Ourmine: An Open Source Data Mining Toolkit

Adam Nelson

Masters Defense
Committee: Tim Menzies, Ph.D; Tim McGraw, Ph.D; Frances VanScoy, Ph.D
West Virginia University, LCSEE

June 13, 2010

Outline

Data Sharing

Since 2006, the PROMISE [1] community has addressed data sharing. This is important because

- researchers can use standardized data
- the data is freely available
- experimentation is encouraged

Experiment Sharing

But in 2009, Gay et al. [2] explored a step beyond this. After *data* sharing comes *experiment* sharing.

Experiment sharing

- allows researchers to confirm prior results
- lets authors publish entire experiments alongside results
- increases understandability of experiments
- gives way to modifications of experiments

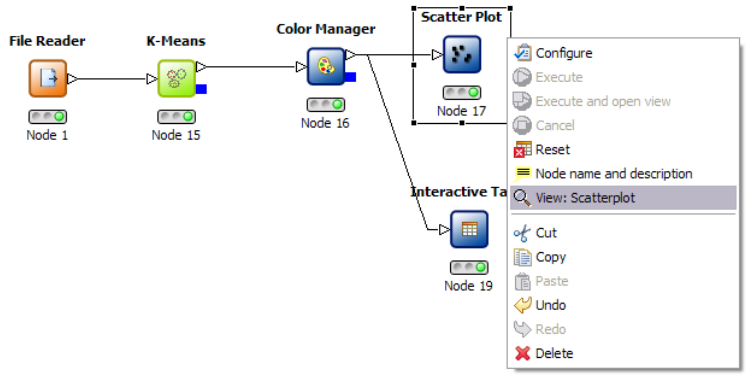
How?

Repeatable experiments in publications require a textual environment capable of not only running data mining algorithms, but also performing analyses and report generation.

To demonstrate Ourmine for this use:

Outline

KNIME



The KNIME [3](Konstanz Information Miner) is a visual-based modular data exploration environment. It incorporates

- over 100 processing nodes for various tasks (data preprocessing, mining and analysis)
- personalized workflows via the *workflow editor*
- editable nodes within each workflow

RapidMiner

RapidMiner [4] is a popular data mining system available worldwide.

- Provides a graphical interface as well as a custom scripting environment
- Uses workflows to build streams of execution
- Allows data to be imported from databases

Important!

Only commercially available editions allow integration into closed-source software

Weka

Weka [5](the Waikato Environment for Knowledge Analysis) is an extremely popular, open source toolset.

- Written in Java. Any modifications to source requires knowledge of Java and rebuilding.
- Offers:
  - *Knowledge Flow*: construction of experiments through interconnected nodes
  - *Explorer*: allows quick exection of data mining algorithms, data visualization, etc.
  - *Experimenter*: builds experiments using a standard GUI (drop-down boxes, text fields, etc.)
  - *CLI*: Weka's command-line interface. Gives standard interaction but through text commands.

Others

Other important open source toolkits include:

- Orange [6] - data mining through visual programming, or Python scripting
- ADaM [7](Algorithm Development and Mining System) - hundreds of components included as executables and Python modules
- Gnome Data Mining Tools [8] - requires Gnome and Python. Operates using command-driven GUIs.
- Rattle [9] (the R Analytical Tool To Learn Easily) - attempts to teach "R" through the use of its data mining GUIs

Textual vs. Visual Appraoches

According to Menzies [10], typical VPLs seen in many standard toolkits are

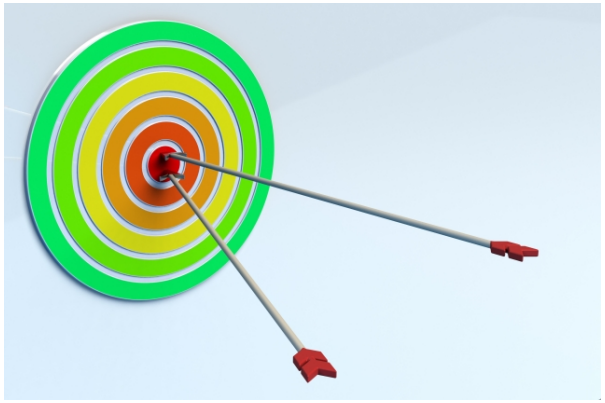
- adequate for beginning operations
- ...but *discourage* further experimentation because
  - developers spend too much time supporting the environment - i.e., never actually data mining

Rithoff et al. [11] found that

- Real-world applications are more intricate than running one algorithm
- These applications require intricate *combinations* of data miners, preprocessors, report generators
- Widely used tools (e.g. WEKA) do not support rapid generations of these combinations

Outline

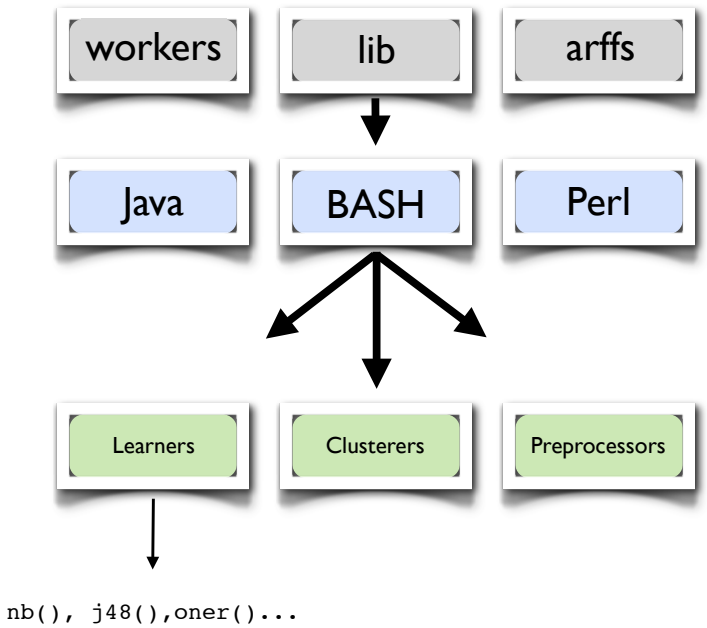
The Goal



To develop a data mining toolkit that promotes experiment sharing *in addition* to

- modularity
- self-containment
- brevity of code
- availability at no cost

Modularity



A modular environment makes extending and modifying code easy.

Self-containment

Required modules should be included with the environment upon download:

- data sets
- learners
- experiment utilities (e.g., cross-validation, evaluation)

Brevity of Code

Publishable code should be short and sweet; large Java or C++ files are much more difficult to inspect and learn from.

Example

```
1 clean(){
2    local docdir=$1
3    local out=$2

4   for file in $docdir/*; do
5      cat $file | tokes | caps | stops $Lists/stops.txt > tmp
6      stems tmp >> $out
7      rm tmp
8   done
9 }
```

Availability at No Cost

Open source environments

- advocate modification to the source code for customization/improvements
- reduces "risk factor" of trying the new software unlike other tools (e.g. Matlab)
- beneficial changes can be shared among all users of the environment

Outline

What is Ourmine?

Ourmine is a data mining scripting environment developed at WVU.

- Designed to assist graduate data mining students
- In 2009 was mostly rebuilt to be *modular*
- Has tools written in BASH, GAWK, JAVA, PERL
- Uses shell scripting as the "glue" between command-line APIs
- Comes with standard data sets, machine learning algorithms, evaluation functions, etc.
- Available for free from a publicly available repository

Why a Scripting Environment?

Shell scripting has many advantages:

- allows publishable experiments to be constructed
- requires limited resources
- executes on any UNIX-based OS

Note

Tool-specific operations can be limiting.

Important Functions

- `MakeTrainAndTest` - builds training and testing sets separately
- `abcd` - evaluates classification performance (*accuracy*, *pd*, *pf*, *precision*, *balance*)
- `winLossTie` - allows statisical rankings of measures
- `quartile` - provides ASCII/numeric representations of quartile charts
- `show` - prints function code
- `funs` - lists available functions
- `gotwant` - extracts predicted and actual classes

Installing and Running Ourmine

Ourmine is an open source toolkit licensed under GPL 3.0. It can be downloaded and installed from `http://code.google.com/p/ourmine`.

Once there, follow the instructions to get started using Ourmine.

Outline

Experiments Using Ourmine

To prove Ourmine as a tool usable in both industry and research, experiments were conducted using it:

- Commissioning a Learner
- Cross-Company Defect Prediction using Relevancy Filtering
- Component vs. Whole-based Defect Prediction
- Analyzing the Scalability of Clustering Text Documents

Commissioning a Learner

In a recent journal paper under review [12], reviewers requested proof of Ourmine's use in industry.

Problem

Many factors can lead to better results in software defect prediction:

- Learners?
- Discretize the data?
- Data size?

Commissioning a Learner

Problem

Choosing these is important

- Some treatments perform better on certain data sets
- As software is released in stages, it's important to know how *early* learners can be applied

Purpose

To demonstrate, for practicioners, how learners/discretized data/data sizes can be selected for use at a local site.

Commissioning a Learner

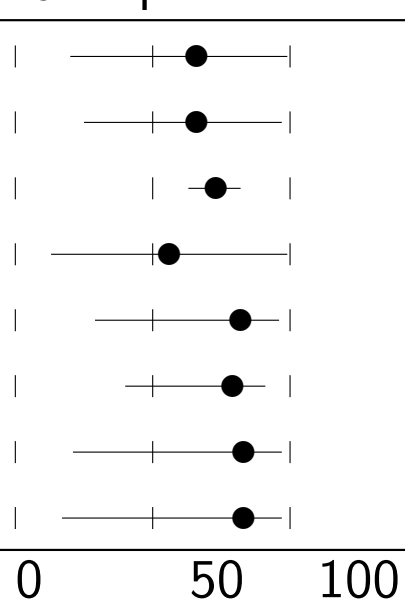
Setup I

Using seven PROMISE defect data sets (CM1, KC1, KC2, KC3, MC2, MW1, PC1), determine the best combinations of learners/discretizers to use.

Evaluate using *pd* and *pf* in a 10X10-way cross-validation.

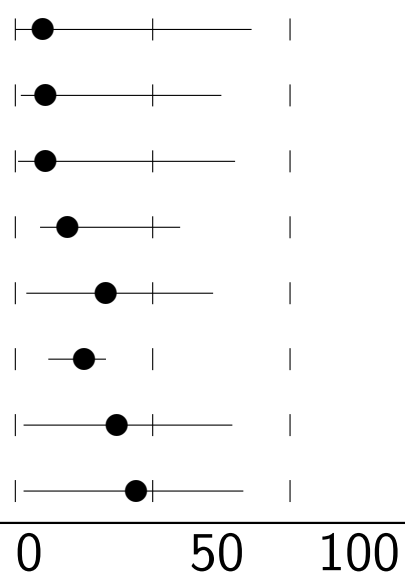
Results 1a

Rank	Treatment	pd percentiles 25%	50%	75%	2nd quartile median, 3rd quartile
1	J48 + discretization	20	66	99	
1	ADTree + discretization	25	66	97	
2	NB + discretization	63	73	82	
2	One-R + discretization	13	56	99	
3	J48	29	82	96	
3	NB	40	79	91	
4	ADTree	21	83	97	
4	One-R	17	83	97	



Results 1b

Rank	Treatment	pf percentiles 25%	50%	75%	2nd quartile median, 3rd quartile
1	One-R + discretization	0	10	86	
1	ADTree + discretization	2	11	75	
1	J48 + discretization	1	11	80	
2	NB	9	19	60	
2	J48	4	33	72	
3	NB + discretization	12	25	33	
3	ADTree	3	37	79	
4	One-R	3	44	83	



Commissioning a Learner

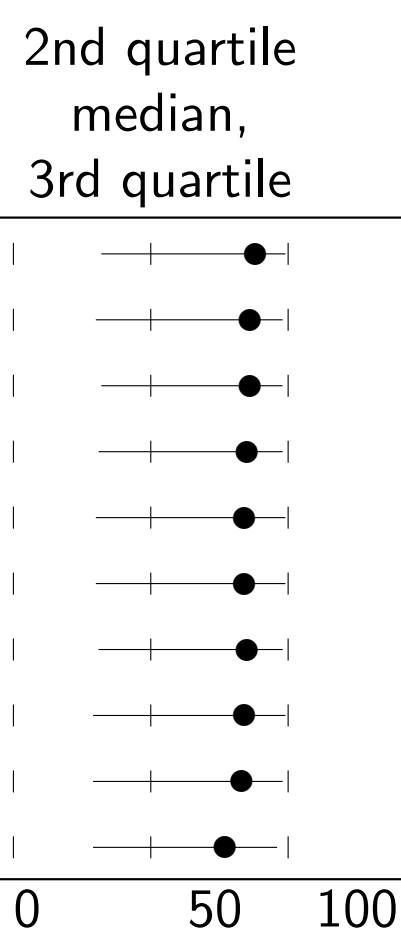
Setup II

Using the same data sets, find the minimal number of training examples required.

Using the winning method above, incrementally select instances from $N = 100, 200$ to 1000. $Train = 90\% * N$ and $Test = 100$, both selected at random.

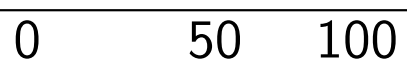
Results 2a

Rank	Training Size	pd percentiles 25%	50%	75%	2nd quartile median, 3rd quartile
1	600	32	88	99	
1	700	30	86	98	
1	900	32	86	98	
1	800	31	85	98	
1	500	30	84	99	
1	1000	30	84	99	
2	300	31	85	98	
2	400	29	84	99	
3	200	29	83	98	
3	100	29	77	96	



Results 2b

Rank	Training Size	pf percentiles 25%	50%	75%	2nd quartile median, 3rd quartile
1	600	1	12	68	
1	700	1	14	70	
1	900	1	14	68	
1	1000	1	14	69	
1	500	1	15	69	
1	800	1	15	69	
1	300	2	15	69	
1	400	1	15	71	
2	200	2	17	71	
3	100	4	23	71	



Conclusion

The best method is ADTree + discretization.

- Given by the best Mann-Whitney ranking and medians for *pd* and *pf*.

The smallest/best size of data to train on is:

- 600 instances
- However, 300 is nearly as good
  - Loss in *pd* ranking of 1 and 3% decrease in medians over 600 instances
  - *Pf* ranking is *identical*, and medians increased by another 3%.

Cross-Company Defect Prediction using Relevancy Filtering

When companies want to determine the quality of their software, they can use

- Within Company data (WC)
- Cross Company data (CC)

Problem

Adequate local data is expensive to acquire and may be impossible for newer companies. Instead, freely available, public defect data can be used.

Cross-Company Defect Prediction using Relevancy Filtering

In 2008, Turhan et al. [13] determined that by using *relevancy filtering* on cross-company training data, results are highly benefited.

The authors found that

- in a 10-way cross-validation, CC predictions weren't useful because of high false alarm rates
- however, with relevency filtered training data, CC results were nearly as good as those using WC training data.

Cross-Company Defect Prediction using Relevancy Filtering

Purpose

To verify Turhan et al.'s relevancy filtering results. This experiment can also be found in [12].

Setup

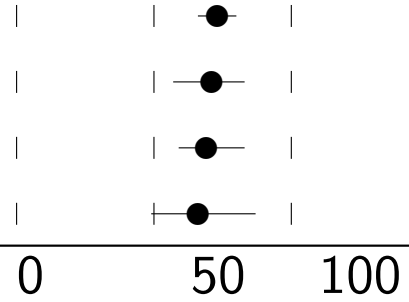
Using seven *combined* defect data sets (CM1, KC1, KC2, KC3, MC2, MW1, PC1).

To test relevancy filtering,

- a k-NN filter was used
- duplicate instances were removed, remaining instances became the new training set

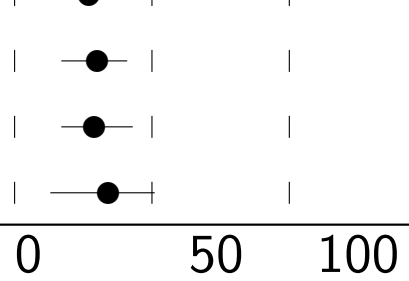
Results 1a

Rank	Treatment	pd percentiles 25%	50%	75%	2nd quartile median, 3rd quartile
1	WC (local data) + filter	66	73	80	
2	CC (imported data) + filter	57	71	83	
2	WC (local data)	59	69	83	
3	CC (imported data)	49	66	87	



Results 1b

Rank	Treatment	pf percentiles 25%	50%	75%	2nd quartile median, 3rd quartile
1	WC (local data) + filter	20	27	34	
2	WC (local data)	17	30	41	
3	CC (imported data) + filter	17	29	43	
3	CC (imported data)	13	34	51	



Conclusion

In this experiment, we find that:

- filtered WC always outperforms CC
- however, CC + *relevancy filtering* performs nearly as well as WC

Thus,

- When local data is available, that data should be used to build defect predictors
- If needed, imported data can be used to build defect predictors when using relevancy filtering

Component vs. Whole-based Defect Prediction

Problem

Despite efforts to improve software defect predictors through new algorithms (i.e. classifiers), results have reached a ceiling.

Purpose

To develop a new method of treating the *data itself* to improve performance.

What's a Component?

Hongyu Zhang, Tsinghua University, noted that most software defects occur in small numbers of components.

- Components are portions of software that encapsulate certain functionality (e.g. a specific tool in a GUI).
- Components contain 1 or more modules. Modules are functions/methods, depending on the programming language.

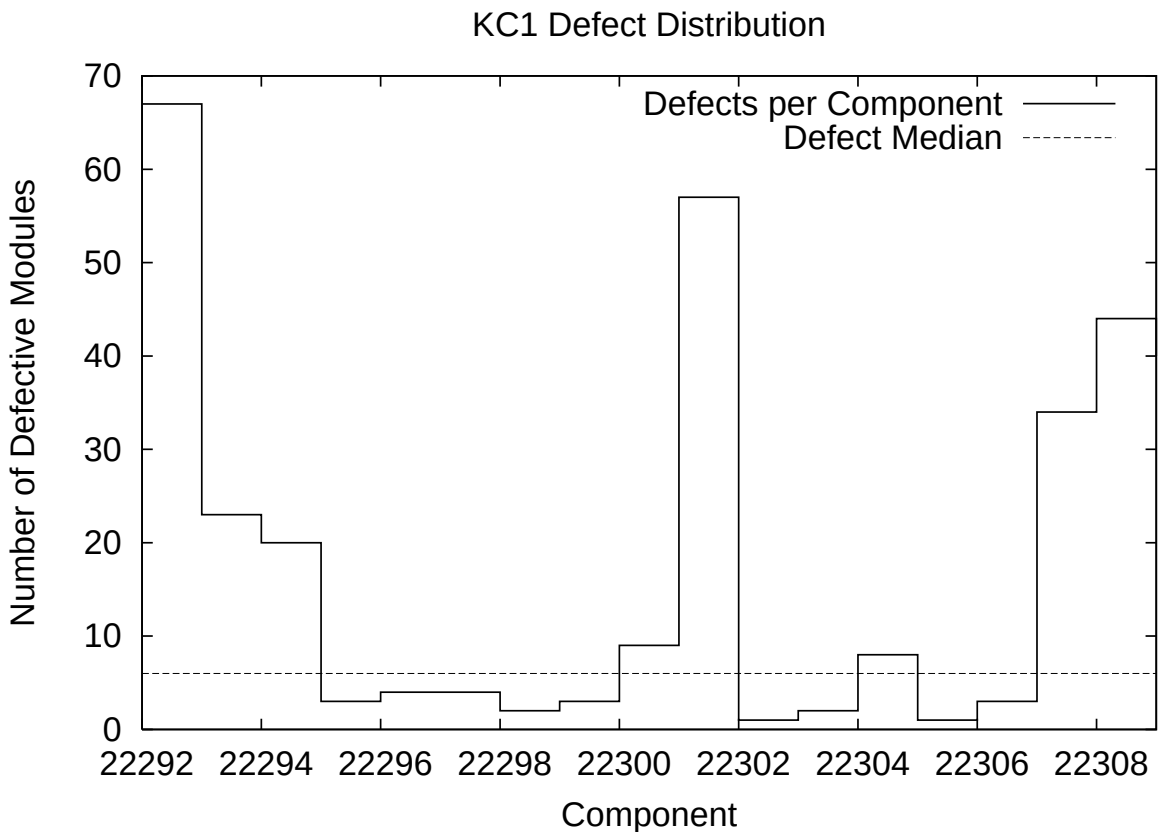
Component vs. Whole-based Defect Prediction

Setup

Using five NASA defect data sets (CM1, KC1, MC1, PC1, PC3), training instances were selected as follows:

- Pick components whose number of defective modules exceeded the median number of defective modules across *all* components.
- From those components, extract modules for training. This was the *dense* treatment.

Distribution of Defects



Component vs. Whole-based Defect Prediction

Setup cont.

The experiment was conducted as follows:

```
1 For run = 1 to 10
2   For each dense component C in data set D
3     Train = C
4     Test  = D - C
5     For bin = 1 to 10
6        Test'  = 10% of Test (picked at random)
7        Train' = 90% of Train (picked at random)
8        Naive Bayes (Train', Test')
9     end bin
10   end component
11 end run
```

Component vs. Whole-based Defect Prediction

Setup cont.

Other methods have been proposed [15] to alter the class distribution of defect training data:

- Under-sampling: randomly minimize the non-target class to equal the size of the target class
- Over-sampling: randomly maximize the target class to equal the size of the non-target class

For rigor, in this experiment, comparisons are made between all four treatments: over/under sampling, all/dense component learning.

Results per Data set

Project	Recall		0% 50% 100%	Prob. False Alarm (Pf)		0% 50% 100%	Precision		0% 50% 100%
CM1	1	All		1	Dense		1	All	
	2	Over		2	Over		1	Over	
	2	Under		2	Under		1	Under	
	3	Dense		2	All		1	Dense	
KC1	1	Dense		1	All		1	Dense	
	1	Under		1	Over		1	All	
	1	All		1	Under		1	Under	
	1	Over		1	Dense		1	Over	
MC1	1	Over		1	Over		1	Dense	
	1	Under		1	Under		1	All	
	2	All		2	All		1	Under	
	3	Dense		3	Dense		1	Over	
PC1	1	Dense		1	Dense		1	Dense	
	2	Over		2	Over		2	All	
	2	All		2	All		2	Under	
	3	Under		3	Under		3	Over	
PC3	1	Under		1	Dense		1	Dense	
	1	Over		2	Over		1	Under	
	2	All		2	Under		1	All	
	2	Dense		3	All		1	Over	

Summary Table

data set	performance measure	all components	dense components	over sampling	under sampling
CM1	precision	0	0	0	0
	recall	+	-	-	-
	pf	-	+	-	-
KC1	precision	0	0	0	0
	recall	0	0	0	0
	pf	0	0	0	0
MC1	precision	0	0	0	0
	recall	-	-	0	0
	pf	-	-	0	0
PC1	precision	-	+	-	-
	recall	-	+	-	-
	pf	-	+	-	-
PC3	precision	0	0	0	0
	recall	-	-	0	0
	pf	-	+	-	-
summary	+	1	5	0	0
	0	6	6	9	9
	-	8	4	6	6

Conclusion

Focusing on defect dense regions yields

- better predictors when considering *recall*, *pf* and *precision*.
- smaller, more managable data sets.

Future Work

Applying training on defect-dense components in a cross-company experiment. Example:

- combine all training data's (CC) components
- extract modules as per the previously discussed filter
- train predictors using these modules

Analyzing the Scalability of Clustering Text Documents

Document clustering is widely used for document organization, topic extraction, search result categorization, etc.

Problem

Rigorous clustering is slow.

- K-means: Uses $K * I * D$ document comparisons *plus* centroid update computations.

Purpose

To analyze the scalability and, thus, usability of faster heuristic methods to cluster and reduce the dimensions (terms) of the data.

Analyzing the Scalability of Clustering Text Documents

Setup

Data sets used in this experiment:

- EXPRESS schemas: AP-203, AP-214
- Text mining datasets: BBC, Reuters, The Guardian (multi-view text datasets), 20 Newsgroup subsets: sb-3-2, sb-8-2, ss-3-2, sl-8-2

Analyzing the Scalability of Clustering Text Documents

Setup cont.

Clustering: K-means vs. GenIc [16] and Canopy Clustering [17]

- GenIc

```
1  Select original candidate centers randomly
2  For each point p in the stream of points
3    Find the nearest candidate centroid to p
4    Move the nearest centroid toward p
5    Increment its weight
6    When n, 2n, 3n, ... instances have been seen
7  Compute the centroid's probability of survival
8  If low, kill it and replace it. Set its weight to 1.
9  If high, keep it and go to the next generation.
10 Candidate centers can be used in final clusters.
```

Analyzing the Scalability of Clustering Text Documents

Setup cont.

- Canopy clustering

```
1  Each "canopy" has two distance thresholds, T1/T2
2  Randomly select a point from the data
3  Use a cheap distance measure create a canopy
     with that point as a centroid. Canopy members
      are within T2
4  Expensive distance measures are only used within
the same canopies
5  Points inside of T1 are excluded from becoming
other canopy members
```

Analyzing the Scalability of Clustering Text Documents

Setup cont.

- Dimensionality Reduction
  - TF*Idf (Term Frequency * Inverse Document Frequency) For each term t in document D_j

$$Tf * df(t, D_j) = \frac{tf(t_i, D_j)}{|D_j|} log(\frac{|D|}{df(t_i)}) \quad (1)$$

  - PCA (Principal Component Analysis) - Find axes of the data such that variation is maximized (i.e. lack of variation in an axis suggests this is a less important dimension)
    - Find eigenvalues and eigenvectors of the covariance matrix

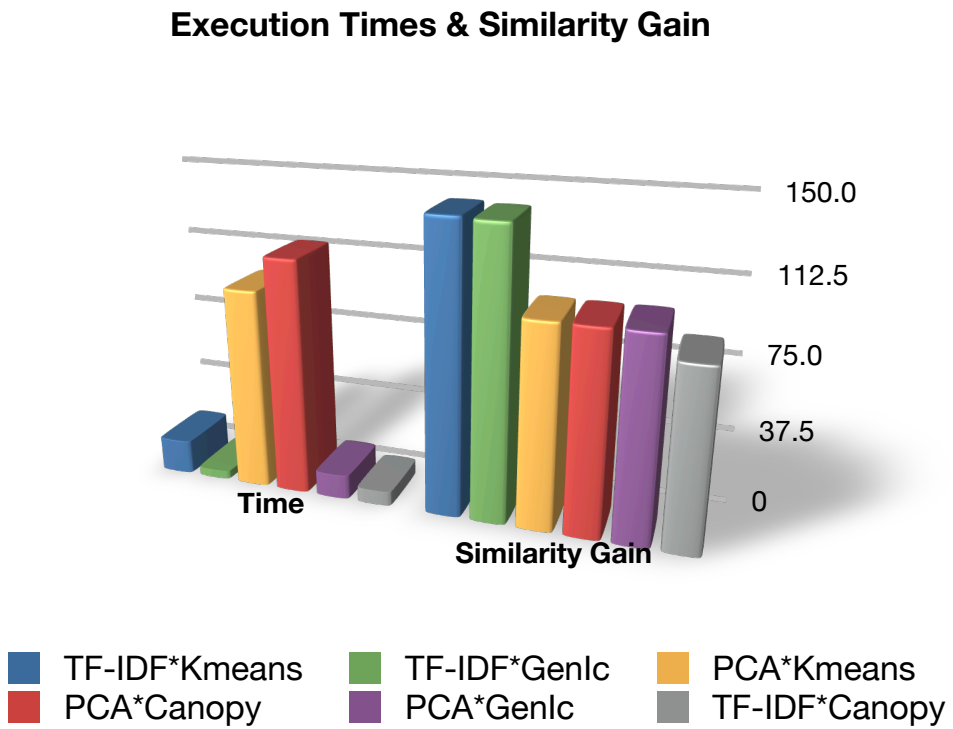
Analyzing the Scalability of Clustering Text Documents

Setup cont.

Evaluation of the clustering/reduction methods was done through

- Runtime of clustering/reducing the data
- $Gain = intra\text{-}similarity - inter\text{-}similarity$
- Cluster intra-similarity
  - For each document d in cluster C, find the similarity between d and all documents belonging to C [maximized]
- Cluster inter-similarity
  - For each document d in cluster C, find the similarity between d and all documents belonging to all other clusters. [minimized]

Results


Execution Times & Similarity Gain
150.0
112.5
75.0
37.5
0
Time
Similarity Gain
TF-IDF*Kmeans
TF-IDF*GenIc
PCA*Kmeans
PCA*Canopy
PCA*GenIc
TF-IDF*Canopy

Conclusion

By examining runtimes and similarity gain, it was shown that

- Faster clustering algorithms are competitive because of their *scalability* (GenIc vs. K-means)
- Faster dimensionality reduction methods sometimes outperform more rigorous ones on text mining data sets (Tf-IDF vs. PCA).

In summary, faster heuristics can lead to adequate results on some text mining data sets.

Outline

Conclusion

Ourmine lends itself to being a competitive toolkit for data mining because

- it's freely available
- it remains optimal for publishing experiments through printable scripts representing experiments
- it supports any language/libraries with a command-line API
- its use builds multi-functional knowledge
- the environment is functional as a teaching tool
  - novices can learn data mining concepts through simple scripts
- it's capable of complex experiments

Conclusion

Most existing, free environments suffer from

- lack of experiment sharing abilities (must publish *instructions* to build an experiment, not the experiment itself!)
- conformity to standards (custom scripting languages)
- complexity
- language restriction (Java, C++, Python, etc.)

Outline

Future Work

Note

Ourmine is far from complete!

Possible future work could include:

- Regulated contributions from the open source community
- Rapid syntax for less complex experiments.
  - Example:

```
data="weather mushroom iris"
learners="nb j48"
_crossval=10x10
$data->$learners
```

- Automated plotting/visualization of results

Questions/Comments?


"The promise (predictor models in software engineering) repository." accessible via http://promisedata.org.


G. Gay, T. Menzies, and B. Cukic, "How to build repeatable experiments," in *PROMISE '09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. New York, NY, USA: ACM, 2009, pp. 1–9.


"Knime: Konstanz information miner." available from http://www.knime.org/.


"Rapidminer." available from http://rapid-i.com/content/view/181/190/lang,en/.


"Weka: Waikato environment for knowledge analysis." available from http://www.cs.waikato.ac.nz/ml/weka/.


"Orange." available from http://www.ailab.si/orange/.


"Adam: Algorithm development and mining system." available from http://datamining.itsc.uah.edu/adam/index.html.


"The gnome data mine." available from http://www.togaware.com/datamining/gdatamine/.


"Rattle: the r analytical tool to learn easily." available from http://rattle.togaware.com/.


T. Menzies, "Evaluation issues for visual programming languages," 2002, available from http://menzies.us/pdf/00vp.pdf.


O. Ritthoff, R. Klinkenberg, S. Fischer, I. Mierswa, and S. Felske, "Yale: Yet another learning environment," in *LLWA 01 - Tagungsband der GI-Workshop-Woche, Dortmund, Germany*, October 2001, pp. 84–92, available from http://ls2-www.cs.uni-dortmund.de/~fischer/publications/YaleLLWA01.pdf.


B. Turhan, T. Menzies, A. B. Bener, and J. D. Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, 2009, available from http://menzies.us/pdf/08ccwc.pdf.


M. Druzdzel and C. Glymour, "What Do College Ranking Data Tell Us About Student Retention: Causal Discovery In Action."


T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, "Implications of ceiling effects in defect predictors," in *PROMISE '08: Proceedings of the 4th international workshop on Predictor models in software engineering*. New York, NY, USA: ACM, 2008, pp. 47–54.


C. Gupta and R. Grossman, "Genic: A single pass generalized incremental algorithm for clustering," in *In SIAM Int. Conf. on Data Mining*. SIAM, 2004.


A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," in *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2000, pp. 169–178.

Images sources:

Slide 7: `http://www.knime.org/documentation/gettingstarted`

Slide 13: `http://www.freedigitalphotos.net/images/view_photog.php?photogid=987`