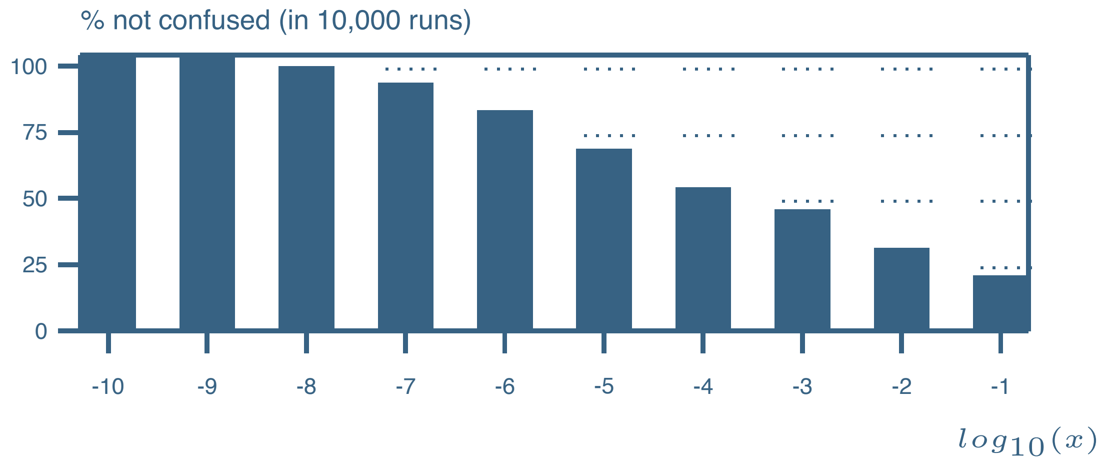
$0 < i \leq 20$	; treatment size
$b < a$	; *apex* is better than *base*
$10^{-10} < x \leq y \leq 0.25$	; see graphs
$0 < a \leq x^i \leq x \leq 0.25$	; *a* combines many *x*-like numbers
$0 < b \leq y^i \leq y \leq 0.25$	; *b* combines many *y*-like numbers





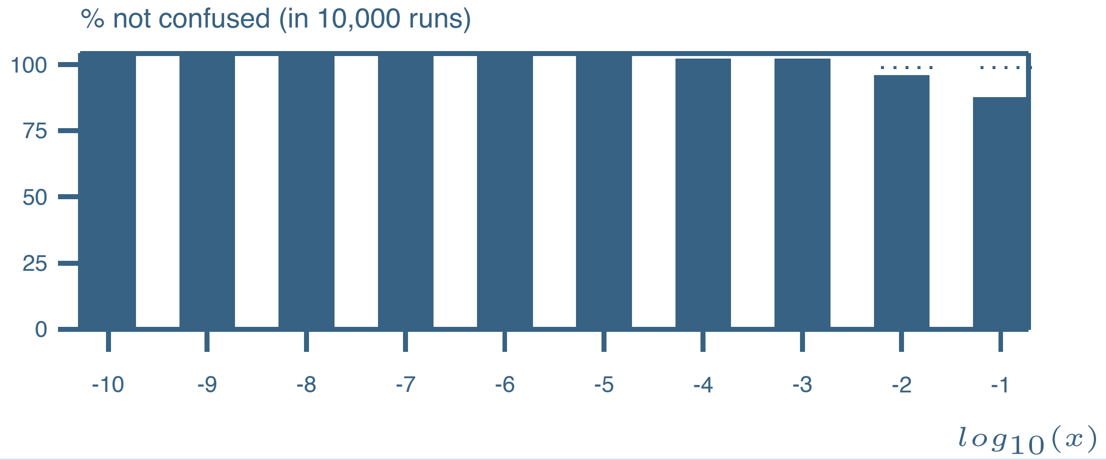
TAR4.1 Works

- 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 $\left(\frac{(a/x)^2}{a/x+b/y} > \frac{a^2}{a+b}\right)$



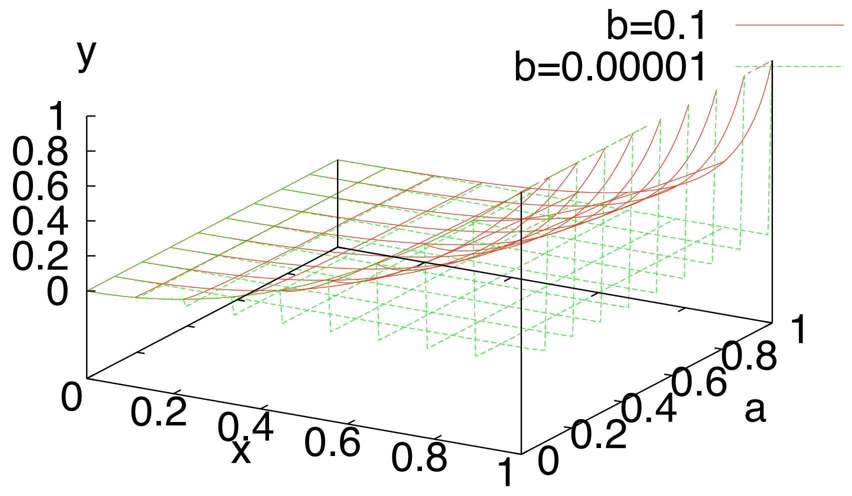
Rarely confused.

So What?

- 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.

Related Work

References

- SAWTOOTH
  - Tim Menzies and Andres Orrego. Incremental discreatization 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, Unviersity of British Columbia, Department of Electrical and Computer Engineering. In preperation
  - 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)

- 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)

- 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)

- 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)

- 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`

And so...

Success Despite Complexity

- Maybe....
  - The world is not as complex as we thing
  - 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

- 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?

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

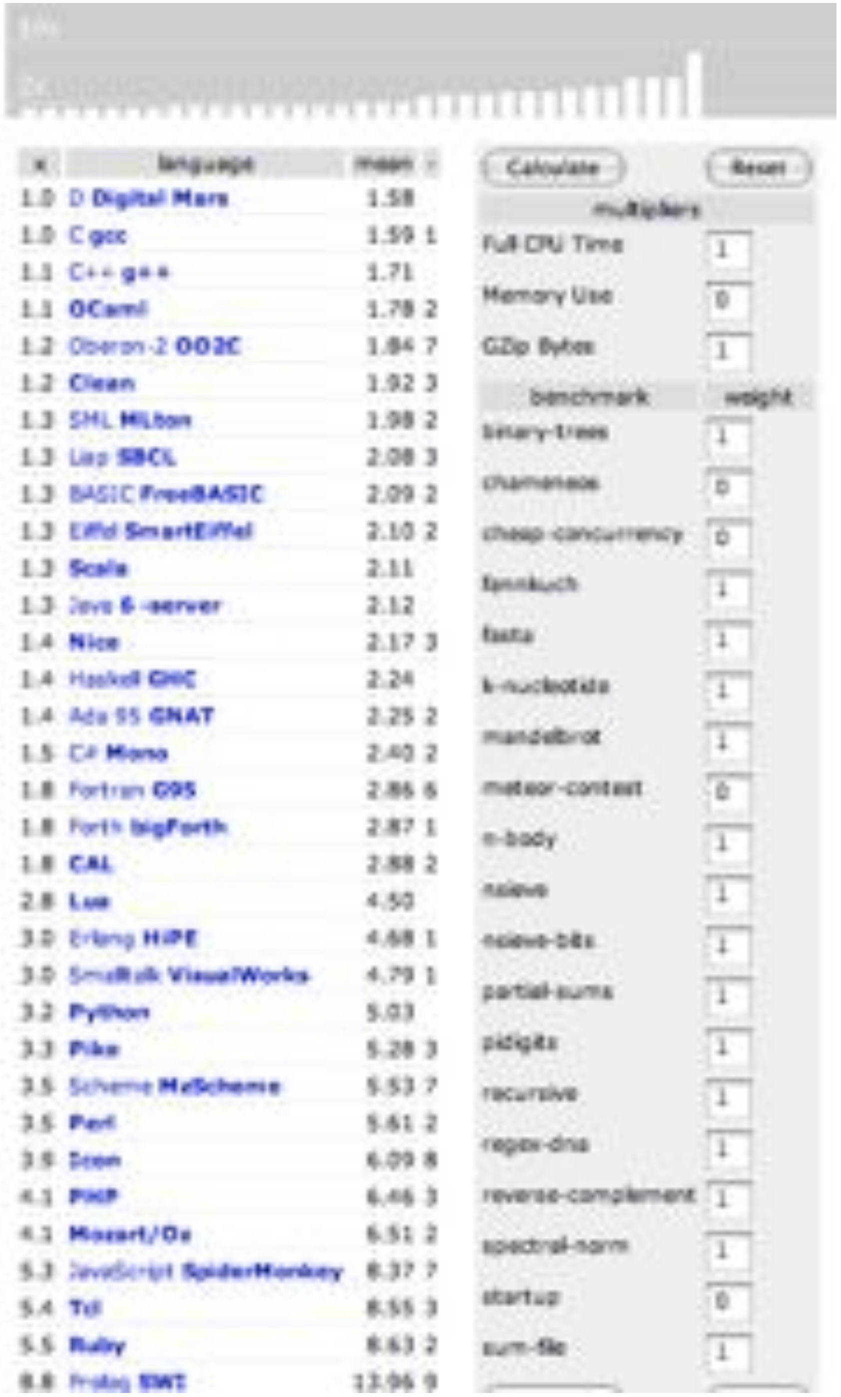
Ruby by Example by Kevin Baird (Paperback)

14.
Tag Score: 1

Learn to Program with Java by John Smiley

15.
Tag Score: 1

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

multipliers	
Full CPU Time	1
Memory Use	0
GZip Bytes	1

benchmark	weight
binary-trees	1
chameneos	0
cheap-concurrency	0
fannkuch	1
fasta	1
k-nucleotide	1
mandelbrot	1
meteor-contest	0
n-body	1
nsieve	1
nsieve-bits	1
partial-sums	1
pidigits	1
recursive	1
regex-dna	1
reverse-complement	1
spectral-norm	1
startup	0
sum-file	1

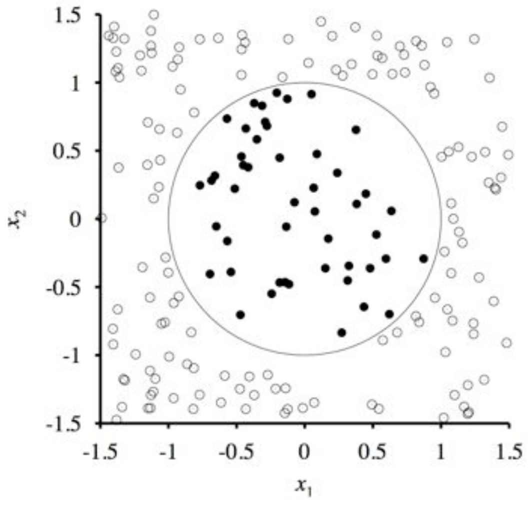
Bias is unavoidable

- 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)

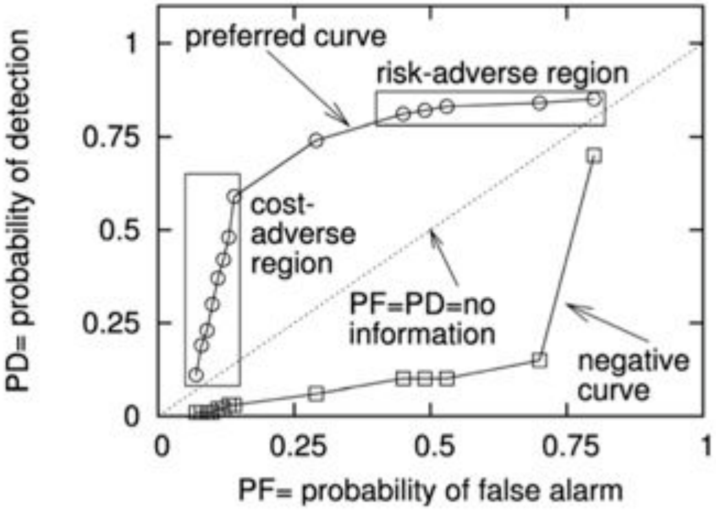
YOU WRITE WHAT YOU'RE TOLD!
THANKS, CORPORATE NEWS!
We Couldn't Control The People Without You

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?

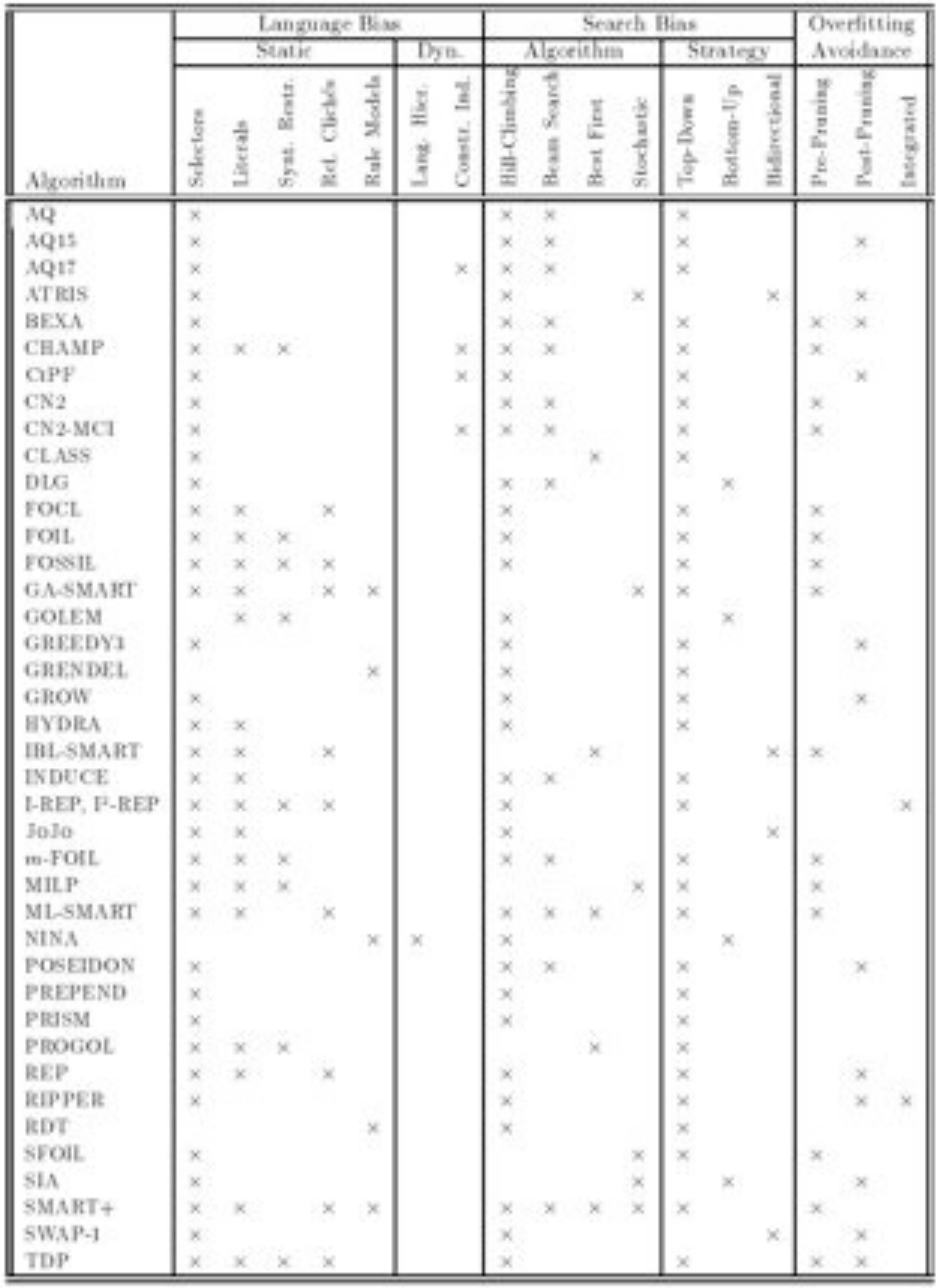


e.g. language bias. Hard to describe a circle if your language Is restricted to "Z op Value"



Different learners use different biases

- 48 learners, 320 combinations of biases
  - 48/320 = 15%
- Separate-and-conquer rule learningJ. FurnkranzArtificial Intelligence Review, 13, pages 3--54, 1999. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.4894



	Language Bias							Search Bias							Overfitting Avoidance		
	Static					Dyn.		Algorithm				Strategy					
Algorithm	Selectors	Literals	Synt. Restr.	Rel. Clichés	Rule Models	Lang. Hier.	Constr. Ind.	Hill-Climbing	Beam Search	Best First	Stochastic	Top-Down	Bottom-Up	Bidirectional	Pre-Pruning	Post-Pruning	Integrated
AQ	×							×	×			×					
AQ15	×							×	×			×				×	
AQ17	×						×	×	×			×					
ATRIS	×							×			×			×		×	
BEXA	×							×	×			×			×	×	
CHAMP	×	×	×				×	×	×			×			×		
CiPF	×						×	×				×				×	
CN2	×							×	×			×			×		
CN2-MCI	×						×	×	×			×			×		
CLASS	×									×		×					
DLG	×							×	×				×				
FOCL	×	×		×				×				×			×		
FOIL	×	×	×					×				×			×		
FOSSIL	×	×	×	×				×				×			×		
GA-SMART	×	×		×	×						×	×			×		
GOLEM		×	×					×					×				
GREEDY3	×							×				×				×	
GRENDEL					×			×				×					
GROW	×							×				×				×	
HYDRA	×	×						×				×					
IBL-SMART	×	×		×						×				×	×		
INDUCE	×	×						×	×			×					
I-REP, I²-REP	×	×	×	×				×				×					×
JoJo	×	×						×						×			
m-FOIL	×	×	×					×	×			×			×		
MILP	×	×	×								×	×			×		
ML-SMART	×	×		×				×	×	×		×			×		
NINA					×	×		×					×				
POSEIDON	×							×	×			×				×	
PREPEND	×							×				×					
PRISM	×							×				×					
PROGOL	×	×	×							×		×					
REP	×	×		×				×				×				×	
RIPPER	×							×				×				×	×
RDT					×			×				×					
SFOIL	×										×	×			×		
SIA	×										×		×			×	
SMART+	×	×		×	×			×	×	×	×	×			×		
SWAP-1	×							×						×		×	
TDP	×	×	×	×				×				×			×	×	

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

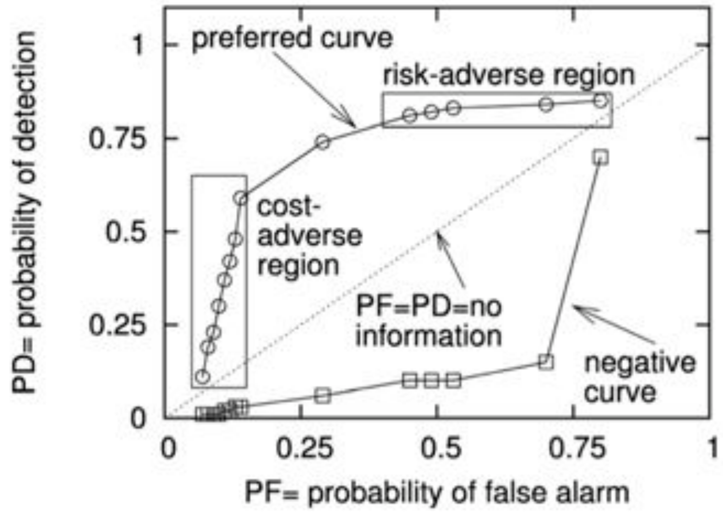
- When reporting a conclusion, report the biases that generated it.
- Make it a first class modeling construct
- Example #1: "W"
  - Recall the sampling bias of "W"
  - Different biases (the query "q") lead to different conclusions
  - Case-Based Reasoning vs Parametric Models Software Quality Optimization, 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?

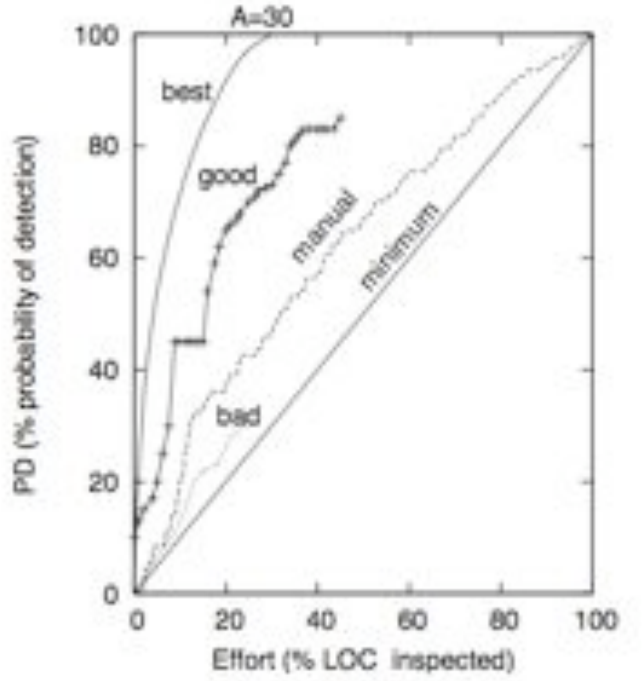


Evaluation Bias #2 : AUC(Pd,effort)

- 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 ≤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 ratoos 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

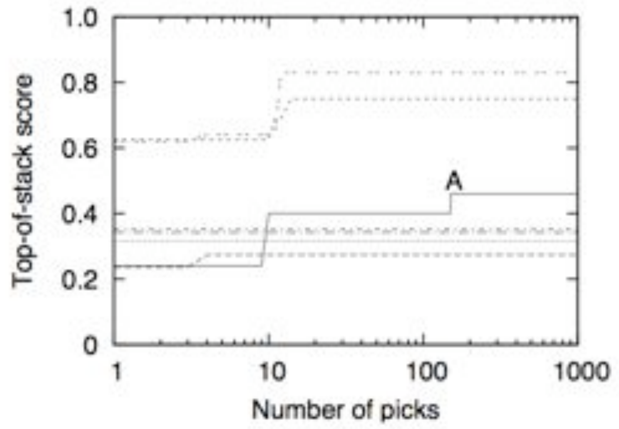
- All learners have an search bias S and an evaluation bias E . e.g. C4.5:
  - S = infogain
  - E = pd, pf, accuracy, etc
- Note: usually, not(S = E)
- Question: What if we make S = E ?
  - Answer: “WHICH”

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

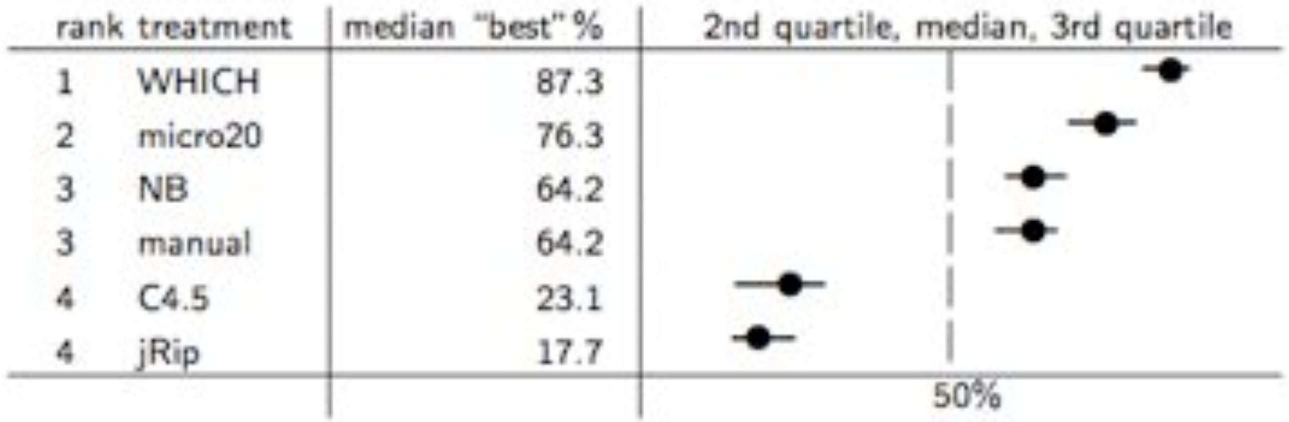
- Note: no S and E is customizable
- But when to stop? (Use 200 picks)



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(effort, pd)
- Micro20: training on 20 defective + 20 non-defective



rank	treatment	median "best" %	2nd quartile, median, 3rd quartile
1	WHICH	87.3	
2	micro20	76.3	
3	NB	64.2	
3	manual	64.2	
4	C4.5	23.1	
4	jRip	17.7	

- 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