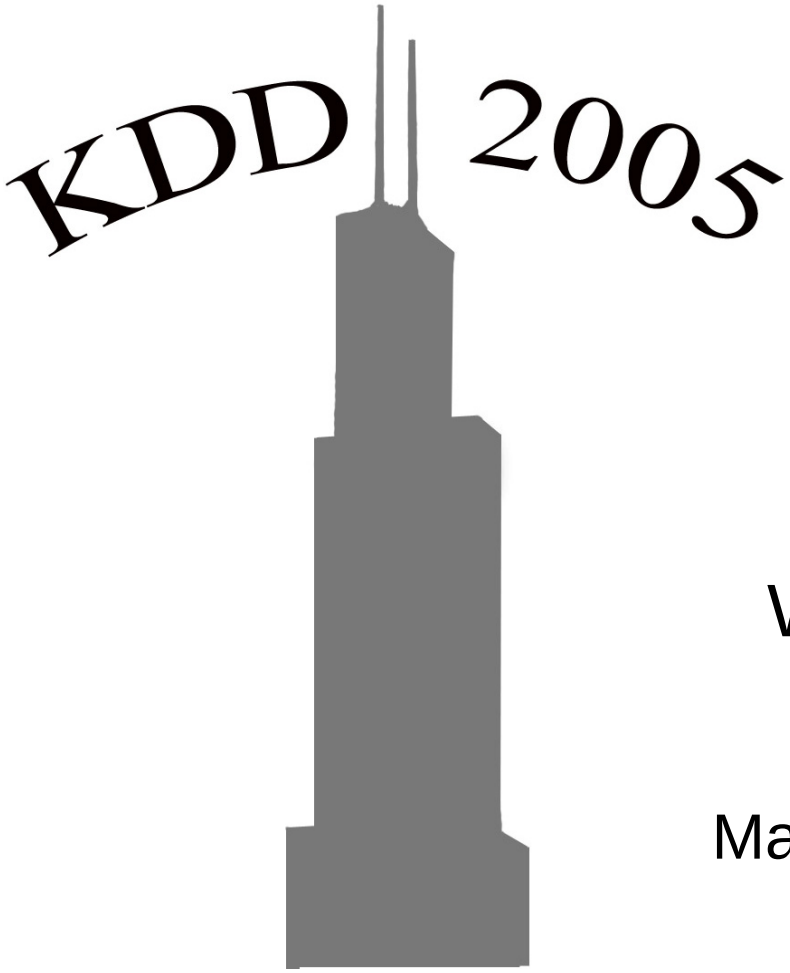
First International Workshop on Utility-Based Data Mining



Workshop Chairs:

Gary Weiss
Maytal Saar-Tsechansky
Bianca Zadrozny

August 21, 2005
Chicago, Illinois, USA

The Association for Computing Machinery, Inc.
1515 Broadway
New York, New York 10036

ACM ISBN: 1-59593-208-9

Additional copies may be ordered prepaid from:

ACM Order Department	Phone: 1-800-342-6626
P.O. BOX 11405	(U.S.A. and Canada)
Church Street Station	+1-212-626-0500
New York, NY 10286-1405	(All other countries)
	Fax: +1-212-944-1318
	E-mail: acmhelp@acm.org

Printed in the U.S.A.

Preface

Early work in predictive data mining and machine learning rarely addressed the complex circumstances in which knowledge is extracted and applied. It often assumed that training data were freely available and focused on simple objectives, namely predictive accuracy. Over time, there has been a growing interest in the machine learning and data mining communities in research addressing economical data acquisition, utility-based methods for knowledge induction and application and methodologies for evaluating the utility derived from data mining techniques.

This workshop explores the notion of economic utility and how it can be maximized throughout the data mining process. As of today much of the work focuses on a single aspect data mining. The workshop aims to bring together researchers from data mining and machine learning to share their perspective on key challenges in utility-based data mining and how individual contributions made thus far can be combined towards a comprehensive utility-based methodology.

We believe the very positive response we have had from both academia and industry indicates the importance of utility-based data mining research and hope that the workshop will promote a fruitful exchange of ideas to further advance the field.

We would like to thank all the researchers that submitted their recent work, our Program Committee, and our invited speakers, Naoki Abe, Robert Holte, and Foster Provost for their generous contributions to this workshop.

Gary Weiss
Maytal Saar-Tsechansky
Bianca Zadrozny

Table of Contents

UBDM-2005 Workshop Organizing and Program Committee

Workshop Program

8:30 – 8:45	**Opening Remarks and Welcome**
8:45 – 9:15	Invited Talk: Toward Economic Machine Learning and Utility-based Data Mining *Foster Provost*
9:15 – 9:35	Budgeted Learning of Bounded Active Classifiers* *Aloak Kapoor and Russell Greiner*
9:35 – 9:55	Economical Active Feature-value Acquisition through Expected Utility Estimation *Prem Melville, Maytal Saar-Tsechansky, Foster Provost and Raymond Mooney*
9:55 – 10:05	Reinforcement Learning for Active Model Selection *Aloak Kapoor and Russell Greiner*
10:05 – 10:30	**Break**
10:30 – 11:00	Invited Talk: Cost-Sensitive Classifier Evaluation *Robert Holte (work in conjunction with Chris Drummond)*
11:00 – 11:10	Wrapper-based Computation and Evaluation of Sampling Methods for Imbalanced Datasets *Nitesh Chawla, Lawrence Hall and Ajay Joshi*
11:10 – 11:20	Noisy Information Value in Utility-Based Decision Making *Clayton Morrison and Paul Cohen*
11:20 – 11:40	Learning Policies for Sequential Time and Cost Sensitive Classification *Andrew Arnt and Shlomo Zilberstein*
11:40 – 12:00	Improving Classifier Utility by Altering the Misclassification Cost Ratio *Michelle Ciraco, Michael Rogalewski and Gary Weiss*
12:00 – 1:30	**Lunch**
1:30 – 1:50	One-Benefit Learning: Cost-Sensitive Learning with Restricted Cost Information *Bianca Zadrozny*
1:50 – 2:00	Utility based Data Mining for Time Series Analysis—Cost Sensitive Learning for Neural Network Predictors *Sven Crone, Stefan Lessmann and Robert Stahlblock*
2:00 – 2:10	Does Cost-Sensitive Learning Beat Sampling for Classifying Rare Classes? *Kate McCarthy, Bibi Zabar and Gary Weiss*
2:10 – 2:40	Invited Talk: Machine Learning Paradigms for Utility-based Data Mining *Naoki Abe*
2:40 – 2:50	Interruptible Anytime Algorithms for Iterative Improvement of Decision Trees *Saher Esmeir and Shaul Markovich*
2:50 – 3:00	Position Paper: Contextual Recommender Problems *Omid Madani and Dennis DeCoste*
3:00 – 3:30	**Break**
3:30 – 3:50	A Fast High Utility Itemsets Mining Algorithm *Ying Liu, Wei-keng Liao and Alok Choudary*
3:50 – 4:30	Panel Discussion: Utility Based Data Mining—Research Challenges and Issues in Industry
4:30 – 4:40	**Concluding Remarks**

* Not included in proceedings due to copyright issues

Toward Economic Machine Learning and Utility-based Data Mining

Foster Provost
Stern School of Business
New York University
44 W. 4th St. New York, NY 10012

fprovost@stern.nyu.edu

ABSTRACT

Data mining requires certain information—for example, supervised learning requires training data. Some prior research has recognized that this information often does not simply present itself for free, but involves various acquisition costs. In addition, applying the learned models involves costs and benefits. I introduce a general economic setting that includes as special cases the settings of many different streams of prior research, such as cost-sensitive learning, traditional active learning, semi-supervised learning, active feature acquisition, progressive sampling, and budgeted learning, which are interwoven inextricably. For data mining in the general setting I suggest a strategy of maximum expected-utility data acquisition. Finally, I discuss how there are many open research issues that must be addressed. As a simple example, we must be able to deal with the seemingly straightforward problem of handling missing values in induction and inference.

See http://pages.stern.nyu.edu/~fprovost/ for more details (forthcoming).

UBDM '05 , August 21, 2005, Chicago, Illinois, USA.

Machine Learning Paradigms for Utility-based Data Mining*

Naoki Abe
IBM T. J. Watson Research Center
P.O.Box 218 Yorktown Heights, NY 10598
nabe@us.ibm.com

ABSTRACT

In this talk, I will describe a number of machine learning paradigms that are relevant to utility-based data mining, and review some key techniques and results in each.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning - Induction; H.2.8 [**Database Management**]: Applications - Data Mining

General Terms

Algorithms

Keywords

Machine Learning, Cost-sensitive Learning, Reinforcement Learning, Active Learning, Data Mining

There are a number of ways to introduce *utility* in machine learning, depending on the application scenario. One natural way to introduce utility is in terms of the *cost* assigned to misclassification errors, and this is the so-called *cost sensitive learning* [4]. Another way in which utility can be introduced is by considering the *cost* of data acquisition. This aspect has been rigorously formulated as *Economic Machine Learning* by Provost (c.f. [7].) One paradigm of machine learning that pays special attention to the cost of data acquisition, in addition to the predictive quality of the obtained hypotheses, is *active/query learning* [2]. The standard active learning paradigm assumes, in effect, that acquiring each example is equally costly, but it readily admits generalizations to accout for general cost structure. Another machine learning paradigm, which we might collectively refer to as *active on-line learning* addresses the issue of optimizing the combination, and trade-off, of losses incurred during data acquisition, and those associated with the predictive quality of the final hypothesis. Some examples of learning paradigms that fall within this general class include the classic bandit problem [3] and its generalizations and *associative reinforcement learning* [5, 1]. Theories have been developed on these learning paradigms, which provide learning strategies that come with theoretical guarantee on the total losses, inclusive of the two types of losses. Finally, a comprehensive paradigm of machine learning, which includes all of the ones mentioned so far as special cases, is reinforcement learning. Indeed, some authors have embedded instances of utility-based data mining problems within the MDP framework (e.g. [6]). While the MDP formulation is the most general, it does not necessarily follow that it will be the most effective in practice. When the problem at hand falls into one of the special cases discussed, the theory and methodology in that special case may be the most effective. I hope to draw some examples of real world applications, for which some of these special cases have indeed proved to be satisfactory.

1. REFERENCES

[1] N. Abe, A. W. Biermann, and P. M. Long. Reinforcement learning with immediate rewards and linear hypotheses. *Algorithmica*, 37(4):263–293, 2003.

[2] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[3] D. A. Berry and B. Fristedt. *Bandit Problems*. Chapman and Hall, New York, 1985.

[4] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 973–978, Aug. 2001.

[5] L. Kaelbling. Associative reinforcement learning: Functions in k-dnf. *Machine Learning*, 15(3):279–298, 1994.

[6] A. Kapoor and R. Greiner. Reinforcement learning for active model selection. In *Proc. ACM SIGKDD Workshop on Utility-based Data Mining*, 2005.

[7] F. Provost. Economic machine learning. In *Proc. ACM SIGKDD Workshop on Utility-based Data Mining*, 2005.

Cost-Sensitive Classifier Evaluation

Robert C. Holte
Department of Computing Science,
University of Alberta,
Edmonton, Alberta, Canada, T6G 2E8
holte@cs.ualberta.ca

Chris Drummond
Institute for Information Technology,
National Research Council Canada,
Ottawa, Ontario, Canada, K1A 0R6
Chris.Drummond@nrc-cnrc.gc.ca

ABSTRACT

Evaluating classifier performance in a cost-sensitive setting is straightforward if the operating conditions (misclassification costs and class distributions) are fixed and known. When this is not the case, evaluation requires a method of visualizing classifier performance across the full range of possible operating conditions. This paper reviews the classic technique for classifier performance visualization – the ROC curve – and argues that it is inadequate for the needs of researchers and practitioners in several important respects. It then shows that a different way of visualizing classifier performance – the cost curve introduced by Drummond and Holte at KDD'2000 – overcomes these deficiencies. A software package supporting all the cost curve analysis described in this paper is available by contacting the first author.

1. INTRODUCTION

In this paper[1], our focus is on the visualization of a classifier's performance. This is one of the attractive features of ROC analysis – the tradeoff between false positive rate and true positive rate can been seen directly. A good visualization of classifier performance allows an experimenter to immediately see how well a classifier performs and to compare two classifiers – to see when, and by how much, one classifier outperforms others.

We restrict the discussion to classification problems in which there are only two classes. The main point of this paper is to show that, even in this restricted case, ROC curves are not a good visualization of classifier performance. In particular, they do not allow any of the following important experimental questions to be answered visually:

- what is classifier C's performance (expected cost) given specific misclassification costs and class probabilities?
- for what misclassification costs and class probabilities does classifier C outperform the trivial classifiers?
- for what misclassification costs and class probabilities does classifier C1 outperform classifier C2?
- what is the difference in performance between classifier C1 and classifier C2?
- what is the average of performance results from several independent evaluations of classifier C (e.g. the results of 5-fold cross-validation)?
- what is the 90% confidence interval for classifier C's performance?
- what is the significance (if any) of the difference between the performance of classifier C1 and the performance of classifier C2?

The paper is organized around these questions. After a brief review of essential background material, there is a section devoted to each of these questions.

2. BACKGROUND

For 2-class classification problems ROC space is a two-dimensional plot with true positive rate (TP) on the y-axis and false positive rate (FP) on the x-axis. A single confusion matrix thus produces a single point in ROC space. An ROC curve is formed from a sequence of such points, including (0,0) and (1,1), connected by line segments. The method used to generate the sequence of points for a given classifier (or learning algorithm) depends on the classifier. For example, with Naive Bayes [5, 9] an ROC curve is produced by varying its threshold parameter.

An ROC curve implicitly conveys information about performance across all possible combinations of misclassification costs and class distributions[2]. We use the term "operating point" to refer to a specific combination of misclassification costs and class distributions.

One point in ROC space dominates another if it has a higher true positive rate and a lower false positive rate. If point A dominates point B, A will have a lower expected cost than B for all operating points. One set of points A is dominated by another B when each point in A is dominated by some point B and no point in B is dominated by a point in A.

[1] An early version of this paper appeared in [4].

[2] "All" distributions and costs with certain standard restrictions. For class distributions "all" means any prior probabilities for the classes while keeping the class-conditional probabilities, or likelihoods, constant [16]. For costs "all" means all combinations of costs such that a misclassification is more costly than a correct classification.

UBDM'05, August 21, 2005, Chicago, Illinois, USA.

Cost curves were introduced in [2]. Performance (expected cost normalized to be between 0 and 1) is plotted on the y-axis. Operating points are plotted on the x-axis after being normalized to be between 0 and 1 by combining the parameters defining an operating point in the following way:

$$PC(+) = \frac{p(+)\mathtt{C}(-|+)}{p(+)\mathtt{C}(-|+) + p(-)\mathtt{C}(+|-)} \quad (1)$$

where `C(-|+)`is the cost of misclassifying a positive example as negative, `C(+|-)`is the cost of misclassifying a negative example as positive, $p(+)$ is the probability of a positive example, and $p(-) = 1 - p(+)$. The motivation for this PC definition, and cost curves more generally, originates in the simple situation when misclassification costs are equal. In this case $PC(+) = p(+)$ and the y-axis becomes error rate, so the cost curve plots how error rate varies a function of the prevalence of positive examples. The PC definition generalizes this idea to the case when when misclassification costs are not equal. The PC formula is intimately tied to the definition of the slope of a line in ROC space, which plays a key role in ROC analysis. The x-axis of cost space is "slope in ROC space" normalized to be between 0 and 1 instead of being between 0 and infinity (historically this is how cost curves were invented).

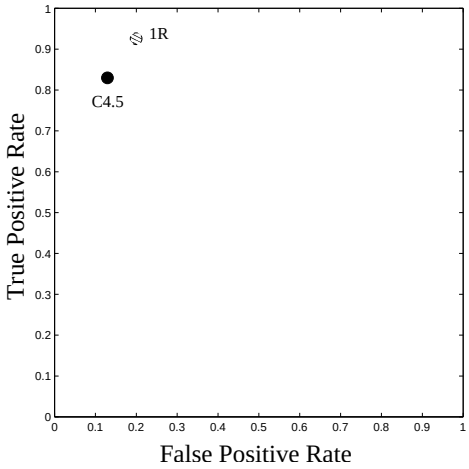


Figure 1: Two ROC points

There is a point/line duality between ROC space and cost space, meaning that a point in ROC space is represented by a line in cost space, a line in ROC space is represented by a point in cost space, and vice versa. A classifier represented by the point (FP,TP) in ROC space is a line in cost space that has $y = FP$ when $x = 0$ and $y = 1 - TP$ when $x = 1$. The set of points defining an ROC curve become a set of lines in cost space For example, Figure 1 shows the ROC points for two classifiers for the Japanese credit dataset from the UCI repository [1]: the dashed point is for the decision stump produced by 1R [7], the solid point is for the decision tree produced by C4.5 [12]. Each point becomes a line in cost space, as shown in Figure 2.

Given the cost lines for a set of classifiers, a cost curve is created by deciding which classifier to use for every possible operating point. If, for each operating point, the classifier is chosen that minimizes normalized expected cost, the resulting cost curve is the lower envelope of the given cost lines, the dual of the ROC convex hull.

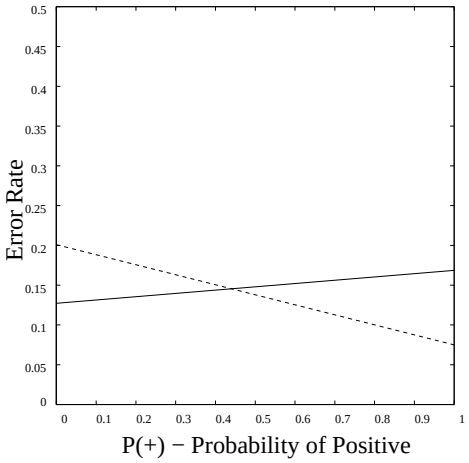


Figure 2: Corresponding Cost Lines

3. VISUALIZING PERFORMANCE

ROC analysis does not directly commit to any particular measure of performance. This is sometimes considered an advantageous feature of ROC curves. For example, Van Rijsbergen [15] quotes Swets [13] who argues that this is useful as it measures "discrimination power independent of any 'acceptable criterion' employed". Provost and Fawcett substantiate this argument by showing that ROC dominance implies superior performance for a variety of commonly-used performance measures [10]. The ROC representation allows an experimenter to see quickly if one classifier dominates another and therefore, using the convex hull, to identify potentially optimal classifiers visually without committing to a specific performance measure.

For example, Figure 3 shows a set of ROC points for C4.5 on the sonar data set from the UCI collection. Each point corresponds to a different setting of the classification threshold parameter. Even though ROC analysis does not commit to any particular measure of performance it is still possible to read certain performance-related information from this figure. For example, certain ROC points are obviously dominated by others, and from the visually obvious fact that all the ROC points are well above the chance line, the diagonal joining (0,0) to (1,1), one can easily see that the decision trees perform well overall.

Being independent of any particular performance measure can be a disadvantage when one has a particular performance measure in mind. ROC curves do not visually depict the quantitative performance of a classifier or the difference in performance between two classifiers.

The solid lines in Figure 4 are the the cost lines for the classifiers whose ROC points are shown in Figure 3. Each cost line in Figure 4 corresponds to one of the individual ROC points in Figure 3. All the conclusions drawn from the ROC plot, and more, can be made from a quick visual inspection of this cost curve plot. The lower envelope is visually obvious as is the fact that C4.5's decision tree will never have a normalized expected cost higher than 25%. One can also see that there are many choices of classification

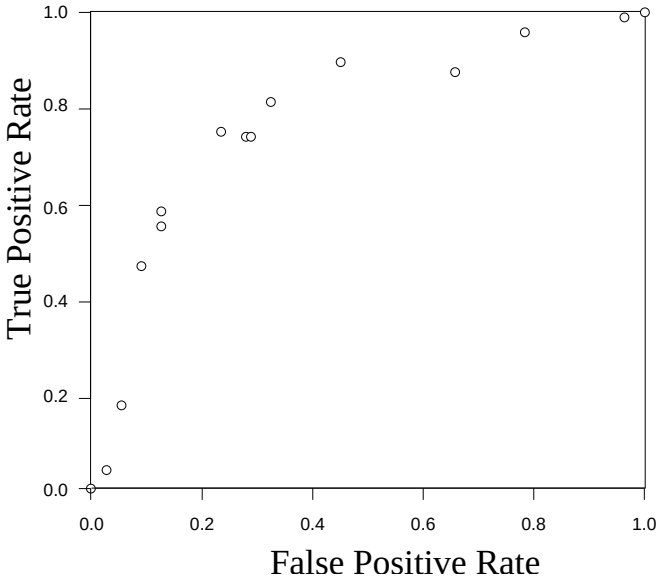


Figure 3: ROC Points for C4.5 on the Sonar dataset

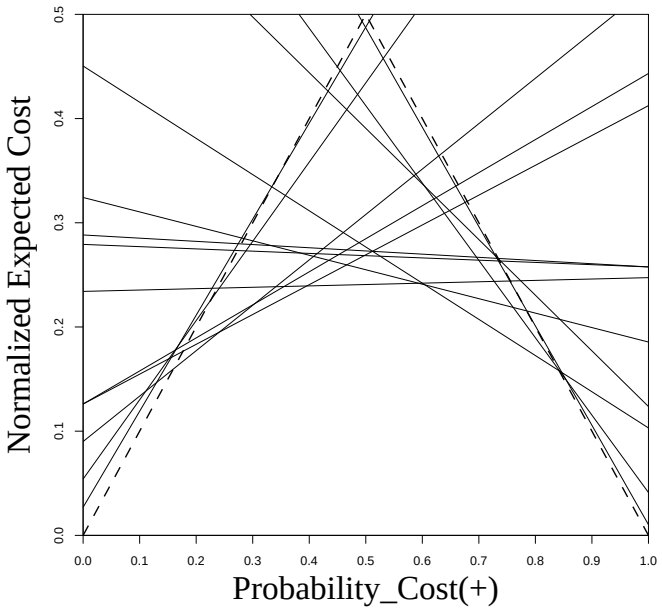


Figure 4: Cost Lines Corresponding to Figure 3

threshold that result in near-optimal normalized expected cost when $PC(+)$ is near 0.5.

4. COMPARING A CLASSIFIER TO THE TRIVIAL CLASSIFIERS

In an ROC diagram points (0,0) and (1,1) represent the trivial classifiers: (0,0) represents classifying all examples as negative, and (1,1) represents classifying all points as positive. The cost lines for these classifiers are the dashed lines shown in Figure 4. The dashed line from (0,0) to (0.5,0.5) is the cost line for the classifier that classifies all examples as negative, and the diagonal line from (0.5,0.5) to (1,0) is the cost line for the classifier that classifies all examples as positive.

The operating range of a classifier is the set of operating points where it outperforms the trivial classifiers. A classifier should not be used outside its operating range, since one can obtain superior performance by assigning all examples to a single class.

The operating range of a classifier cannot be seen readily in an ROC curve. It is defined by the slopes of the lines tangent to the ROC curve and passing through (0,0) and (1,1). By contrast, a classifier's operating range can be immediately read off of a cost curve: it is defined by the PC values where the cost curve intersects the diagonal lines representing the trivial classifiers. For example, in Figure 4 it can be seen immediately that all the classifiers being considered perform worse than a trivial classifier when $PC < 0.15$ or $PC > 0.85$.

5. CHOOSING BETWEEN CLASSIFIERS

If the ROC curves for two classifiers cross, each classifier is better than the other for a certain range of operating points. Identifying this range visually is not easy in an ROC diagram and perhaps surprisingly the crossover point of the ROC curves has little to do with the range. Consider the ROC curves for two classifiers, the dotted and dashed curves of Figure 5. The solid line is the isoperformance line tangent to the two ROC curves. Its slope represents the operating point at which the two classifiers have equal performance. For operating points corresponding to steeper slopes, the classifier with the dotted ROC curve performs better than the classifier with the dashed ROC curve. The opposite is true for operating points corresponding to shallower slopes.

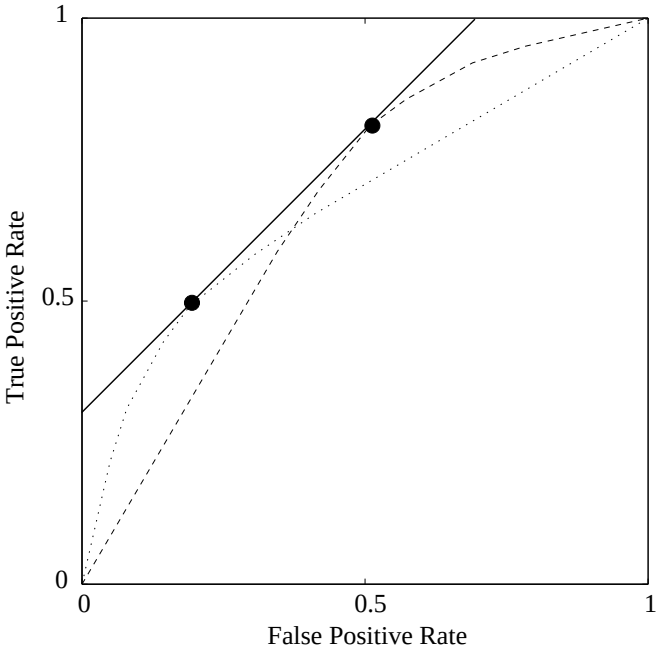


Figure 5: ROC Space Crossover

Figure 6 shows the cost curves corresponding to the ROC curves in Figure 5. It can immediately be seen that the dotted line has a lower expected cost and therefore outperforms the dashed line when $PC < 0.5$ and vice versa.

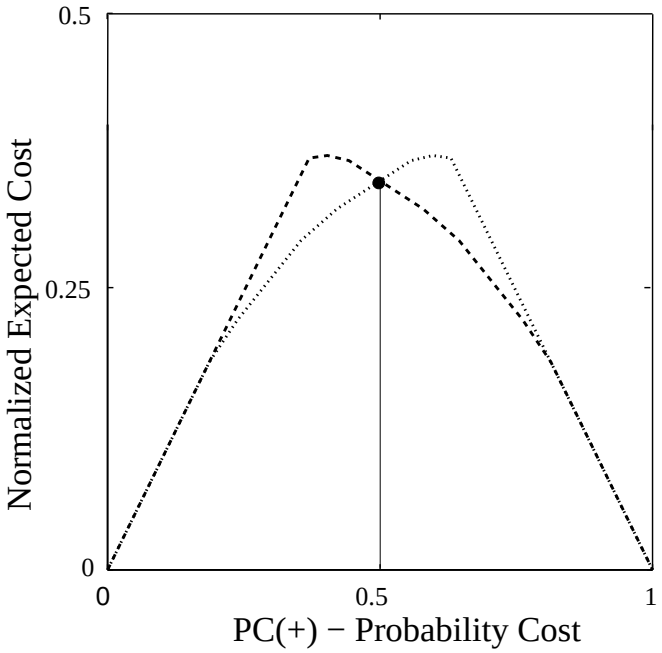


Figure 6: Corresponding Cost Space Crossover

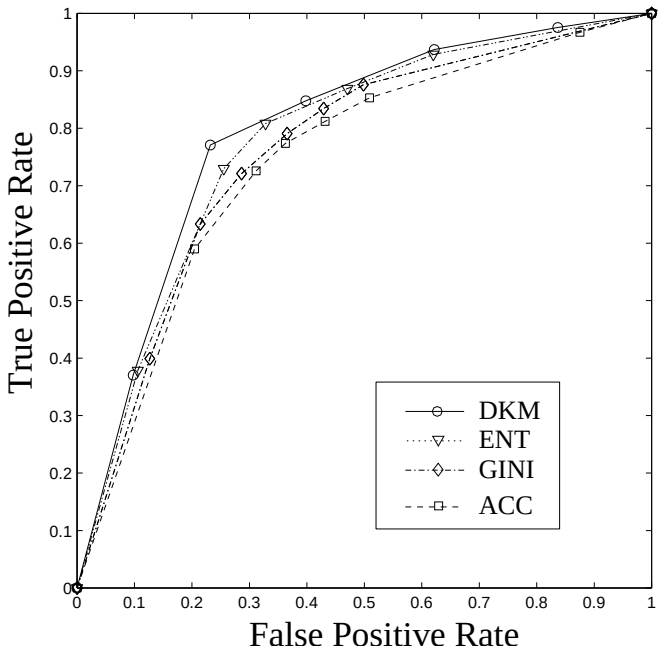


Figure 7: ROC Curves for Various C4.5 Splitting Criteria on the Sonar Dataset

6. COMPARING PERFORMANCE

The ROC curves in Figure 7 show the performance of the decision trees built on the Sonar dataset by C4.5 with different splitting criteria [3]. The ROC curves are close together and somewhat tangled, making visual analysis difficult. These are typical of the comparative experiments in machine learning. While it is clear that the DKM splitting criterion dominates the others, there is no indication of how much better DKM is than them or how much their performances differ from one another.

Figure 8 shows the corresponding cost curves. The tangled ROC curves are now cleanly separated, and the vertical distance between two cost curves directly indicates the difference in their performance. Although DKM dominates, it can now be seen that its performance differs little from ENT's over a fairly broad range, $0.3 < PC(+) < 0.6$. These two splitting criteria have similar operating ranges and are clearly superior to the other two. It can also be clearly seen that GINI dominates ACC over most of their operating range.

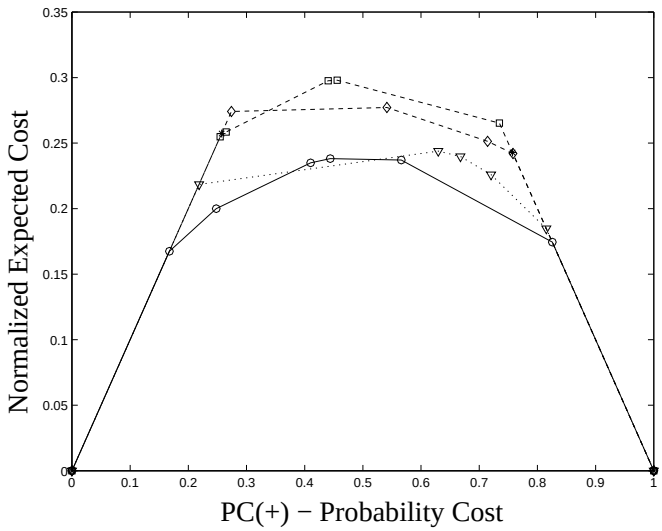


Figure 8: Cost Curves Corresponding to Figure 7

7. AVERAGING MULTIPLE CURVES

Each solid line in Figure 9 is an ROC curve based on a single non-trivial classifier. One is based on the point $(FP_1, TP_1) = (0.04, 0.4)$, the other is based on the point $(FP_2, TP_2) = (0.3, 0.8)$. We assume that they are the result of learning or testing from different random samples, or some other cause of random fluctuation in performance, and therefore their average can be used as an estimate of expected performance.

There is no universally agreed-upon method of averaging ROC curves. Swets and Pickett [14] suggest two methods, pooling and "averaging", and Provost et al. [11] propose an alternative averaging method. The Provost et al. method is to regard y, here the true positive rate, as a function of x, here the false positive rate, and to compute the average y value for each x value. We call this method "vertical averaging". In Figure 9 the vertical average is one of the dotted lines in between the two ROC curves. The other dotted line is the "horizontal" average - the average false positive rate (x) for each different true positive rate (y).

An important shortcoming of all these methods of averaging ROC curves is that the performance (error rate, or cost) of the average curve is not the average performance of the two given curves. The easiest way to see this is to consider the isoperformance line that connects the central vertices of the two ROC curves in Figure 9. The vertical and horizontal averages do not touch this line, they are well below it.

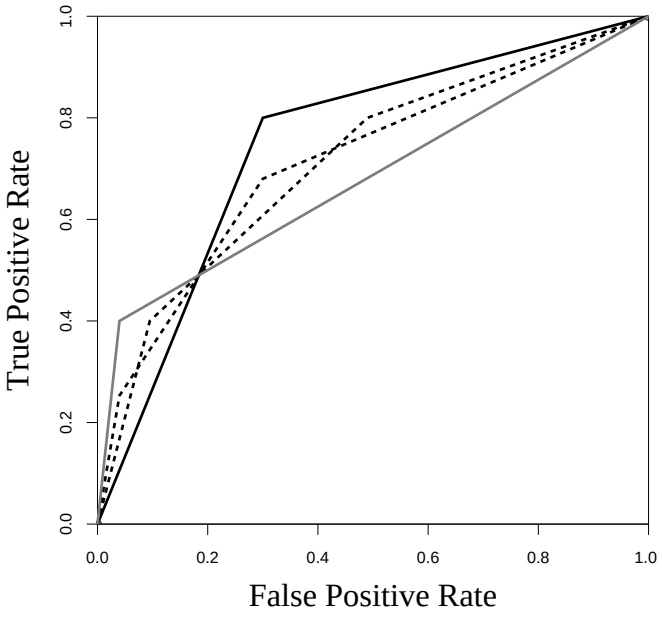


Figure 9: Vertical and Horizontal Averages of two ROC curves

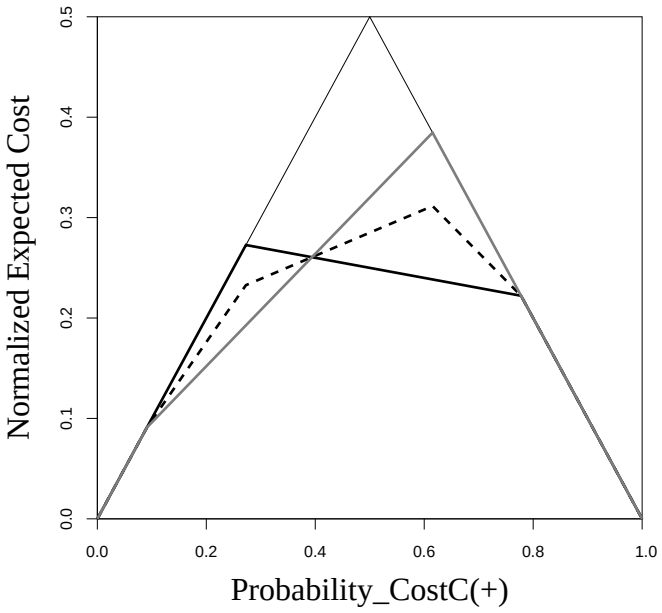


Figure 10: Average Cost Curves

Now consider what vertical averaging would do in cost space, where each x value is an operating point and y is performance (normalized expected cost). The vertical average of two cost curves is the average performance at each operating point – precisely what we wish to estimate. The solid lines in Figure 10 are the ROC curves from Figure 9 translated into cost curve lower envelopes. The expected performance based on these two cost curves is given by the bold dotted line.

8. CONFIDENCE INTERVALS ON COSTS

The measure of classifier performance is derived from a confusion matrix produced from some sample of the data. As there is likely to be variation between samples, the measure is, itself, a random variable. So some estimate of its variance is useful, which usually takes the form of a confidence interval. The most common approach to producing a confidence interval is to assume that the distribution of the estimate belongs to, or is closely approximated by, some parametric family such as Gaussian or Student-t. An alternative, data driven, method has become popular in recent times which does not make any parametric assumptions. Margineantu and Dietterich [8] described how one such non-parametric approach called the bootstrap [6] can be used to generate confidence intervals for predefined cost values. We use a similar technique, but for the complete range of class distributions and misclassification costs.

The bootstrap method is based on the idea that new samples generated from the available data are related to that data in the same way that the available data relates to the original population. Thus the variance of an estimate based on the new samples should be a good approximation to its true variance. Confidence limits are produced by resampling from the original matrix to create numerous new confusion matrices. The exact way bootstrapping is carried out depends on the sampling scheme. We propose a resampling method analogous to stratified cross validation, in which the class frequency is guaranteed to be identical in every sample.

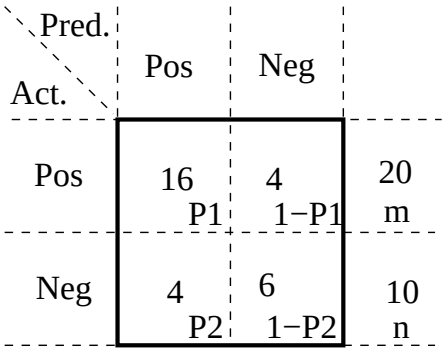


Figure 11: Binomial Sampling

For example, consider the confusion matrix of Figure 11. There are 30 instances, 20 of which are positive and 10 negative. The classifier correctly labels 16 out of 20 of the positive class, but only 6 out of 10 of the negative class. We fix the row totals at 20 and 10, and treat the two rows as independent binomial distributions with probabilities $P1 = 16/20 = 0.8$ and $P2 = 4/10 = 0.4$, respectively, of assigning a positive label to an example.

A new matrix is produced by randomly sampling according to these two binomial distributions until the number of positive and negative instances equal the corresponding row totals. For each new confusion matrix, a dotted line is plotted in Figure 12 representing the new estimate of classifier performance. For ease of exposition, we generated 100 new confusion matrices (typically at least 500 are used for an

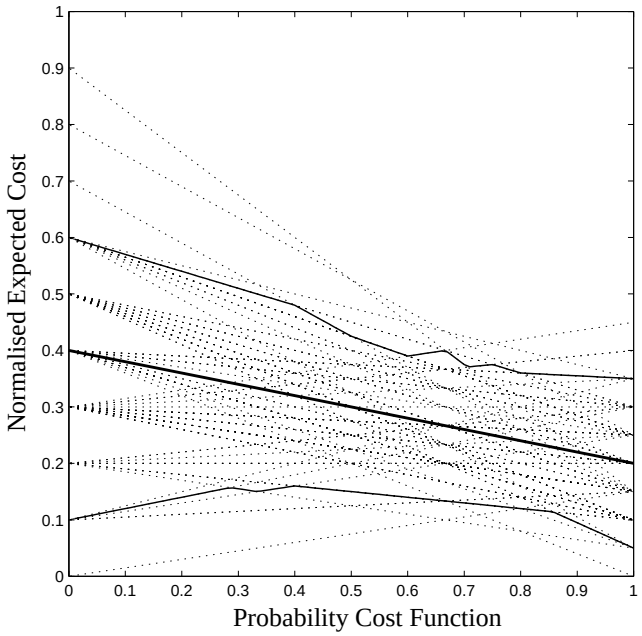


Figure 12: 90% Confidence Interval on a Cost Curve

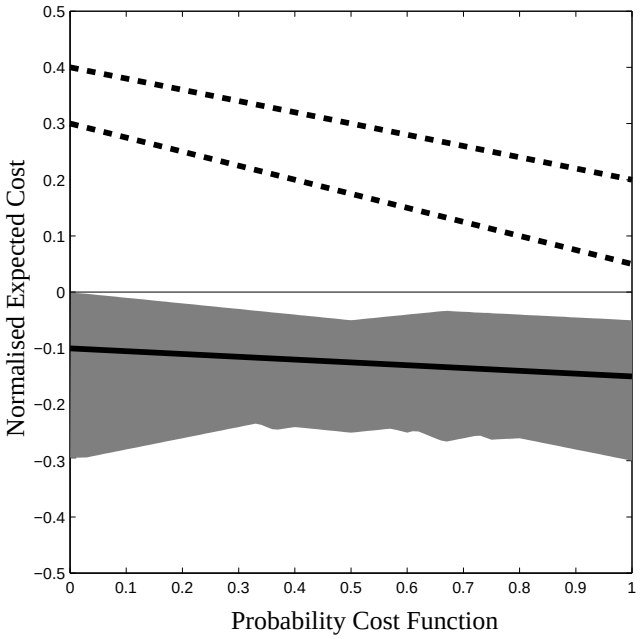


Figure 13: Confidence Interval for the Difference, High Correlation

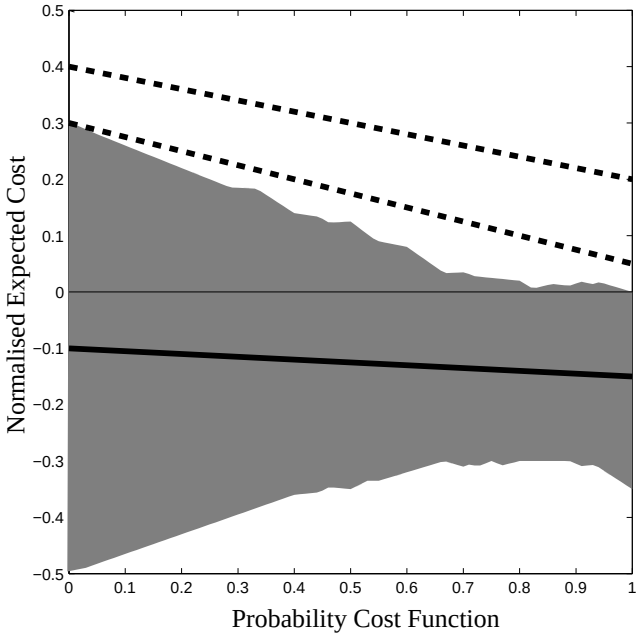


Figure 14: Confidence Interval for the Difference, Low Correlation

accurate estimate of variance). To find the 90% confidence limits, if we had values just for one specific x-value, the fifth lowest and fifth highest value could be found. This process is repeated for each small increment in the PC(+) value. The centre bold line in Figure 12 represents the performance of the classifier based on the original confusion matrix. The other two bold lines are the upper and lower confidence limits for this classifier.

9. TESTING IF PERFORMANCE DIFFERENCES ARE SIGNIFICANT

The difference in performance of two classifiers is statistically significant if the confidence interval around the difference does not contain zero. The method presented in the previous section can be extended to do this, by resampling the confusion matrices of the two classifiers simultaneously, taking into account the correlation between the two classifiers. A single resampling thus produces a pair of confusion matrices, one for each classifier, and therefore two lines in cost space. However, instead of plotting the two lines, we plot the difference between the two lines. We can repeat this process a large number of times to get a large number of lines and then, as above, extract a 90% confidence interval from this set of lines. This is the confidence interval around the difference between the classifiers' performances.

The thick continuous line at the bottom of Figure 13 represents the mean difference between performance of the two classifiers (which are shown in the figure as bold dashed lines). The shaded area represents the confidence interval of the difference, calculated as just described. As the difference can range from -1 to $+1$ the y-axis has been extended. Here we see that the confidence interval does not contain zero, so the difference between the classifiers is statistically significant. Figure 14 shows two classifiers with the same individual confusion matrices but with their classifications less correlated. Notably, the confidence interval is much wider and includes zero, so the difference is not statistically significant. Thus cost curves give a nice visual representation of the difference in expected cost between two classifiers across the full range of misclassification costs and class frequencies. The cost curve representation also makes it clear that performance differences might be significant for some range of operating points but not others. An example of this is shown in Figure 15, where the difference is significant only if $PC > 0.7$.

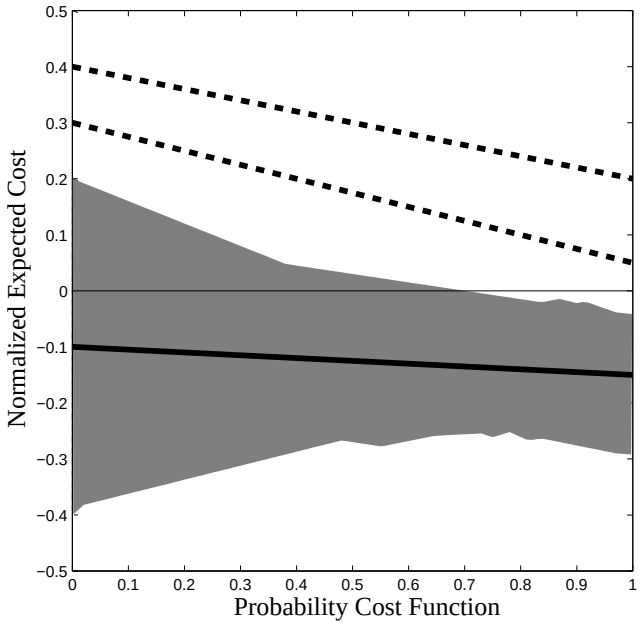


Figure 15: Confidence Interval for the Difference, Medium Correlation

10. CONCLUSIONS

This paper has demonstrated shortcomings of ROC curves for visualizing classifier performance, and shown that cost curves overcome these problems. We do not, however, contend that cost curves are always better than ROC curves. For example, for visualizing the workforce utilization measure of performance[10], ROC curves are distinctly superior to cost curves. But for many common visualization requirements, cost curves are by far the best alternative and we recommend their routine use instead of ROC curves for these purposes.

A software package supporting all the cost curve analysis described in this paper is available by contacting the first author.

11. ACKNOWLEDGEMENTS

We would like to acknowledge Alberta Ingenuity Fund for its support of this research through the funding of the Alberta Ingenuity Centre for Machine Learning.

12. REFERENCES

[1] C. L. Blake and C. J. Merz. UCI repository of machine learning databases, University of California, Irvine, CA . www.ics.uci.edu/∼mlearn/MLRepository.html, 1998.

[2] Chris Drummond and Robert C. Holte. Explicitly representing expected cost: An alternative to roc representation. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 198–207, 2000.

[3] Chris Drummond and Robert C. Holte. Exploiting the cost (in)sensitivity of decision tree splitting criteria. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 239–246, 2000.

[4] Chris Drummond and Robert C. Holte. What ROC Curves can't do (and Cost Curves can). In *Proceedings of the 1st Workshop on ROC Analysis in Artificial Intelligence (held in conjunction with ECAI 2004)*, pages 19–26, 2004.

[5] R. O. Duda and P. E. Hart. *Pattern Classification and scene analysis*. Wiley, New York, 1973.

[6] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, London, 1993.

[7] Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11(1):63–91, 1993.

[8] Dragos D. Margineantu and Thomas G. Dietterich. Bootstrap methods for the cost-sensitive evaluation of classifiers. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 582–590, 2000.

[9] M. Pazzani, C. Merz, P. Murphy, K. Ali, T. Hume, and C. Brunk. Reducing misclassification costs. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 217–225, San Francisco, 1994. Morgan Kaufmann.

[10] Foster Provost and Tom Fawcett. Robust classification systems for imprecise environments. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 706–713, Menlo Park, CA, 1998. AAAI Press.

[11] Foster Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 43–48, San Francisco, 1998. Morgan Kaufmann.

[12] J. Ross Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.

[13] J. A. Swets. *Information Retrieval Systems*. Bolt, Beranek and Newman, Cambridge, Massachusetts, 1967.

[14] John A. Swets and Ronald M. Pickett. *Evaluation of diagnostic systems : methods from signal detection theory*. Academic Press, New York, 1982.

[15] C. J Van Rijsbergen. *Information retrieval*. Butterworths, London, 1979.

[16] G. Webb and K. M. Ting. On the application of ROC analysis to predict classification performance under varying class distributions. *Machine Learning*, 58(1):25–32, 2005.

Economical Active Feature-value Acquisition through Expected Utility Estimation

Prem Melville
Dept. of Computer Sciences
Univ. of Texas at Austin
melville@cs.utexas.edu

Maytal Saar-Tsechansky
Red McCombs School of Business
Univ. of Texas at Austin
maytal@mail.utexas.edu

Foster Provost
Stern School of Business
New York University
fprovost@stern.nyu.edu

Raymond Mooney
Dept. of Computer Sciences
Univ. of Texas at Austin
mooney@cs.utexas.edu

ABSTRACT

In many classification tasks training data have missing feature values that can be acquired at a cost. For building accurate predictive models, acquiring all missing values is often prohibitively expensive or unnecessary, while acquiring a random subset of feature values may not be most effective. The goal of *active feature-value acquisition* is to incrementally select feature values that are most cost-effective for improving the model's accuracy. We present two policies, *Sampled Expected Utility* and *Expected Utility-ES*, that acquire feature values for inducing a classification model based on an estimation of the expected improvement in model accuracy per unit cost. A comparison of the two policies to each other and to alternative policies demonstrate that *Sampled Expected Utility* is preferable as it effectively reduces the cost of producing a model of a desired accuracy and exhibits a consistent performance across domains.

General Terms

Algorithms

Keywords

machine learning, data mining, active learning, cost-sensitive learning

1. INTRODUCTION

In many predictive modeling problems, feature values for training data are missing, but can be acquired at a cost. Often the cost of acquiring the missing information varies according to the nature of the information or of the particular instance for which information is missing. Consider, for example, patient data used to induce a model to predict whether or not a treatment will be effective for a given patient. Some patient data may have missing demographic information that can be obtained at a low cost. In contrast, acquiring diagnostic test results from different health-care providers can be significantly more expensive and time-consuming. Various solutions are available for learning models from incomplete data, such as imputation methods [8], and learners that ignore missing feature values such as the Naive Bayes classifier. However, these solutions almost always undermine model performance as compared to that of a model induced from complete information. Since obtaining all missing values may be prohibitively expensive, it is desirable to identify what information would be most cost-effective to acquire. In this paper we address this generalized version of the *active feature-value acquisition* (AFA) task for classifier induction [10]: given a model built on incomplete training data, select feature values that would be most cost-effective to acquire for improving the model's accuracy. The problem of feature-value acquisition is different from traditional active learning [2] in which class labels rather than feature values are missing and are costly to acquire.

Unlike prior work [9], we study AFA in a setting where the total cost to be spent on acquisitions is not determined *a priori*, but rather can be determined on-line based on the model's performance as learning progresses. This setting is motivated by the inherent uncertainty regarding the tradeoff between costs and improvement in model accuracy. An incremental spending strategy enables a decision maker to re-evaluate the desirability of further expenditures by incrementally exploring the performance curve resulting from a series of acquisition decisions. For example, one may choose not to acquire additional information if the current model accuracy is satisfactory, or if additional information is unlikely to provide a significant improvement in the model. We propose a general setting for AFA that specifies an incremental acquisition schedule. Given the current model, an AFA strategy identifies feature-value acquisitions that are estimated to be most cost-effective with respect to model accuracy.

We present a solution to the AFA task that ranks alternative feature-value acquisitions based on an estimation of the expected improvement in model performance per unit cost. Our approach is general, i.e., it can be applied to select acquisitions for any learner, and to attempt to improve any performance metric. Experimental results on decision tree induction to improve classification accuracy demonstrate that our method does consistently result in significantly improved model accuracy per unit cost compared to random feature-value acquisition. The method is particularly advantageous in challenging tasks for which there is a significant variance across

UBDM '05, August 21, 2005, Chicago, Illinois, USA.

potential acquisitions with respect to their contribution to learning per unit cost.

2. TASK DEFINITION AND ALGORITHM

2.1 Active Feature-value Acquisition

Assume a classifier induction problem where each instance is represented with n feature values and a class label. A training set of m instances can be represented by the matrix F, where $F_{i,j}$ corresponds to the value of the j-th feature of the i-th instance. Initially, the class label, y_i, of each instance is known, and the matrix F is incomplete, i.e., it contains missing values. The learner may acquire the value of $F_{i,j}$ at the cost $C_{i,j}$. We use $q_{i,j}$ to refer to the query for the value of $F_{i,j}$. The general task of active feature-value acquisition is the selection of these instance-feature queries that will result in building the most accurate model (classifier) at the lowest cost. The framework for the generalized AFA task is presented in Algorithm **??**ach step the learner builds a classifier trained on the current data, and scores the available queries based on this classifier. The query with the highest score is selected and the feature value corresponding to this query is acquired. The training data is appropriately updated and this process is repeated until some stopping criterion is met, e.g. a desirable model accuracy has been obtained. To reduce computation costs in our experiments, we acquire queries in fixed-size batches at each iteration.

Algorithm 1 General Active Feature-value Acquisition Framework

Given:
F – initial (incomplete) instance-feature matrix
$Y = \{y_i : i = 1, ..., m\}$ – class labels for all instances
T – training set = $< F, Y >$
$\mathcal{L}$ – base learning algorithm
b – size of query batch
C – cost matrix for all instance-feature pairs

1. Initialize $TotalCost$ to cost of F
2. Initialize set of possible queries Q to $\{q_{i,j} : i = 1, ..., m; j = 1, ..., n;$ such that $F_{i,j}$ is missing$\}$
3. Repeat until stopping criterion is met
4. Generate a classifier, $M = \mathcal{L}(T)$
5. $\forall q_{i,j} \in Q$ compute $score(M, q_{i,j}, \mathcal{L}, T)$
6. Select a subset S of b queries with the highest $score$
7. $\forall q_{i,j} \in S$,
8. Acquire values for $F_{i,j}$
9. $TotalCost = TotalCost + C_{i,j}$
10. Remove S from Q
11. Return $M = \mathcal{L}(T)$

Alternate problem settings of feature-value acquisition have been explored in the literature. In particular, Melville et al. [10] studied a specialized version of the AFA problem addressed here, where *all* the missing feature values for an instance are acquired at once and an acquisition policy selects the instances for which acquiring all missing values would result in the most accurate classifier. Lizotte et al. [9] studied the *budgeted learning* scenario, in which the total cost (budget) to be spent on feature-value acquisitions is determined *a priori*. We discuss these and other related research in more detail in the related work section.

2.2 Expected Utility Estimation

Specific solutions to the AFA problem differ based on the method used to score and rank queries. In our approach we provide scores based on the *expected utility* of each query (defined below). For now we assume all features are nominal, i.e., they can take on values from a finite set of values. Assume feature j has K distinct values $V_1, ..., V_K$. The expected utility of the query $q_{i,j}$ can be computed as:

$$E(q_{i,j}) = \sum_{k=1}^{K} P(F_{i,j} = V_k)\mathcal{U}(F_{i,j} = V_k) \quad (1)$$

where $P(F_{i,j} = V_k)$ is the probability that $F_{i,j}$ has the value V_k, and $\mathcal{U}(F_{i,j} = V_k)$ is the utility of knowing that the feature value $F_{i,j}$ is V_k, given by:

$$\mathcal{U}(F_{i,j} = V_k) = \frac{\mathcal{A}(F, F_{i,j} = V_k) - \mathcal{A}(F)}{C_{i,j}} \quad (2)$$

where $\mathcal{A}(F)$ is the accuracy of the current classifier; $\mathcal{A}(F, F_{i,j} = V_k)$ is the accuracy of the classifier trained on F assuming $F_{i,j} = V_k$; and $C_{i,j}$ is the cost of acquiring $F_{i,j}$. For this paper, we define the utility of an acquisition in terms of improvement in model accuracy per unit cost. Depending on the objective of learning a classifier, alternate utility functions could be used.

If we were to plot a graph of accuracy versus model cost after every iteration of AFA, our *Expected Utility* approach would correspond to selecting the query that is expected to result in the largest slope for the next iteration. If all feature costs are equal, this corresponds to selecting the query that would result in the classifier with the highest expected accuracy.

Since the true distribution of each missing feature value is unknown, we estimate $P(F_{i,j} = V_k)$ in Eq. 1 using a learner that produces class probability estimates. For each feature j, we train a classifier M_j, using this feature as the target variable and all other features along with the class as the predictors. When evaluating the query $q_{i,j}$, the classifier M_j is applied to instance i to produce the estimate $\hat{P}(F_{i,j} = V_k)$.

In Eq. 2, the true values of $\mathcal{A}(.)$ are also unknown. However, since the class labels for the training data are available at selection time we can estimate $\mathcal{A}(F)$ and $\mathcal{A}(F, F_{i,j} = V_k)$ based on the training set accuracy. In our experiments, we used 0-1 loss to measure the accuracy of the classifiers. However, other measures such as class entropy or GINI index could also be used [9]. In our preliminary studies we did not observe a consistent advantage to using entropy.

When the *Expected Utility* method described here is applied to learn a Naive Bayes classifier and feature costs are assumed to be equal, it is similar to the *greedy loss reduction* approach presented in [9]. Similar approaches to expected utility estimation have also been used in the related task of traditional active learning [12, 7, 14].

Computing the estimated expectation $\hat{E}(.)$ for query $q_{i,j}$ requires training one classifier for each possible value of feature j. Selecting the best from *all* available queries would require exploring, in the worst case, mn queries. So exhaustively selecting a query that maximizes the expected utility is computationally very intensive and is infeasible for most interesting problems. We make this exploration tractable by reducing the search space to a random sub-sample of the available queries. We refer to this approach as *Sampled Expected Utility*. This method takes a parameter α ($1 \leq \alpha \leq \frac{mn}{b}$) which controls the complexity of the search. To select a batch of b queries, first a random sub-sample of αb queries

is selected from the available pool, and then the expected utility of each query in this sub-sample is evaluated. The value of α can be set depending on the amount of time the user is willing to spend on this process. One can expect a tradeoff between the amount of time spent and the effectiveness of the selection scheme.

2.3 Instance-based Active Feature-value Acquisition

In *Sampled Expected Utility* we use a random sample of the pool of available queries to make the *Expected Utility* estimation feasible. However, it may be possible to improve performance by applying *Expected Utility* estimation to a sample of queries that is better than a random sample. One approach could be to first identify potentially informative *instances*, and then select candidate queries only from these instances. In previous work we studied a specialized version of AFA, where *all* the missing feature values for an instance are acquired at once and an acquisition policy selects the instances for which acquiring all missing values would result in the most accurate classifier[10]. The method proposed in this work, *Error Sampling* (ES), can be readily used to identify informative instances from which we can then choose candidate queries. *Error Sampling* orders incomplete instances in terms of potential informativeness in the following way. It ranks instances that have been misclassified by the current model as the most informative. Next, it ranks correctly classified instances in order of decreasing uncertainty in the model's prediction. *Error Sampling* requires building only one model at each step of AFA, and hence is not too computationally intensive to use in place of random sampling in our *Sampled Expected Utility* approach. We call this new approach *Expected Utility-ES*, in which *Error Sampling* is used to rank instances from which the first αb missing instance-feature pairs are selected as candidate queries. Where b is the desired batch size and α is the exploration parameter.

Though *Error Sampling* was designed for selecting instances, it can also be modified to acquire single feature values in our general AFA setting. The method ranks instances for acquisition, but does not provide a mechanism for selecting the most informative features for a given instance. We therefore examine a version of *Error Sampling* in which instances are ordered using the *Error Sampling* ranking, and the first b missing feature values are selected for acquisition.

3. EXPERIMENTAL EVALUATION

3.1 Methodology

We begin by evaluating our proposed approaches on four datasets from the UCI repository [1], the details of which are presented in Table 1. For the sake of simplicity, we selected datasets that have only nominal features. In the future work section, we describe how we can extend our approach to handle numeric features. None of the UCI datasets provide feature acquisition costs – in our initial experiments we simply assume all costs are equal. Later, we present additional experiments with different cost structures.

Table 1: Summary of Data Sets

Name	Instances	Features	Classes
vote	435	16	2
car	1727	6	4
lymph	148	18	4
audio	226	69	24

We compare all the proposed methods to *random feature acquisition*, which selects queries uniformly at random to provide a representative sample of missing values. For the *Sampled Expected Utility* and *Expected Utility-ES* we set the exploration parameter α to 10. Given the computational complexity of *Expected Utility* it is not feasible to run the exhaustive *Expected Utility* approach on all datasets. However, we did run *Expected Utility* on the *vote* dataset. For all methods, as a base learner we used J48 decision-tree induction, which is the Weka [16] implementation of C4.5 [11]. Laplace smoothing was used with J48 to improve class probability estimates.

The performance of each acquisition scheme was averaged over 10 runs of 10-fold cross-validation. In each fold of cross-validation, we generated learning curves in the following fashion. Initially, the learner is given a random sample of feature values, i.e. the instance-feature matrix is partially filled. The remaining instance-feature pairs are used to initialize the pool of available queries. At each iteration, the system selects a batch of queries, and the values for these features are acquired. This process is repeated until a desired number of feature values is acquired. Classification accuracy is measured after each batch acquisition in order to generate a learning curve. One system (A) is considered to be *significantly* better than another system (B) if the average accuracy across the points on the learning curve of A is higher than that of B according to a paired t-test ($p < 0.05$). As in [10], the test data contains only complete instances, since we want to approximate the true generalization accuracy of the constructed model given complete data for a test instance. For each dataset, we selected the initial random sample size to be such that the induced model performed at least better than majority class prediction. The batch size for the queries was selected based on the difficulty of the dataset. For problems that were harder to learn, we acquired a larger number of feature-values and consequently used larger batch sizes.

3.2 Results

Our results are presented in Figure 1. For all datasets, *Sampled Expected Utility* builds more accurate models than random sampling for any given number of feature acquisitions. These results demonstrate that the estimation of the expected improvement in the current model's accuracy enables effective ranking of potential queries. Consequently, *Sampled Expected Utility* selects queries that on average are more informative for the learner than an average query selected at random. The differences in performance between these two systems on all datasets is significant, as defined above. Since *Sampled Expected Utility* was proposed in order to reduce the computational costs of our original *Expected Utility* approach, we also examined the performance and computational time of the exhaustive *Expected Utility* algorithm for *vote*. We computed the average time it took to select queries in each iteration for each of the methods. These timing results are summarized in Table 2. The results show that constraining the search in *Expected Utility* by random sampling (or *Error Sampling*) can significantly reduce the selection time (by two orders of magnitude in this case) without a significant loss in accuracy.

While *Error Sampling* can rank acquisitions of complete instances effectively, it does not consider the value of individual feature values. Despite this, we observed that *Error Sampling* performs quite well. In particular, it often performs significantly better than random sampling and it sometimes performs better than *Sampled Expected Utility*. However, the performance of *Error Sampling* in this general setting of AFA is inconsistent, as it may perform significantly worse than random selection, as is seen on the *lymph* dataset.

The performance of *Expected Utility-ES* shows that the method

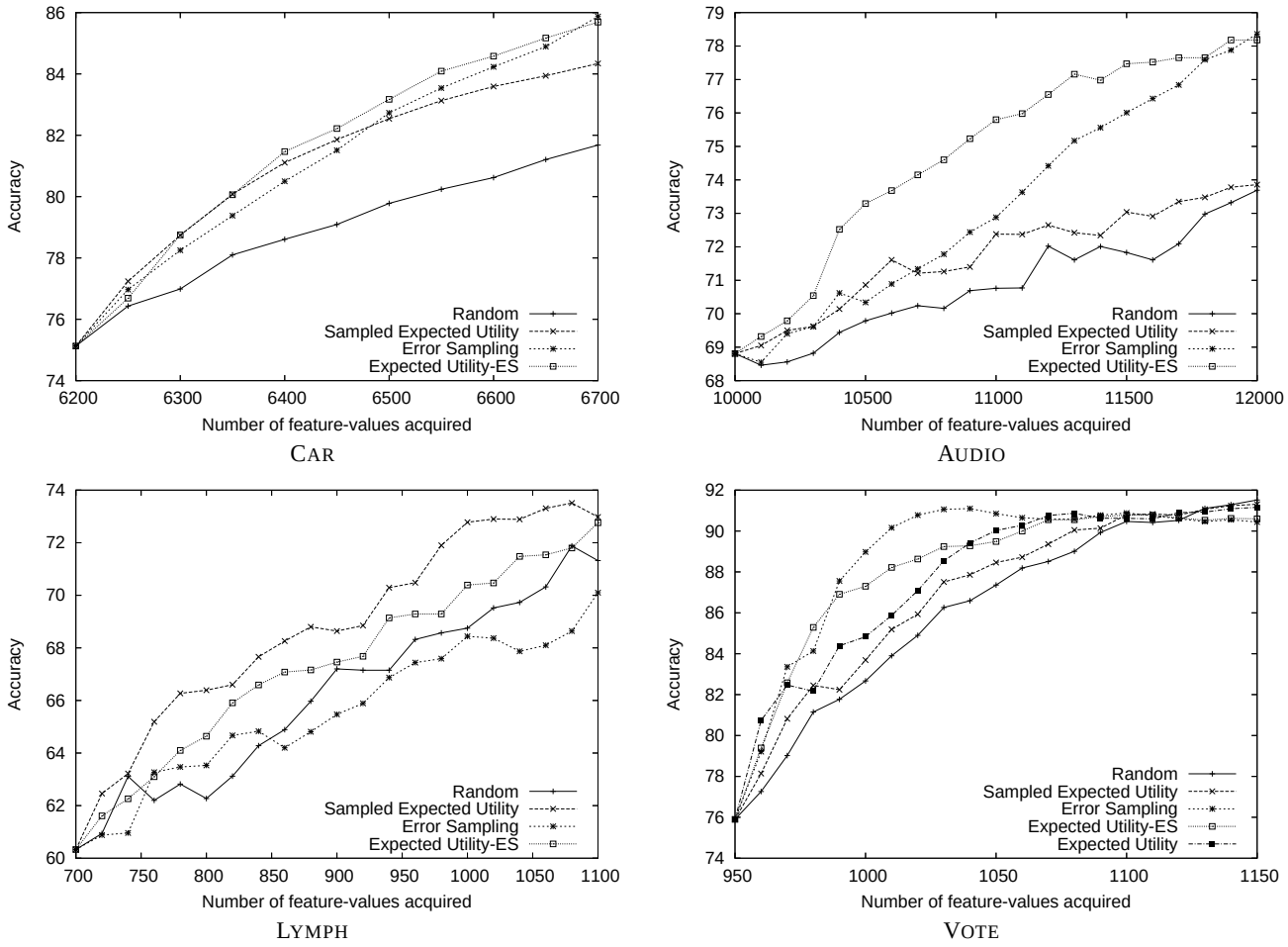


Figure 1: Comparing alternative active feature-value acquisition approaches.

Table 2: Average selection times on *vote*.

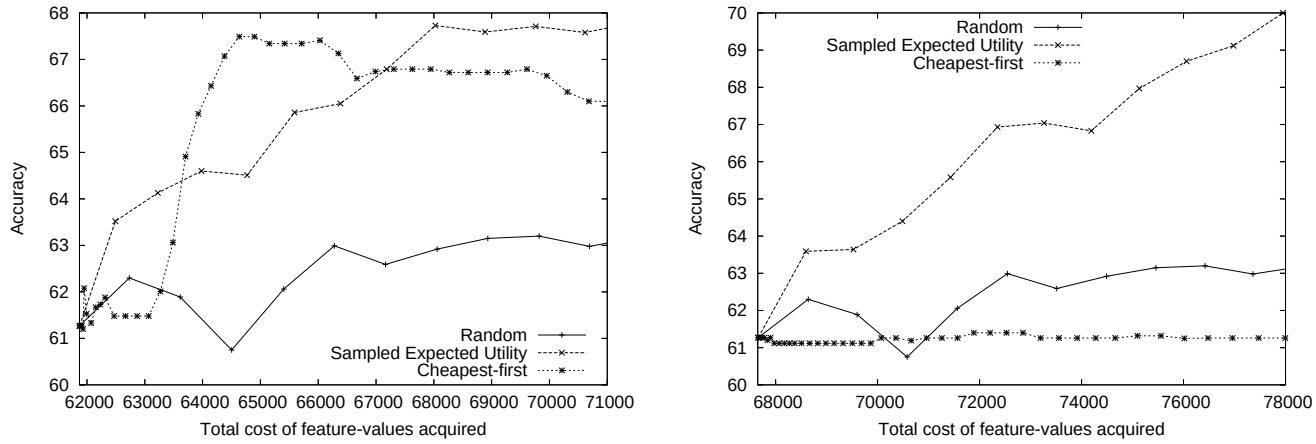
AFA Method	Selection time (msec)
Random	3.8
Expected Utility	3.77×10^5
Sampled Expected Utility	6.64×10^3
Error Sampling	8.04
Expected Utility-ES	7.44×10^3

can effectively benefit from each of its components. When *Error Sampling* performs better than random sampling, the acquisitions made by *Expected Utility-ES* result in better models than those induced with *Sampled Expected Utility*. The *vote* dataset seems to be an exception, in which *Error Sampling* can at times perform even better than *Expected Utility*, so the combined *Expected Utility-ES* method does not outperform *Error Sampling* here. *Error Sampling*'s inconsistent performance can also undermine the *Expected Utility-ES* acquisition policy, so that when *Error Sampling* fails to improve upon random acquisitions, *Expected Utility-ES* produces inferior models than those induced with *Sampled Expected Utility*. These results suggest that the use of *Error Sampling* in our current AFA setting is a promising direction for future work, but is dependent on improving the *Error Sampling* strategy such that *Error Sampling* consistently performs better than random selection. Note that, in the *instance-completion* setting of AFA for which *Error Sampling* was originally designed, it always performs better than random [10].

In summary, *Expected Utility-ES* often exhibits superior performance with respect to *Sampled Expected Utility* and random selection. However, it is susceptible to the inconsistent performance of *Error Sampling* and thus may potentially perform worse than random sampling. On the other hand, *Sampled Expected Utility* exhibits consistent improvements over random sampling on all datasets.

3.3 Artificial Data and Feature Costs

As no feature-acquisition costs are provided for the domains we employ here, we initially assumed uniform feature costs. In addition, some of the features in the data are equally discriminative so that there may be little value in selecting between them. In the extreme case, where feature costs are uniform and all features provide equal information about the target concept, random sampling is likely to be a very effective strategy. In order to make the problem setting more challenging, we constructed artificial data in the following way. We took the *lymph* dataset, which is composed of 18 features, and added an equal number of binary features with randomly-selected values, so as to provide no information about



(a) Feature cost structure 1

(b) Feature cost structure 2

Figure 2: Comparing different algorithms on artificial data under different cost structures

the class variable. In addition, we experimented with different cost structures. For the sake of simplicity, instead of having a cost associated with each instance-feature pair, we assume that the cost of acquiring a particular feature is the same irrespective of the instance. With each feature, we associate a cost selected uniformly at random from 1 to 100. Experiments were run as before for 5 different assignments of feature costs. Along with recording the accuracy after each batch acquisition of queries, we also record the current model cost based on the cost of the features acquired. Since random sampling does not take feature costs into account, we also compare *Sampled Expected Utility* with a simple baseline strategy that incorporates feature costs. This approach, which we call *Cheapest-first*, selects feature values for acquisition in order of increasing costs. Given the inconsistent performance of *Error Sampling* and *Expected Utility-ES*, we do not apply them to these datasets.

Figure 2 presents plots of accuracy versus model cost for two representative cost structures. The results for all randomly assigned costs structures show that for the same cost, *Sampled Expected Utility* consistently builds more accurate models than random sampling. The differences in performance between these two systems is more substantial than those observed for the UCI datasets with uniform costs. In contrast, the performance for *Cheapest-first* is quite varied for different cost assignments. When highly informative features are assigned low costs, *Cheapest-first* can perform quite well (Figure 2(a)). Since the underlying assumption of the *Cheapest-first* strategy, that the cheapest features are also informative, often holds in this case, it sometime performs better than *Sampled Expected Utility*, which imperfectly estimates the expected improvement in accuracy from each acquisition. However, when many inexpensive features are also uninformative, *Cheapest-first* performs worse than a random acquisition policy (Figure 2(b)). *Sampled Expected Utility*, however, estimates the tradeoff between cost and expected improvement in accuracy, and although the estimation is clearly imperfect, it consistently selects better queries than random acquisitions for all cost structures.

4. RELATED WORK

To the best of our knowledge, the methods we propose here are the first approaches designed for the general problem of incrementally ranking and selecting feature values for inducing any classifier under a general acquisition cost structure. In this section, we discuss alternate settings for the AFA task.

Lizotte et al. [9] study AFA in the *budgeted learning* scenario, in which the total cost to be spent towards acquisitions is determined *a priori* and the task is to identify the best set of acquisitions for this cost. In contrast, our setting aims to enable the user to stop the acquisition process at any time, and as such the *order* in which acquisitions are made is important. Given this criterion, we attempt to select the next acquisition that will result in the most accurate model per unit cost. Lizotte et al. also assume that feature values are independent given the class, and as such consider queries of the form "Give me the value of feature j for any instance in class k." However, our approach evaluates feature-value acquisitions of specific instances, which allows us to 1) incorporate feature-value costs that vary per instance; and 2) to better estimate the expected value of an acquisition by capturing improvements from better modeling of feature interactions. Note that a set of features may exhibit different interactions for different instances, in which case evaluating potential acquisitions for individual instances is critical.

In this paper, we explored the use of the *Error Sampling* policy designed for the *instance-completion* setting, in which all missing feature values are acquired for a selected training instance [17, 10]. *Sampled Expected Utility* selects individual features, and hence can be also employed in the instance-completion setting, e.g., by selecting the instance with the highest sum of utilities of individual feature-value acquisitions.

Some work on *cost sensitive* learning [15] has addressed the issue of inducing economical classifiers when there are costs associated with obtaining feature values. However, most of this work assumes that the *training* data are complete and focuses on learning classifiers that minimize the cost of classifying incomplete *test* instances. An exception, CS-ID3 [13], also attempts to minimize the cost of acquiring features during training; however, it processes examples incrementally and can only request additional information for the current training instance. CS-ID3 uses a simple greedy strategy that requests the value of the cheapest unknown feature when the existing hypothesis is unable to correctly classify the current

instance. It does not actively select the most useful information to acquire from a pool of incomplete training examples. The LAC* algorithm [5] also addresses the issue of economical feature acquisition during both training and testing; however, it also adopts a strategy that does not actively select the most informative data to collect during training. Rather, LAC* simply requests complete information on a random sample of instances in repeated *exploration* phases that are intermixed with *exploitation* phases that use the current learned classifier to economically classify instances.

Traditional *active learning* [2, 4] assumes access to unlabeled instances with complete feature data and attempts to select the most useful examples for which to acquire class labels. Active feature-value acquisition is a complementary problem that assumes labeled data with incomplete feature data and attempts to select the most useful additional feature values to acquire.

5. LIMITATIONS AND FUTURE WORK

In *Sampled Expected Utility* we used a random sample of the pool of available queries to make the *Expected Utility* estimation feasible; and in *Expected Utility-ES*, we explored the possibility of limiting the set of candidate queries to only potentially informative instances. Alternatively, we can restrict the set of candidate queries to only the most informative features. A subset of such features could be picked using a *feature selection* technique that can capture the interactions among feature values, such as the wrapper approach of John et al. [6].

The performance of *Expected Utility* relies on having good estimates of the feature-value distributions and of the improvement in model accuracy for each potential acquisition. Thus *Expected Utility* is likely to benefit from improving upon the methods we applied to perform these estimations. For example, we could use probability estimation methods that better approximate the feature-value distributions, specifically when there are many missing values.

The *Expected Utility* framework allows us to consider model performance objectives other than accuracy. For example, when the benefits from making different accurate predictions and the error costs are specified, *Expected Utility* can be applied to identify acquisitions that result in the highest growth in benefits per unit cost. Experimenting with such alternate measures of model performance is an avenue for future work.

Our current study was restricted to datasets that are composed of only nominal features. Since many interesting domains include both numeric and nominal features, we would like to extend this study to datasets which also have numeric features. We could apply our current *Expected Utility* method after converting the numeric features to nominal features using a discretization technique, as in [3].

6. CONCLUSION

In this paper, we propose an expected utility approach to active feature-value acquisition, that obtains feature values based on the estimated expected improvement in model accuracy per unit cost. We demonstrate how this computationally intensive method can be made significantly faster, without much loss in performance, by constraining the search to a sub-sample of potential feature-value acquisitions. Experiments with uniform feature costs show that this *Sampled Expected Utility* approach consistently builds more accurate models than random sampling for the same number of feature-value acquisitions, and exhibits consistent performances across domains as compared to policies employing an instance-based ranking of features. Additional experiments on artificial datasets with different cost structures demonstrate that for the same cost, *Sampled Expected Utility* builds more accurate classifiers than the cost-agnostic random feature acquisition approach. Its performance is also more consistent than that of a simple cost-sensitive method which acquires feature values in order of increasing cost.

Acknowledgments

Prem Melville and Raymond Mooney were supported by DARPA grant HR0011-04-1-007.

7. REFERENCES

[1] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. http://www.ics.uci.edu/˜mlearn/MLRepository.html, 1998.

[2] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[3] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1022–1027, Chambéry, France, 1993. Morgan Kaufmann.

[4] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28:133–168, 1997.

[5] Russell Greiner, Adam Grove, and Dan Roth. Learning cost-sensitive active classifiers. *Artificial Intelligence*, 139(2):137–174, 2002.

[6] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML-94)*, pages 121–129, 1994.

[7] Michael Lindenbaum, Shaul Markovitch, and Dmitry Rusakov. Selective sampling for nearest neighbor classifiers. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 366–371, 1999.

[8] R. Little and D. Rubin. *Statistical Analysis with Missing Data*. John Wiley and Sons,, 1987.

[9] Dan Lizotte, Omid Madani, and Russell Greiner. Budgeted learning of naive-Bayes classifiers. In *Proceedings of 19th Conference on Uncertainty in Artificial Intelligence (UAI-2003)*, Acapulco, Mexico, 2003.

[10] Prem Melville, Maytal Saar-Tsechansky, Foster Provost, and Raymond Mooney. Active feature-value acquisition for classifier induction. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM-04)*, 2004.

[11] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo,CA, 1993.

[12] Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of 18th International Conference on Machine Learning (ICML-2001)*, pages 441–448. Morgan Kaufmann, San Francisco, CA, 2001.

[13] Ming Tan and Jeffery C. Schlimmer. Two case studies in cost-sensitive concept acquisition. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 854–860, Boston, MA, July 1990.

[14] Simon Tong and Daphne Koller. Active learning for parameter estimation in Bayesian networks. In *Advances in Neural Information Processing 13 (NIPS)*, pages 647–653, 2000.

[15] P. D. Turney. Types of cost in inductive concept learning. In *Proceedings of the Workshop on Cost-Sensitive Learning at the 17th International Conference on Machine Learning*, Palo Alto, CA, 2000.

[16] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, 1999.

[17] Z. Zheng and B. Padmanabhan. On active learning for data acquisition. In *Proceedings of IEEE International Conference on Data Mining*, 2002.

Reinforcement Learning for Active Model Selection

Aloak Kapoor
aloak@cs.ualberta.ca

Russell Greiner
greiner@cs.ualberta.ca

Department of Computing Science
University of Alberta
Edmonton, AB T6J 2E8

ABSTRACT

In many practical Machine Learning tasks, there are costs associated with acquiring the feature values of training instances, as well as a hard learning budget which limits the number of feature values that can be purchased. In this budgeted learning scenario, it is important to use an effective "data acquisition policy", that specifies how to spend the budget acquiring training data to produce an accurate classifier. This paper examines a simplified version of this problem, "active model selection" [10]. As this is a Markov decision problem, we consider applying reinforcement learning (RL) techniques to learn an effective spending policy. Despite extensive training, our experiments on various versions of the problem show that the performance of RL techniques is *inferior* to existing, simpler spending policies.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Induction, Knowledge acquisition, Parameter learning*

General Terms

Algorithms

Keywords

budgeted learning, data acquisition, learning costs, training costs

1. INTRODUCTION

Traditional learning theory assumes the existence of a sufficient number of training examples for learning the target concept. In practice, however, there are often costs for acquiring the value of each feature for each training example. Here, the actual quantity of training data is limited by the learner's finite budget for purchasing features. In this budgeted learning scenario, it is critical to develop an intelligent purchasing policy which collects feature values in a way that maximizes the expected accuracy of the resulting classifier.

UBDM August 21, 2005, Chicago, Illinois, USA

To gain insight into the budgeted learning problem, we examine a simpler variant known as *active model selection*: Use a set of b "probes" to determine which of a given set of n objects has the highest expected value, where each probe of a specified object returns a value drawn from that object's distribution. Here we need to identify a strategy for deciding when to probe each object, based on the results of the previous probes and any prior knowledge of the objects. After exhausting the budget of b probes,[1] we select a single object to return, and receive a reward corresponding to the difference between true expected values of the object returned and the best possible object — i.e., the one with the highest expected value. This formulation allows pure exploration of the objects with the budget, as it delays all reward until the final time step. In addition, this budgeted problem accurately captures the training phase of budgeted learning in general, in which features of labelled training instances can be purchased in any way, with a single one-time reward (i.e., the classification accuracy) being received once the budget is exhausted and the final learned classifier is applied. Moreover, previous research has shown that policies that perform well on active model selection translate directly to effective spending policies for the budgeted learning problem. As a result, active model selection can be used as a way to prototype the performance of strategies for general budgeted learning.

Our focus in this paper is on applying reinforcement learning (RL) techniques to the active model selection problem. Although the existing (heuristic) purchasing policies for active model selection perform well, we demonstrate in this work that they still leave room for improvement. We also show that RL is well-suited to the task because (from a Bayesian viewpoint) the active model selection problem is a finite Markov Decision Process with a completely known environment model. We proceed to learn various purchasing policies with RL, where each policy is learned using a different set of features for function approximation. Despite the variety of feature sets tested, we observe that the policies produced by reinforcement learning are inferior to existing, simpler policies. This provides evidence that RL techniques, if they can be applied at all, will require a more sophisticated choice of features in order to be effective for active model selection, and hence for the more complex problem of budgeted learning.

The remainder of the paper is organized as follows. Sec-

[1] Actually, we can assume different objects have different probe costs; we require only that the total cost of the probes be under b. See Section 2.

tion 2 presents the active model selection problem formally, and introduces some of the simple purchasing policies from previous research. Section 3 frames the problem as a Markov Decision Process and discusses the reinforcement learning techniques that we will use. Section 4 provides empirical results comparing RL to the existing policies. In closing, Section 5 discusses related literature and Section 6 provides conclusions and a discussion of future work. All proofs can be found in the Appendix.

We will assume the reader is already familiar with the basic notions of reinforcement learning; if not, we refer the interested reader to [17].

2. ACTIVE MODEL SELECTION

The input to the problem is:

- A set of n independent Bernoulli random variables $\{C_1, \ldots, C_n\}$ with unknown success probabilities. For simplicity of exposition, we can think of these C_i as a set of coins, where the unknown success probability is the probability of the coin turning up heads when flipped.
- A set of n prior distributions (i.e., density functions) over the head probability of each coin C_i. That is, the *head probability* of each coin C_i is itself treated as a random variable X_i, and a prior density function $f(X_i)$ is provided as a distribution over the possible head probabilities of coin C_i.
- A set of n (known) costs $\{S(C_i)\}$ for flipping the coins.
- A finite (known) budget $b \geq 0$ that can be spent flipping the coins.

Given these inputs, the active model selection problem proceeds as follows. Any coin C_i can be flipped at any time, as long as the remaining budget, denoted by b', satisfies $b' \geq S(C_i)$. We use the outcome of each coin flip to update the density function for the flipped coin. For example, if coin C_i is flipped and turns up heads, then its density function is updated to $f(X_i|C_i = heads)$; of course, a similar update occurs for a tails outcome. (We describe the exact format of the density function and the updates in our simplifying assumptions below.) Coin flips and density updates continue until the budget is exhausted ($b' = 0$). Once the budget is exhausted, the learning period is over, and a single coin must be chosen — this coin C^* (and only this coin) will be used in all future flips, for which we will receive rewards for head outcomes. Of course, even when $b' = 0$, we will still not *know* the true head probability for this (or any) coin, and so will not know whether coin C^* actually has a better head probability than the other coins. The best we can do is to choose the coin that minimizes our future regret of selecting it. To do this, we define a new random variable X_{max} to be the maximum head probability over all of the coins: $X_{max} = max_i(X_i)$, and now the Bayesian regret of choosing coin C_i is:

$$\text{Regret}(C_i) \;=\; \int_{\vec{X}} (X_{max} - X_i) \prod_{j=1}^{n} f(X_j)\; d\vec{X} \qquad (1)$$

Notice that we minimize regret by choosing the coin whose mean head probability $E(X_i)$ is largest [10]. Let this maximum mean coin be $C^* = \arg\max_{C_i} E[X_i]$. Thus, when the budget is exhausted, C^* *should* be selected. (We will later use $C^*(\mathbf{o}) = \arg\max_{C_i} E[X_i|\mathbf{o}]$, as a function of the observed coin flip outcomes $\mathbf{o}$.)

Before introducing the overall (regret-related) objective function we wish to minimize, we must first introduce the notion of a policy. A policy π for active model selection specifies which coin to flip at each time step. Formally, a policy is a mapping π: $\langle\, b',\ f(X_1),\ \ldots,\ f(X_n)\,\rangle \rightarrow [1, n]$ that specifies the index of the coin to flip, given the current state defined by the remaining budget and the posterior distributions over the coins.

Since the result of every coin flip is stochastic, a policy for flipping the coins can result in *several* different "outcome" states in which the budget is exhausted. Thus, a policy π for active model selection is scored based on its expected regret:

$$\text{ER}(\pi) \;=\; \sum_{\mathbf{o} \in outcomes(\pi)} P(\mathbf{o})\ \text{Regret}(C^*(\mathbf{o})) \qquad (2)$$

where the sum is over the various "outcomes" of the policy when the budget b' has been exhausted. The objective of active model selection is to find the optimal policy π^* that minimizes Equation 2.

Conveniently, minimizing Equation 2 is equivalent to maximizing the expected head probability of the chosen coin:

$$\pi^* \quad = \quad \arg\max_{\pi} \sum_{\mathbf{o} \in outcomes(\pi)} P(\mathbf{o})\ \max_i \{E(X_i|\mathbf{o})\} \qquad (3)$$

Equation 3 is an easier objective to remember for active model selection: flip the coins so that the expected$_1$ maximum expected$_2$ head probability of the chosen coin is as large as possible. (Note that both "expected" are required as the first expectation$_1$ is over possible outcomes of the policy, while the second expectation$_2$ is over the head probability distribution of the chosen coin.)

2.1 Simplifying Assumptions

Coin C_i's head probability is represented as a random variable X_i. We assume that X_i is a Beta random variable with density function $f(X_i) = Z\ (X_i)^{\alpha-1}(1 - X_i)^{\beta-1}$ (here Z is a normalizing constant and α and β are two positive hyperparameters that define the Beta distribution).

For a Beta(α, β) distribution, the mean is $\mu = \frac{\alpha}{\alpha+\beta}$ while the variance is $\sigma^2 = \frac{\mu(1-\mu)}{\alpha+\beta+1}$. Loosely speaking, when α (β) is much larger than β (α), it means that a coin is likely to have a high (low) head probability. On the other hand, when both α and β are 1, the distribution over head probabilities is uniform.

One attractive property of the Beta distribution is that it is computationally simple to calculate posterior densities. After observing h heads and t tails on coin C_i, its posterior density is just $f(X_i|h\ \ heads,\ t\ \ tails) = Beta(\alpha + h, \beta + t)$. For example, if a coin is initially a Beta$(6, 2)$ and we observe three heads and two tails, then it becomes a Beta$(9, 4)$ coin. In some sense, the Beta hyperparameters can be viewed as simple frequency counts for a random variable with two possible outcomes.

Although the formal description allows for any coin costs, in this paper we will assume that the costs are uniform: $S(C_i) = 1\ \forall i$, and that the budget b is a positive integer. Finally, as we are studying active model selection because of its relationship to budgeted learning, we are typically in-

terested in values of b that are not much greater than n (typically $b = n \times k$, with k a small positive integer), as most budgeted learning algorithms will act reasonable when b is much larger than n. In fact, in the case where b is very large relative to n, even a simple policy (e.g. purchasing every feature of every instance) will yield a training set that can produce an accurate classifier, and so these scenarios are not of great interest from a budgeted learning point of view.

2.2 Mapping to budgeted learning

As mentioned in Section 1, active model selection is highly related to budgeted learning because it mimics the pure exploration phase (i.e., purchasing features of labelled training data), followed by the one-time reward phase (i.e., the classification accuracy of the final learned classifier). In addition to this relationship, optimal active model selection is equivalent to optimal budgeted learning of a depth-one decision tree, albeit with some rather strict assumptions. Specifically, given a binary class Y and n candidate features $\{R_i\}_{i=1..n}$ for a classification task, assume $P(R_i = 1|Y = 0) = 0 \ \forall i$. Then the best feature to use as a depth-one decision tree is: $\arg\max_{R_i} P(R_i = 1|Y = 1)$. Set coin C_i to be feature R_i, X_i to be $P(R_i = 1|Y = 1)$, and let flipping coin C_i be equivalent to purchasing feature R_i on a random $Y = 1$ instance. Then a policy π^* that maximizes the expected head probability of the chosen coin (Equation 3) also maximizes the expected accuracy of the chosen depth-one decision tree.

2.3 Standard Policies

Previous research has considered some simple policies for active model selection.

Round Robin (RR). The most intuitive algorithm is to allocate flips evenly over the coins, proceeding in a round-robin fashion. When $b = n \times k$ for an integer k, and all coins have unit cost, RR will flip each of the n coins k times. Despite its fair distribution of flips, the ratio of RR's expected regret to the expected regret of the optimal policy can be made arbitrarily large [11]. Fortunately, more effective policies than RR are known.

Biased Robin (BR). The BR algorithm repeatedly flips a coin C_i until a tail outcome occurs. Once a tail is observed, BR moves to the next coin, C_{i+1}, and repeats the process. (Of course when the last coin turns up tails, BR moves back to the first coin). This simple algorithm is well known in statistics as "play the winner" [14] and has been previously studied as a sampling method for clinical trials [6]. Its performance on the coins problem has been very strong in the case of identical starting priors. Despite its competitive performance, BR is a suboptimal policy. In fact, we can show that the number of suboptimal decisions made by BR can be made arbitrarily large:

PROPOSITION 1. *Given any positive integer $K \geq 1$, there exists a problem with $n = (K+2)$ $Beta(1,1)$ coins, and budget $b = (2n+3)$ such that the BR policy takes a suboptimal action at least K times.*

Single Coin Lookahead (SCL). The SCL algorithm computes the expected regret (Equation 2) of the policy that devotes *all* remaining flips in the budget to a single coin C_i. Whichever coin yields the policy with lowest expected regret is flipped *once*, and then SCL repeats the previous calculation with its reduced budget to choose the next coin. Like BR, SCL has strong performance, but is still suboptimal. In particular, SCL suffers in situations where multiple coins must interact heavily to produce the optimal policy. This occurs because SCL computes a score for coin C_i without considering how the remaining $n-1$ coins could interact with C_i to improve its policy. These deficiencies in the simple strategies offered by BR and SCL motivate the need for a more robust policy.

3. THE MDP FRAMEWORK

The active model selection problem is a finite-horizon Markov Decision Process [15] consisting of a set of states S, a set of actions A, a reward function R, and a transition function T. Specifically, we identify a state $s \in S$ of the MDP by the remaining budget b', and by the collection of Beta hyperparameters over the coins. That is, a state is a $2n+1$ element vector of the form: $\langle\, b', \alpha_1, \beta_1, \ldots, \alpha_n, \beta_n \,\rangle$. The complete set of reachable states corresponds to all the possible posterior Beta distributions that can occur over the n coins by spending some portion x of the original budget b, with $x \leq b$. Since no more actions can be taken once the budget is exhausted, the terminal states are those in which $b' = 0$. The actions of the MDP correspond to the n different coins that can be flipped, where action $a_i \in A$ corresponds to flipping coin i.

The reward function $R(s, a, s')$ specifies the reward of taking action a from state s and reaching state s'. In the coins problem, the reward received in any non-terminal state (i.e., where the remaining budget is positive) is zero, while the reward at a terminal state is the regret of choosing a coin C^* that might not be the best: Regret(C^*). Since our goal is to minimize the expected regret, it makes more sense to think of the rewards as costs (to be minimized) in our context.

In many MDPs, the reward at future time steps is valued less than immediate reward, and so a discount factor $\gamma \leq 1$ is used to multiply future rewards to reduce their value. In the coins problem, future rewards are no less valued than immediate rewards (in fact the *only* reward that matters is the one received on the last time step), and so we have $\gamma = 1$ in our MDP formulation.

Finally, the transition function $T(s, a, s')$ specifies the probability of reaching state s' after taking action a from state s. In our Bayesian formulation, $T(s, a, s')$ can be conveniently computed using the mean of the Beta distributions over the coins. For example, $T(Beta(4,2), C_i, Beta(5,2))$ is calculated as the expected probability of coin C_i turning up heads: $E(X_i) = 4/6$. As the transition function specifies probabilities, we often use $P(s, a, s')$ in place of $T(s, a, s')$.

3.1 The Optimal Policy

Given the MDP formulation, there are a number of techniques that can be used to solve for the optimal policy exactly [17]. For example, a bottom-up dynamic program can use the Bellman optimality equation to learn V^{π^*}, the expected value of each state under an optimal policy:

$$V^{\pi^*}(s) = \max_a \sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V^{\pi^*}(s')] \quad (4)$$

Beginning at the end states in which $b' = 0$ and performing

Table 1: Feature sets used for approximating the value function

Set #	Features Groups Included In Set
1	Budget, Beta Hyperparameters
2	Budget, Means and Standard Deviations
3	Budget, Confidence Interval Stats
4	Budget, Mean Stats, Confidence Interval Stats
5	Budget, Lookahead Stats, Confidence Interval Stats

a backward sweep toward the initial state $b' = b$, the optimal value function can be completely determined. With the known transition and reward functions, the optimal policy π^* then follows immediately via greedy one-step lookahead. Unfortunately, the state space of active model selection grows exponentially with b and n, making it intractable to compute the optimal policy using exact methods such as dynamic programming. This leads to the natural alternative: approximate dynamic programming via reinforcement learning.

3.2 Applying RL to Active Model Selection

The MDP formulation coupled with the known environment dynamics (i.e., the transition and reward functions) motivates a reinforcement learning approach to active model selection. Repeated episodes of the MDP can be simulated, with an RL agent exploring the space of actions, observing the resulting rewards, and gradually learning a policy to minimize the expected regret.

For our RL investigation, we use tile-coding [17] as our function approximation method and combine it with a temporal difference (TD) learner [16] using an epsilon-greedy policy. In general the combination of epsilon-greedy TD(λ) and linear tile-coding is attractive because there are strong guarantees that the mean squared error between the approximate learned value function and the true value function will be near minimal [18]. As with any RL-agent, the number of free variables that must be manually set for a TD(λ) tile-coding agent is extensive. The large number of degrees of freedom makes any exhaustive investigation of the parameters infeasible. Still, in our experiments, we explored a wide range of points in the parameter space, including various values for the learning step-size (α), the probability of exploration (ϵ), and the weight of n-step backups (λ).

For function approximation, we collected the obvious features (e.g. the Beta hyperparameters, the remaining budget, the means and standard deviations of the coins), along with some more subtle attributes (e.g. confidence intervals, budget based confidence intervals, variation among the coins, odds ratios). Although our experiments tested numerous combinations of features, we focus here on five feature sets that are representative of the general trends we observed. For each one of the five sets, Table 1 gives the names of the different *feature groups* that are included in the set. (The interested reader should refer to the appendix to see exactly which *features* are included in each *feature group*.) For our experiments of the next section, we trained five different RL agents, where each agent used one of the five feature sets for its function approximation.

Table 2: Expected regret of various policies

Policy	(n=5, b=15)	(n=8, b=16)	(n=10, b=20)
BR	0.05669	0.07544	**0.07210**
SCL	**0.05413**	**0.07342**	0.07211
RL(set1)	0.05747	0.07830	0.07473
RL(set2)	0.05791	0.07896	0.07390
RL(set3)	0.05555	0.07528	0.07385
RL(set4)	0.05545	0.07464	0.07248
RL(set5)	0.05537	0.07507	0.07280

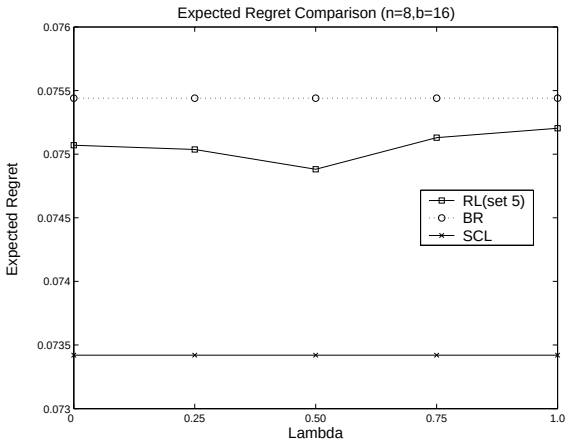


Figure 1: Various values of lambda – BR and SCL still superior to RL

4. EMPIRICAL RESULTS

We conducted experiments on three problems of increasing difficultly, where each initial coin prior was a uniform Beta(1, 1). For each experiment, the expected regret (Equation 2) was calculated for BR, SCL, and the five different policies learned using a TD(0) RL agent. The number of training episodes for the RL agents was set to 1.8 million for the two smaller problems and to 2.8 million for the larger problem. The results are shown in Table 2.

The results indicate that for all problems considered, either BR or SCL produced the smallest expected regret. In fact, no RL policy is able to beat either of the simple policies in the case of ten coins and a budget of twenty, and no RL policy is able to beat SCL on *any* of the problems. We have observed that on larger problems (e.g. ten coins and a budget of thirty), BR beats SCL and RL policies easily. The results of the experiments reveal that despite the extensive number of states observed during training, the RL policies are not generalizing well enough between states to beat the simpler policies.

Additional experiments were run on the $n = 8$, $b = 16$ problem with different values of λ for the TD(λ) learner. For example, Figure 1 shows the results of varying λ when using the fifth set of features for function approximation. For all values of λ considered, the policies learned by RL do only slightly better than BR and are inferior to SCL. The difference between the various TD(λ) learners is not dramatic, but the expected regret is lowest with an intermediate value of $\lambda = 0.5$.

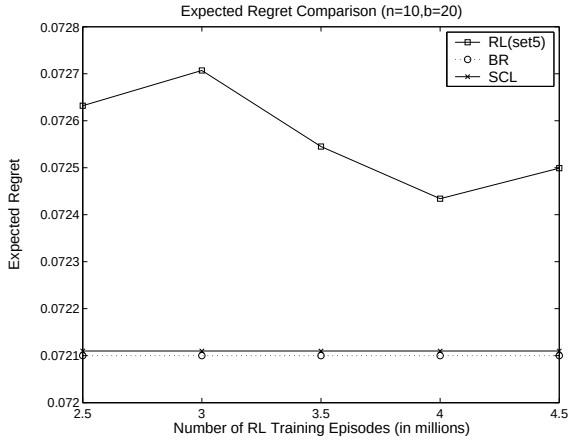


Figure 2: Various amounts of training, RL still exhibits lower performance than simple policies

Table 3: Resources used by each policy on n=10, b=20

Policy	Training time (mins)	Memory Used (MB)
BR	0	0
SCL	0	0
RL(set 5)	630	760

A possible explanation for the lower performance of RL is that not enough training episodes are being experienced. Additional training should permit an RL agent to increase its exploration of the state space, and yield a better policy. To test the effect of increased training, we conducted experiments on the $n = 10, b = 20$ problem in which we varied the number of training episodes from two and a half million up to an even more generous four and a half million. Learning took place with a TD(0.5) learner, using one of the strongest RL feature sets we tested, set number five. The downward sloping trend of Figure 2 suggests that increased training does improve the resulting policy; however, even after four million episodes, the expected regret of the RL policy is still larger than BR and SCL.

For further comparison, we consider the training time and memory required by BR, SCL, and the RL policy after four and half million training episodes. The memory considered is only the policy specific storage (i.e., above and beyond the basic elements such as the beta hyperparameters and the budget that is generally required by all policies). Examining Table 3, we see that even using almost 800 MB of main memory, RL does not gain a significant advantage over the virtually memoryless BR and SCL routines.

As these experiments show, the performance, speed, and low memory requirements make the simper BR and SCL policies preferable to the use of reinforcement learning. Although it should be possible for an RL agent to do better than these heuristic policies, the experimental results indicate that (at least) more cleverly designed features or a better type of function approximator will be required to achieve this.

Perhaps the clearest argument *against* using RL for active model selection (and hence general budgeted learning) is the opportunity cost of conducting the necessary training. That is, although experience is easy to generate, the time and memory used to train RL methods could be equally well spent running a bottom-up dynamic program (as in Section 3.1) that solves for the optimal value of each state. The dynamic program could compute the optimal policy from some select set of states in the same amount of time it takes a reinforcement learning agent to complete training. In effect, the optimal values from this select set of states could be easily combined with the BR or SCL policies to lower their regret even further, and make it yet more difficult for RL methods to compete with these heuristic policies.

Overall, the poor performance demonstrated by RL methods in our experiments suggests that when considering the larger and much more complicated problem of general budgeted learning, the MDP formulation should be avoided, and one should focus on more tractable heuristic policies that have been shown to work effectively [7, 9].

5. RELATED WORK

Active model selection was originally introduced in [10], although several similar problems have been previously studied. The well-known multi-armed bandit problem [14] is concerned with finding the best object among a set, but rewards are typically accrued throughout, without distinguishing training from testing phases. By contrast, active model selection gives no reward until the final coin is selected, and thus more accurately represents the pure training phase of budgeted learning. Strategies from the adversarial bandit formulation [1] could also be adopted for our problem, but the adversarial assumption is unnecessarily strong for our case, and thus less defensive algorithms can usually perform better on active model selection. A more recent bandit-variant, the max k-arm bandit [3], shares our notion of maximizing a *single* reward over the duration of the MDP. However, [3] allows the single reward to occur on any time step, as opposed to strictly at the terminal states.

Duff [5] studied the Bayesian MDP formulation in active model selection, as a Bayes Adaptive Markov Decision Process (BAMDP). That study also considers various RL methods to approximate an optimal policy for BAMDPs, and chooses some of the same types of features for function approximation that we consider in this work. Moreover, the experimental results concur with our findings, as [5] also reports a gap between the reward of the learned RL policies and the optimal policy. Besides RL, another potential strategy for active model selection is online sparse lookahead [19, 8]. Unfortunately, given the size of the state space, we have found that any tractable (truncated) lookahead usually yields a higher regret than the simple BR and SCL policies.

Numerous results from Machine Learning are related to the coins problem, as they too are concerned with the notion of costs at training time. There are too many works to mention here, but some relevant examples include budgeted learning [7, 9], active learning [4], active feature value acquisition [12], and progressive sampling [13]. The field of experimental design [2] is also related, as it deals with how to make finite allocations among objects.

6. CONCLUSIONS AND FUTURE WORK

Despite their known shortcomings, the Biased Robin and Single Coin Lookahead strategies yield low expected regret, and thus an interesting open problem is determining their approximability characteristics. Since features will often have different costs in the general budgeted learning problem, another avenue for future work involves removing the assumption that coins have identical costs, and finding effective strategies in this context. Finally, the fact that the optimal solution can be computed for small versions of the active model selection problem may be useful in deriving an approximation algorithm. One possibility worth exploring is to use careful abstractions to transform large problems into a more tractable, solvable size.

Contributions: This paper investigates the use of reinforcement learning to develop policies to address the active model selection task. We describe deficiencies in the best existing policies, which motivate the need for a more robust solution. We extensively train multiple RL agents using different feature sets for function approximation. Our experiments, on various size problems, demonstrate that the simple policies are able to achieve lower regret with far less computation than the learned RL policies. These results are most helpful when considering approaches for the general problem of budgeted learning. Specifically, in the absence of better features for function approximation, we recommend *not* applying RL techniques to the higher dimensional and more difficult budgeted learning problem, as we anticipate they will prove ineffective.

Acknowledgments

Both authors wish to thank NSERC and iCORE for their generous support. We also thank Dan Lizotte and Omid Madani for insights on various related problems, and the anonymous reviewers for their comments. Russell Greiner also thanks the Alberta Ingenuity Centre for Machine Learning.

7. REFERENCES

[1] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 1995.

[2] K. Chaloner and I. Verdinelli. Bayesian experimental design: A review. *Statistical Science*, 1995.

[3] V. A. Cicirello and S. F. Smith. The max k-armed bandit: a new model of exploration applied to search heuristic selection. In *AAAI*, 2005.

[4] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. In *Advances in Neural Information Processing Systems*, 1995.

[5] M. Duff. *Optimal learning: computational procedures for Bayes-adaptive Markov Decision Processes.* PhD thesis, University of Massachusetts Amherst, 2002.

[6] D. Hoel and M. Sobel. Comparisons of sequential procedures for selecting the best binomial population. In *Sixth Berkeley Symposium on Mathematical Statistics and Probability*, 1971.

[7] A. Kapoor and R. Greiner. Learning and classifying under hard budgets. In *European Conference on Machine Learning*, 2005.

[8] M. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 2002.

[9] D. J. Lizotte, O. Madani, and R. Greiner. Budgeted learning of naives-bayes classifiers. In *Proceedings of Uncertainty In Artificial Intelligence*, 2003.

[10] O. Madani, D. J. Lizotte, and R. Greiner. Active model selection. In *Proceedings of Uncertainty in Artificial Intelligence*, 2004.

[11] O. Madani, D. J. Lizotte, and R. Greiner. Active model selection. Technical report, University of Alberta, 2004.

[12] P. Melville, M. Saar-Tsechansky, F. Provost, and R. Mooney. Active feature-value acquisition for classifier induction. In *ICDM*, 2004.

[13] F. Provost, D. Jensen, and T. Oates. Efficient progressive sampling. In *International Knowledge Discovery and Data Mining Conference*, 1999.

[14] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 1952.

[15] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2002.

[16] R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 1988.

[17] R. S. Sutton and A. G. Barto. *Reinforcement Learning.* The MIT Press, 1998.

[18] J. N. Tsitsiklis and B. V. Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 1997.

[19] T. Wang, D. Lizotte, M. Bowling, and D. Schuurmans. Bayesian sparse sampling for on-line reward optimization. In *International Conference on Machine Learning*, 2005.

APPENDIX

A. PROPOSITION ONE

PROOF. Proposition 1. There are numerous ways to prove the proposition; we use a simple subproblem to obtain the result. Consider a state, Q, in which $b' = 1$, there exists two Beta$(3, 2)$ coins, and $(n-2)$ Beta$(2, 2)$ coins. It is easy to verify (using Equation 3) that the optimal action in Q is strictly to flip a Beta$(3, 2)$ coin. To prove the proposition, we show that BR encounters at least K different variants of Q in which it chooses to flip a Beta$(2, 2)$ coin.

Let there be $n = K + 2$ coins, and a budget of $b = 2n + 3$. Notice the budget is such that state Q is guaranteed to occur under BR's strategy. In fact, Q occurs multiple times because there are $\binom{n}{2}$ distinct ways to place the two Beta$(3, 2)$ coins. We also note that since the number of tails on all n coins is equal, we are guaranteed that BR will be currently flipping the first coin in the set. Thus, BR will make a suboptimal decision whenever it reaches state Q with the first coin being one of the Beta$(2, 2)$s. Observe that there are $\binom{n-1}{2}$ distinct versions of state Q in which the first coin is a Beta$(2, 2)$. Now the proposition follows from the fact that: $\binom{n-1}{2} = \frac{(n-1)(n-2)}{2} = \frac{(K+1)K}{2} \geq K$

for all $K \geq 1$. $\square$

B. FEATURE GROUPS

Budget

- remaining budget (b')

Beta Hyper Parameters

- $\alpha_i \quad \forall i = 1..n$
- $\beta_i \quad \forall i = 1..n$

Means and Standard Deviations

- $\mu_i \quad \forall i = 1..n$
- $\sigma_i \quad \forall i = 1..n$

Mean Stats

- $\max_i \mu_i$
- $\min_i \mu_i$
- $\sum_i \mu_i$

Lookahead Stats

- $\max_i \frac{\alpha_i + b'}{\alpha_i + \beta_i + b'}$
- $\frac{\sum_i \left(\frac{\alpha_i + b'}{\alpha_i + \beta_i + b'} \right)}{n}$

Confidence Interval Stats

- $\max_i (\mu_i + 1.96\sigma_i)$ (95% interval)
- $\max_i (\mu_i + 1.28\sigma_i)$ (80% interval)
- $\max_i (\mu_i + 0.67\sigma_i)$ (50% interval)
- $\max_i (\mu_i + 0.126\sigma_i)$ (10% interval)
- $\sum_i (\mu_i + 1.96\sigma_i)$
- $\sum_i (\mu_i + 0.126\sigma_i)$
- $\max_i (\mu_i + b' \times \sigma_i)$
- $\sum_i (\mu_i + b' \times \sigma_i)$

Wrapper-based Computation and Evaluation of Sampling Methods for Imbalanced Datasets

Nitesh V. Chawla
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN
nchawla@cse.nd.edu

Lawrence O. Hall
Department of Computer Science and Engineering
University of South Florida
Tampa, FL
hall@cse.usf.edu

Ajay Joshi
Department of Computer Science and Engineering
University of South Florida
Tampa, FL
ajoshi2@cse.usf.edu

ABSTRACT

Learning from imbalanced datasets presents an interesting problem both from modeling and economy standpoints. When the imbalance is large, classification accuracy on the smaller class(es) tends to be lower. In particular, when a class is of great interest but occurs relatively rarely such as cases of fraud, instances of disease, and regions of interest in large-scale simulations, it is important to accurately identify it. It then becomes more costly to misclassify the interesting class. In this paper, we implement a wrapper approach that computes the amount of under-sampling and synthetic generation of the minority class examples (SMOTE) to improve minority class accuracy. The *f-value* serves as the evaluation function. Experimental results show the wrapper approach is effective in optimization of the composite *f-value*, and reduces the average cost per test example for the datasets considered. We report both average cost per test example and the cost curves in the paper. The true positive rate of the minority class increases significantly without causing a significant change in the *f-value*. We also obtain the lowest cost per test example, compared to any result we are aware of for the KDD Cup-99 intrusion detection data set.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning - Induction; H.2.8 [**Database Management**]: Applications - Data Mining

General Terms

Algorithms, Performance, Design, Experimentation

Keywords

cost-sensitive learning and evaluation, imbalanced datasets, wrapper, under-sampling, SMOTE

1. INTRODUCTION

In this work, we focus on the problem of learning a classification model from imbalanced data sets. An imbalanced data set is one in which there is a significant difference in the number of examples in a set of classes. The imbalanced datasets pose an economic or utility problem, as there is usually a higher cost in misclassifying the interesting class. A simple consistent guess can become an accurate classifier, by classifying everything as the majority class, but that is not useful for the problem at hand. On the other hand, a simple guess classifying everything as the interesting class will also not work due to the number of false positives. One wants a high number of true positives, while maintaining a low false-positive rate.

There are many examples of imbalanced data sets where the minority class is of interest. For example, cellular-phone fraud or credit card fraud data are typically comprised of a very small proportion of the the fraudlent cases (minority class) [19, 29, 11]. However, it is quite important to predict a fraudulent transaction. It is also important to minimize the false positives (the nonfraudulent transactions that are predicted to be fraudulent) because these cost time to investigate and can potentially upset the customer. Thus, there is a non-zero cost associated with the false positives as well. Typically, the cost with the false negatives will be the cost of the transaction. We don't want a system that will strongly target true positives at the expense of a high false positive rate, thereby increasing the total cost of the operation.

As another example, large-scale simulations can be based on extremely large data sets. Some simulations are replacing or augmenting physical experiments. This requires that they be done in great detail [26, 9]. However, the process of building very large-scale simulations and examining them for correctness when looking for important, subtle details may prevent all areas of interest from being viewed [6]. In any event, the process of validating a simulation can take weeks to months. A similar amount of time is required to actually utilize and explore the simulation. This is indicative of the great opportunity for building intelligent tools which can help the simulation designers/users find regions of interest and/or anomalies quickly. There is a cost involved not only in correctly displaying the regions of interest but also the costs in time. Hence, the intelligent tool should not only be "fast", but also accurately identify the interesting regions, without too many false alarms. Having many false alarms for the user to browse through can inadvertently increase the cost in terms of the time spent. As these two examples highlight, there is a "utility" associated with the usage of

UBDM'05 August 21, 2005, Chicago, Illinois, USA.

a technique. That utility is comprised of various costs of errors, time spent, etc.

We investigate an enhancement to a particular composite approach, combining over-sampling by creating new examples and under-sampling, for dealing with imbalanced data sets. The Synthetic Minority Oversampling TEchnique (SMOTE) creates synthetic examples from minority classes [14]. We also under-sample the majority class(es) to obtain higher accuracy on the minority class(es) without greatly increasing the number of false positives. However, previous work has not shown how to effectively set the amounts of under-sampling and SMOTE for a given dataset. In this paper, we explore an automated method to do this. We set up the method such that the *f-value* [10] is optimized. We chose the *f-value* as it is a composite measure that incorporates both the false positives and false negatives. Hence, if an approach significantly increases the true positives, but also increases the false positives, then the *f-value* will appropriately reflect that. We evaluate the final performance of the classifiers under a cost-based framework using cost-curves and average cost per test example.

It is important to identify the potentially optimal under-sampling and SMOTE percentages. The amount of sampling performed to mitigate the imbalance in class distribution will have an effect on the performance of the classifier. We want to reduce the costs per test example. The utility of the learning algorithm for a particular domain or task is strongly dependent on the right amount of sampling and the examples distribution in the dataset. Each dataset and the corresponding class distribution will have its own requirements [32]. The computational time and resources spent deploying the wrapper technique should be mitigated by the reduced cost per test example, and a higher detection of the interesting class or regions in the dataset. There is a tradeoff between the time spent in learning or searching for the parameters, and the relative reduction in the costs or improvement in the true positives on the testing set. We will show that minority class accuracy is improved on several data sets with only small increases in false positive predictions. In addition, we will also show that our approach produces much reduced costs per test example. The approach is shown to be both tractable computationally and effective in choosing the parameters.

2. LEARNING FROM IMBALANCED DATASETS

Researchers in the machine learning community have dealt with the problem of class imbalance by using various approaches like over-sampling the minority classes, undersampling the majority classes, assigning different costs for different misclassification errors, learning by recognition as opposed to discrimination, etc [3, 25, 27, 32, 4, 1, 24, 2, 34]. There is a significant body of research comparing the various sampling methods [12, 28, 17, 22, 7]. Sampling strategies have almost become the de facto standard for countering the imbalance in datasets [13]. With all this there is still no answer on how to do the sampling required for obtaining good classifier accuracies on minority classes.

There are a number of different approaches that can be applied to build classifiers on imbalanced data sets. In this work, we examined under sampling and over-sampling by creating synthetic examples of minority classes. Under-sampling the majority class can reduce the bias of the learned classifier towards it and thus improve the accuracy on the minority classes.

Some studies [27, 21] have been done which combined under-sampling of majority classes with over sampling by replication of minority classes. While Japkowicz [21] found this approach very effective, Ling and Li [27] were not able to get significant improvement in their performance measures. Japkowicz experimented with only one-dimensional artificial data of varying complexity whereas Ling and Li used real data from a Direct Marketing problem. This might have been the reason for the discrepancy between their results. On the whole, from the body of literature, it was found that under-sampling of majority classes was better than over-sampling with replication of minority classes [17, 12] and that the combination of the two did not significantly improve the performance over under sampling alone.

Chawla et al. [14] introduced a new over-sampling approach for two class problems that over-sampled the minority class by creating synthetic examples rather than replicating examples. They pointed out the limitation of over-sampling with replication in terms of the decision regions in feature space for decision trees. They showed that as the minority class was over sampled by increasing amounts, for decision trees, the result was to identify similar but more specific regions in the feature space. A preferable approach is to build generalized regions around minority class examples.

The synthetic minority over-sampling technique (SMOTE) was introduced to provide synthetic minority class examples which were not identical but came from the same region in feature space. The over-sampling was done by selecting each minority class example and creating a synthetic example along the line segment joining the selected example and any/all of the k minority class nearest neighbors. In the calculations of the nearest neighbors for the minority class examples a Euclidean distance for continuous features and the value Distance Metric (with the Euclidean assumption) for nominal features was used. For examples with continuous features, the synthetic examples are generated by taking the difference between the feature vectors of selected examples under consideration and their nearest neighbors. The difference between the feature vectors is multiplied by a random number between 0 and 1 and then added to the feature vector of the example under consideration to get a new synthetic example. For nominal valued features, a majority vote for the feature value is taken between the example under consideration and its k nearest neighbors. This approach effectively selects a random point along the line segment between the two feature vectors. This strategy forces the decision regions of the minority class learned by the classifier to become more general and effectively provides better generalization performance on unseen data.

However, an investigation into how to choose the number of examples to be added was not done. In addition, the amount of under-sampling also needs to be determined. Given the various costs of making errors, it is important to identify potentially optimal values for both SMOTE and under-sampling. This is equivalent to discovering the operating point in the ROC space giving the best trade-off between True Positives and False Positives. In this paper, we develop an approach to automatically set the parameters. We discuss a wrapper framework using cross-validation that

performs a step-wise and greedy search for the parameters. Note that while the computational aspects of the automated approach induces certain costs, we do not incorporate that into our framework. We optimize based on the different types of errors made. However, we do try to restrict our search space. We show that this approach works on three highly skewed datasets. We also utilized a cost-matrix to indicate the costs per test example based on the different kinds of errors.

3. WRAPPER

In this work, a wrapper [23] approach was utilized to determine the percentage of minority class examples to add to the training set and the percentage to under-sample the majority class examples. The wrapper approach works by doing a guided search of the parameter space. In this case the underlying classifier is used to evaluate the chosen performance function for every considered amount of under-sampling and SMOTE. A particular parameter or set of parameters is chosen and a five-fold cross validation on the train data is done to get the performance average. The parameters are varied in a systematic way such that a set of parameter candidates are generated, training sets are updated, and the classifiers built and evaluated. The candidate associated with the highest performance is chosen to have its parameters systematically modified to create new candidate solutions. This process is a type of best-first search. In order to evaluate the effectiveness of the wrapper approach in selecting the parameters for under-sampling and SMOTE, we need to use a metric other than strict accuracy. With imbalanced data, accuracy can be misleading, because it causes you to favor high prediction accuracy on the majority class which is often uninteresting. Hence, the *f-value* metric was used as the evaluation function [10]. It is made up of two measures: *precision* which gives us the measure of correctness of the classifier in predicting the actual positive or minority class, whereas *recall* gives us the measure of the percentage of positive or minority class examples predicted correctly. The precision, recall and *f-value* were calculated as follows, where β corresponds to the relative importance of *precision* vs *recall*.

$$precision = \frac{TP}{TP + FP} \tag{1}$$

$$recall = \frac{TP}{TP + FN} \tag{2}$$

$$f - value = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times recall + precision} \tag{3}$$

We implemented our wrapper approach as follows. We first do a ten-fold stratified split to separate the original dataset into ten training sets and ten disjoint testing sets. Then, for each of the ten training folds, we implement the wrapper approach using five-fold cross-validation to get more robust amount estimates for under-sampling and SMOTE. Note that these performance estimates will hold true only when either the training data is a good representative of the actual data distribution or the wrapper strategy does not over-fit the training data. If the training data is not a good representative of the actual data, no strategy can help. So the only thing which remains is to see whether the wrapper approach finds under-sampling and SMOTE levels which when used to build a classifier, do not over-fit the training data. Once the wrapper selects the particular amount of SMOTE and under-sampling, we apply those amounts five different random times on the training set, since both SMOTE and under-sampling randomly remove or create new instances. The classifiers learned from the updated training sets are evaluated on the same testing set, and those performances are averaged. This is done for each of the 10 folds. Thus, the final ten-fold average reported is essentially over fifty classifiers.

The two search parameters for the wrapper are the under-sampling and SMOTE percentages. The search space becomes large if the search is done simultaneously for both the under-sampling percentage and the SMOTE percentage (creation of new synthetic examples). Hence, we chose to first use wrappers to find the best under-sampling percentage. The wrapper starts with no under sampling for all majority classes and obtains baseline results on the training data. Then in a step-by-step greedy fashion it traverses through the search space of under-sampling percentages to seek better performance over the minority classes. The search process continues as long as it does not reduce the *f-value* of the minority classes or reduce the *f-value* over the majority classes more than some specified amount (generally 5%). Note that for under-sampling we look at both the minority and majority class *f-values*. We also looked at the *f-value* for the majority class as we only want to remove the redundant examples through undersampling, and not remove some of the important majority class examples. By looking at both the values simultaneously we are maintaining the decision regions for all the classes. Also, we wanted to identify the amount of under-sampling before introducing any synthetic minority class examples as that could have inadvertently penalized the *f-value* for the majority class. We want to first remove the majority class examples, that add no learning value to the base classifier.

Then with the under-sampling percentage fixed, we used the wrapper approach, to find the SMOTE percentage. Oversampling by creating synthetic examples is done until no minority class *f-value* increase is obtained for 3 candidate expansions. Now, for SMOTE we are only interested in improving the performance of the minority class. The *f-value* takes into account the increase in false positives (lowered precision), if any, by SMOTE increments. Thus, an overwhelming increase in the precision will stop the SMOTE process. This provides significantly improved computation times at the cost of a potential loss in accuracy. Once the best percentages for under-sampling and over-sampling via SMOTE are found, the training folds are updated with the requisite SMOTE and undersampling amounts. A classifier is then learned and evaluated on the unseen test data. We would like to be able to put this in a cost-framework if the time spent in searching for the "optimal" and "best" under-sampling and SMOTE percentages, justifies the performance improvement. We are investigating that line of work, as future work.

4. EXPERIMENTS

We report results on three data sets:

- Mammography Dataset,
- Forest Cover Dataset, and

Table 1: Summary of Datasets. The percentages indicate the proportion of minority class in the complete dataset.

Dataset	# of Examples	# classes	# of Majority class examples	# of Minority class examples	# of attributes	# of continuous attributes
Mammography	11183	2	10923	260 (2.3%)	6	6
Forest cover	38501	2	35754	2747 (7.13%)	54	54
Modified KDD cup 99 (intrusion data)	69980	5	Normal: 35000; Dos: 25988; Probe: 4813	U2R: 267 (0.41%); R2L: 3912 (5.95%)	41	34

- KDD-cup 99: Network Intrusion Detection Dataset (two versions).

A brief summary of the datasets is presented in Table 1 and further details are given in later subsections. The Forest Cover dataset is available from the UCI repository [8] and our modifications to it will be described in the proceeding. The network intrusion data set comes from the KDD cup competition in 1999 [20] and the mammography data set is one that we locally extracted [33]. It is clear from Table 1 that there is significant imbalance between the two classes of each of these data sets. Hence, there is an opportunity to improve the minority class recognition accuracy because a typical classifier will be highly accurate but focused on the majority class. We report the *f-value* for all our experiments. The *f-value* assumed a β of 1. We introduced a cost-matrix for the mammography dataset, as there can be a large cost associated with misclassification of a potentially malignant calcification (cancer) as non-calcification (non-cancerous). Moreover, there is also a slight cost associated with misclassifying the non-calcifications as calcifications. While there wasn't a natural application of costs to the forest cover dataset, we still constructed a cost-matrix for the sake of analysis. The KDD-cup dataset comes with a cost-matrix for each of the relative type of errors.

However, we did not incorporate the cost-matrix during the validation stage to select the amount of SMOTE and under-sampling. We are going to investigate that as a future line of work. It requires a definite cost matrix to be known for a dataset. It will be interesting to compare the SMOTE and undersampling parameters discovered using cost matrices during validation with the SMOTE and undersampling parameters discovered without using the cost-matrices (assuming the same loss).

4.1 Classifiers

Experiments were done with two types of classifiers, decision trees using software (USFC4.5) which emulates C4.5 release 8 [30] and a rule learning technique called RIPPER [15]. USFC4.5 was used with the default settings. By default, RIPPER will build rules first for the smallest class and will not build rules for the largest class. In the case of two class problems with imbalanced classes, such as here, only rules for the minority class are going to be built. Hence, one might expect that RIPPER will be better than a decision tree in accuracy on the minority class.

The wrapper algorithm that uses five fold cross-validation on the training set finds the undersampling and SMOTE percentages for a particular training fold (one of the ten folds for cross-validating the system). Then under-sampling and SMOTE are applied to each fold with wrapper selected percentages, a classifier was built on the updated training data and evaluated on the test data, unseen during the wrapper process. Due to the inherent random nature of under-sampling and SMOTE, the process of training and testing with wrapper selected under-sampling and SMOTE percentages is done five times to get an averaged (more stable) performance measure. To summarize, on each of the 10 folds, training and testing for wrapper selected SMOTE and under-sampling percentages was done five times i.e. SMOTE and under-sampling was done for a total of 50 times for cross-validation to get average stable results. All results reported in the proceeding are averages obtained in this way. In the tables, t-stat indicates the results of a significance test at the 95% level. This was a paired t-test. The x% of under-sampling means that x% of majority class examples were retained; and the y% of SMOTE means that many more examples of the minority class were created. For example, 200% of SMOTE means that twice as many (than the original number) minority class examples were created.

5. RESULTS

We did a ten-fold cross-validation, for mammography and forest cover datasets, in which the original dataset is stratified into ten disjoint sets or folds from which ten distinct testing sets and ten training sets are created. For the intrusion dataset, we utilized the training and testing sets as provided. We also used the cost-matrix as provided for the intrusion dataset and report the average cost per test example to compare with other published results [18]. For the mammography and forest cover datasets, we report various perfomance metrics, including *TPrate*, *FPrate*, *f-values*, average cost per test example at different cost ratios, and cost curves. Our main goal is to compare the classifiers in terms of reduction in the expected cost across different cost ratios. Drummond and Holte [16] introduced the cost space representation that allows for comparing different classifiers in terms of the expected cost. Let p(+) be the prior probability of the positive class, and p(-) be the prior probability of the negative class. $C(-|+)$ is cost of misclassifying a positive example as a negative example (false negative); and $C(+|-)$ is cost of misclassifying a negative example as a positive example (false positive). The Normalized Expected Cost (NE[C]) can then be expressed in terms of *TPrate*, *FPrate*, and Probability Cost Function (PCF) as follows:

$$PCF(+) = \frac{p(+)C(-|+)}{p(+)C(-|+) + p(-)C(+|-)} \qquad (4)$$

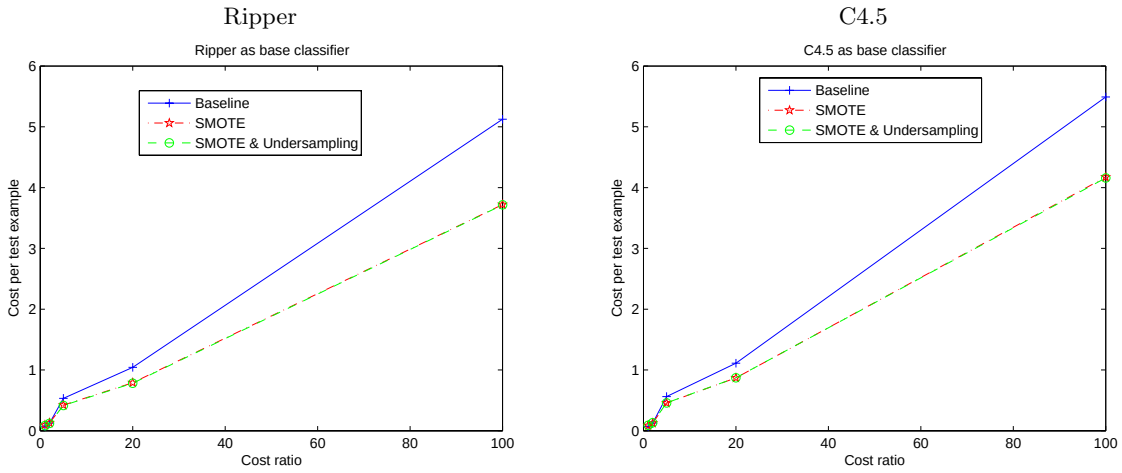


Figure 1: Average Cost per test example at different cost ratios for the Mammography dataset.

$$NE[C] = (1 - TPrate - FPrate) \times PCF(+) + FP \quad (5)$$

The performance of classifier using a fixed threshold, as used in this paper, is represented by a pair of (TP, FP). It can thus be represented as a line in the cost space, comprising of the normalized expected cost ($NE[C]$) in the y-axis and $PCF(+)$ in the x-axis. The range of both the measures is between 0 and 1. Given a family of such classifiers, if a classifier is lower in the normalized expected cost across a range of PCF, it dominates the other. One can, thus, choose a classifier that has a minimum cost either over a range of $PCF(+)$ or at a particular operating range.

5.1 Mammography Data

The Mammography Dataset was used in [33] and consists of 11,183 total samples with six numeric features and two classes representing calcification (cancerous) and non-calcification (non-cancerous). The minority class which represents calcification contained only 260 examples in the dataset i.e. only 2.32% of the total examples. The results obtained are shown in Table 2. The negative sign before the number in the '% increase' row indicates reduction in the associated value. It can be seen from Table 2, that for all four experimental trials, the wrapper algorithm was able to statistically significantly improve TP-rates for the minority class at the expense of a statistically significant reduction in *f-values* for the majority class. However, the wrapper method also produces significantly higher FP-rates than the baseline methods. But, the correspnding decrease in the *f-value* was not significant. Hence, the goal of a higher true positive rate is attainable without a significant reduction in the overall *f-value*.

We then constructed a cost-matrix for the mammography dataset by considering the the following costs of making errors between the positive and negative examples (using the convention $(C(+|-), C(-|+))$: (5, 5); (5, 10); (5, 50); (5, 100); and (5, 500). Figure 1 shows the results using these varied cost ratios with the different methods presented in Table 2. As one would expect, if using the same costs of errors, the baseline method produces the lowest cost per test example, and is indeed the preferred method. However, varying the costs from twice as much for false negative to 100 times, we see that the SMOTE classifier achieves the least cost. Under-sampling in conjunction with SMOTE provides very little reduction in the cost, if any. Both the C4.5 and Ripper classifiers exhibit similar behavior with SMOTE — significant improvement in performance over baseline. Ripper is well-suited for the task, as it is able to produce lower cost estimates per test example. We believe that incorporating a *f-value* in the wrapper framework maintains the relative importance of false positives and false negatives, as the number of true positives increases. However, one might vary the relative importance of precision and recall in the equation based on the specified costs. We assumed uniform costs in the *f-value*.

We also implemented cost-curves over the range of $PCF(+)$ established by varying $C(+|-)$ and $C(-|+)$ [16]. Figure 2 shows the result. Again over the wide range of $PCF(+)$, the Ripper-SMOTE classifier achieves the lowest expected costs. Until a $PCF(+)$ of 0.05 all the classifiers achieve similar performances, but beyond that the Ripper-SMOTE classifier dominates over the others.

5.2 Forest Cover Data

Originally, the Forest Cover dataset [8] consisted of 581,012 examples with 54 numeric features related to cartographic variables and seven classes representing the type of the forest cover. For our study, the data samples from two classes were extracted while the rest were ignored as done in [14]. The two classes we considered are Ponderosa Pine with 35,754 samples and Cottonwood/Willow with 2,747 samples. The results obtained on this dataset are tabulated below in Table 3.

For the Forest cover dataset, the results for the minority class were as expected, with the wrapper TP rate increasing with statistical significance. But the interesting thing about these results was that, the wrapper *f-values* obtained on the majority class using RIPPER in both scenarios actually increased slightly instead of decreasing which was the general trend. For the 'SMOTE only' scenario using RIPPER, the wrapper *f-values* were better than baseline *f-values* with statistical significance. For C4.5, the drop in the wrapper *f-values* over the majority class though statistically significant was extremely small. These were almost perfect results which one might always hope for, where the minority ex-

Table 2: Results for the Mammography Data. $>$ indicates Wrapper is statistically significantly greater than Baseline; $<$ indicates Wrapper is statistically significantly lower than Baseline; and $\equiv$ indicates there is no statistically significant difference between the Wrapper and Baseline methods.

		C4.5		Ripper	
		SMOTE only	Undersampling and SMOTE	SMOTE only	Undersampling and SMOTE
	Average SMOTE %	210%	180%	300%	180%
	Average Under- sampling %	100%	87%	100%	94%
Average Minority Class TP-rate	Baseline	0.546	0.546	0.577	0.577
	Wrapper	0.658	0.659	0.696	0.665
	% increase	16.96%	17.15%	17.13%	13.29%
	t-stat	-4.7	-3.913	-4.276	-3.322
	significance	$>$	$>$	$>$	$>$
Average Minority Class FP-rate	Baseline	0.0031	0.0031	0.00439	0.00439
	Wrapper	0.0088	0.0099	0.0114	0.0099
	% increase	64.58%	68.63%	61.47%	55.96%
	t-stat	-9.626	-6.387	-8.952	-5.461
	significance	$>$	$>$	$>$	$>$
Average Minority Class *f-value*	Baseline	0.644	0.644	0.652	0.652
	Wrapper	0.647	0.634	0.643	0.639
	% increase	0.49%	-1.61%	-1.38%	-1.90%
	t-stat	-0.128	0.398	0.484	0.727
	significance	$\equiv$	$\equiv$	$\equiv$	$\equiv$
Average Majority class *f-value*	Baseline	0.993	0.993	0.993	0.993
	Wrapper	0.992	0.991	0.991	0.991
	% decrease	0.16%	0.21%	0.21%	0.18%
	t-stat	3.678	3.923	5.674	4.224
	significance	$<$	$<$	$<$	$<$

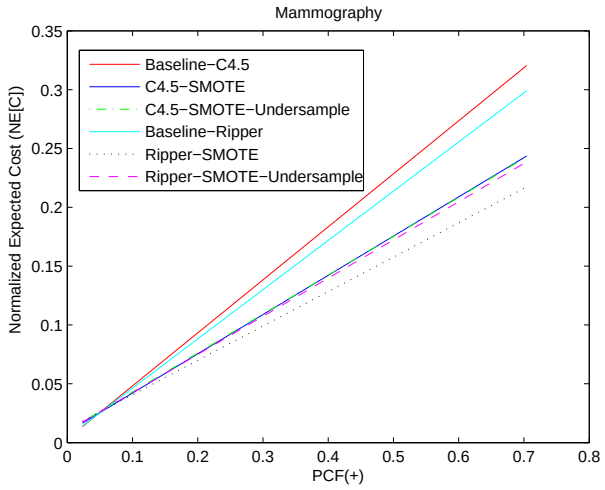


Figure 2: Cost Curve for Mammography dataset.

Table 3: Results for the Forest Cover Data. $>$ indicates Wrapper is statistically significantly greater than Baseline; $<$ indicates Wrapper is statistically significantly lower than Baseline; and $\equiv$ indicates there is no statistically significant difference between the Wrapper and Baseline methods.

		C4.5		Ripper	
		SMOTE only	Undersampling and SMOTE	SMOTE only	Undersampling and SMOTE
	Average SMOTE %	600%	430%	560%	580%
	Average Under- sampling %	100%	99%	100%	93%
Average Minority Class TP-rate	Baseline	0.873	0.873	0.834	0.834
	Wrapper	0.905	0.903	0.900	0.905
	% increase	3.59%	3.28%	7.34%	7.87%
	t-stat	-4.889	-4.223	-7.544	-7.532
	significance	$>$	$>$	$>$	$>$
Average Minority Class FP-rate	Baseline	0.0072	0.0072	0.009	0.009
	Wrapper	0.0109	0.0105	0.0133	0.014
	% increase	33.72%	30.82%	28.42%	32.58%
	t-stat	-10.996	-8.626	-9.507	-5.719
	significance	$>$	$>$	$>$	$>$
Average Minority Class *f-value*	Baseline	0.887	0.887	0.852	0.852
	Wrapper	0.884	0.885	0.868	0.866
	% increase	-0.35%	-0.24%	1.88%	1.69%
	t-stat	0.771	0.549	-3.963	-3.178
	significance	$\equiv$	$\equiv$	$>$	$>$
Average Majority class *f-value*	Baseline	0.992	0.992	0.989	0.989
	Wrapper	0.991	0.991	0.989	0.989
	% decrease	0.06%	0.05%	-0.06%	-0.04%
	t-stat	2.225	1.884	-2.19	-1.087
	significance	$<$	$<$	$>$	$\equiv$

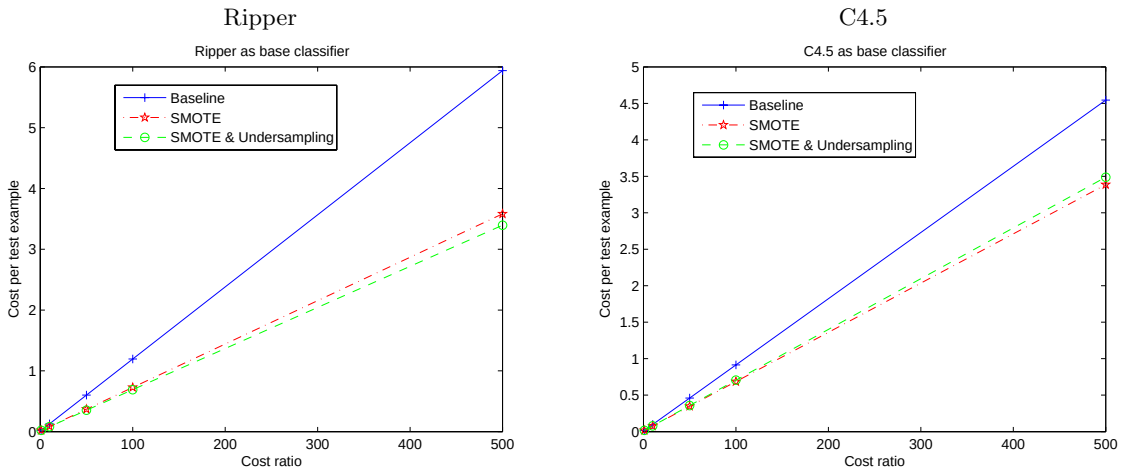


Figure 3: Average Cost per test example at different cost ratios for the Forest Cover dataset.

amples which were previously misclassified were correctly classified without increasing the number of majority class examples being classified as belonging to the minority class. The reason for these good results might be due to the similar distribution of the minority class examples in training and test data when cross-validation is performed. For example, in the forest cover dataset which contains 2,747 total minority class examples, the training data will contain approximately 2,472 examples while test data will contain 275 examples. Since there were a fair number of examples in the minority class SMOTE may have been more effective. It is unlike the mammography dataset where the number of minority class examples in the testing set is only 27.

We also looked at the Forest cover dataset under a cost framework. While, there weren't any obvious cost matrices that could be constructed, we simply utilized the following relative costs of $(C(+|-), C(-|+))$: (1, 1); (1, 10); (1, 50); (1,100); and (1,500). As evident in Figure 3, Ripper and C4.5 provide different performances as the cost matrices change for this dataset. Ripper is helped by undersampling, while C4.5 is not. This further justifies the use of wrapper techniques for different classifiers when considering sampling as a strategy for imbalanced datasets. Moreover, Ripper at (1,1) also benefited by SMOTE. Figure 4 shows the cost-curves across the range of $PCF(+)$. The wrapper based SMOTE and SMOTE-Undersampling for C4.5 and Ripper, respectively, produce the lowest expected for the broad range of $PCF(+)$. The choice of the classifier with the sampling methods doesn't seem to make a significant difference in the expected costs, while the individual classifiers are significantly apart in the cost space.

An interesting addition to our work will be analysis of the behavior of SMOTE and the rules thus constructed with both Ripper and C4.5. We would also investigate combination of the outputs of both the classifiers if they are making different kinds of errors to reduce the overall costs.

5.3 KDD-99 Cup Intrusion Dataset

This data set we treat differently. We look at it in a way that allows for comparisons with previous published work. A particular interesting example for comparison is to look at the results from of the KDD-99 cup data. A cost matrix was used in the scoring of the competition as shown in Table 4 [18]. It was used to produce the results in Table 7 below. There were many duplicate examples in the original 5 million example training set. All duplicate examples were removed. We also under-sampled both the normal and neptune (dos) class by removing examples which occurred only once. For Training Data 1 as in the Table 4, we under-sampled the normal class, and for the Training Data 2 we under-sampled both the normal and neptune classes. Note that for both these set of experiments, the test set remained unchanged. Our assumption was that some of them could be mislabeled or they were not very representative. These changes resulted in the training data set used here. Only SMOTE was applied to the modified data with the percentages for each minority class shown in Table 5.

It can be seen that our approach with RIPPER as the classifier produced the lowest cost per example after under-sampling both the normal and neptune classes (Training Data 2), and applying 100% SMOTE to the u2r class, while keeping the r2l class unchanged. This was better than the winner of the contest and better than the succeeding results from the literature. Even C4.5 as the base classifier with SMOTE (200% u2r and 300% for r2l) performed better than the other published techniques.

6. CONCLUSIONS

In this work, a wrapper [23] approach was utilized to determine the percentage of minority examples to add to the training set and the percentage to under-sample the majority class examples. The wrapper approach works by doing a guided search of the parameter space. The evaluation function was applied with a five fold cross validation done on the training set. Once the best percentages for under sampling and SMOTE are found it can be used to build a classifier on the updated training set and applied on the unseen testing set.

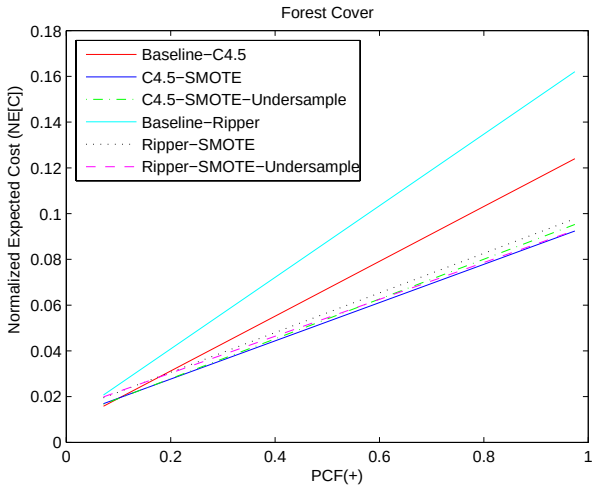


Figure 4: Cost Curve for Forest Cover dataset.

The *f-value* metric was used as the evaluation function. By using such a composite measure, we are able to control the relative increases in precision and recall, as both are essentially dependent on the different types of errors — false positives and false negatives. To statistically validate the results, we applied a 10-fold cross-validation framework to all but one of the datasets. Within each 10-folds, the wrapper utilized 5-fold cross-validation to identify the potentially optimal amounts of under-sampling and SMOTE.

We show results from applying this approach to the mammography dataset, the forest cover dataset, and the KDD-cup 99: Network Intrusion Detection dataset. Two learning algorithms were used, RIPPER a rule learning algorithm and C4.5 a decision tree learning algorithm. For the experiments, it was shown that it was possible to significantly increase the accuracy on the minority class, and reduce the overall expected costs. Our approach for imbalanced datasets significantly outperformed the baselines both in the true positive rate and the average cost per test example. Note that the *f-values* did not differ significantly because of the reduction in precision at the expense of the increase in recall. However, the relative increase in false positives does not impact the costs computation, because it is more costly to err as a false negative than a false positive. We achieved the lowest cost per test example of any approach

Table 4: Cost matrix used for scoring entries in KDD CUP 99 competition.

Actual/predicted	dos	u2r	r2l	probe	normal
dos	0	2	2	1	2
u2r	2	0	2	2	3
r2l	2	2	0	2	4
probe	2	2	2	0	1
normal	2	2	2	1	0

Table 5: Comparison of results obtained on the original KDD CUP 99 test data. The numbers beside u2r and r2l indicate the SMOTE percentage utilized for the experiments.

	dos	u2r	r2l	probe	normal	Cost per test example
Winning Strategy [18]	97.10%	13.20%	8.40%	83.30%	99.50%	0.2331
Decision Tree [5]	96.57%	13.60%	0.45%	77.92%	99.43%	0.2371
Nave Bayes [5]	96.65%	10.96%	8.66%	88.33%	97.68%	0.2485
Multi-classifier [31]	97.30%	29.80%	9.60%	88.70%	-	0.2285
Using C4.5 on Training Data 1 u2r (200) - r2l (0)	97.08%	14.47%	1.21%	93.52%	97.87%	0.2478
Using RIPPER on Training Data 1 u2r (100) - r2l (0)	97.45%	22.37%	6.96%	81.64%	96.18%	0.2444
Using C4.5 on Training Data 2 u2r (200) - r2l (300)	99.41%	14.47%	7.39%	93.61%	97.34%	0.2051
Using RIPPER on Training Data 2 u2r (100) - r2l (0)	97.33%	19.74%	13.73%	91.98%	95.62%	**0.2049**

we know of for the intrusion detection data. We also introduced artificial costs for both the mammography and forest cover data. Our approach again produces lowest cost per test example, when compared to the baseline approach. It is very compelling that for the Forest cover dataset, our approach produces lower cost per test example even for (1, 1). Hence, the wrapper approach for automatically selecting the amounts of SMOTE with under-sampling is very promising. The proposed framework should be applicable to any sampling technique and evaluation measure.

In this paper, we did not include the costs in the *f-value* by varying the β parameter to reflect the relative ratios. We believe that will be an interesting addition to our work. If the costs are known then they can expressed within validation framework for selecting the amounts of SMOTE and under-sampling quantities. We believe incorporating costs should again reduce the overall costs of the errors. The sampling quantities are also discovered using the same costs for both the classes as they will be used during evaluation. Thus, a stronger utilitarian framework can be implemented.

7. ACKNOWLEDGMENTS

Larry Hall was partially supported by the Department of Energy through the Advanced Strategic Computing Initiative (ASCI) Visual Interactive Environment for Weapons Simulation (VIEWS) Data Discovery Program Contract number: DEAC04-76DO00789. Nitesh Chawla was partially supported by National Science Foundation grant CNS-0130839, by the Central Intelligence Agency, and Department of Justice grant 2004-DD-BX-1224. We would like to thank Chris Drummond for his helpful input on the cost curves. We would also like to thank the anonymous reviewers for their useful comments.

8. REFERENCES

[1] In T. Dietterich, D. Margineantu, F. Provost, and P. Turney, editors, *Proceedings of the ICML'2000 Workshop on COST-SENSITIVE LEARNING*. 2003.

[2] In N. V. Chawla, N. Japkowicz, and A. Kołcz, editors, *Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Data Sets*. 2003.

[3] In N. V. Chawla, N. Japkowicz, and A. Kolcz, editors, *SIGKDD Explorations Special Issue on Learning from Imbalanced Datasets*. SIGKDD, 2004.

[4] In C. Ferri, P. Flach, J. Orallo, and N. Lachice, editors, *ECAI' 2004 First Workshop on ROC Analysis in AI*. ECAI, 2004.

[5] N. B. Amor, S. Benferhat, and Z. Elouedi. Naive Bayes vs. Decision Trees in Intrusion Detection Systems. In *Proceedings of the ACM Symposium on Applied Computing*, pages 420–424, 2004.

[6] R. E. Banfield, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. Ensembles of Classifiers from Spatially Disjoint Data. In *Proceedings of the Sixth International Conference on Multiple Classifier Systems*, 2005.

[7] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explorations*, 6(1), 2004.

[8] C. Blake and C. Merz. UCI Repository of Machine Learning Databases. Department of Information and Computer Sciences, University of California, Irvine, 1998.

[9] K. W. Bowyer, L. O. Hall, N. V. Chawla, and T. E. Moore. A parallel Decision Tree Builder for Mining Very Large Visualization Datasets. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 2000.

[10] M. Buckland and F. Gey. The Relationship Between Recall and Precision. *Journal of the American Society for Information Science*, 45:12–19, 1994.

[11] P. Chan, W. Fan, P. Prodromodis, and S. Stolfo. Distributed Data Mining in Credit Card Fraud Detection. *IEEE Intelligent Systems*, 14(6):67–74, 1999.

[12] N. V. Chawla. C4.5 and imbalanced datasets: Investigating the effect of ampling method, probabilistic estimate, and decision tree structure. In *Proceedings of the ICML'03 Workshop on Class Imbalances*, 2003.

[13] N. V. Chawla. Editorial: Learning from Imbalanced Datasets. *SIGKDD Explorations*, 6(1), 2004.

[14] N. V. Chawla, L. O. Hall, B. K. W., and W. P. Kegelmeyer. SMOTE: Synthetic Minority Oversampling TEchnique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[15] W. W. Cohen. Fast Effective Rule Induction. In *Proc. 12th International Conference on Machine Learning*, pages 115–123, Lake Tahoe, CA, 1995. Morgan Kaufmann.

[16] C. Drummond and R. Holte. Explicitly representing expected cost: An alternative to ROC representation. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 198–207, 2001.

[17] C. Drummond and R. Holte. C4.5, class imbalance, and cost sensitivity: Why under-sampling beats over-sampling. In *Proceedings of the ICML'03 Workshop on Learning from Imbalanced Data Sets*, 2003.

[18] C. Elkan. Results of the kdd'99 Classifier Learning Contest. http://www.cse.ucsd.edu/∼elkan/clresults.html, 1999.

[19] J. Ezawa, K., M. Singh, and W. Norton, S. Learning Goal Oriented Bayesian Networks for Telecommunications Risk Management. In *Proceedings of the International Conference on Machine Learning, ICML-96*, pages 139–147, Bari, Italy, 1996. Morgan Kauffman.

[20] S. Hettich and S. D. Bay. The UCI KDD Archive [http://kdd.ics.uci.edu]. Department of Information and Computer Sciences, University of California, Irvine, 1998.

[21] N. Japkowicz. The Class Imbalance Problem: Significance and Strategies. In *Proceedings of the 2000 International Conference on Artificial Intelligence (IC-AI'2000): Special Track on Inductive Learning*, Las Vegas, Nevada, 2000.

[22] N. Japkowicz and S. Stephen. The Class Imbalance Problem: A Systematic Study. *Intelligent Data Analysis*, 6(5):203–231, 2002.

[23] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[24] M. Kubat, R. Holte, and S. Matwin. Machine Learning for the Detection of Oil Spills in Satellite Radar Images. *Machine Learning*, 30:195–215, 1998.

[25] M. Kubat and S. Matwin. Addressing the Curse of Imbalanced Training Sets: One Sided Selection. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 179–186, Nashville, Tennesse, 1997. Morgan Kaufmann.

[26] B. S. Lee, R. R. Snapp, and R. Musick. Toward a Query Language on Simulation Mesh Data: An Object Oriented Approach. In *Proceedings of the International Conference on Database Systems for Advanced Applications*, 2001.

[27] C. Ling and C. Li. Data Mining for Direct Marketing Problems and Solutions. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, New York, NY, 1998. AAAI Press.

[28] M. Maloof. Learning when data sets are imbalanced and when costs are unequal and unknown. In *Proceedings of the ICML'03 Workshop on Learning from Imbalanced Data Sets*, 2003.

[29] F. Provost and T. Fawcett. Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distributions. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 43–48, New Port Beach, CA, 1997. AAAI Press.

[30] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1992.

[31] M. R. Sabhnani and G. Serpen. Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset with Misuse Detection Context. In *Proceedings of the International Conference on Machine Learning: Models, Technologies, and Applications*, pages 209–215, 2003.

[32] G. Weiss and F. Provost. Learning when Training Data are Costly: The Effect of Class Distribution on Tree Induction. *Journal of Artificial Intelligence Research*, 19:315–354, 2003.

[33] K. Woods, C. Doss, K. Bowyer, J. Solka, C. Priebe, and P. Kegelmeyer. Comparative Evaluation of Pattern Recognition Techniques for Detection of Microcalcifications in Mammography. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(6):1417–1436, 1993.

[34] B. Zadrozny and C. Elkan. Learning and Making Decisions When Costs and Probabilities are Both Unknown. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 204–213, 2001.

Noisy Information Value in Utility-based Decision Making

Clayton T. Morrison
USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292, USA
1-310-448-8430
clayton@isi.edu

Paul R. Cohen
USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292, USA
1-310-448-9342
cohen@isi.edu

ABSTRACT
How much is information worth? In the context of decisions, the value of information is the expected increase in utility of the decision as a result of having the information. When the information might be noisy, the model is slightly more complicated. We develop a model of the value of noisy information in the context of a plausible intelligence information gathering and decision making scenario.

Categories and Subject Descriptors
H.4.2 [**Information Systems Applications**]: Types of Systems – *decision support*; G.3 [**Probability and Statistics**]; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving – *uncertainty and probabilistic reasoning*.

General Terms
Economics, Management, Theory.

Keywords
Value of information, Uncertainty, Noise, Economic Utility, Decision Theory, Intelligence Analysis.

1. INTRODUCTION
In this paper we describe a method for evaluating the value of information with respect to making a decision. The method answers the question, if paying more for information ensures better information, how much should I pay? Or, what is the value of corrupted or noisy information? Our goal is to develop a rational theory of intelligence information gathering as it relates to decision-making, particularly in scenarios where information quality depends on the amount of resources spent. We develop this model within statistical decision theory, which combines probability, statistics and utility theory to provide a coherent framework for evaluating and choosing actions under conditions of uncertainty [20, 13, 15, 11]. Here we consider a class of decision-theoretic models made up of three components: a probabilistic model of the states of the world and their causal relations, decisions that link actions to consequences, and a system of values assigned to those consequences. We review the standard framework for assessing the value of information, develop the model of the value of noisy information, and provide an example of its use in an intelligence information gathering and decision scenario.

UBDM'05, August 21, 2005, Chicago, Illinois, USA.

2. OLYMPIC SECURITY
Consider the following scenario. You are the head of security for the 2004 Olympic Games in Athens, Greece. You have been tasked with protecting the games from potential terrorist attacks. You have the authority to raise a terrorist threat alert that will mobilize special forces and implement extreme counter-terrorism measures. If you raise the alert just prior to an actual attack, the attack will be thwarted and your security firm will be awarded a handsome sum. However, raising the alert is costly, especially if it is a false alarm – it will disrupt the games and your security firm will be held accountable for the resulting loss of potential revenue. On the other hand, not raising the alert prior to an attack will be devastating. The city will be unprepared, lives will be lost and the security firm will be held accountable for a very large sum of money. If no attack is imminent and you do not raise the alert, you are assumed to be doing your job and will be paid as per your contract.

At your disposal you have a team of agents to collect information; each piece of information must be purchased. Suppose one piece of information is the location of terrorist group members, and another is about how prepared they are (in the sense of having the right materials and manpower) to carry out an attack. Based on your experience, you have developed a model of how knowledge of the location or level of preparedness will affect your belief in the likelihood of an attack. You have been provided a fixed budget and it is up to you to manage your information-gathering resources and make the decision about the terror alert. You must decide which information to pay for (if any) while minimizing overall expenditures and still make the right decision about the alert level.

3. A DECISION MODEL
The value of information is always relative to some *target decision*. In the Olympic Security scenario, the target decision is a choice among the actions of raising or not raising the terror alert. The target decision is associated with a *target hypothesis T*, a state of the world that has a direct bearing on the outcome of the target decision. The target hypothesis for our security firm is whether an attack is about to take place. In a probabilistic model, T is a random variable with possible states t, and the available actions a are represented by a decision variable A. In general, the states and actions of T and A are the domain of the variables T and A. A random variable has a probability distribution over its

possible states, while a decision variable is assumed to be deterministically controlled by the decision maker.

The outcomes of actions taken in the context of world states may be assigned values or *utilities*, which represent the relative desirability of outcomes. In the Olympic scenario, the outcome of deciding whether to raise the terror alert is represented in terms of money being gained or lost, depending on whether an attack is or is not about to happen. More generally, a utility function $U(a,t)$ maps action a and a target hypothesis state t to a utility value.

Given a target hypothesis, a set of actions and a mapping of target states and actions to utilities, we can frame the target decision problem. Under a widely accepted characterization of rational decision making [20, 15], the optimal decision is to take the action that maximizes the expected utility given beliefs about the target state of the world. Given a distribution over the possible values t of target state T, the expected utility of taking action a is

$$EU(T) = \sum_{t \in T} P(t)U(a,t). \quad (1)$$ [1]

Some actions may yield a higher expected utility than others. The utility of taking the *optimal* action out of the possible action choices in A is the utility of taking the action that *maximizes* expected utility, expressed as follows:

$$MEU(T) = \max_{a \in A} \sum_{t \in T} P(t)U(a,t). \quad (2)$$

4. UTILITY-BASED VALUE OF INFORMATION

In this section we present the standard approach to assessing the value of information with available utilities. In Section 5 we develop the value of noisy information out of this basic framework.

In many situations, we cannot simply observe the state of the target hypothesis T and make our decision. Instead, we must rely on other states, which are observable, and (we hope) tell us something about the state of the target. The Olympic Security scenario describes two potential sources of information that have some relation to the target hypothesis about whether there is an imminent terrorist attack: the proximity of terrorists to Athens and the capabilities or readiness of the terrorists to commit a terrorist act. These information sources are also about states of the world (e.g., how far away the terrorists are from the Games) and may themselves be represented by random variables (e.g., a distribution over whether the terrorists are within the city limits, in the country, or outside of the country). We use the generic term *indicator variable* for a variable that has some relationship with a target hypothesis. We denote an indicator random variable by I, which represents a distribution over possible states i. We also assume we have a complete joint probability distribution representing the relationship between I and T. This means that we have the information required to derive prior probabilities over I and T, as well as their conditional relationships, $P(I|T)$ and $P(T|I)$. In practice, joint distributions are efficiently represented as Bayesian belief networks, and algorithms exist for effectively deriving and estimating probability distributions from them [15, 11].

Usually the state of our target hypothesis is not directly observable. Instead, we may need to rely on one or more indicators, and determining their state may come at a cost, C_I. In this case we are faced with an *information gathering* decision, which is to be made in the service of our target decision. Out of the set of available indicators, which should we spend resources on? Making this decision requires assessing the value of the information the indicators may provide about the target hypothesis.

The value of any information source is defined as the difference between the utilities of two decision strategies, one in which we choose the optimal action after finding out the state an indicator variable is in, the other choosing the optimal action without that information [10, 13, 15, 8, 11, 12]. The expected utility of taking the optimal action given the outcome i of indicator variable I is

$$MEU(T \mid i) = \max_{a \in A} \sum_{t \in T} P(t \mid i)U(a,t) \quad (3)$$

Since the outcome of I is not known ahead of time, we can calculate the expected utility of having evidence I by marginalizing over the possible values of I:

$$MEU(T \mid I) = \sum_{i \in I} P(i)MEU(T \mid i) \quad (4)$$

$$VOI(T \mid I) = MEU(T \mid I) - MEU(T). \quad (5)$$

Taking into account the cost C_I of acquiring the information about the state of I, the net value or *expected profit* of purchasing the information is

$$netVOI(T \mid I) = VOI(T \mid I) - C_I. \quad (6)$$

If the net value is greater than zero, then the information is worth paying for.

4.1 Myopic Value of Information

Equations 3 through 6 allow us to calculate the value of information about a particular indicator given our current state of knowledge. However, once we consult one source of information, our state of knowledge may change, affecting what we may learn from other information sources, and this in turn affects their value. In general, when considering sequences of information gathering decisions, every permutation of the available information sources must be considered [15, 8, 11]. A *myopic* approximation of information value assesses each information source independently of the others. The myopic assessment is made as if the information source were the only one available, and under the assumption that immediately after gathering the information a final decision is made that incurs some utility [15]. While not perfect, this method has been found to perform well in medical diagnostic systems [7, 9, 14]. Heckerman, Horvitz and Middleton [8] have proposed an *approximate nonmyopic* method for computing information value given certain constraints. In this paper we will consider only myopic value of information.

[1] A note about notation: All of the probability terms in this paper are assumed to be in the context of all currently available evidence. That is, $P(t)$ could be expressed as $P(t|E)$ where E is the set of all other known variables, some of which are known to be in specific states. For clarity, we omit E from our equations. Similarly, all utility calculations are assumed to be in the context of a particular decision variable A and we will only explicitly note A in the context of maximization functions.

5. THE VALUE OF NOISY INFORMATION

We now add another wrinkle to the story, one that has been alluded to [10, 15, 16], but to our knowledge has not received extended treatment, except in very different terms in the economics literature [1, 6]. Suppose the amount you pay affects the quality of information you receive from an information source and the more you pay the more accurate the information is. Now our information gathering decision is to determine which level of payment is optimal for this potentially noisy information. We start by considering paying for reports at a particular cost level and then present the more general formulation of the choice of cost level.

We use R_C to represent a distribution over possible reports about the state of an indicator I at a particular cost level C. We emphasize that what we are paying for at this cost level is not a particular report, but a *distribution* over reports, as the report we receive depends on both the state of I as well as the probabilistic, and therefore noisy, relationship $P(R_C|I)$ we are paying for.

Assessing the expected utility of paying for possible reports R_C about the state of I is no different than the standard Value of Information calculation presented as Equations 3 through 6 in Section 4. Only now we're considering the state of R_C as an indicator of state I, which in turn is an indicator of our target hypothesis T. That is, we could re-write Equations 3 through 6 by replacing R_C for I. Nonetheless, it is useful to highlight the relationship between R_C and I because, again, it is *this* relationship we are paying for.

The following recasts the *VOI* calculation in terms that make the relation between R and I explicit:

$$MEU(T \mid R_C) = \sum_{r \in R_C} P(r) \max_{a \in A} \left(\sum_{i \in I} \sum_{t \in T} P(t \mid i) P(i \mid r) U(a,t) \right) \quad (7)$$

All we have done is add a term that conditions the probability of i on r, and marginalized the effects of r on expected utility by multiplying by and summing over $P(r)$. In other words, Eq. 7 represents the expected maximum utility given that our only source of information is r.

Equations 8 and 9 express the expected benefit and profit of paying for noisy reports at cost C.

$$VONI(T \mid R_C) = MEU(T \mid R_C) - MEU(T) \quad (8)$$

$$netVONI(T \mid R_C) = MEU(T \mid R_C) - MEU(T) - C \quad (9)$$

With $netVONI(T|R_C)$ we can now determine the report distribution R_C at cost level C that yields the highest expected utility:

$$maxVONI(T \mid R_C) = \max_{R_C \in \Re} \left[netVONI(T \mid R_C) \right], \quad (10)$$

where $\Re$ is a set of sources, R, of information about I, each with its own distribution $Pr(I|R)$, distinguished only by how much R costs.

5.1 A Simple Model of Noisy Reports

There are many possible representations of the relationship between I and R. This is a general topic for intelligence analysis and modeling research and will depend on the domain being represented. To demonstrate the value of noisy information in the Olympic Security scenario, we provide a simple linear noise model to generate R_C.

We define a "*noise* level" as a real-valued number between 0.0 and 1.0 (inclusive), where 0.0 means perfect information (no noise) and 1.0 means complete noise. Suppose the reports we are paying for are about an indicator with possible states {*close*, *near*, *far*}. For each possible state of I that report r could say I is in, given that the indicator is actually in state i, we determine the probability $P(R_C=r|i)$ as follows:

$$P(R_C = r \mid i) = \left(\left(\frac{1}{m} - d \right) \times noise \right) + d, \quad (11)$$

where m is the number of possible states of the indicator and $d = 1$ when r reports the same as the target state value i of the indicator, otherwise $d = 0$. We repeat this for each state of I to arrive at $P(R_C|I)$, the conditional probability distribution over reports given the states of I at a particular noise level. Finally, we provide a cost function that maps costs to noise levels, so that given a particular payment C, we can generate $P(R_C|I)$. Under the assumption that information becomes exponentially more expensive with accuracy, we chose the cost function depicted in Figure 1.

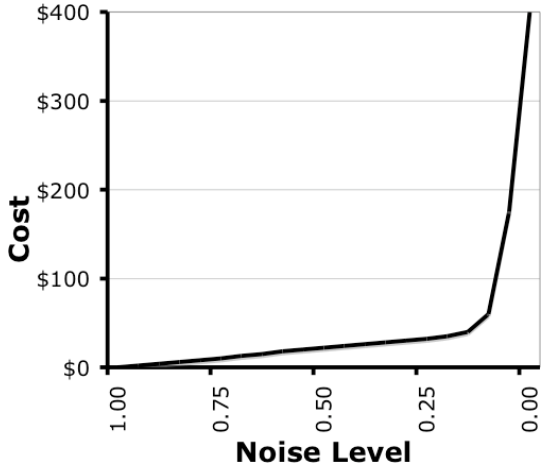


Figure 1. An exponential cost function mapping noise levels to costs of information in dollars.

5.2 Back to the Olympics

Figure 2 shows a *decision graph* representing the Olympic Security scenario. A decision graph is a useful formalism for representing relationships between variables in decision problems. In the graph, decision variables are represented by squares, utility functions by diamonds, and random variables by circles. A directed arrow indicates that the state of a parent node participates in determining the state of a child node (where the child is the node being "pointed to"). The labels in the nodes represent random variables, and we have included text near the nodes indicating which part of the Olympic Security decision problem the variable corresponds to. To complete the specification of the model, we need the prior and conditional probability relationship between random variables as well as a utility function. The tables on either side of the graph in the figure provide this information.

In this scenario, we consider purchasing a distribution over reports about the capabilities of the terrorists. With all of the information represented in Figure 2, we can calculate the value of noisy information of a report about terrorist capabilities at a given level of noise (Eq. 8). Figure 3 does this for noise levels ranging from 1.0 (complete noise) down to 0.0 (no noise). The state of knowledge about proximity affects the value of information, and proximity can be in one of four states (*unobserved, close, near* or *far*). Because of this, Figure 3 plots four different curves representing the value of information across noise levels *given* that proximity is in one of its four possible states. Figure 4 factors in the cost of information for the *net* value of information (Eq 9). The *max* value of noisy information, Equation 10, provides us with a strategy by selecting the cost level at which the expected utility peaks on each curve. We should select the level of payment for a noisy report according to the noise level that yields the greatest expected utility.

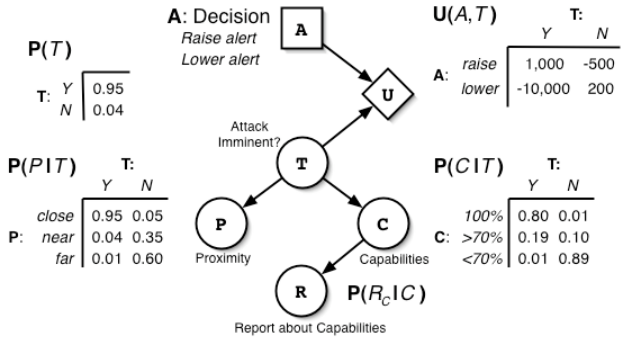


Figure 2. Decision graph and probability and utility tables characterizing the Olympic Security scenario.

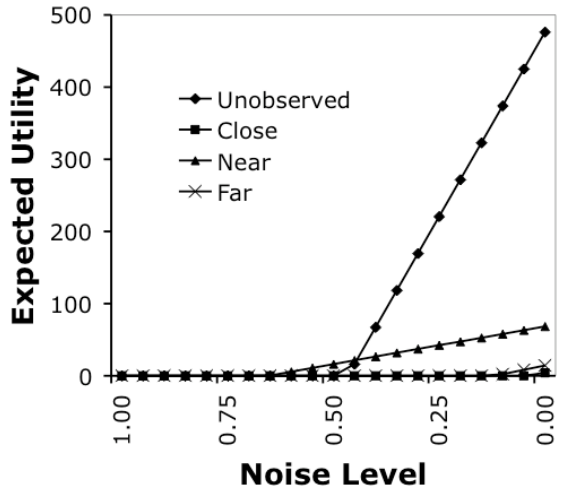


Figure 3. The VONI at varying noise levels. Each curve represents VONI given a state of proximity.

As Figure 4 shows, whether to pay for reports, and if so, how much, depends on our belief about the proximity of terrorists to the Games. When proximity is *unobserved*, paying for a report has the greatest benefit. In particular, the benefit is maximized in the peak of the curve, when noise level is 0.10, costing $60. When proximity is *near*, the benefit of paying for a report peaks at noise level 0.15, with a cost of $40. When proximity is either *close* or *far*, a report is simply not worth paying for at any level. When proximity is known to be *far*, it is likely that an attack is not about to take place; when proximity is *close*, then an attack is almost certain. Under these conditions, paying for more information is simply not worth it. However, as Figure 4 shows, when proximity is *unknown* or *near*, then knowing about the state of the terrorists capabilities is useful in determining whether an attack is about to occur, and paying the price of perfect information is not as cost effective as paying less for somewhat degraded information.

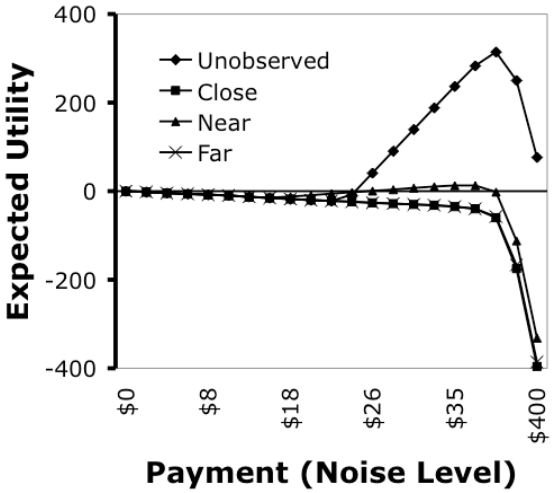


Figure 4. The netVONI at varying noise levels (as represented by their cost) given proximity.

6. CONCLUSION

We have presented a decision model based on the value of information and demonstrated its use in a simple intelligence analysis decision scenario. We demonstrated that with VONI we can determine the optimal amount to pay for information where the amount of effort or cost invested affects information quality. The value of noisy information is an incremental extension to the standard value of information framework, making it possible to assess the value of reports about an indicator that informs a target decision.

As we noted in Section 5.1, the representation of the relationship between and indicator variable I and possible reports R about the indicator is a general topic for intelligence analysis and modeling research. Our simple linear noise model is just one example. Robust models of these relationships will depend on specific scenarios, the expertise of trained analysts, and possibly learned from collected data.

Although we have not presented this framework in terms of machine learning, there are connections between the VONI framework and recent work in cost-sensitive [11, 19] and active [6, 10, 17, 18] learning. In particular, VONI makes explicit the role of data acquisition costs and the impact that acquiring costly information has on decision-making. Recently, [18] explicitly argues for the importance of making active learning decision-

centric, demonstrating in an active learning scenario that simply improving the accuracy of the target classification on which a decision is based does not necessarily lead to overall improvement in decision-making. An important next step in the VONI model is to explore how a model of the noisy relationship between reports and indicators can be learned.

7. ACKNOWLEDGEMENTS

We thank Jafar Adibi and Aram Galstyan for stimulating discussions about portions of the model presented. Work on this project was funded by the Air Force Research Laboratory, account number 53-4540-0588.

8. REFERENCES

[1] Athey, S. C. and Levin, J. D. 2001. The value of information in monotonic decision problems. MIT Department of Economics Working Papers.

[2] Ben-Bassat, M. 1978. Myopic policies in sequential classification. *IEEE Transactions of Computing* 27: 170-174.

[3] Cohn, D. A., Atlas, L. and Ladner, R. E. (1994). Improving Generalization with Active Learning.

[4] Cohn, D. A., Ghahramani, Z. and Jordan, M. I. (1996). Active Learning with Statistical Models. *Journal of Artificial Intelligence Research*, 4: 129-145.

[5] Elkan, C. (2001). The Foundations of Cost-Sensitive Learning. Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01).

[6] Gauthier, L. and Morellec, E. 1997, June. Noisy information and investment decisions: A note. MIT Department of Economics Working Papers.

[7] Gorry, G. and Barnett, G. 1968. Experience with a model of sequential diagnosis. *Computer and Biomedical Research* 1: 490-507.

[8] Heckerman, D, Horvitz, E. and Middleton, B. 1993. An approximate nonmyopic computation for value of information. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15(3): 292-298.

[9] Heckerman, D., Horvitz, E. and Nathwani. 1992. Toward normative expert systems: Part i. the pathfinder project. *Methods Inform. Med.* 31: 90-105.

[10] Howard, R. A. 1966. Information value theory. *IEEE Transactions on System Science and Cybernetics* 2(1): 22-26.

[11] Jensen, F. 2001. *Bayesian networks and decision graphs*. New York: Springer.

[12] Jensen, F. and Liang, J. 1994. A system for value of information in Bayesian networks. In Proceedings of the 1994 Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, 178-183.

[13] Lindley, D. V. 1971. *Making Decisions*. New York: John Wiley & Sons.

[14] Mussi, S. 2004. Putting value of information theory into practice: a methodology for building sequential decision support systems. *Expert Systems* 21(2): 92-103.

[15] Pearl, J. 1988. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Los Altos, CA: Morgan-Kaufman.

[16] Russell, S. J. and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. New Jersey: Prentice Hall.

[17] Saar-Tsechansky, M. and Provost, F. 2004. Active Sampling for Class Probability Estimation and Ranking. *Machine Learning* 54(2), 153-178.

[18] Saar-Tsechansky, M. and Provost, F. Active Learning for Decision-Making. Working paper (CeDER-04-06, Stern School of Business, New York University) . Available at http://www.mccombs.utexas.edu/faculty/Maytal.Saar-Tsechansky/home/GOAL.pdf

[19] Turney, P. (2000). Types of Cost in Inductive Concept Learning. Proceedings of the Cost-Sensitive Learning Workshop at the 17th ICML-2000 Conference, Stanford, CA. pp.15-21.

[20] Von Neumann, J. and Morgenstein, O. 1953. *Theory of Games and Economic Behavior*. 3rd Ed. New York: John Wiley & Sons.,

Learning Policies for Sequential Time and Cost Sensitive Classification

Andrew Arnt
arnt@cs.umass.edu

Shlomo Zilberstein
shlomo@cs.umass.edu

Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003

ABSTRACT

In time and cost sensitive classification, the utility of labeling an instance depends not only on the correctness of the labeling, but also the amount of time taken to label the instance. Instance attributes are initially unknown, and may take significant time to measure. This results in a difficult problem, trying to manage the tradeoff between time and accuracy. The problem is further complicated when we consider a sequence of time-sensitive classification instances, where time spent measuring attributes in one instance can adversely affect the costs of future instances. We solve these problems using a decision theoretic approach. The problem is modeled as an MDP with a potentially very large state space. We discuss how to intelligently discretize time and approximate the effects of measurement actions in the current instance given all waiting instances. The results offer an effective approach to attribute measurement and classification for a variety of time sensitive applications.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning - Induction; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search - Graph and tree search strategies; H.2.8 [**Database Management**]: Applications - Data Mining

General Terms

Algorithms

Keywords

cost-sensitive learning, data mining, AO* search

1. INTRODUCTION

Cost sensitive classification (CSC) has been the subject of a growing body of research. In CSC, the goal is to train a classifier that minimizes the expected cost incurred on future test instances, rather than trying to maximize the predictive accuracy. Penalties for misclassifying instances vary based on the actual label of the instance. For example, in medical diagnosis domains classifying a sick patient as well is often far more costly than labeling a healthy patient as sick. In a spam filtering system, legitimate email flagged as spam is significantly more costly than spam judged as legitimate.

Additionally, in some CSC problems, attributes of an instance are not initially known. Instead, the CSC classifier must explicitly decide which attributes to measure. Some of these attributes may have a fixed cost to measure; an example from the medical diagnosis domain are those that require an expensive test to be performed. In this problem, an attribute measurement and classification policy that specifies what attributes to measure and in what order is designed to minimize not only misclassification penalties, but also the sum of attribute measurement costs.

In this work, we examine a previously unexplored dimension of CSC. In many domains, the value of a classification result depends not only on the correctness of the labeling, but also the timeliness with which it is computed. Furthermore, measuring some of these attributes may be either computationally intensive or rely on slow external sources of information. For example, in medical diagnosis, tests are often sent away for processing while the patients condition may be deteriorating. In the spam filtering case, retrieving or verifying hyperlinked information can take significant time and delay the arrival of email to a user's inbox. It is impractical to measure all possible attributes for each instance when the final result has time-dependent utility. We call this problem time and cost sensitive classification (TCSC).

Managing the tradeoff between classifier accuracy and time costs incurred is a challenging problem. Myopic methods such as those used in [12] will not perform well due to interactions between attributes: when not all attributes can be measured, the ordering of measurements becomes very important. There have been several methods designed by researchers for handling the CSC problem, but very little attention has been paid to the TCSC case. We develop a model that allows the system to quickly decide which attributes to measure, what order to measure them in, and when to cease any further measurement and classify the current instance.

We take a decision theoretic approach, where we try to minimize the expected value of a cost function reflecting the quality of service of the system. In the cost sensitive classifier developed in [17, 16], the attribute measurement problem was framed as a Markov Decision Process (MDP)

UBDM '05 August 21, 2005, Chicago, Illinois, USA

where the state was the current attribute vector. We build upon that work by adding the current time to that state. Due to the potentially large size of this state space, AO* heuristic search is used to compute the policy. It is not necessary in AO* to compute values for all possible states as would be required in a dynamic programming approach. The addition of time to that state space requires that time be intelligently discretized to provide a balance between the quality of the computed policy and the memory required to compute it.

We then examine the case where a sequence of time dependent instances must be classified over time. In this sequential TCSC problem, classification instances arrive at the classifier over time and are processed in a first-in first-out manner. Time spent measuring attributes in the instance at the head of the queue can increase the costs incurred on waiting instances by delaying the start of their processing. Clear examples of this type of problem are spam filtering on an overloaded mail server or estimating the value of messages posted to a newsgroup or online forum [1]. This model can also apply to diagnosis tasks. We show how to extend the MDP model to find policies that minimize cost over all instances processed and discuss the approximations used to solve this even larger MDP.

2. TYPES OF COST IN TCSC

2.1 Misclassification costs

In many applications, not all misclassifications have the same value. There may be a significant difference between the problems caused by a false negative versus those caused by a false positive. We denote this portion of the cost function which handles misclassification penalties as $C_L(l_p|l_a)$, which is the cost incurred by classifying an instance with actual label l_a with the predicted label l_p.

The misclassification (MC) cost component depends on the actual label l_a of an instance, which is unknown, except in training data. Thus, in practice we need to use the *expected* MC cost, given that the classifier predicts label l_p:

$$EC_L(\mathrm{cl}(l_p)|\mathbf{f}) = \sum_{l_a \in L} p(l_a|\mathbf{f}) C_L(l_p|l_a)$$

where L is the complete set of labels that an instance may have. The probability that an instance with the current measured attribute vector $\mathbf{f}$ has the actual label of l_a is estimated, when necessary, from training data: $p(l_a|\mathbf{f}) = \frac{|train(l_a,\mathbf{f})|}{|train(\mathbf{f})|}$, where $train(\mathbf{f})$ is the set of all training instances such that for every measured attribute in $\mathbf{f}$, the training instance has the same value, and $train(l_a, \mathbf{f})$ is the subset of instances in $train(\mathbf{f})$ that have label l_a. In practice this estimate is smoothed to reduce overfitting and avoid divisions by zero:

$$p(l_a|\mathbf{f}) = \frac{|train(l_a,\mathbf{f})| + 1}{|train(\mathbf{f})| + |L|}$$

When classifying a partially measured instance $\mathbf{f}$, the algorithm will choose the label that incurs the minimum expected MC cost on the given set of training data:

$$l_p = \arg\min_{l \in L} EC_L(\mathrm{cl}(l)|\mathbf{f}) \quad (1)$$

There has been a significant volume of work on the problem of minimizing MC costs: some general methods are the weighted boosting algorithm of [8], and the MetaCost algorithm of [7].

2.2 Attribute measurement costs

The action of measuring an attribute f_i is indicated as $\mathrm{m}(f_i)$. This action may incur a deterministic cost: $C_M(\mathrm{m}(f_i))$. We assume the value of a measured attribute is constant and will not change upon repeated measurements.

Research that handles both attribute measurement costs and misclassification costs includes the genetic algorithm based decision tree inducer of [15], the POMDP (Partially Observable MDP)-based decision tree learner of [5], a dynamic programming algorithm described in [10], a POMDP for computing attribute measurement policies with respect to a given naive Bayes classifier in [11], the test-cost sensitive naive Bayes classifer of [6], and finally an MDP framework with heuristic search to find good attribute measurement and classification policies [17, 16]. Note that none of these can handle any kind of time-dependent costs.

2.3 Response time costs

The cost function should reflect the timeliness with which we wish the classifier to act. In many systems (especially those that interact with humans), a labeling decision made quickly will be worth more than one that takes a very long time. In general, any system that has a component of human interaction should be fairly responsive and not spend unreasonable amounts of time measuring all instance attributes so as to minimize the expected MC cost.

Therefore the cost function has a final component $C_T(t)$, which will typically have a super-linear form: the cost of a quick result is small and fairly constant, but as the waiting time increases, the time cost grows at an increasing rate. This function provides a good approximation of a user's perceived utility of a system when they are forced to wait for a result. In general, the time cost function can take on any form that is nondecreasing over time.

Note that if the time cost function is linear and the measurement times for each attribute are deterministic, the time cost function can be simply folded into the attribute measurement costs.

2.4 Combining cost function components

To combine the three components of the cost function, it suffices to perform a simple weighted addition. The expected cost of assigning predicted label l_p to an instance $\mathbf{f}$ with measured attributes $\mathrm{meas}(\mathbf{f})$ in t time units is:

$$C(\mathbf{f}, t) = w_L EC_L(\mathrm{cl}(l_p)|\mathbf{f}) + w_T C_T(t) + w_M \sum_{f_i \in \mathrm{meas}(\mathbf{f})} C_M(\mathrm{m}(f_i))$$

The variables w_L, w_T, w_M are system parameters that are manually tuned to provide a good balance between the conflicting goals of low MC costs, attribute measurement costs, and timely responses.

3. SINGLE INSTANCE TCSC

Given a set of training data, we want to find the attribute measurement and classification policy that minimizes the expected cost of classification of future instances where cost is made up of the three components discussed in Section 2.

Our strategy for time and cost sensitive policy learning builds on the work of [17]. We frame the attribute measurement and classification problem as a Markov Decision

Process (MDP). The "optimal" policy (quoted because it is optimal only with respect to a set of labeled training data) can then be found using AO* search, a classical heuristic search technique. We extend this model to handle time-sensitive utility costs.

3.1 TCSC as an MDP

MDPs are a popular framework for sequential decision making problems. An agent in an MDP takes *actions* which cause stochastic transitions between *states*. A typical formulation (and the one used here) has an agent with the goal of minimizing the costs incurred while transitioning to some terminal state. Each state in the MDP satisfies the Markov property: the current state effectively summarizes all previous activity of the agent in the environment. The mapping from states to actions that minimizes cost is called the *optimal policy*.

The states $s \in S$ in the model presented in [17] are simply the set of all possible attribute vectors $\mathbf{f}$. This includes those with unmeasured attributes. An additional absorbing terminal state E is transitioned to when an instance is classified. We augment that state space to include the current waiting time of the instance: $s = \langle \mathbf{f}, t \rangle$. The starting state of the MDP is the state with no measured attributes and zero waiting time: $\langle \mathbf{f}_?, 0 \rangle$.

The actions in this model are to either measure an unmeasured attribute f_i, denoted '$\mathrm{m}(f_i)$', or to classify the current instance using the label l_p, denoted '$\mathrm{cl}(l_p)$'.

There are two types of cost related to taking a measurement action. $C_M(\mathrm{m}(f_i))$ is the deterministic cost to measure attribute f_i. There is also the incremental time cost $C_\Delta(\delta|t)$ which indicates the portion of the end cost $C_T(t)$ incurred by waiting δ additional time units to classify an instance that has already been waiting t time units. Given a time cost function $C_T(t)$, it is straightforward to compute the incremental time cost function:

$$C_\Delta(\delta|t) = C_T(t+\delta) - C_T(t)$$

The expected immediate cost of taking the action $\mathrm{m}(f_i)$ is then

$$C_M(\mathrm{m}(f_i)) + \sum_{\delta \in T_i} p(T_i = \delta) C_\Delta(\delta|t)$$

where T_i is the set of all possible durations that the measurement action can take.

The probability of transitioning from state $s = \langle \mathbf{f}, t \rangle$ to state $s' = \langle \mathbf{f} \cup f_i = x, t+\delta \rangle$ (where $\mathbf{f} \cup f_i = x$ refers to $\mathbf{f}$ with attribute f_i set to x) is

$$p(s'|s, \mathrm{m}(f_i)) = p(f_i = x|\mathbf{f}) p(T_i = \delta)$$

The probability that attribute f_i will take on value x given the incomplete attribute vector $\mathbf{f}$ is estimated from training data:

$$p(f_i = x|\mathbf{f}) = \frac{|train(\mathbf{f} \cup f_i = x)|}{|train(\mathbf{f})|}$$

In practice this value is smoothed to:

$$p(f_i = x|\mathbf{f}) = \frac{|train(\mathbf{f} \cup f_i = x)| + 1}{|train(\mathbf{f})| + |f_i|}$$

where $|f_i|$ indicates the total number of distinct values the ith attribute can have.

The probability that attribute f_i takes δ time units to measure is denoted as $p(T_i = \delta)$, and is estimated from training data or from some other source of prior experience. Note that these transition probabilities are computed only when necessary during the AO* search.

Taking the classification action incurs the MC cost $C_L(l_p|l_a)$ and transitions to the terminal state with probability

$$p(E|s, \mathrm{cl}(l_p)) = 1$$

Recall that l_p is chosen to minimize expected cost as in Equation 1.

3.2 AO* search

AO* search is an heuristic search algorithm for searching AND/OR graphs [13]. It is akin to A* search for standard directed graphs. MDP policies can be represented as an AND/OR graph: at an OR node, the agent must choose a single action to take so as to minimize future cost. However, since the environment is stochastic, taking an action causes the agent to transition probabilistically to one of a number of states. Therefore all these states are successors of the original state and their costs must be AND-ed together (computing the expected cost) to find the best expected action.

AO* works by iteratively improving upon the current best partial solution policy until an optimal policy is found. Each iteration of AO* search is composed of two parts. First, the current best partial solution is expanded (its successors are added to the search graph) by picking an unexpanded search state within the current policy. Next, state values and best action choices are updated in a bottom-up manner, starting from the newly expanded state. The estimated value of a state s during the search is $F(s)$: an optimistic estimate of the cost to get from s to a terminal state.

A heuristic is necessary to guide AO*. The heuristic value of a state is the optimistic estimate of how much cost will be incurred before reaching a terminal state. For an optimal policy to be found, the heuristic must be *admissible*: it must never overestimate the cost from a state to the terminal state. An optimistic one-step lookahead heuristic derived by [17] for the case where there is no time dependent utility was extended to include incremental time costs in [2].

Given an unexpanded state s, the heuristic value $F(s)$ is the cost of the action (classifying or measuring an attribute) giving the smallest immediate cost:

$$F(s) = \min_{f_i \notin \mathrm{meas}(\mathbf{f})} \begin{cases} EC_L(\mathrm{cl}(l_p)|\mathbf{f}) \\ C_M(\mathrm{m}(f_i)) + \sum_{\delta \in T_i} p(T_i = \delta) C_\Delta(\delta|t) \end{cases} \quad (2)$$

This heuristic is admissible because it gives the smallest possible immediate cost incurred when taking any action from the current state.

3.3 AO* as an anytime algorithm

For classification problems where instances have a large number of measurable attributes, each of which can take on many values, pruning the search space is essential for efficient search. A pruning strategy that preserves the optimality of the policy hinges on the fact that the terminal state E can be reached from any state of this MDP by taking a single classification action [17]. This property, which is not applicable for general MDP models, allows for significant pruning of the search space. An upper bound $\hat{F}(s)$ value is computed at each node; this value represents the expected

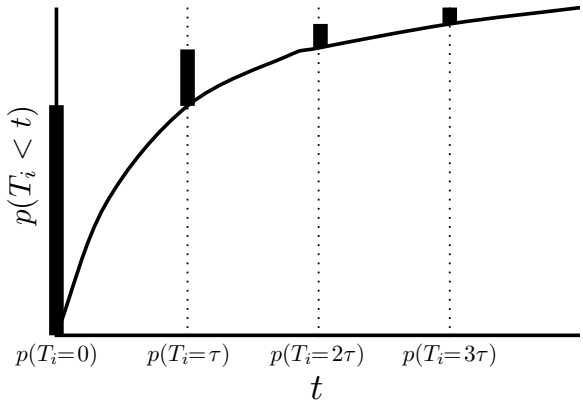


Figure 1: An attribute time distribution discretized in a 'round-down' manner. The curve shows the cumulative density function, the length of the black bars represents the probability of each discrete value.

cost of following the current *best known* policy from search state s. Therefore, any unexpanded search node s' with parent node s where $\hat{F}(s) < F(s')$ can be pruned, as the expansion of s' cannot lead to an improved policy since we will always choose the action at s that provides the minimal $\hat{F}(s)$.

Furthermore, maintaining the best known action at each search node gives the AO* search the properties of an anytime algorithm. At any time during the search, the process can be halted and the current best policy returned. For large classification problems where there are many attributes taking on many possible values in each instance, this must be done when memory is exhausted.

3.4 Discretization of time

While continuous attributes can be discretized using metrics such as information gain relative to the class attribute (for example [9]), finding an appropriate discretization of time for a instance is a more difficult problem. A very fine grained discretization of time results in the best policies, except when search terminates early due to available memory being exhausted by the larger state spaces. Furthermore, in some cases, using a coarser time representation may still find a policy of approximately equal value.

In this work we take an iterative approach to finding a suitable time unit size. Starting from an initial coarse time unit τ, we iteratively solve the MDP using a unit of $\tau' = \tau/2$. This process repeats until the cost incurred on training data by the policy $\pi_{\tau'}$ is greater than or approximately equal to the cost incurred by following π_τ; the π_τ policy is retained for actual use. Note that the cost of $\pi_{\tau'}$ can be greater than that of π_τ when the memory limit is reached and search is forced to terminate with the current best known policy.

To compute the policy for time unit τ, we use 'rounded-down' versions of the continuous attribute measurement time distributions $p(T_i)$. That is: $p_\tau(T_i = k\tau) = \int_{k\tau}^{(k+1)\tau} p(T_i = x)dx$. See Figure 1 for an example distribution. This rounding down combined with the nondecreasing nature of the time cost function $C_T(t)$ means that the F value of a state s in π_τ always underestimates of the actual cost of state s.

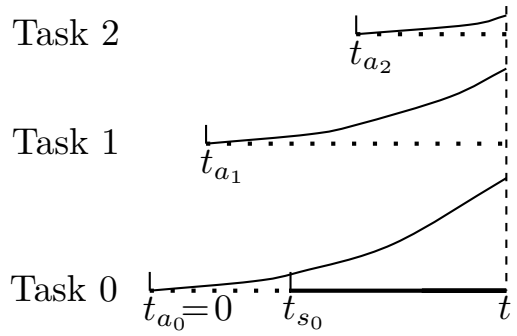


Figure 2: instances arriving over time. Dotted lines represent waiting time, solid lines active processing of an instance. Time cost curves are shown for each instance. Instance 0 arrived at t_{a_0}, but processing did not begin until t_{s_0} due to delay from measuring attributes in previous instances.

Therefore, we can use the F values computed in the τ iteration as part of a new admissible heuristic function F' in the τ' iteration; The new heuristic value at a state is the smaller of the heuristic value computed as before in Equation 2 and the final F value computed in the previous iteration (rounding down time from $k\tau'$ to $j\tau$, such that $j\tau \leq k\tau' < (j{+}1)\tau$):

$$F'(\langle \mathbf{f}, k\tau' \rangle) = \min(F(\langle \mathbf{f}, k\tau' \rangle), F_\tau(\langle \mathbf{f}, j\tau \rangle))$$

4. SEQUENTIAL TCSC

The above procedures do not account for other instances that need to be classified. Suppose that instead of a single classification instance to process, the system has to handle a *stream* of classification instances arriving over time. Therefore, when deciding which attributes to measure in the current instance, we must also consider the potential for utility loss due to delay in processing of all other instances waiting to be classified. [3] refer to this as the *opportunity cost*, the loss of expected value due to delay in the starting of work on the remaining instances. They show that for a similar problem, the opportunity cost function can be quickly and effectively approximated by examining simple attributes of the queue of waiting instances.

There are no known existing methods for classifying sequences of time sensitive instances. [14] study a sequential CSC problem where the cost of each instance is dependent on the labels assigned to prior instances. Reinforcement learning is used to minimize costs over a sequence of interacting instances; however, there is no time-sensitive cost component.

4.1 MDP model for sequential classification

We call the instance that is currently having attributes measured the 'active' instance. Time t is measured relative to the arrival time of the active instance $t_{a_0} \equiv 0$. As attributes are measured in the active instance, new instances arrive at t_{a_i}. Figure 2 shows a possible configuration.

Sequential classification instances can be introduced to the MDP model by expanding the state space to $s = \langle \mathbf{f}, t, \mathbf{q} \rangle$, where $\mathbf{q} = \{t_{a_1}, \ldots, t_{a_{|\mathbf{q}|}}\}$ describes the queue of instances waiting to be classified.

The MDP transition model must also be augmented to

include $\mathbf{q}$. The probability of transitioning from state $s = \langle \mathbf{f}, t, \mathbf{q} \rangle$ to state $s' = \langle \mathbf{f} \cup f_i = x, t + \delta, \mathbf{q}' \rangle$is

$$p(s'|s, \mathrm{m}(f_i)) = p(f_i = x|\mathbf{f})p(T_i = \delta)p(\mathbf{q}'|\mathbf{q}, \delta)$$

where $p(\mathbf{q}'|\mathbf{q}, \delta)$ is the probability of the queue going from state $\mathbf{q}$ to $\mathbf{q}'$during a time interval of δ time units. This quantity can be estimated from past experience.

We could then change the transition model so that classification actions transfer to terminal state E only when the queue is empty. In all other cases, the next state would be

$$s' = \langle \mathbf{f}_?, t - t_{a_1}, \{\forall_{1<i<|\mathbf{q}|} t'_{a_{i-1}} = (t_{a_i} - t_{a_1})\} \rangle$$

and processing begins on the new active instance. Solving this MDP would give an optimal solution to the TCSC problem. The size (possibly infinite) of this MDP makes it intractable to solve exactly.

An alternative approach is to estimate the opportunity cost of investing δ time units on the active instance given that $\mathbf{q}$ instances are waiting. Once estimated, this cost augments the incremental time cost component $C_\Delta(\delta|t)$:

$$C_{\Delta'}(\delta|t, \mathbf{q}) = C_\Delta(\delta|t) + C_{OC}(\delta|\mathbf{q}, t)$$

The MDP can be solved with the updated incremental time cost function using the techniques discussed in Section 3.

Note that instances under the model may now have start time $t \neq 0$. Therefore, when solving the MDP, a new start state is introduced where t is unknown. The only action available at this state is a 'fan-out' action which transitions to 'sub start' states with $t = 0$ to $t = T$ with uniform probability. T is chosen to be large enough so that no time consuming attributes are measured from that state. After the policy has been computed and instances are being classified online, we start the measurement policy at the appropriate sub start state. If the instance has been waiting longer than T, the policy starting at sub start state T is followed.

4.2 Estimating opportunity costs

We will examine three methods for estimating the opportunity cost incurred by delaying processing on the instances in the queue.

A very conservative estimate of opportunity cost simply sums the incremental time cost incurred on each instance in the queue, assuming that processing will begin on every one of these instances, simultaneously, after δ time units have elapsed:

$$C_{OC}(\delta|\mathbf{q}, t) = \sum_{i \in \mathbf{q}} C_\Delta(\delta|t - t_{a_i}) \tag{3}$$

A second estimate takes into account that before processing can begin on instance $i+1$, instance i must first be classified. Given a classification/measurement policy π computed on the single instance problem, the estimated time to complete the active instance given the current state can be computed in a bottom up manner. The time remaining distribution from state s given the time remaining distributions of all child states s' reached by following the action $\pi(s)$ is

$$p(t_d|s) = \sum_{s'} p(s'|s, \pi(s))p(t_d - t(s, s')|s')$$

where $t(s, s')$ is the time difference between states s and s'. With this probability, the starting times of all instances in the queue can be estimated:

$$p(t_{s_{i+1}} = t_{d_i} + t_{s_i}) = p(t_{s_i})p(t_{d_i}|\langle \mathbf{f}_?, t_{s_i} - t_{a_i} \rangle) \tag{4}$$

The start time of instance 1 is set to be the current time t of the active instance.

Once the start time distributions of all instances in the queue are computed, the opportunity cost can be estimated as the total estimated time cost incurred by delaying the estimated start times of all instances in the queue by δ:

$$C_{OC}(\delta|\mathbf{q}, t) = \sum_{i \in \mathbf{q}} \sum_{t_{s_i}} p(t_{s_i}) C_\Delta(\delta|t_{s_i} - t_{a_i}) \tag{5}$$

The above methods assume that the only costs incurred by delaying instances will be time costs. In reality, the policy for an instance that begins processing with substantial waiting time already elapsed will generally measure less time consuming attributes to avoid the continually increasing time penalties. We can then expect smaller attribute measurement costs at the expense of higher MC costs.

A third opportunity cost estimation looks at the difference in expected costs incurred for all queued instances before and after an action taking δ time units is taken in the active instance. Given the single instance policy π, the expected total cost incurred C from state s can be computed in a similar manner as the time remaining distributions:

$$p(C|s) = \sum_{s'} p(s'|s, \pi(s))p(C - c(s, s')|s')$$

where $c(s, s')$ is the cost incurred between states s and s'. With this expected cost distribution, the expected cost incurred on all queue instances can be computed, given the current time of the active instance.

$$C_{\langle \mathbf{f}, t \rangle} = \sum_{i \in \mathbf{q}} \sum_{C} \sum_{t_{s_i}} p(t_{s_i}) p(C|\langle \mathbf{f}_?, t_{s_i} - t_{a_i} \rangle) C$$

where the start time distributions $p(t_{s_i})$ are computed using equation 4 starting from time t.

The final opportunity cost estimate is the difference in expected costs when making a transition from state $\langle \mathbf{f}, t \rangle$ to $\langle \mathbf{f}', t' \rangle$ is:

$$C_{OC}(\delta|\mathbf{q}, t) = C_{\langle \mathbf{f}', t' \rangle} - C_{\langle \mathbf{f}, t \rangle} \tag{6}$$

4.3 Queue approximations

An exact representation of the state of the instance queue would contain the arrival times (relative to the arrival time of the active instance) for every waiting instance. Clearly this representation would cause an exponential blowup in the total size of the state space. Instead we choose to represent the queue using simple features describing the state of the queue. In this work, we use two features: the total number of instances in the queue and the average arrival time of those instances. A manually tuned parameter for each feature controls the resolution: as the resolution increases there are less total queue states, which results in smaller searches but lower quality opportunity cost estimates.

5. EXPERIMENTAL

5.1 Setup

We use three data sets in the following experiments: 'breast,' 'pima,' and 'bupa' from the UCI repository [4]. All attributes are discretized into three bins so as to maximize the information gain of the class labels for the entire data set. The class labels for all data sets are binary. The breast

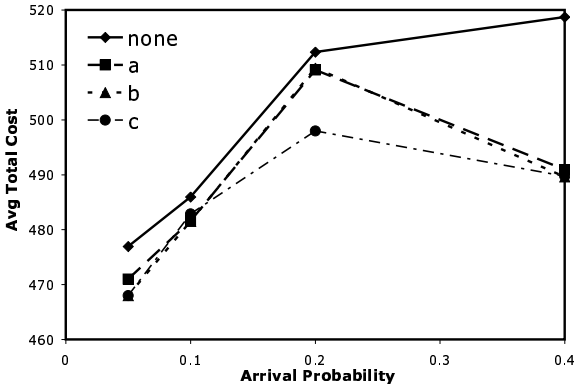


Figure 3: Average total costs for bupa with $\tau = 4$

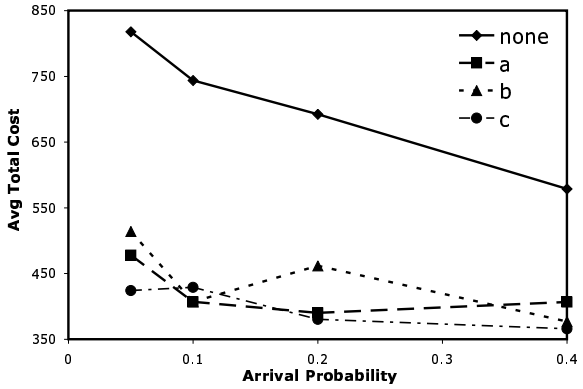


Figure 4: Average total costs for pima with $\tau = 8$]

data set has nine attributes in 683 instances (444 negative, 239 positive). The pima data has eight attributes in 768 instances (500 negative, 268 positive). The bupa data set has five attributes in 345 instances (169 negative. 176 positive).

These data sets all have associated attribute measurement costs, but in these experiments we set w_M to zero so as to ignore attribute measurement costs, as time and MC costs are the primary focus of this study. For each data set the attributes with highest information gain (seven in breast, six in pima, four in bupa) are given time measurement distributions that are normal with mean 6 and standard deviation 2. The remaining attributes can be measured instantly. This assignment of time does not reflect the actual time to measure for these attributes, which for these data sets are unknown. In lieu of that knowledge, we can most effectively study the tradeoff between misclassification and time costs by imposing measurement times on the most predictive attributes. The time cost weight w_T is set to 1. The time cost function $C_T(t) = t^2$. Time is discretized using the method discussed in Section 3.4 on the single instance problem. We use $\tau = 8$ for pima and breast, and $\tau = 4$ for bupa.

MC costs are set up for each instance so that correct predictions have penalty zero, the more frequent class A has penalty 1, and the less frequent class B has misclassification cost penalty $(|A|/|B|)^2$. This reflects the fact that in most CSC problems, the rarer class is more costly to label incorrectly. Misclassification cost weights are set to 1000.

Arrival of new instances in the queue at each time unit is sampled from a Bernoulli distribution where a single instance arrives at each time unit with probability a. This parameter is varied to simulate a variety of loads on the sequential classifier: $a = 0.05, 0.1, 0.2, 0.4$. When solving the MDP using $\tau > 1$, the probability of n instances arriving during the unit of τ is computed from the binomial distribution $P_a(n|\tau)$. The resolution of the queue average waiting time parameter was set to τ, and the number of instances parameter to $\tau/2$.

Five fold stratified cross validation was performed on each data set/arrival rate pair, with a 2/3 vs. 1/3 split between training and testing data.

5.2 Results

Figures 3, 4, and 5 show the average total cost per instance incurred in each of the three data sets for each of the four instance arrival rates. Figure 5 shows a typical breakdown between time and MC costs. The labels 'a', 'b', and 'c' represent the OC estimation methods presented in equations 3, 5, and 6 respectively, while 'none' indicates the performance of the single instance policy that is ignorant of waiting instances. We see that estimating the opportunity cost results in lower cost policies in all data sets and all arrival rates than following the single instance policy. There is no clear winner among the three OC approximation schemes.

One obvious question is why didn't method 'c' perform better? This stems from the fact that the estimated cost distributions in method 'c' are computed using a single instance policy. The distribution of costs in the single instance policy is likely to be significantly different from the distribution that would be seen given the current state of the queue. Costs in the current instance are likely to be larger (due to MC costs) when many instances are waiting. The costs from the single policy distribution are thus often too small, resulting in substantial underestimates of the actual OC. Further research will address this shortcoming by iteratively re-estimating the cost distribution: the policy π_c is first computed using cost and time distributions from the single instance policy, as usual. We can then solve for a new policy using cost and time distributions estimated from π_c. This process can iterate until the policy no longer changes from iteration to iteration.

Additionally, we can observe evidence of overfitting in the bupa and pima cases. For example, the single instance policy on pima incurs lower costs as the arrival rate increases. Because queued instances are ignored, start times for individual instances increase with arrival rate. The policies for higher start times measure less instances, which results in smaller MC costs on test data versus policies that measured many attributes and overfit the training data. Overfitting must be given careful consideration in further research.

6. CONCLUSIONS

Existing cost-sensitive classification algorithms have focused solely on misclassification and attribute measurement costs. Yet for many applications, good responsiveness is a desirable and often necessary property. In this research we have shown novel methods for dealing with time sensitive classification problems in both the single instance and

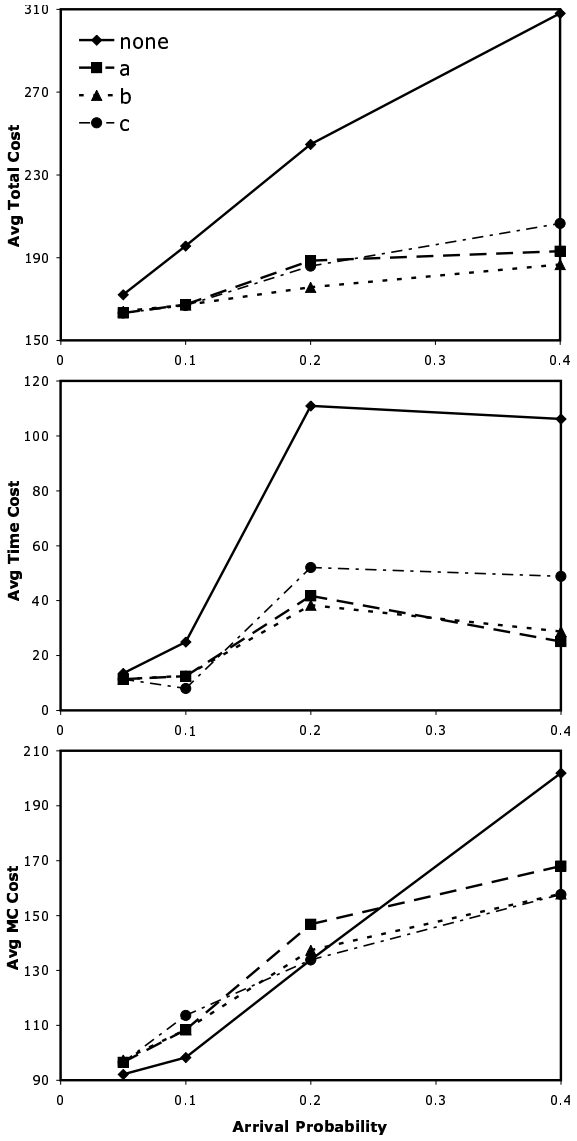


Figure 5: Costs for breast with $\tau = 8$

sequential cases. By intelligently discretizing time and effectively approximating opportunity costs, policies can be computed for a wide range of classification problems.

Avenues for further research involve relaxing the attribute measurement model. These include allowing for stochastic attribute values and repeated measurement of said attributes. A differentiation should also be made between 'blocking' and 'nonblocking' measurement actions. During a blocking measurement, the processor is busy performing the measurement computations. However, during a nonblocking measurement, such as retrieving a document over the Internet, other measurements (including those in subsequent instances) can simultaneously be performed.

7. ACKNOWLEDGMENTS

Support for this work was provided in part by the National Science Foundation under Grant No. 0328601.

8. REFERENCES

[1] A. Arnt and S. Zilberstein. Learning to perform moderation in online forums. In *Proc. IEEE/WIC Intl. Conf. on Web Intelligence*, 2003.

[2] A. Arnt and S. Zilberstein. Attribute measurement policies for time and cost sensitive classification. In *Proc. 4th IEEE International Conference on Data Mining (ICDM'04)*, pages 323–326, 2004.

[3] A. Arnt, S. Zilberstein, J. Allan, and A. I. Mouaddib. Dynamic composition of information retrieval techniques. *Journal of Intelligent Info. Systems*, 2004.

[4] C. Blake and C. Merz. UCI repository of machine learning databases, 1998.

[5] B. Bonet and H. Geffner. Learning sorting and decision trees with POMDPs. In *Proc. 15th Intl. Conf. on Machine Learning*, pages 73–81, 1998.

[6] X. Chai, L. Deng, Q. Yang, and C. X. Ling. Test-cost sensitive naive bayes classification. In *Proc. 4th IEEE International Conference on Data Mining (ICDM'04)*, pages 51–58, 2004.

[7] P. Domingos. Metacost: A general method for making classifiers cost-sensitive. In *Proc. 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 155–164, 1999.

[8] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. AdaCost: misclassification cost-sensitive boosting. In *Proc. 16th Intl. Conf. on Machine Learning*, pages 97–105, 1999.

[9] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. 13th Intl. Joint Conf. on Articial Intelligence*, pages 1022–1027, 1993.

[10] R. Greiner, A. J. Grove, and D. Roth. Learning cost-sensitive active classifiers. *Artificial Intelligence*, 139(2):137–174, 2002.

[11] A. Guo. Decision-theoretic active sensing for autonomous agents. In *Proc. 2nd Intl. Conf. on Computational Intelligence, Robotics, and Autonomous Systems*, 2003.

[12] E. Horvitz and G. Rutledge. Time-dependent utility and action under uncertainty. In *Proc. 7th Conf. on Uncertainty in Artificial Intelligence*, pages 151–158, 1991.

[13] N. J. Nilsson. *Principles of artificial intelligence*. Tioga Publishing Company, 1980.

[14] E. Pednault, N. Abe, and B. Zadrozny. Sequential cost-sensitive decision making with reinforcement learning. In *Proc. 8th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 259–268, 2002.

[15] P. D. Turney. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, 2:369–409, 1995.

[16] V. B. Zubek. Learning diagnostic policies from examples by systematic search. In *Proc. 20th Conf. on Uncertainty in Artificial Intelligence*, pages 27–34, 2004.

[17] V. B. Zubek and T. Dietterich. Pruning improves heuristic search for cost-sensitive learning. In *Proc. 19th Intl. Conf. on Machine Learning*, pages 19–26, 2002.

Improving Classifier Utility by Altering the Misclassification Cost Ratio

Michelle Ciraco, Michael Rogalewski and Gary Weiss

Department of Computer Science
Fordham University
Rose Hill Campus
Bronx, New York 10458

{ciraco, rogalews, gweiss}@cis.fordham.edu

ABSTRACT

This paper examines whether classifier utility can be improved by altering the misclassification cost ratio (the ratio of false positive misclassification costs to false negative misclassification costs) associated with two-class datasets. This is evaluated by varying the cost ratio passed into two cost-sensitive learners and then evaluating the results using the actual (or presumed actual) cost information. Our results indicate that a cost ratio other than the true ratio often maximizes classifier utility. Furthermore, by using a hold out set to identify the "best" cost ratio for learning, we are able to take advantage of this behavior and generate classifiers that outperform the accepted strategy of always using the actual cost information during the learning phase.

Categories and Subject Descriptors

I.2.6 [Artificial Intelligence]: Learning – *Induction*
H.2.8 [Database Management]: Applications - *Data Mining*

General Terms

Algorithms

Keywords

Data mining, machine learning, induction, cost-sensitive learning, utility-based data mining

1. INTRODUCTION

Classifier induction programs have traditionally made a number of assumptions. One such assumption is that the best class distribution for learning is the true underlying distribution—the one that the classifier will eventually be applied to. Recent research has shown that this assumption is often not true and that improved performance can be achieved by using a modified class distribution [6]. This leads us to ask a similar question, "can classifier performance be enhanced by employing cost-sensitive learning and altering the ratio of true misclassification costs?" That is, can we obtain better classifier performance by training with a cost ratio that is not the same as the one that will be used to evaluate the classifier? This is the principle question that we investigate in this paper and, somewhat to our surprise, the answer is "yes".

Much early work on machine learning and data mining did not consider issues of utility, including how a classifier will be used. In particular, misclassification errors typically have non-uniform costs. These misclassification costs are often determined by the class associated with an example, such that for two-class problems, the cost of a false positive prediction is not equal to the cost of a false negative prediction. In these circumstances accuracy is a poor utility metric [4] and for that reason we consider cost information when evaluating the utility of a classifier.

In this paper we vary the cost ratios employed in the learning phase, and analyze how this impacts the performance of the classifier, based on the presumed "actual" cost information associated with the data set. Besides helping us answer the practical question posed earlier, about whether one can improve performance by altering the cost ratio during learning, a secondary benefit is that we gain insight into cost-sensitive learning.

2. BACKGROUND AND TERMINOLOGY

We begin by introducing the basic terminology used to describe the behavior of a classifier for a two-class data set. This is provided in Table 1, which shows a confusion matrix. Note that in Table 1 "ACTUAL" represents the true class for an example, whereas "PREDICTED" represents the label assigned by the classifier. Misclassified examples are those that are either false positives or false negatives and accuracy is defined as: (TP + TN)/(TP + FP + TN + FN).

Table 1: Confusion Matrix Terminology

		ACTUAL	
PREDICTED		Positive class	Negative class
	Positive class	True positive (TP)	False positive (FP)
	Negative class	False negative (FN)	True negative (TN)

One can apply different cost (or profit) values to each of the four possible outcomes listed in Table 1. For cost-sensitive learning, one typically specifies only the costs for the false positives and false negatives and assigns a cost of zero to the true positives and

UBDM '05 , August 21, 2005, Chicago, Illinois, USA.

true negatives. In this case, the construction of the classifier will only be affected by the ratio of the two non-zero misclassification costs. Throughout this paper we specify the cost ratios as the ratio of false positive costs to false negative costs. So, a cost ratio of 1:5 indicates that the cost of a false negative is five times that of a false positive. Holding with established convention, the minority class is considered the positive class. This is often the class of primary interest, as in the case of medical diagnosis (most patients do not have the sickness for which they are being tested). In these situations, the cost assigned to a false negative is generally greater than the cost assigned to a false positive, since this leads to classifiers that correctly classify a greater percentage of the minority-class examples.

In this paper, we are interested in two cost ratios. The actual cost ratio is based on the characteristics of the domain and is typically provided by a domain expert. We refer to this as the evaluation misclassification cost ratio, or, more simply, as the Evaluation Cost Ratio, abbreviated ECR. This name reflects the fact that this cost ratio is used when *evaluating* the utility of the classifier. In particular, we measure the quality of the classifier based on total cost, which is calculated using the equation below (the evaluation cost ratio is FP_{cost}: FN_{cost}).

$$\text{Total cost} = FP * FP_{cost} + FN * FN_{cost} \quad [1]$$

Under normal circumstances, the evaluation cost ratio is passed to the classifier induction program, assuming that it is capable of cost-sensitive learning. However, in this paper we often utilize a different cost ratio in the learning phase. We refer to this ratio as the Training misclassification Cost Ratio, abbreviated TCR. For most of our experiments, TCR ≠ ECR. TCR affects the *construction* of the classifier but not the evaluation of the classifier.

Our paper is organized as follows. In Section 3 we describe our experiments. Results are then described in Section 4 and are discussed in Section 5. Section 6 describes related work and Section 7 presents our conclusions.

3. EXPERIMENTS

3.1 Data Sets

Our empirical study analyzes the relationship between TCR and ECR for the twelve data sets described in Table 2.

Table 2: Description of Data Sets

Data Set	Number of Attributes	Minority Class %	Total Size
Cover-Type*	55	48%	581,012
Adult	15	24%	21,281
Coding	16	50%	20,000
Letter-Vowel*	17	24%	16,122
Blackjack+	5	36%	15,000
Boa1+	69	50%	11,000
Mushroom	23	48%	8,124
Weather+	36	40%	5,597
Splice-Junction*	62	24%	3,175
Move+	11	50%	3029
OCR1	577	18%	2,283

The majority of the twelve data sets are available from the UCI Repository [3]. The remaining ones, identified by a "+" sign in Table 2, were originally supplied by AT&T and have been used in several published research papers. All of the data sets employed in this study are two-class data sets. Those that originally had more than two classes are identified by a "*". These were converted to two-class data sets by assigning one class, typically the least frequently occurring class, to the minority class, and then mapping all remaining classes to a single, majority, class.

3.2 Classifier Induction Programs

The majority of experiments described in this paper utilize C5.0, a commercial decision tree induction tool by Rulequest research. C5.0 is an updated, commercial version of Quinlan's popular C4.5 program [5]. To demonstrate the generality of our results, some experiments were repeated using the decision tree tool incorporated into Enterprise Miner™, an integrated suite of data mining tools from SAS Corporation. Both learners are capable of cost-sensitive learning.

3.3 Experimental Methodology

The primary purpose of the experiments described in this paper is to determine if one can improve classifier learning by using a training cost ratio (TCR) that is not equal to the evaluation cost ratio (ECR). Thus, the experiments in this paper vary the TCR value used during the learning phase and then evaluate the induced classifiers using the evaluation cost ratio. In order to run our experiments, we need to specify the TCR and ECR values for each data set. Because actual cost information is not available for most of the data sets we analyze (either because the costs are not known or not provided), we evaluate a wide range of ECR values rather than just one value. Because we are interested in how different TCR values impact performance, we also evaluate a wide variety of these values. For simplicity, we evaluate the same set of cost ratios for training (TCR) and evaluation (ECR). For each of the twelve datasets we evaluate the following nineteen cost ratios, for training and evaluation: 1:10, 1:9, … 1:2, 1:1, 2:1, …, 9:1, 10:1.

If we only wanted to understand the relationship between TCR and ECR (i.e., how a TCR value affects the performance of a classifier with a specified ECR), or whether a TCR equal to ECR yields the best classifier performance, then we would only need a training set for learning and a test set for classifier evaluation. However, we want to go further; we want to come up with a strategy for selecting, during the learning phase, a good TCR for learning, such that the utility of the induced classifier is improved. We therefore utilize a hold out set to identify the best TCR for learning (i.e., the one that yields the lowest total cost on the hold out set). We refer to this strategy as the TCR identification strategy, or simple TCR identification. The results using the TCR identification strategy are therefore based on using this identified TCR for classifier construction and then evaluating this classifier on the independent test set. Note, however, that since we are also interested in understanding how the choice of TCR and ECR impact classifier performance, we also report results using other TCR values. For the C5.0 experiments in this paper, each data set is partitioned as follows: 40% for the training set, 30% for the hold out set and 30% for the test set.

The above description was for our primary learner, C5.0. Our experiments that use the decision tree tool that is part of Enterprise Miner are slightly different. When using Enterprise Miner, we used the following seven cost ratios for training and evaluation:

1:10, 1:5, 1:2, 1:1, 2:1, 5:1, 10:1. We used fewer cost ratios for our Enterprise Miner experiments because of our time constraints and because it is much harder to automate the experiments when using Enterprise Miner (automation is more difficult because Enterprise Miner employs a graphical user interface, whereas C5.0 uses a command line interface). Furthermore, we did not utilize a hold out set for these experiments. For this reason, for our Enterprise Miner results we focus on how the choice of TCR affects total cost for different ECR values. Nonetheless, as we discuss later, the Enterprise Miner results tend to support the results and conclusions obtained using C5.0.

4. RESULTS

Experiments were run to determine the total costs produced by C5.0 and Enterprise Miner when the TCR and ECR values were varied for the data sets used in our study. In Section 4.1 we provide a detailed analysis of the coding data set. In Section 4.2 we briefly describe the detailed results for some other data sets and provide pointers to an on-line appendix [7] that includes the detailed statistics for all twelve data sets. Section 4.3 then provides the data set specific results using Enterprise Miner. This is followed by Section 4.4, which provides summary results for both C5.0 and Enterprise Miner. Our most important results are presented in Section 4.4.

4.1 Detailed Analysis of the Coding Data Set

We begin by showing the results for the Coding data set using C5.0. We chose to show detailed results of the coding data set because they are representative of the data sets that benefited from the TCR identification strategy. Table 3 shows the confusion matrix values for the Coding data set. Table 3 shows that as the cost ratio of FP to FN changes from 1:10 to 10:1 and false positive predictions become much more costly, then more negative predictions are made and the number of false positives decreases while the number of false negatives increases. We see that once the cost ratio reaches 3:1, every example is classified as a negative example.

Table 3: Confusion Matrix Values for Coding Data Set

TCR	FP	FN	TP	TN
1:10	2073	119	3381	427
1:9	1981	180	3320	519
1:8	1876	234	3266	624
1:7	1812	259	3241	688
1:6	1752	287	3213	748
1:5	1465	482	3018	1035
1:4	1303	634	2866	1197
1:3	1177	767	2733	1323
1:2	933	1104	2396	1567
1:1	592	1676	1824	1908
2:1	245	2502	998	2255
3:1	0	3500	0	2500
…	0	3500	0	2500
10:1	0	3500	0	2500

The utility of the classifier, as measured by total cost, is impacted by the evaluation cost ratio. Figure 4 shows the ECR curves for the coding data set, for ECR values 1:4, 1:3, 3:1 and 4:1 (for readability we do not show the ECR curves for all nineteen evaluated ECR values). Note that all of the curves flatten out at a TCR of 3:1, the point at which all examples are classified as the negative class. The different total cost values are due to the fact that three of the four curves have different false positive costs.

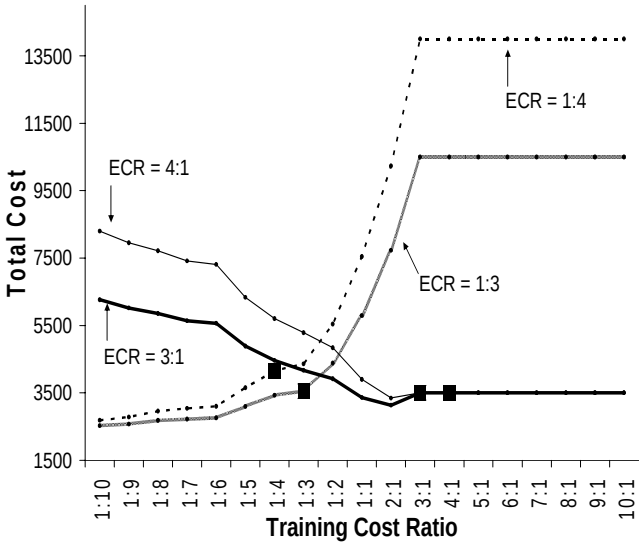


Figure 1: ECR Curves for the Coding Data Set

Each curve in Figure 1 has a square marker to indicate the point at which TCR=ECR. In none of the cases does this point yield the minimum total cost. This is especially true when the ECR is 1:4 or 1:3, in which case the savings appears to be very significant. This indicates that one might be able to improve classifier performance by selecting a TCR that is not equal to the ECR. However, for the TCR identification strategy to yield improved classifier performance, the results on the test data must be similar to the results for the hold out set, which are shown in Figure 1.

This leads us to Table 4, which compares the effectiveness of the TCR identification strategy to two other strategies for selecting the training cost ratio. The default strategy is what is commonly done—always setting the TCR to the ECR. The omniscient strategy involves selecting the TCR that produces the best results on the *test* set and then using that for learning. This strategy is not one that can be fairly used in practice, since it requires an oracle capable of knowing the performance of the classifier on future examples. However, it does provide an upper bound on the possible savings due to the TCR identification strategy and also allows us to see how effective the hold out set results are at identifying the optimal TCR for learning. For TCR identification and the omniscient strategy, the selected TCR value is specified along with the resulting total cost (for the default strategy the TCR is always equal to the ECR and so is not specified explicitly). The last two columns compare TCR identification to the default strategy and the omniscient strategy to the default strategy. The savings that are listed are relative savings. Note that all results in Table 4 are based on the test set. A TCR value of 2-10:1 is equivalent to 2:1-10:1 and means that all nine TCR values in this range yield identical results.

Table 4: Comparison of Three TCR Selection Strategies

ECR	Default	TCR Identification		Omniscient		% Savings (vs. Default)	
	Cost	TCR	Cost	TCR	Cost	TCR Identification	Omniscient
1:10	3664	1:10	3664	1:10	3664	0	0
1:9	3843	1:10	3501	1:10	3501	8.9	8.9
1:8	4052	1:10	3338	1:10	3338	17.6	17.6
1:7	3987	1:10	3175	1:10	3175	20.4	20.4
1:6	3782	1:10	3012	1:10	3012	20.4	20.4
1:5	4200	1:10	2849	1:10	2849	32.2	32.2
1:4	4153	1:10	2686	1:10	2686	35.3	35.3
1:3	3552	1:10	2523	1:10	2523	29.0	29.0
1:2	3229	1:6	2422	1:9	2359	25.0	26.9
1:1	2289	1:4	1972	1:3	1930	13.9	15.7
2:1	2926	1:1	2826	1:1	2826	3.4	3.4
3:1	3500	2:1	3136	2:1	3136	10.4	10.4
4:1	3500	2:1	3346	2:1	3346	4.4	4.4
5:1	3500	2-10:1	3500	2-10:1	3500	0	0
6:1	3500	2-10:1	3500	2-10:1	3500	0	0
7:1	3500	2-10:1	3500	2 -10:1	3500	0	0
8:1	3500	2-10:1	3500	2:-10:1	3500	0	0
9:1	3500	2-10:1	3500	2:-10:1	3500	0	0

The results in Table 4 demonstrate that in many cases very substantial reductions in cost are possible if TCR identification is used instead of the default strategy. The majority of situations where this is not true are when the TCR value is greater than 4:1. Note, however, that we are generally most concerned with the TCR values where the positive (minority-class) examples are given more weight—and this occurs in the range 1:2 to 1:10. Table 4 also shows us that the TCR identification strategy yields the same performance as the omniscient strategy in all but two cases (for ECR values of 1:2 and 1:1). The underlying data indicates that the hold out set identifies the TCR that gives the best test set results for all but these two cases.

4.2 Additional Detailed Results

In section 4.1 we provided the detailed results for the coding data set, using C5.0. In this section we briefly describe the results for the other eleven data sets included in our study, using C5.0. Due to space considerations, the detailed results are included in the on-line appendix [7], available at *http://storm.cis.fordham.edu/~gweiss/ubdm05-appendix.html*. However, summary statistics for all twelve data sets are provided in Section 4.3.

Figures analogous to Figure 1, but for the letter-vowel and adult data set, are provided in Figures A1 and A2 in the on-line Appendix, respectively. Tables analogous to Table 4 are also provided in the on-line appendix for all twelve data sets (Tables A1-A12). The one difference is that the tables in the appendix include one additional field, labeled "Hold Out Set Effectiveness". This field describes how close the hold out data set came to predicting the TCR with the lowest total cost. A value of '1' in that column means that the hold out set successfully predicted the TCR with the lowest total test-set cost. A value of *n* means that the hold out set predicted the TCR with the *n*th lowest total cost, when evaluated on the test data; thus a value of 19 would indicate that the TCR identification strategy identified the worst TCR value (since we evaluate 19 total TCR values). Based on Tables A1-A12, we see that the TCR identification strategy often identifies the TCR with the lowest total cost, and when it does not, it usually comes close.

If we analyze the behavior of the letter-vowel data set, using Figure A1 and Table A4, we see that TCR identification improves classifier performance over the default strategy in almost all cases, but unlike the results for the coding data set, the most substantial improvements are in the range 2:1-9:1, where the false positive cost is greater than the false negative cost. If we look at the results for the adult data set, in Figure A2 and Table A2, we see more ambiguous results. For that data set, TCR identification leads to small reductions in total cost in some cases and small increases in others. For detailed dataset-specific results, see Tables A1-A12 in the on-line appendix.

4.3 Enterprise Miner Results

Figures 2 and 3 show the ECR curves for the Adult data set and the Boa1 data set, respectively, when using SAS Enterprise Miner. As with the majority of C5.0 results, we see that the TCR that yields the best results is not always the one that equals the ECR. For Figure 2, the TCR value that generates the lowest total cost is always either 1:10 or 10:1; thus when the ECR value is not equal to one of these values (for an ECR of 1:5 or 5:1), suboptimal results are produced.

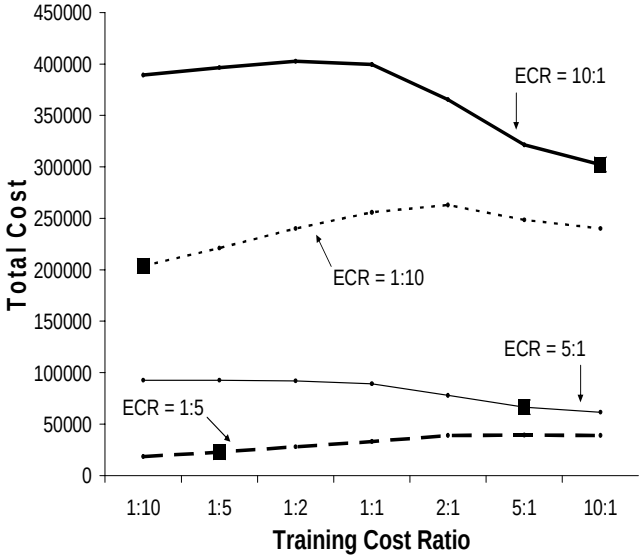


Figure 2: Enterprise Miner ECR Curves for Adult Data Set

Figure 3 shows that suboptimal results occur when the ECR is 1:2, since the best results occur for TCR values of 1:5 and 1:10, and for an ECR of 2:1 the best results occur for a TCR of 5:1 or 10:1. A detailed analysis of the Enterprise Miner results shows that, as with C5.0, the default strategy of setting the TCR equal to the ECR often does not provide the best performance. Section 4.4,

which provides summary results for all C5.0 and Enterprise Miner data sets, will quantify the potential savings.

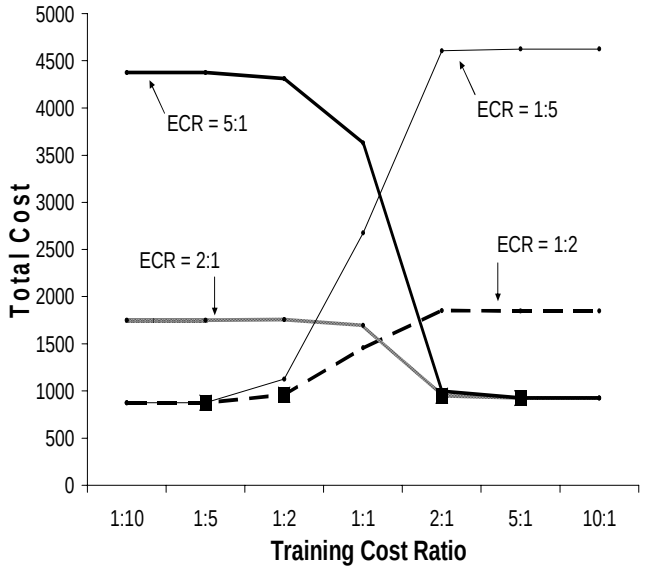


Figure 3: Enterprise Miner ECR Curves for Boa1 Data Set

4.4 Summarized Results over all Data Sets

This section summarizes the results over all of the data sets. The purpose of this section is to quantify the effectiveness of the TCR identification strategy and to identify any patterns or trends in the results. Table 5 shows how TCR identification compares to the default strategy of setting the TCR equal to the ECR, when evaluating classifier performance using total cost. The comparison is based on the performance over all nineteen cost ratio values.

Table 5: Comparison of TCR Identification vs. Default (C5.0)

Data Set	% Avg. Savings (1:10 – 10:1)	% Avg. Savings (1:10 -1:1)	Win/Loss/Tie
Cover-Type	-30.8% (-33.6%)	0.7%	4/13/2
Adult	0.0% (0.0%)	0.3%	6/8/5
Coding	11.6% (18.4%)	20.3%	12/0/7
Letter-Vowel	7.4% (8.3%)	1.4%	15/2/2
Blackjack	-0.2% (-1.1%)	-0.4%	1/3/15
Boa1	0.0% (0.0%)	0.0%	0/0/19
Mushroom	47.4% (100.0%)	60.0%	9/0/10
Weather	-0.2% (-0.4%)	0.3%	4/8/7
Network1	5.4% (7.3%)	5.7%	12/2/5
Splice-Junction	2.0% (2.2%)	-5.9%	8/9/2
Move	3.9% (9.3%)	3.4%	4/4/11
OCR1	23.4% (24.7%)	11.4%	15/3/1
TOTAL	5.8% (11.3%)	8.1%	90/52/105

The second column in Table 5 specifies the savings (i.e., relative reduction in cost) averaged over the full range of nineteen cost ratios. In parenthesis next to this number is the average savings if all ties between the TCR identification and default strategies are omitted. The third column shows the average savings over the ten cost ratios from 1:10 – 1:1. We break down the results for these cost ratios separately because, as mentioned earlier, one is typically most interested in this range since it leads to improved performance on the minority class, which otherwise might rarely be predicted. The last column in Table 5 provides the win/loss/tie record for each data set over the nineteen cost ratios.

Table 5 shows that the TCR identification performs substantially better than the default strategy of always using the ECR for training, although the performance varies widely for each data set. For the five data sets highlighted in bold and for which the row is shaded, TCR identification greatly outperforms the default strategy, in that it performs better for almost all of the nineteen cost ratios and consistently produces substantial savings. The four data sets that produced essentially neutral results are underlined. These include the boa1 data set, which produced perfectly identical results to the default strategy for all nineteen cost ratios, as well as the adult, blackjack and weather data set, which produced either very slight positive or negative savings. The move and splice-junction data sets can be considered moderate wins, when the entire range of cost ratios is considered. Finally, the Cover-type data set is the only true loss, over all nineteen cost ratios. However, for this data set the TCR identification strategy provides no loss (i.e., a very slight win) over the TCR range we are most interested in (1:10-1:1). If we focus exclusively on the range 1:10 – 1:1, then we see that there is only one moderate loss (splice-junction), while there are still many substantial wins.

In summary, our results indicate that of the twelve cases, averaged over the nineteen cost ratios, TCR identification produces substantially better results than the default strategy in five cases and worse results in one case; the remaining six cases produce ambiguous results. If we focus on the more important 1:10-1:1 TCR range, then the one significant loss is eliminated and the only loss at all is a moderate one, for the splice-junction data set. Due to space considerations, Table 5 does not compare the TCR identification strategy to the omniscient strategy, but that information is provided, at a more detailed level, in tables A1-A12 in the on-line appendix.

We now turn to the summary results for Enterprise Miner (EM). Because we did not utilize a hold out set for identifying the best TCR to use, we can only compare the omniscient strategy to the default strategy. This will put an upper bound on the savings that are possible. However, we expect that most of the savings we see with the omniscient strategy could be realized if a hold out set were used, since this is what we saw with C5.0. The results of this comparison are shown in Table 6.

Table 6: Comparison of Omniscient vs. Default Strategy (EM)

Data Set	% Avg. Savings
Cover-Type	22.7%
Adult	11.1%
Boa1	4.1%
Weather	19.1%
Network2	46.2%
Splice-Junction	1.9%
Move	18.7%
TOTAL	17.7%

Table 6 clearly demonstrates that the best TCR for learning is not equal to the ECR. While we have not demonstrated that these savings can be realized, we feel that the use of a hold out set would allow us to realize most of these savings.

5. DISCUSSION

This paper looks at two questions: does the evaluation cost ratio always produce the best results when used for training and 2) can we identify an alternative cost ratio for training such that classifier performance is improved. Our results indicate that the answer to the first question is "no", the best cost ratio for training is generally not the evaluation cost ratio and that the answer to the second question is "yes", we can identify an cost ratio for learning that outperforms the evaluation cost ratio.

Our main results show only one substantial failure for the TCR identification strategy, for the cover-type data set when all nineteen cost ratios were considered. Why were the results poor in this case? The detailed results for this data set, available in Table A1 in the on-line appendix, shows us the reason is that the hold out set did not effectively identify the best TCR, with respect to the test set performance, when the cost ratio was between 1:4 and 10:1. Even in the cases where our strategy did poorly, the omniscient strategy performed well. Our results may have been better for this data set, and for all data sets, had we employed multiple runs and averaged the results. In the only other failure, for the splice-junction data set for cost ratios 1:10 – 1:1, the failure resulted because the best TCR for learning, based on the test set, was 1:6, and the TCR identification strategy identified the best TCR (using the hold out data) as 1:7 (see Table A10). Thus, this failure was due to a slight difference between the selected and optimal TCR values.

Our results with Enterprise Miner support the conclusions reached using C5.0. Enterprise Miner also performed better when the TCR was set to a value other than the ECR. In the future we plan to incorporate the use of a hold out set to show that much of the potential reduction in total cost that is available can be realized.

Our study evaluated nineteen different evaluation cost ratios. An ECR of 1:1 is special, since it corresponds to the accuracy metric, which is very commonly used and therefore is, most likely, the metric for which most classifiers are optimized. Thus, it is worthwhile to analyze our results for an ECR of 1:1. For this ECR, for five of the twelve data sets a TCR of 1:1 provides the best performance and for three data sets the best performance is achieved with a TCR of 2:1. For the remaining four data sets, the best performance is achieved with a cost ratio of 1:3, 1:4, 3:1, or 4:1. No other ECR performed as well using the default strategy. For instance, with an ECR of 4:1, the best TCR ranged from 1:4 to 9:1 and a TCR of 4:1 yielded the lowest total cost only for one data set. Thus, our results indicate that the default strategy of setting the TCR to the ECR *is* generally quite appropriate for maximizing classifier accuracy.

This paper has focused on the issue of how the choice of the training cost ratio impacts classifier performance, as measured by total cost. Because one can effectively modify the misclassification cost ratio for a data set by altering the class distribution of the training set[1], our results have some additional implications. Specifically, our results seem to imply that one should often be able to benefit by altering the class distribution of the domain. In fact, this conclusion is seemingly supported by existing research [6], which showed that the naturally occurring class distribution often does not provide the best classifier performance. This connection, however, is not as clear. The research previously noted [6] showed that altering the class distribution improves learning when the training set size is held fixed; it did not examine the case where training examples were added. Also, the results in this paper can only be used to conclude that altering the class distribution will improve learning only if one can draw new training examples. However, most research on altering the class distribution of a training set utilizes sampling to change the distribution. Sampling involves either discarding examples of one class (undersampling) or duplicating examples of one class (oversampling). Both of these methods have drawbacks; undersampling throws away potentially useful data and oversampling may lead to overfitting the data. Thus, even though there is an equivalency between changing the cost ratio and altering the training sets class distribution, it is not clear that our results show that changing the class distribution of the training data will necessarily improve classifier performance.

Finally, we turn to the question of why the best cost ratio for training is not always the evaluation cost ratio. It is useful to start with accuracy, which is based on an ECR of 1:1. Our results indicate that in this case the default strategy of setting TCR to ECR performs quite well. This is encouraging, since it doesn't seem likely that a classifier would consistently perform sub-optimally for accuracy, the metric that most classifiers were originally optimized for. However, that still leaves us with the question as to why C5.0 and Enterprise Miner perform best for a training cost ratio other than the evaluation cost ratio when there are non-uniform misclassification costs. The only apparent answer is that these cost-sensitive learners do not handle non-uniform misclassification costs well. We are not sure why this is true, but feel that is an important area for future research. The only insight we have is that, for decision tree learning, it may be more difficult to label a leaf node with the correct class for non-uniform error costs, where the class probability threshold will not be 0.5. Our conjecture is that as the decision threshold moves away from 0.5, it becomes more difficult to accurately label the node, especially if there are only a small number of training examples. However, the evaluation of this conjecture is left for future work.

6. RELATED WORK

Research by Weiss and Provost [6] looked at the impact of class distribution on learning, when training data is costly and one is only able to purchase a fixed number of examples. That research found that improved classifier performance was possible by using a class distribution other than then naturally occurring distribution. That article then went on to show that one could use an adaptive, progressive, sampling strategy to identify a "good" class distribution for learning and thus actually improve classification performance. In many ways the research in this paper parallels that research, except that here we alter the cost ratio of the training set instead of its class distribution. In fact the results here might appear to be implied by those earlier results, since altering the class distribution of the training data is, as Elkan [2] pointed out, in some ways equivalent to altering the cost ratio. However, there is an important difference. In the earlier work, when the class distribution was changed, measures were taken to adjust the

[1] For example, to impose a misclassification cost ratio of 1:2 without using a cost-sensitive learner, one need only increase the ratio of positive to negative examples by a factor of 2 [2].

classifier so that it was not biased to favor the over-sampled class. Thus, changing the class distribution was not equivalent to altering the cost ratio, and the results in this paper are not implied by that earlier work.

Domingos [1] developed a method called Metacost that can transform a wide variety of error-based classifiers into cost-sensitive classifiers. Metacost re-labels the training set examples with their optimal class, or the class that minimizes conditional risk, and then relearns the classifiers with the modified training data. While both Metacost and the TCR identification method alter the training data or how the classifier treats the training data, the methods are incomparable because 1) TCR identification requires a cost-sensitive learner and 2) TCR identification identifies a cost ratio that is different from the one that would be derived from the cost matrix, while Metacost uses the one derived from the cost matrix (i.e., the ECR). However, Metacost could be adapted to do something similar to TCR identification and alter the class probability thresholds so that TCR ≠ ECR for the entire classifier, or could even use different class probability thresholds for different parts of the classifier (e.g., different rules).

There is great deal of additional research on cost-sensitive learning, but we are not aware of any studies that examine the impact of the changing the cost ratio during the learning process in order to assess the impact that this has on the quality of the induced classifier. This can be contrasted to the wealth of research that examines how changing the class distribution can improve learning from skewed class distributions.

7. CONCLUSION

In this paper we demonstrated that the performance of a classifier, when the misclassification costs are not uniform, is generally maximized when the training cost ratio is set equal to a value other than the evaluation cost ratio. This was shown for commercially available classifier induction programs, C5.0 and Enterprise Miner. We furthermore showed that, for C5.0, we could successfully identify good training cost ratios for learning, using a hold out test set, such that classifier performance could be improved over the standard practice of setting the training cost ratio to the evaluation cost ratio. Our results showed that for five of twelve data sets, the TCR identification strategy led to substantial reductions in total cost and only in one case did it lead to substantial increases in total cost. Furthermore, when the evaluation of the TCR evaluation strategy considered only cost ratios between 1:10 and 1:1, this one loss disappeared. The implications of our results are significant—that classifier induction programs may perform poorly—unnecessarily poorly—when handling non-uniform misclassification costs. These results are particularly notable since we analyzed popular state-of-the-art commercial classifiers. The "wrapper" approach we introduced in this paper can be used to overcome some of the weaknesses of these cost-sensitive learners.

The fact that the TCR identification strategy yields a net improvement in classifier performance indicates that the induced learners exhibit a systematic bias (altering the cost ratio will always increase the frequency that one class is predicted over the other). The main areas for future work are to better understand why these cost-sensitive learners perform sub-optimally and how this behavior can be remedied. Other areas for future work include analyzing additional cost-sensitive learners, analyzing data sets which exhibit more extreme class imbalance and analyzing additional data sets.

8. REFERENCES

[1] Domingos, P. Metacost: A General Method for Making Classifiers Cost-Sensitive. *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining, 155-164, 1999.*

[2] Elkan, C. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 973-978, 2001.

[3] Hettich, S., Blake, C. and Merz, C. UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. University of California, Department of Information and Computer Science, 1998.

[4] Provost, F. Fawcett, T., and Kohavi, R. The case against accuracy estimation for comparing classifiers. *In Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[5] Quinlan, J. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco, CA, 1993.

[6] Weiss, G. and Provost, F. Learning when training data are costly: the effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 19: 315-354.

[7] Ciraco, M., Rogalewski, M., and Weiss, G. Appendix I: Additional results from improving classifier utility by altering the misclassification cost ratio, 2005. *http://storm.cis.fordham.edu/~gweiss/ubdm05-appendix.html*

One-Benefit learning: Cost-sensitive learning with restricted cost information

Bianca Zadrozny
IBM T.J. Watson Research Center
1101 Kitchawan Road, Route 134
Yorktown Heights, NY 10598
zadrozny@us.ibm.com

ABSTRACT

This paper presents a new formulation for cost-sensitive learning that we call the One-Benefit formulation. Instead of having the correct label for each training example as in the standard classifier learning formulation, in this formulation we have one possible label for each example (which may not be the correct one) and the benefit (or cost) associated with that label. The goal of learning in this formulation is to find the classifier that maximizes the expected benefit of the labelling using only these examples. We present a reduction from One-Benefit learning to standard classifier learning that allows us to use any existing error-minimizing classifier learner to maximize the expected benefit in this formulation by correctly weighting the examples. We also show how to evaluate a classifier using test examples for which we only the benefit for one of the labels. We present preliminary experimental results using a synthetic data generator that allows us to test both our learning method and our evaluation method.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning - Induction; H.2.8 [**Database Management**]: Applications - Data Mining

General Terms

Algorithms

Keywords

data mining, cost-sensitive learning

1. INTRODUCTION

In standard classifier learning, we are given a training set of examples of the form (x, y), where x is a feature vector and y is a class label. These examples are assumed (at least, implicitly) to be drawn independently from a fixed distribution D with domain $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is a feature space and $\mathcal{Y}$ is a (discrete) class label space. The goal is to learn a classifier $h : \mathcal{X} \rightarrow \mathcal{Y}$ that minimizes the expected error rate on examples drawn from D, given by

$$E_{x,y\sim D}[I(h(x) \neq y)] \quad (1)$$

where $I(\cdot)$ is the indicator function that has value 1 in case its argument is true and 0 otherwise.

The traditional formulation assumes that all errors are equally costly. However, this is not true for many domains for which one would like to obtain classifiers. For example:

- In one-to-one marketing, the cost of making an offer to a person who does not respond is small compared to the cost of not contacting a person who would respond.
- In medicine, the cost of prescribing a drug to an allergic patient can be much higher than the cost of not prescribing the drug to a nonallergic patient.
- In image or text retrieval, the cost of not displaying a relevant item may be lower or higher than the cost of displaying an irrelevant item.

One extension to the standard classifier learning formulation that has received considerable attention in the past few years is the cost matrix formulation [2, 4, 3]. In this formulation, we specify a cost matrix C for the domain in which we would like to learn a classifier. If there are k classes, the cost matrix is a $k \times k$ matrix of real values. Each entry $C(i, j)$ gives the cost of predicting class i for an example whose actual class is j. Now, instead of minimizing the error rate given by equation 1, we would like to find a classifier h that minimizes the expected cost of the labeling, given by

$$E_{x,y\sim D}[C(h(x), y)]. \quad (2)$$

Research on cost-sensitive learning has traditionally been couched in terms of costs, as opposed to benefits or rewards. However, in many domains, it is easier to talk consistently about benefits than about costs. The reason is that all benefits are straightforward cash flows relative to a baseline wealth of \$0, while some costs are counterfactual opportunity costs [3]. Instead of specifying a cost matrix, we can equivalently specify a benefit matrix B, where each entry of the matrix describes the benefit (or reward) of predicting

class i for an example whose actual class is j. Then, instead of minimizing 2, we maximize

$$E_{x,y\sim D}[B(h(x), y)].$$

The benefit matrix formulation assumes that the benefits are fixed, i.e., that they only depend on the predicted and actual classes, but not on the example itself. However, more often than not, benefits in real-world domains are example-dependent. For example, in direct marketing, the benefit of classifying a respondent correctly depends on the profit that the customer generates. Similarly, in credit card fraud detection, the benefit of correctly identifying a fraudulent transaction depends on the amount of the transaction.

Zadrozny and Elkan [6] extend the benefit matrix formulation to the example-dependent case by allowing each entry to depend on the particular feature vector x. In this case, the benefits are given by a function $B(i, j, x)$, where i is the predicted class, j is the actual class and x is the feature vector of the example. Accordingly, we would now like to find a classifier h that maximizes the expected benefit of the labeling, given by

$$E_{x,y\sim D}[B(h(x), y, x)]. \tag{3}$$

Because the benefit matrix formulation assumes that the benefits are fixed for all examples, it also implicitly assumes that they are known in advance. However, when we allow example-dependent benefits, it might be the case that the benefits are not known for all of the possible labels of all the training examples. An example of an application where this is the case is direct marketing. In this case, x is the description of a customer (which may include, for example, past purchases) and y is a marketing action (such as mailing a catalog or a coupon). The benefit for each y is the profit attained if the customer responds to the action or \$0 if he does not respond to it. Therefore, in order to measure the benefits for each possible y, it would be necessary to take all possible actions with the same customer, which is not feasible[1].

A similar situation occurs in medical treatment, here x is the description of a patient and y is a possible treatment. Usually only one treatment is assigned to each patient, so we only have information about the benefit of one treatment per person. Therefore, the example-dependent formulation is not directly applicable here.

In this paper, we introduce a new formulation of the cost-sensitive learning problem that does not require that all of the benefits (or costs) for each of the labels are known for each training example. We call this formulation the One-Benefit formulation. We present an algorithm for learning under this formulation that is in fact a reduction to standard classifier learning. In other words, it is an algorithm that transforms a cost-sensitive learning problem of this type into a standard classifier learning problem. We call this reduction the One-Benefit reduction. The main advantage of a reduction is that it allows the use of any classifier learner as a black box. For other details and advantages of reductions, see Beygelzimer et al. [1]. We also present a method for evaluating classifiers under the One-Benefit formulation. Finally, we show some preliminary results on a synthetic dataset.

[1]Note that previous work in cost-sensitive learning applied to direct marketing[6] dealt with a special case in which there were only two possible actions (mail/not mail). The "not mail" action had a fixed benefit of zero and all the customers in the training set had been mailed. Therefore, the benefits for each label were known for each training example

2. CLASSIFIER LEARNING UNDER THE ONE-BENEFIT FORMULATION

We assume that we have m training examples (x, y, b) drawn from a joint distribution D with domain $\mathcal{X} \times \mathcal{Y} \times \mathcal{B}$ where $\mathcal{X}$ is an (arbitrary) space, $\mathcal{Y}$ is a (discrete) label space and $\mathcal{B}$ is a (nonnegative, real) benefit space, where the benefit of assigning label y to example x is is given by a stochastic function $B : \mathcal{X} \times \mathcal{Y} \to [0, \infty]$ (that is $b \sim B(x, y)$).

We call this formulation the One-Benefit formulation. Note that instead of having the correct label for each training example as in the standard classifier learning formulation, in this formulation we have one possible label (which may not be the correct one) and the benefit associated with that label. If we know the benefit of more than one possible label, we can create one example per benefit.

Our goal is to find the classifier $h : \mathcal{X} \to \mathcal{Y}$ that maximizes the expected value of the benefit given by

$$E_{x\sim D}[B(x, h(x))] \tag{4}$$

using only the available examples.

We also want to be able to evaluate an existing classifier using only examples of the form (x, y, b). That is, we want to be able to obtain an estimate of the expected benefit of the classifier (given by equation 4).

Standard classifier learners try to find H to maximize the accuracy

$$\frac{1}{m} \sum_{(x,y)} I(H(x) = y)$$

but, according to the translation theorem in Zadrozny et al. [8], can be made to maximize a weighted loss

$$\frac{1}{m} \sum_{(x,y,w)} wI(H(x) = y), \tag{5}$$

where w is a importance weight given to each example.

The following theorem shows that the expected benefit in (4) can be rewritten in a way that allows us to use a classifier learner that maximizes (5) to learn the classifier h.

THEOREM 1. *For all distributions, D, for any deterministic function, $h : \mathcal{X} \to \mathcal{Y}$ and for any stochastic function $B : \mathcal{X} \times \mathcal{Y} \to [0, \infty]$, if we assume that $P(y|x) > 0 \ \forall x, y$ then*

$$E_D[B(x, h(x))] = E_D\left[\frac{b}{P(y|x)} I(h(x) = y)\right]$$

PROOF.

$$
\begin{aligned}
& E_D\left[\tfrac{b}{P(y|x)} I(h(x)=y)\right] \\
&= E_{x,y,b\sim D}\left[\tfrac{B(x,y)}{P(y|x)} I(h(x)=y)\right] \\
&= E_D\left[\tfrac{B(x,y)}{P(y|x)}\Big| h(x)=y\right] P(h(x)=y) \\
&= E_D\left[\tfrac{b(x,h(x))}{P(h(x)=y|x)}\Big| h(x)=y\right] P(h(x)=y) \\
&= \int_x \tfrac{B(x,h(x))}{P(h(x)=y|x)} P(x|h(x)=y)P(h(x)=y)dx \\
&= \int_x \tfrac{B(x,h(x))}{P(h(x)=y|x)} P(x,h(x)=y)dx \\
&= \int_x B(x,h(x))P(x)dx \\
&= E_D[B(x,h(x))]
\end{aligned}
$$

□

From this theorem, it follows that

$$\frac{1}{m}\sum_{(x,y,b)} \frac{b}{P(y|x)} I(h(x)=y) \tag{6}$$

is an unbiased empirical estimate of the expected benefit of classifier h. Thus, if we know $P(y|x)$, that is, the probability that label y is assigned to example x in the training data, we can use a classifier learner to learn the classifier from these examples. Looking back at (5) we see that we simply have to weigh each example (x, y, b) by $\frac{b}{P(y|x)}$.

Note that the theorem holds only if $\forall x, y\ P(y|x) > 0$, that is, in order to guarantee convergence to the optimal classifier, we require that in the training data each label have non-zero probability of being assigned to each example. However, the reduction degrades gracefully even when this is not the case if we define that

$$\frac{I(h(x)=y)}{P(y|x)} = 0$$

when $I(H(x) = y) = 0$ and $P(y|x) = 0$. In this case, it is easy to see that the reduction will converge to a classifier that is optimal, except that label y is not allowed for example x.

This theorem demonstrates that in this formulation we have to account both for the benefits (given by the numerator in the first factor of equation 6) and for the fact that the labels are not assigned at random to the examples (given by the denominator in the first factor of equation 6).

Zadrozny et al.[8] showed that learning from a weighted distribution of examples is not straightforward with many classifier learners but that "costing", a method based on rejection sampling, achieves good results in practice. For this reason, we recommend using costing here, where instead of using misclassification costs as weights we use the ratio $\frac{b}{P(y|x)}$ as a weight for each example (x, y, b). Another option is to use learners that accept weights directly, such as naive Bayes and SVM.

In practice, we may not know the probabilities $P(y|x)$ for the training examples in advance. However, we can estimate these using the available training data by applying a classifier learning method that estimates conditional probabilities or by transforming the outputs of a classifier into accurate probability estimates [7].

One-Benefit Reduction(Training Set $S = (x, y, b)$)

1. **Learn a model for $P(y|x)$ using S.**
2. **Calculate a weight for each example (x, y, b): $w = \frac{b}{P(y|x)}$**
3. **Learn a classifier h using a cost-sensitive learner on $S' = (x, y, w)$.**
4. **Output h.**

Table 1: The One-Benefit Reduction.

Table 1 shows the pseudo-code for this reduction. Given a training set of the form (x, y, b), we first learn a model for $P(y|x)$. This can be accomplished by using a classifier learner that outputs class membership probability estimates. We then calculate weights for each example (x, y, b) by dividing b by $P(y|x)$. We can now use a cost-sensitive learning method that takes examples (x, y, w) as input, such as the ones presented in Zadrozny et al.[8] to learn a classifier that maximizes the expected benefit.

3. CLASSIFIER EVALUATION UNDER THE ONE-BENEFIT FORMULATION

The most obvious way to estimate the expected benefit of a classifier h in the One-Benefit formulation is to select the test examples (x, y, b) for which $h(x) = y$, since these are the examples that tell us the benefit of the label predicted by the classifier. Then, we can average the benefits b from each of the selected examples to obtain an estimate of the expected benefit of the classifier. This is reasonable if the number of possible labels is small, so that we can obtain enough examples that agree with the classifier.

Nonetheless, even when this condition is true, selecting the examples in this manner may result in a biased estimate of the expected benefit of the classifier. This can happen because the examples are being selected according to a criteria that is not necessarily independent of the feature vector x. For example, in the direct marketing case, each example x describes a particular customer. If we have a classifier that is more likely to agree with the data for the "rich customers" (who presumably tend to buy more), by using this kind of evaluation we may think it is a very good classifier. However, if we apply the classifier to the general population of customers it may not perform as well.

As we did for learning, we can use theorem 1 for evaluation. According to theorem 1, the expected benefit of a classifier h is given by

$$E_D[B(x,h(x))] = E_D\left[\frac{b}{P(y|x)} I(h(x)=y)\right].$$

Therefore, an empirical estimate of the expected benefit of the classifier is the following sum for a set of m test examples:

$$\frac{1}{m}\sum_{(x,y,b)} \frac{b}{P(y|x)} I(h(x)=y),$$

that is, we sum over the test examples whose labels agree with the label selected by the classifier h, but we weigh

them by the ratio of their benefit divided by the conditional probability that the label appears in the data. Again, the probabilities $P(y|x)$ have to be estimated (and validated) using the training data.

4. EXPERIMENTAL RESULTS

We present experimental results using a synthetic data generator that is a modification of the IBM Quest Synthetic Data Generation Code for classification (Quest)[5]. Quest randomly generates examples for a person data set in which each person has the nine attributes described below.

- `Salary`: uniformly distributed between 20000 and 150000.
- `Commission`: if `Salary` $\geq$ 75000, `Commission` $= 0$, else uniformly distributed between 10000 and 75000.
- `Age`: uniformly chosen from 60 integer values (20 to 80).
- `Education`: uniformly chosen from 4 integer values.
- `CarMake`: uniformly chosen from 20 integer values.
- `ZipCode`: uniformly chosen from 9 integer values.
- `HouseValue`: uniformly distributed from $50000\,k$ to $150000\,k$, where $0 \leq k \leq 9$ and depends on the `ZipCode`.
- `YearsOwned`: uniformly distributed from 1 to 30.
- `Loan`: uniformly distributed between 0 and 500000.

In the original Quest generation code, there are a series of classification functions of increasing complexity that used the above attributes to classify people into different groups. After determining the values of different attributes of an example and assigning it a group label according to the classification function, the values for non-categorical attributes are perturbed. If the value of an attribute A for an example x is v and the range of values of A is a, then the value of A for x after perturbation becomes $v + r * a$, where r is a uniform random variable between -0.5 and +0.5.

We modified Quest to include both label generation functions and benefit generation functions. These are used to generate examples of the form (x, y, b), where x is a person described by the attributes above, y is a label describing a marketing action taken for that person (such as mailing a particular catalog) and b is the benefit received after the action described by y is taken (such as the amount purchased from the catalog).

We use two different label generation functions that were created based on classification functions already implemented in Quest. The functions are non-deterministic. For each example, they specify a probability for each label. The labels are then randomly drawn according to these probabilities. The label generation functions are shown in Table 2.

Given an example x and a label y, the benefit generation function determines a benefit for labeling person x as belonging to class y. We use two different benefit generation functions, which are shown in Table 3.

Labelling Function 1	Labelling Function 2

`if (Age< 40)`
`  if (50000≤Salary≤100000)`
`  probClass1 = 0.3;`
`  else`
`  probClass1 = 0.7;`
`else`
`if (40 ≤Age< 60)`
`  if (75000≤Salary≤125000)`
`  probClass1 = 0.1;`
`  else`
`  probClass1 = 0.9;`
`else`
`if (25000≤Salary≤75000)`
`  probClass1 = 0.4;`
`  else`
`  probClass1 = 0.6;`
 	
`if (probClass1>rand())`
`  y=1;`
`else`
`  y=0;`
	`if (Age<40)`
	`  probClass1 = 0.2;`
	`else`
	`if (40≤Age<60)`
	`  probClass1 = 0.8;`
	`else`
	`  probClass1 = 0.2;`
	
	`if (probClass1>rand())`
	`  y=1;`
	`else`
	`  y=0;`

Table 2: Label generation functions. The function `rand()` generates a random number drawn uniformly from the interval $[0, 1]$.

Benefit Function 1	Benefit Function 2

`if (YearsOwned<20)`
`  equity = 0;`
`else`
`  equity = 0.1∗YearsOwned`
`  - 2;`
 	
`disposable = 2∗Salary/3`
`  - Loan/5`
`  + 5000∗Education`
`  + equity/5`
`  - 10000;`
 	
`if (disposable>0)`
`  if (y = 0)`
`  b = randn(250,20);`
`  else`
`  b = randn(200,20);`
`else`
`  if (y = 0)`
`  b = randn(80,20);`
`  else`
`  b = randn(150,20);`
	`if (Age<40)`
	`  if (Education∈ {0,1})`
	`  if (y= 0)`
	`  b = randn(100,20);`
	`  else`
	`  b = randn(80,20);`
	`  else`
	`  if (y= 0)`
	`  b = randn(50,20);`
	`  else`
	`  b = randn(120,20);`
	`else`
	`if (40 ≤Age< 60)`
	`  if (Education∈ {1,2,3})`
	`  if (y= 0)`
	`  b = randn(100,20);`
	`  else`
	`  b = randn(150,20);`
	`  else`
	`  if (y= 0)`
	`  b = randn(120,20);`
	`  else`
	`  b = randn(140,20);`
	`else`
	`  if (Education∈ {2,3,4})`
	`  if (y= 0)`
	`  b = randn(90,20);`
	`  else`
	`  b = randn(70,20);`
	`  else`
	`  if (y= 0)`
	`  b = randn(50,20);`
	`  else`
	`  b = randn(70,20);`

Table 3: Benefit generation functions. The function `randn(`μ`,`σ`)` generates a random number drawn from a Gaussian with mean μ and standard deviation σ.

The advantage of using a synthetic data generator is that we can evaluate any classifier by generating the benefits for each possible action, which is not possible with real data. In the real-world, we cannot "reset" customers to the same state and mail a different catalog as if the customer had not received the first one, but we can do this with Quest.

We applied the One-Benefit reduction 1 to three training sets of 50000 examples generated using three settings of the label and benefit generation functions (Label1-Benefit1, Label1-Benefit2 and Label2-Benefit2).For obtaining the estimates of $P(y|x)$ we use naive Bayes followed by the PAV calibration algorithm [7]. For learning the main classifier, we use three methods:

- weighted Naive Bayes,
- costing with Naive Bayes as base learner,
- costing with C4.5 as base learner.

For evaluating the classifiers, we use the simulator to generate three test sets of 50000 examples. We evaluate the classifiers using three methods:

- **True:** use the generator to obtain benefit values for the two labels for each test example and average the benefits for the labels chosen by the classifier (unbiased but unrealistic in a data mining setting).
- **Biased:** select only the test examples that agree with the classifier and average the benefits for those examples.
- **Corrected:** select only the test examples that agree with the classifier and use the bias correction method proposed in Section 3 to calculate the expected benefit of the classifier (unbiased and realistic).

The probabilities $P(y|x)$ necessary for the bias correction method are obtained by applying the model learned on the training set to the test examples.

Table 4 summarizes the results obtained. For comparison purposes, it also includes the average benefit of the training labels, of the best possible labelling and of the worst possible labelling.

In all cases, the One-Benefit reduction improves upon the training labels. Furthermore, by comparing the two settings with the same benefit function and different training labelling, we see that the particular training labelling does not greatly influence the final result. The different learning algorithms (weighted NB, costing NB and costing C4.5) in general led to classifiers that are equally good, except that costing C4.5 resulted in a better classifier for the settings with Benefit2.

Whereas using only the selected examples to evaluate the classifier yields incorrect estimates of the value of the classifier, the evaluation using the bias correction method yields results that are very close to the true (but unrealistic) evaluation.

Labelling Function 1 - Benefit Function 1

	Evaluation Method		
Classifier	True	Biased	Corrected
worst possible	146.94	-	-
best possible	206.31	-	-
training labels	178.06	-	-
weighted NB	192.74	180.74	191.86
costing NB	192.30	180.80	191.80
costing C.45	190.94	180.23	190.78

Labelling Function 1 - Benefit Function 2

	Evaluation Method		
Classifier	True	Biased	Corrected
worst possible	73.87	-	-
best possible	116.01	-	-
training labels	102.99	-	-
weighted NB	107.21	115.65	107.85
costing NB	107.06	115.53	107.76
costing C.45	112.45	120.30	112.56

Labelling Function 2 - Benefit Function 2

	Evaluation Method		
Classifier	True	Biased	Corrected
worst possible	73.87	-	-
best possible	116.01	-	-
training labels	96.12	-	-
weighted NB	107.08	112.20	108.50
costing NB	107.09	112.34	108.56
costing C.45	112.67	116.41	112.52

Table 4: Experimental results.

5. CONCLUSIONS

We present here a new formulation for the cost-sensitive learning problem that we call the One-Benefit formulation. Instead of assuming that the benefits for each of the labels is known for each training example as in previous cost-sensitive formulations, this formulation only assumes that the benefit for one of the possible labels for each training example is known at training time. We argue that this is a realistic setup for some cost-sensitive domains such as direct marketing and medical treatment.

We show that it is possible to learn under this formulation by presenting a reduction from One-Benefit learning into standard classifier learning. The reduction requires that we first learn $P(y|x)$, that is, the conditional probability of the labels that appear in the training data, and then use the benefits and the conditional probabilities to correctly weigh the training data before applying the classifier learner. We also show how to correctly evaluate a classifier when only One-Benefit test examples are available, again by correctly weighting the examples.

We present some preliminary experimental results using a synthetic data generator. The advantage of using the generator is that we can evaluate the classifiers without resorting to the proposed evaluation method and, therefore, we can assess the accuracy of our evaluation method. Our results show that by using the One-Benefit reduction it is possible to learn a classifier that has greater expected benefit than the classifier used to label the training examples. Also, our proposed evaluation method succeeds in correctly measuring the expected benefit of the classifiers.

In future work, we would like to apply this method to real data sets from the direct marketing and medical treatment domains.

6. ACKNOWLEDGEMENTS

I thank Charles Elkan, John Langford and Roberto Oliveira for the many helpful conversations about the topic of this paper.

7. REFERENCES

[1] A. Beygelzimer, V. Dani, T. Hayes, J. Langford, and B. Zadrozny. Error limiting reductions between classification tasks. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, 2005. To appear.

[2] P. Domingos. MetaCost: A general method for making classifiers cost sensitive. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 155–164. ACM Press, 1999.

[3] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 973–978, Aug. 2001.

[4] D. Margineantu. Class probability estimation and cost-sensitive classification. In *Proceedings of the Thirteenth European Conference on Machine Learning*, pages 270–281, 2002.

[5] R. Srikant. IBM Quest Synthetic Data Generation Code, 1999. Available at `http://www.almaden.ibm.com/software/quest/Resources/datasets/syndata.html`.

[6] B. Zadrozny and C. Elkan. Learning and making decisions when costs and probabilities are both unknown. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 204–213, 2001.

[7] B. Zadrozny and C. Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 694–699, 2002.

[8] B. Zadrozny, J. Langford, and N. Abe. Cost-sensitive learning by cost-proportionate example weighting. In *Proceedings of the Third IEEE International Conference on Data Mining*, pages 435–442, 2003.

Utility based Data Mining for Time Series Analysis - Cost-sensitive Learning for Neural Network Predictors

Sven F. Crone
Lancaster University
Department of Management Science
Lancaster, LA1 4YX, UK
+44 1524 592991
s.crone@lancaster.ac.uk

Stefan Lessmann
University of Hamburg
Institute of Information Systems
VMP 5, 20146 Hamburg, Germany
+49 40 42838-5500
lessmann@bis-lab.com

Robert Stahlbock
University of Hamburg
Institute of Information Systems
VMP 5, 20146 Hamburg, Germany
+49 40 42838-3063
stahlbock@bis-lab.com

ABSTRACT
In corporate data mining applications, cost-sensitive learning is firmly established for predictive classification algorithms. Conversely, data mining methods for regression and time series analysis generally disregard economic utility and apply simple accuracy measures. Methods from statistics and computational intelligence alike minimise a symmetric statistical error, such as the sum of squared errors, to model ordinary least squares predictors. However, applications in business elucidate that real forecasting problems contain non-symmetric errors. The costs arising from over- versus underprediction are dissimilar for errors of identical magnitude, requiring an ex-post correction of the prediction to derive valid decisions. To reflect this, an asymmetric cost function is developed and employed as the objective function for neural network training, deriving superior forecasts and a cost efficient decision. Experimental results for a business scenario of inventory-levels are computed using a multilayer perceptron trained with different objective functions, evaluating the performance in competition to statistical forecasting methods.

Categories and Subject Descriptors
H.2.8 [**Database Management**]: Applications – *Data Mining*

General Terms
Algorithms, Management, Economics

Keywords
Data Mining, cost-sensitive learning, asymmetric costs, neural networks, time series analysis

1. INTRODUCTION
Profit and costs drive the utility of every corporate decision. As corporate decision making, from strategic to operational planning, is based upon future realisations of the decision parameters, e.g. telecommunications demand [1] or the likelihood of responders reacting to a mailing campaign [2], predictions or forecasts are a prerequisite for all managerial decisions. The quality of a forecast must be evaluated considering its ability to enhance the quality of the resulting decision. In management decisions, the utility arising to the decision maker from decisions based upon sub-optimal forecasts is measured in profit and costs. As a consequence, costs need to be incorporated to guide the predictions and ultimately derive valid corporate decisions.

In predictive data mining, the relevance of incorporating the costs resulting from a decision is reflected in approaches of cost-sensitive learning [3]. For classification, the costs for accurately predicting class membership of instances are proportional to the amount of accurately predicted instances. In addition, the costs associated with true versus false prediction of positives and negatives are often asymmetric [4] and are routinely used to guide the parameterisation and selection process of a wide range of classifiers, e.g. MetaCost [5] or cost sensitive boosting [6]. Consequently, robust evaluation techniques like the ROC convex hull method [7, 8] or the area under the ROC curve [9] have been proposed to enable classifier assessment in accordance with managerial objectives.

Similarly, for the predictive data mining problems of regression and time series analysis [10, 11] the costs arising from invalid point prediction of the true realisation increase with the magnitude of the error. In addition, the costs of the decisions derived from positive versus negative errors, or underprediction versus overprediction, are also often asymmetric. For example, in inventory management of retail outlets, keeping units of consumer goods in stock or on shelf in order to satisfy customer demand the effect of overstocking a product may induce increased stock holding costs for a single period versus the costs of understocking leading to lost sales revenue and dissatisfied customers. In both cases, the final evaluation of a forecast must be measured by the monetary costs arising from setting suboptimal decisions based on imprecise predictions of future demand [12], for asset transactions or inventory levels alike. Consequently, they depend on the given decision environment and a chosen behavioural strategy resulting from the decisions. These costs arising from over- and underprediction are typically not quadratic in form and frequently non-symmetric [13]. In addition, it is the asymmetry of costs that determines corporate policy, e.g. setting a target of satisfying 95% of demand.

However, for regression problems these asymmetries are largely neglected. Particularly in the field of data driven time series

UBDM '05, August 21, 2005, Chicago, Illinois, USA.

analysis and prediction, predictors of continuous scale are routinely evaluated using accuracy based evaluations through statistical error measures, such as the mean squared error or absolute error, eluding the reality of asymmetric costs of over- and underprediction. While this may seem unsurprising in the domain of econometric modelling of conventional statistical methods such as regression, autoregressive methods and exponential smoothing, these practices also persist in the domain of novel methods from computational intelligence, permitting minimisation of arbitrary objective or error functions through adaptive learning algorithms.

Artificial neural networks (NN) have found increasing consideration in forecasting theory, leading to successful applications in time series and explanatory sales forecasting [14, 15]. Based upon modest research in non-quadratic error functions in NN theory [15, 16] and asymmetric costs in prediction theory [13, 17-19], a set of asymmetric cost functions was recently proposed as objective functions for neural network training [20]. In this paper, we analyse the efficiency of a linear asymmetric cost function in inventory management decisions, training a multilayer perceptron to find a cost efficient stock-level for a set of seasonal time series directly from the data. As a consequence, the NN is trained directly using the ideas developed in utility based or cost-sensitive learning within the data mining domain.

Following a brief introduction to neural network prediction in inventory management, Section 3 assesses statistical error measures and asymmetric cost functions for neural network training. Section 4 gives an experimental evaluation of neural networks trained with asymmetric cost functions, outperforming expert software-systems for time series prediction. Conclusions are given in Section 5.

2. NEURAL NETWORK PREDICTIONS FOR INVENTORY DECISIONS

2.1 Forecasting for inventory management

In inventory management, forecasts of future demands are generated to select an efficient inventory level, balancing inventory holding costs for excessive stocks with costs of lost sales-revenue through insufficient stock [21, 22]. Although the amount of costs will generally increase with the numerical magnitude of the forecast errors, the costs arising from over- and underprediction are frequently neither symmetric nor quadratic [12, 19].

A service-level is routinely determined from strategic objectives or according to the actual costs arising from the decision, e.g. aiming to fulfil 98.5% of customer demand to balance this trade-off. Assuming Gaussian distribution, setting the inventory level to the optimum predictor will only fulfil 50% of all customer demand. Therefore, safety-stocks are calculated to reach the service level, using assumptions of the conditional distribution of the ex post forecast errors of the method applied [22].

For the decision of an inventory level for a single product in a single period of time the classic "newsboy"-problem is applicable. The decision rule for a service level resulting from a given cost of underprediction c_u and overprediction c_o reads

$$p_{y<}(Q^*) = \frac{c_u}{c_u + c_o} \quad , \qquad (1)$$

giving the value for a lookup of k in the probability table of the valid distribution, with $p(\bullet)$ denoting the probability of sales y being lower than an optimal inventory quantity Q^* held in each period. The final stock-level s is calculated using the forecast $\hat{y}_{t+h}$ of the sales volume y and adding a safety stock (SS) of k standard deviations of the forecast errors [22]:

$$s = \hat{y}_{t+h} + k\delta_e \quad . \qquad (2)$$

Consequently, the precision of the forecasts directly determines the safety stocks kept, the inventory level and the inventory holding costs. Hence, forecasting methods with superior accuracy such as NN may significantly reduce inventory holding costs [22].

2.2 Neural networks for time series analysis

Forecasting time series with non-recurrent NNs is generally based on modelling the network in analogy to a non-linear autoregressive AR(p) model [23]. At a point in time t, a one-step ahead forecast $\hat{y}_{t+1}$ is computed using n observations $y_t, y_{t-1}, \ldots, y_{t-n+1}$ from n preceding points in time *t, t-1, t-2, …, t-n+1*, with n denoting the number of input units of the NN. This models a time series prediction of the form

$$\hat{y}_{t+1} = f(y_t, y_{t-1}, \ldots, y_{t-n+1}) \quad . \qquad (3)$$

The architecture of a feed-forward multilayer perceptron (MLP) of arbitrary topology together with the resulting residuals of invalid forecasts denoted as the absolute error (*AE*) is displayed in Fig. 1.

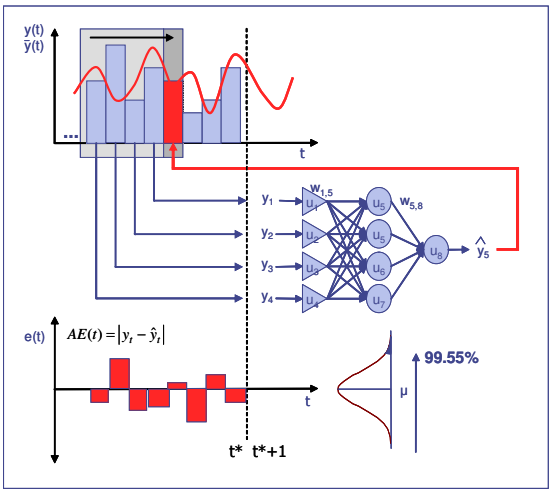


Figure 1. Neural network application to time series forecasting in inventory management, applying a MLP with 4 input units for observations in *t*, *t*-1, *t*-2, *t*-3, 4 hidden nodes and 1 output node for time period *t*+1.

The task of the MLP is to model the underlying generator of the data during training, so that a valid forecast is made when the trained network is subsequently presented with a new value for the input vector [16]. Therefore the objective function used for NN training determines the resulting system behaviour and performance [15].

The objective functions routinely employed in neural network training differ from the objective function of the underlying

inventory management decision in slope, scale and ratio of asymmetry. Following, alternate objective functions are discussed to incorporate the original objective structure in NN training.

3. OBJECTIVE FUNCTIONS FOR COST-SENSITIVE REGRESSION LEARNING

Supervised online-training of a MLP is the task of adjusting the weights of the links w_{ij} between units i,j and adjusting their thresholds to minimise the error δ_j between the actual and a desired system behaviour [24]. Gradient descent algorithms traditionally minimise the modified sum of squared errors (*SSE)* as the objective function, ever since the popular description of the back-propagation algorithm by Rumelhart, Hinton and Williams [25].

The *SSE*, as all statistical error measures, produces a value of 0 for an optimal forecast and is symmetric about $e_t=0$, implying symmetric costs of errors in predicting future demand for inventory levels. The consistent use of the modified *SSE* in time series forecasting with NN is motivated primarily by analytical simplicity [15] and the similarity to statistical regression problems, modelling the conditional distribution of the output variables [24]. As neural network theory and applications consistently focus on the symmetric *SSE*-function for training, therewith modelling least squares predictors as well, the forecasts also need to be adjusted using safety stocks to attain a desired service-level.

Following, we propose an asymmetric cost function (ACF), modelling the objective function of the costs arising in the original decision problem instead of least squares predictors. These costs are often not only non-quadratic, but also non-symmetric in form. The objective function in NN training, determining the size of the error in the output-layer, may thus be interpreted as the actual costs arising from an overprediction or an underprediction of the current pattern *p*, comprising all input and output information for the MLP, in training.

Recently, we introduced a linear ACF to NN training [20], originally developed by Granger for statistical forecasts in inventory management problems [19]. The *LINLIN* cost function *(LLC)* is linear to the left and right of 0. The parameters *a* and *b* give the slopes of the branches for each cost function and measure the costs of error for each stock keeping unit (SKU) difference between the forecast $\hat{y}_{t+h}$ and the actual value y_{t+h}. The parameter c_o corresponds to an overprediction and the resulting stock-keeping costs, while c_u relates to the costs of lost sales-revenue for each underpredicted SKU. The *LLC* yields:

$$LLC(y_{t+h}, \hat{y}_{t+h}) = \begin{cases} c_o \left| y_{t+h} - \hat{y}_{t+h} \right| & \text{for } y_{t+h} < \hat{y}_{t+h} \\ 0 & \text{for } y_{t+h} = \hat{y}_{t+h} \\ c_u \left| y_{t+h} - \hat{y}_{t+h} \right| & \text{for } y_{t+h} > \hat{y}_{t+h} \end{cases} . \qquad (4)$$

The shape of one asymmetric *LLC,* as a valid linear approximation of a real cost function in our corresponding inventory management problem, is displayed in Fig. 2.

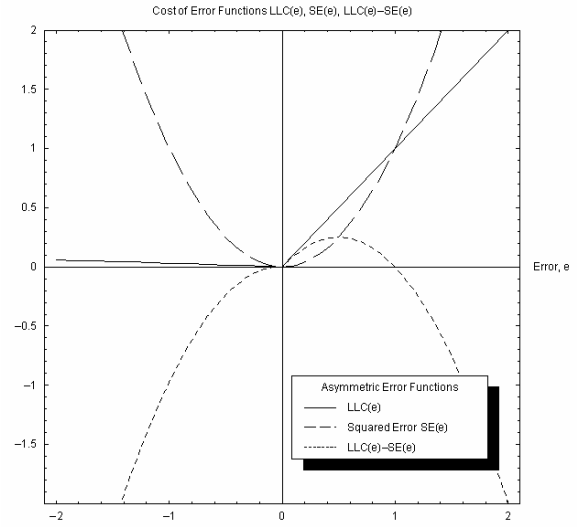


Figure 2. Empirical Asymmetric Cost Function showing cost arising for over- and under-prediction, using c_o=£0.01 and c_u=£1.00 in comparison to the SE.

For $c_u \neq c_o$ these cost functions are non-symmetric about 0 and are hence called asymmetric cost functions. The degree of asymmetry is given by the ratio of c_o to c_u [17]. For $c_o = c_u = 1$ the LLC equals the statistical absolute error measure *AE*. The linear form of the ACF represents constant marginal costs arising from the business decision. Our model therefore coincides with the analysis of business decisions based on linear marginal costs and profits.

Yang, Chan and King introduce a classification-scheme for objective functions, introducing dynamic non-symmetric margins for support vector regression [17]. Applied to objective functions in NN training it allows a classification of all symmetric statistical error functions and asymmetric cost functions previously developed. Linear, non-linear and mixed ACFs have been specified in literature [13, 17-19] while variable or dynamic objective functions to account for varying or heteroscedastic training objectives have not yet been developed for NN-training, as shown in Table 1.

Table 1. Objective functions for neural network training

Variability	Symmetry of objective function	
	Symmetric	**Non-symmetric**
Fixed	SE, AE, ACE. statistical error functions	LINLIN etc. asymmetric cost functions
Variable	-	-

Asymmetric transformations of the error function alter the error surface significantly, resulting in changes of slope and creating different local and global minima. Therefore, using gradient descent algorithms, different solutions are found minimising cost functions instead of symmetric error functions, finding a cost minimum prediction for the inventory management problem. These asymmetric cost functions may be applied in NN training using a simple generalisation of the error-term of the back-propagation rule and its derivatives, amending only the error calculation for the weight adaptation in the output layer [20], but

applying alternative training methods or global search methods to allow network training [15]. For the following simulation experiments, we developed a simulator allowing minimisation of arbitrary, non-differentiable objective functions through the use of gradient decent and code controlling for non-defined derivatives.

4. SIMULATION EXPERIMENT OF COST-SENSITIVE TIME SERIES ANALYSIS

4.1 Experimental time series data

Following, we conduct an experiment to evaluate the ability of a MLP to evolve a set of weights minimising an *LLC* asymmetric cost function for a seasonal time series. We analyse a set of benchmark time series for seasonal time series prediction recently published in a study by Zhang and Qi [26]. In order to exemplify the potential impact of asymmetric cost functions in an empirical setting while controlling for problems of model misspecification and selection, we limit our analysis to the three artificial time series, closely resembling the seasonality and length of real department store sales [26]. The simulated series were created using a multiplicative seasonal model without trend

$$y_t = 100\, SI_t + E_t \ , \tag{5}$$

with SI_t the seasonal index for each month, E_t the additive error term following a normal distribution $N(0, \mu)$ and t denoting the time index. The seasonal indices to calculate each observation are

$$SI_t = \{.75;.80;.82;.90;.94;.92;.91;.99;.95;1.02;1.20;1.80\}. \tag{6}$$

To estimate the effect of different noise levels on the forecasting accuracy the authors apply three levels of error variance $\sigma^2 = \{1;25;100\}$ to construct three time series A, B and C. A total of 1200 points is generated for each time series of a particular noise level. In order to control for external influences we sourced the original time series from the authors, to realign the properties of the random noise with the original series. A part of the time series is presented in Figure 3.

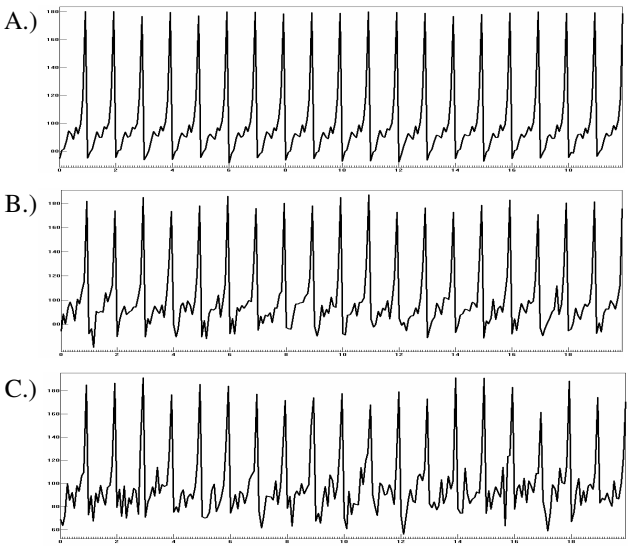


Figure 3. Part of three artificial time series A, B and C.

An analysis of the autocorrelations (AC) and partial autocorrelations (PAC) reveals the purely seasonal pattern of the time series. An analysis of the noise reveals the structure documented by Zhang and Qi and no significant AC or PAC.

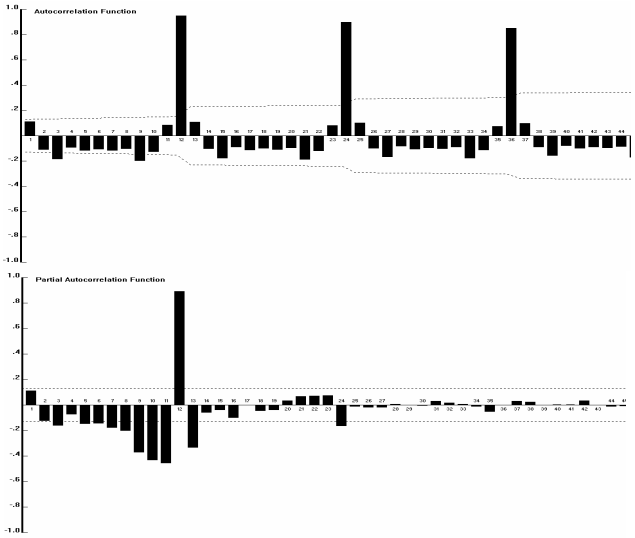


Figure 4. Autocorrelation function and partial autocorrelation function of the seasonal SI_t without added noise.

The autocorrelation function (ACF) of the undifferenced series of the seasonal factors without noise reveals a seasonal pattern with significant spikes at lagged 12 months apart with decaying magnitude, indicating a seasonal autoregressive process an the absence of a moving average process. As expected, first seasonal differencing *D*=1 eliminates all AC and PAC across all lags. Similarly, for all three noisy time series first seasonal differencing eliminates all significant ACs and PACs at all lags.

4.2 Objective functions

To exemplify the effect of different objective functions on NN predictions, we compare three objective functions. Firstly, we train a set of NN using a squared error (*SE*) objective function, NN_{SE}, modelling least squares predictors to find the mean of the distribution, implying equal costs c_u=c_o=1 and 50% service level.

Secondly, we train a set $NN_{LLC\text{-}1}$ using an asymmetric cost function to reflect the estimation of a cost efficient inventory level. In order to specify the underlying costs arising from the decision process we specify a particular cost trade-off reflecting an empirical cost relationships in fast moving consumer goods retailing, also reflecting the original motivation of the artificial time series from the retail domain. A retail outlet needs to allocate products to customer demand for each period. Overprediction of consumer demand leads to unsold items and inventory holding costs c_o for another period while underprediction results in costs c_u through lost sales-revenue per product, assuming $c_u > c_o$ and disregarding fixed costs of the decision. As a consequence, we construct a newsboy decision problem, reflecting the single period inventory model without backordering, as outlined under section 2.1. We create an linear asymmetric cost function *LLC-1* of (c_o=\$0.1; c_u=\$1.00), implying high costs of running out of stock and therefore the need of increased inventory levels or predictions

respectively. The asymmetric cost relationship relates to a 90% service level, by

$$LLC\text{-}1: \quad p_{y<}\left(Q^*\right) = \frac{c_u}{c_u + c_o} = \frac{1.00}{1.00 + 0.10} = 0.90 \quad \cdot \qquad (7)$$

In addition, we train a third set of $NN_{LLC\text{-}2}$ using an objective function *LLC-2* of (c_o=\$1.00; c_u=\$0.10*)* implying high costs of overstocking as the reverse quantile of *LLC-1*, as in

$$LLC\text{-}2: \quad p_{y<}\left(Q^*\right) = \frac{c_u}{c_u + c_o} = \frac{0.10}{0.10 + 1.00} = 0.10 \quad \cdot \qquad (8)$$

While a 10% service level seems implausible from a corporate policy perspective, it may serve to evaluate the NN ability to estimate arbitrary quantiles on both sides of distribution.

4.3 Design of the forecasting methods

Each of the three time series of n=1200 observations is split into three disjoint datasets for NN training, validation and testing, using the last 300 observations for out-of-sample evaluation in the test dataset, 300 observations for early stopping and selection of the best NN model in the validation dataset and the rest of the 600 observations for parameterisation in the training dataset. This results in 588, 300 and 300 predictable patterns in each set. Considering the limited length of the time series for training and testing, only an approximation and no exact estimation of the quantile and service level minimizing the costs appears feasible.

All data was scaled from a range of 0 to 210 into the interval [-1;1] applying a headroom of 20% to avoid saturation effect of the nonlinear activation functions. It should also be noted, that the ability of NN to forecast seasonal and trended time series patterns has recently been questioned [26], leading to recommendations to deseasonalise and detrend time series prior to training the networks. With regard to our own research findings we refrain from preprocessing the time series this way, and train the NN on the original, seasonal auto regressive patterns.

To determine an efficient and parsimonious network architecture while limiting experimental complexity, we pre-evaluated a set of input vectors applying different lag structures from {t-1,…t-36}, a number of {0…20} nodes in a single hidden layer with different sigmoid activation functions {tanh; logistic} and different output functions in the output layer {tanh; logistic; identity} simultaneously. We evaluated 594 network topologies on 10 initialisations each with randomised starting weights to account local minima. The results were analysed conducting a multifactorial analysis of variance (ANOVA) with equal cell sizes to identify significant suboptimal topologies. While topologies with 0 or 1 hidden nodes showed reduced accuracy, no significant differences between topologies with hidden nodes n>2 could be identified using a multiple comparison test of homogeneous subgroups of estimated marginal means. The impact of activation functions showed a negative effect of the sigmiod function in the output layer and a negative interaction effect between tanh and non-tanh functions in the output layer. It found no significant difference in performance between tanh in both hidden and output layer and sigmoid in hidden and identify in the output layer. As a consequence, we selected the most parsimonious topology with the lowest *MSE* and variance on the validation dataset for further experimentation. We chose a fully connected MLP without shortcut connections, applying a topology of 12 input nodes for the time lags t-1,…,t-12 to exploit all feasible yearly time-lags of a monthly series, 2 hidden nodes and 1 output node. Additionally, one bias unit models the thresholds for all units in the hidden and output layer. All units in the hidden layer use a summation as an input-function, the logistic function as a semilinear activation function and the identity function as an output function. The unit in the output layer uses a nonlinear, unbounded identity function.

Three sets of networks NN_{SE}, $NN_{LLC\text{-}1}$ and $NN_{LLC\text{-}2}$ were trained using different objective functions. Each MLP was initialised and trained for twenty times to account for [-0.6;0.6] randomised starting weights. We applied a standard backpropagation algorithm, using an initial learning rate of η=0.5 decreased by a cooling factor of .99 after every epoch, and a momentum term of φ=.4. Training consisted of a maximum of 1000 epochs with a validation after every epoch, applying early stopping if a composite of 50% training and 50% validation error did not decrease by 0.01% for 10 epochs. After training a total of 189 NN across 3 time series and for 3 objective functions, the results for the best network within each subgroup, chosen on its objective function performance on the validation set, was computed for all three data subsets. Consequently, NN_{SE} was selected on lowest mean squared error (MSE) on the validation set, $NN_{LLC\text{-}1}$ on lowest mean LLC1 and NN_{LLC2} on lowest mean LLC2 on the validation set respectively. Only the test dataset is used to measure generalisation, applying a simple hold out method for out-of-sample evaluation or generalisation.

To compare the performance on achieving a 90% service level, we need to extend the NN_{SE} predictions through the calculation of safety stocks. We generate business forecasts based upon ordinary least-squares predictors of the best NN_{SE} and conventional statistical methods and calculate additional safety stocks necessary to achieve the desired service level using the standard formulas. As statistical benchmarks, the Naïve1 method using last periods sales as a forecast, $\hat{y}_{t+1} = y_t$, exponential smoothing and ARIMA were computed. The statistical predictions were computed using the benchmark software system Forecast Pro, which selects and parameterises appropriate models of exponential smoothing or ARIMA intervention models based upon statistical testing and expert knowledge based on the properties of each time series [27]. For the predictions by NN_{SE} and the Naïve method the final inventory level was calculated as in ForecastPro, using

$$s = \hat{y}_{t+h} + 2.33\sigma_e \quad , \qquad (9)$$

for an ex-post correction of the ordinary least squares predictor by adding k=2.33 standard deviations σ to derive a cost efficient service level of 90.0% for the given inventory problem, assuming Gaussian distribution and homoscedastiticity of the residuals, as confirmed by Kolmogorov-Smirnov tests.

In contrast, the asymmetric cost predictor $NN_{LLC\text{-}1}$ was trained to predict the cost efficient inventory level, equal to a 90% service level, for each month directly from the training process. Following the experiments we assess the ex-post performance of the competing approaches in the following section.

All neural network experiments were computed using the NN software simulator “Intelligent Forecaster”, developed within our research group to compute and compare multiple NN time series experiments on arbitrary objective functions. Average runtime for

training a NN, creating predictions and saving results was 2.75 seconds on a Pentium IV 3.8 GHz, 4GB RAM, 1TB disk drive.

4.4 Experimental impact of cost functions

First, we evaluate the ability of a NN to estimate a predetermined service level from the cost relationship of over- versus underprediction. Consequently, we compare the forecasts and resulting service levels of the three sets of NNs to evaluate their ability to adhere to different objectives during the training process across the three time series.

Table 2 displays the results using mean error measures computed on each dataset to allow comparison between datasets of varying length. The results are given in the form (training set / validation set / test set) to allow interpretation. The descriptive performance measure of the alpha-service-level gives the amount of suppressed sales occurrences per dataset. All methods are evaluated ex-post on their performance by mean *SE (MSE)* and the ex post mean *LINLIN* costs (*MLLC*) for each objective function *MLLC*-1 and *MLLC*-2 respectively. In addition, the NN predictions on the test set for out-of-sample evaluation are presented in figures. The results reflect the impact of different objective functions SE, LLC-1 and LLC-2. Each set of NN shows lower mean errors or costs across all time series A, B, and C for its individual training objectives. E.g. NN_{SE} shows significantly lower MSE then NN_{LLC-1} and NN_{LLC-2}. Vice versa, NN_{LLC-1} shows robust minimisation of *MLLC-1* in- and out-of-sample as opposed to both other sets of trained networks. These results are confirmed in a multifactorial ANOVA, revealing two homogeneous subsets of the method trained on minimising the particular error measure versus the two other methods. An analysis of the service levels further reveals, that each method approximates the target service level of 50%, 90% and 10% within and out-of-sample robustly and accurately, considering the achievable degree of accuracy determined through the length of the time series. We may therefore conclude, that NN allow minimisation of arbitrary objective functions to estimate different service levels.

Various additional results may be drawn from the experiment. As expected, the best selected NN_{SE} trained on the standard *SE* approximates the seasonal time series pattern and generates valid and reliable *t*+1 predictions on validation and test dataset across all three time series, as visible in the results of Table 2 and the part of the test set predictions given in Figure 5. With regard to the decreasing signal to noise ratio the predictions show increasing deviations from the actual data from time series A to B and C, as must be expected.

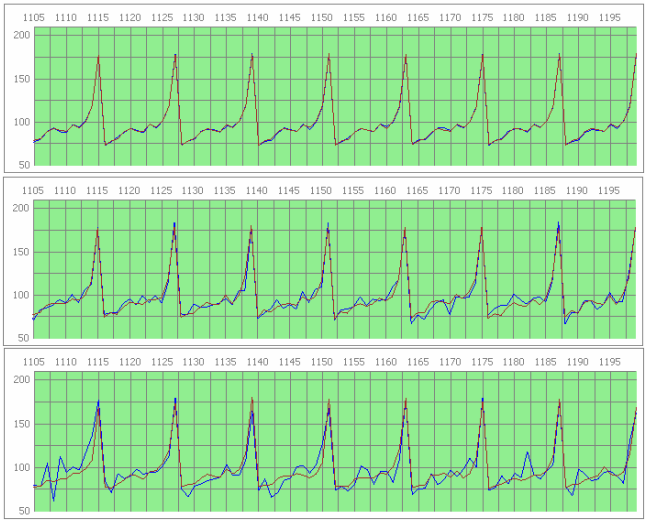


Figure 5. Predictions of a NN_{SE} trained on minimising the symmetric *SE* to forecast monthly retail sales across three time series A, B and C from above. The graph shows the time series of retail demand in blue versus the NN forecast in red on the test dataset.

Nevertheless, the artificial data pattern underlying the generated time series is robustly extracted by NN_{SE} regardless of the increasing noise level, demonstrating only limited overfitting through the training process. In cost and inventory terms, the NN are trained on equal costs of over- versus underprediction in order to estimate a 50% service level relating to the ordinary mean predictor, as shown in Figure 5. As a consequence, we are unable to confirm recent findings in the forecasting and management science domain, that NN are incapable of predicting seasonal time series patterns without prior deseasonalisation.

The level of predictions given by the NNs trained on minimizing the asymmetric cost function *LLC-1* presented in Figure 6 differs significantly from the predictions by the NN_{SE}. Analysing the behaviour of the forecast based upon the asymmetry of the costs function, the neural network NN_{LLC-1} raises its predictions in comparison to the NN_{SE} trained on squared errors to achieve a cost efficient forecast of the optimum inventory level. Predictions on the test set are displayed in Figure 6, with identical patterns on the training and validation set omitted due to the length of the time series and space restrictions.

Table 2. Results on Forecasting Methods and NNs trained on linear Asymmetric Costs and Squared Error Measures

Objective Function	Time Series	NN no.	Error Measures									Service Level		
			MSE(e)			*MLLC-1 (e)*			*MLLC-2 (e)*			*alpha*		
NN_{SE}	A	(#30)	1.35	1.45	1.23	.47	.51	.43	.55	.55	.54	50.9%	49.0%	54.0%
	B	(#96)	26.63	28.00	30.17	2.26	2.14	2.56	2.27	2.44	2.26	49.7%	50.0%	47.0%
	C	(#147)	104.77	88.08	100.35	4.78	4.36	4.27	4.15	3.86	4.32	49.3%	49.7%	54.7%
NN_{LLC-1}	A	(#53)	4.50	4.77	4.14	1.79	1.79	1.67	0.23	0.26	0.21	90.3%	87.3%	93.0%
	B	(#98)	78.44	75.90	83.50	7.11	6.96	7.44	0.99	1.04	1.05	92.5%	91.7%	90.0%
	C	(#158)	246.85	221.95	219.87	12.28	11.89	11.46	1.82	1.77	1.69	90.5%	91.7%	90.7%
NN_{LLC-2}	A	(#1)	4.23	4.28	4.20	.22	.23	.21	1.68	1.66	1.72	7.3%	9.7%	6.7%
	B	(#109)	81.93	84.16	76.71	.93	1.00	.96	7.25	7.50	6.90	7.7%	10.7%	6.3%
	C	(#163)	267.46	251.13	279.30	1.84	1.69	2.00	13.14	12.99	13.73	9.4%	10.7%	13.7%

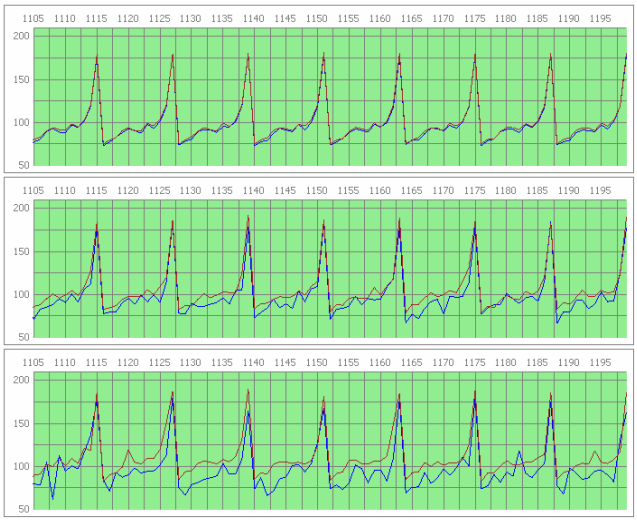


Figure 6. NN_{LLC-1} predictions on a part of the test dataset across time series A, B and C from top to bottom, aiming to minimise inventory costs through an service level of 90%. The upper red line denotes the forecasts, the lower blue the actuals.

The network accounts for higher costs of underpredicition versus overprediction through increased predictions, therefore avoiding costly stock-outs. The predictions estimate a cost minimal point depending on the varying distributions of the error residuals, as visible in level of predictions increasing with the size of the error distribution from time series A to C. This is also evident in the lack of stock-outs represented by the increased service-level of 93.0%, 90.0% and 90.7% on time series A, B and C.

To evaluate the validity and reliability of the NN training, we estimate an inverse ACF of the given problem domain, estimating the 10% quantile. As shown in Figure 7, the NN alters its estimation on asymmetric costs by lowering its predictions to robustly achieve a 10% service level across all three time series.

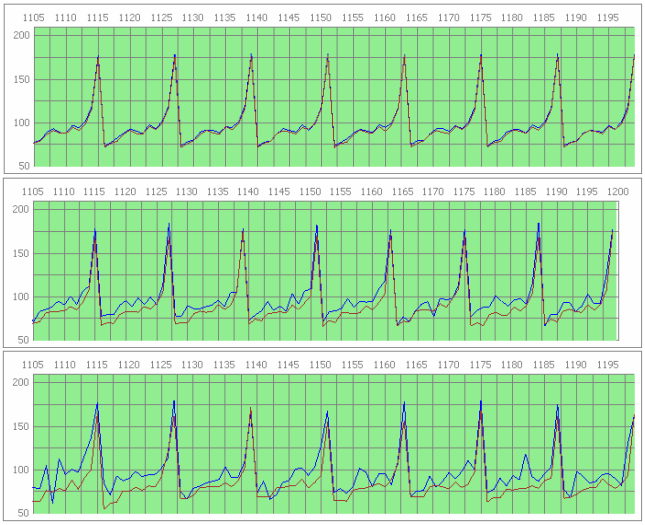


Figure 7. NN_{LLC-2} predictions on a part of the test dataset across time series A, B and C, aiming at a service level of 10%. The lower red line denotes the forecasts, the upper the actuals.

Consequently, a neural network may be trained to not only predict the expected mean of a time series but instead produces a biased optimum predictor, as intended by Grangers original work through ex post correction of the original predictor [19]. This may be interpreted as finding a valid approximation for a point on the conditional distribution of the optimal predictor depending on the standard-deviation, or quantile autoregression. Within an inventory management problem, the network finds a cost efficient inventory level without the separate calculation of safety stocks directly from the cost relationship. This reduces the complexity of the overall management process of stock control, successfully calculating a cost efficient inventory level directly through a forecasting method using only a cost function and the data.

The results of the adjusted in relation to the increasing noise levels become visible in a PQ-scatterplott in Figure 8.

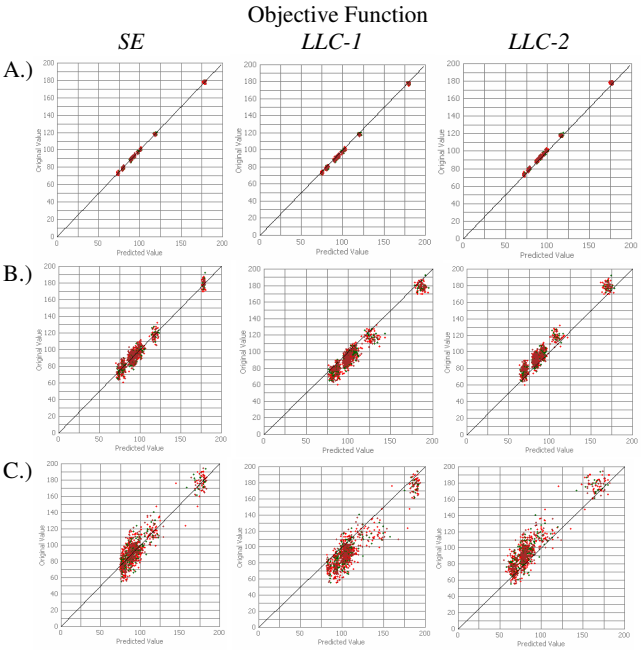


Figure 8. PQ-scatter plots of actual values versus predicted values for three time series A, B and C and objective functions indicating accuracy and lack of biases in predictions

A set of insignificant Kolmogorov-Smirnov tests confirms the normality of the residuals for all predictions, with the residuals of the NN_{SE} being centred around zero while the residuals of NN_{LLC-1} and NN_{LLC-2} are centred around the relevant quantile. As a consequence we may use the standard formula to estimate the 90% quantile or service level to compare the methods performance on the estimated inventory levels in the next section.

4.5 Experimental accuracy of inventory levels

After determining the general ability of NN to predict a cost efficient inventory level for the newsboy problem directly through training with an asymmetric cost function, we seek to evaluate their accuracy in comparison to a conventional ex-post correction of the mean estimator of a statistical forecasting method.

We utilise the predictions of the NN_{SE} for time series A, B and C and add a safety stock of k=2.33 standard deviations to the individual prediction, in accordance with the Gaussian noise and

homoscedasticity of the residuals as confirmed by nonparameric testing. In addition, we use ForecastPro to generate statistical forecasts for each time series, selecting three different seasonal ARIMA (p,d,q)(P,D,Q) models all with log transform as optimum methods. For time series A, ForecastPro selects an ARIMA(2,0,2)*(0,1,1), for time series B an ARIMA(1,0,0)*(0,1,1) and an ARIMA(0,0,3)*(0,1,1) with log transform for time series C. In addition, the software uses an internal expert procedures for residual analysis and safety stock calculation to achieve a 90.0% service level based upon its forecasts and the adequate distribution of the residuals directly.

We compute ex post accuracy on the business objective, using the actual costs occurring form each unit out-of-stock and each item left overstocked for each period *t* in the dataset, approximated by LLC-1. The ex-post costs arising from over- and underprediction alike represent the true variable decision costs and therefore a valid business objective in operational inventory management. In addition, we compute the number of stock-out and overstock occurrences to evaluate the frequency in which suboptimal decisions were made regardless of the magnitude of the errors. The results are provided in Table 3 with parts of the time series and calculated inventory levels provided in separate figures.

Unsurprisingly, all methods outperform the benchmark naïve method, showing significantly better results through robust identification and extrapolation of the seasonal time series pattern in forecast and inventory levels.

The best NN_{LLC-1} trained with the asymmetric *LINLIN* cost function *LLC*-1 gives an overall superior forecast regarding the business objective of minimising costs, achieving the lowest mean costs in-sample on training and validation data as well as out-of-sample on the test-data and across all three time series A, B and C. For the noisy time series C, it exceeds all inventory methods, and clearly outperforms forecasts of NN trained with the *SE* criteria and added safety stocks, as presented in Figure 9. An analysis of the marginal means reveals two homogeneous subsets of costs. While the differences between NN_{LLC-1} and all other methods prove statistically significant, no significant differences could be confirmed between NN_{SE} forecasts and ARIMA forecasts using the conventional calculation of safety stock.

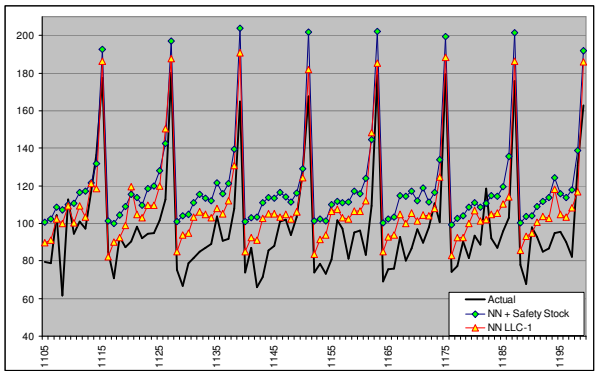


Figure 9. Comparison of predictions by NN_{SE} plus 2.33 standard deviations of safety stock (◊) versus direct inventory calculation by NN_{LLC-1} (Δ) on time series C.

In addition, NN_{LLC-1} outperforms the best automatically selected and parameterised ARIMA model and safety stocks selected by the software expert system, as displayed in Figure 10. Considering the inferior quality of the predictions provided by the Naïve method, its benchmarks may be excluded from further analysis.

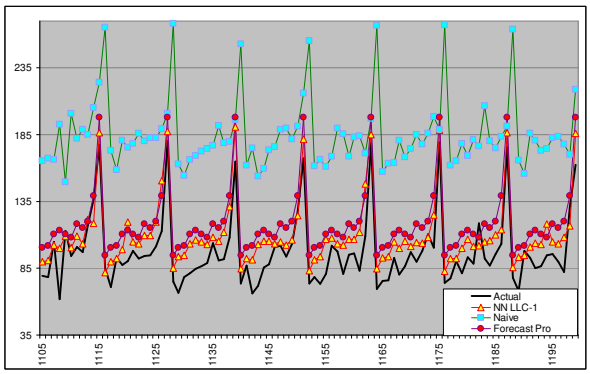


Figure 10. Comparison of inventory levels from the Naïve method (€) and ForecastPro (O) plus 2.33 standard deviations of safety stock (◊) versus NN_{LLC-1} (Δ). on time series C.

Table 3. Results on Forecasting Methods and NNs trained on linear Asymmetric Costs and Squared Error Measures

		Cost Error Measures			Descriptive Error Measures of Inventory Holding					
Time	Forecasting	*Sum of LLC-1 (e_t) with c_o=0.1; c_u=1*			*No. of overstocked occurrences*			*No. of out-of-stock occurrences*		
Series	Method	Training	Validation	Test	Training	Validation	Test	Training	Validation	Test
A	NN_{LLC-1} inventory	129.95	68.61	64.49	531	262	279	57	38	21
	NN_{SE} + safety stock	163.74	82.80	85.05	586	289	289	2	2	2
	ForecastPro	137.71	69.14	71.70	580	296	295	8	4	5
	Naïve Method	4870.74	2495.11	2485.25	588	300	300	0	0	0
B	NN_{LLC-1} inventory	558.57	299.74	291.20	544	275	270	44	55	30
	NN_{SE} + safety stock	725.51	379.59	364.92	587	297	298	1	3	2
	ForecastPro	694.82	358.67	341.35	582	296	299	6	4	1
	Naïve Method	5014.08	2557.91	2558.13	588	300	300	0	0	0
C	NN_{LLC-1} inventory	1099.74	491.78	592.35	532	275	272	56	25	28
	NN_{SE} + safety stock	1361.14	689.33	734.17	580	297	294	8	3	6
	ForecastPro	1257.56	655.08	684.18	578	298	294	10	2	6
	Naïve Method	5171.19	2636.68	2642.80	587	300	298	1	0	2

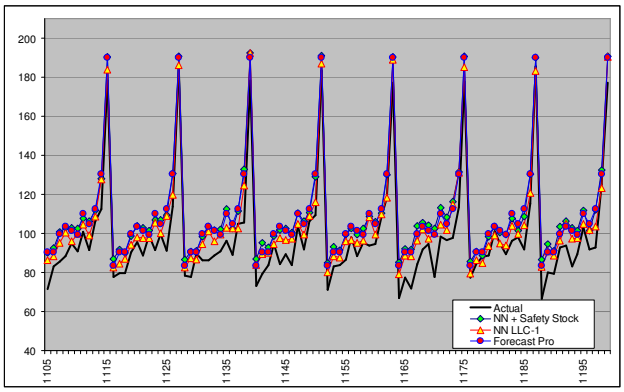


Figure 11. Comparison of predictions by NN_{SE} (◊) and a ForecastPro ARIMA model (O) plus 2.33 standard deviations of safety stock (◊) versus NN_{LLC-1} (Δ) on time series B.

While these results prove consistent across all time series, the differences in prediction for time series B become smaller due to the reduced noise levels and prove insignificant in testing, also apparent in Figure 11 for time series B. For time series C of the lowest noise level, the differences in accuracy between the competing methods prove statistical non-significant. Nevertheless, NN_{LLC-1} demonstrates a competitive performance in comparison to established forecasting and inventory methods.

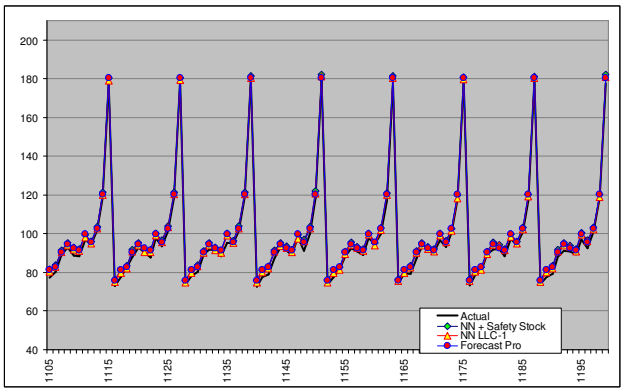


Figure 12. Comparison of predictions by NN_{SE} (◊) and a ForecastPro ARIMA model (O) plus 2.33 standard deviations of safety stock (◊) versus NN_{LLC-1} (Δ) on time series A.

Our earlier experimental results demonstrated that NN may be trained on arbitrary objective functions to predict predetermined quantiles on an empirically estimated distribution. Moreover, the results in comparing conventional statistical approaches versus an integrated modelling though simultaneous prediction and safety stock calculation indicate, that NN trained on minimizing the appropriate cost function directly from the training data may also outperform conventional approaches of inventory level calculations. This may be attributed to a more accurate approximation of the true distribution of the residuals given a reduced sample size in empirical experiments or to reduced errors in the modelling process itself, limiting effects of suboptimal selection and parameterisation of the forecasting model, identification of the error distribution and estimation of the cost efficient point on the distribution. However, these indications require additional experimentation to rationalize the origin of increased validity and reliability of the proposed approach.

5. CONCLUSION

We have examined symmetric and asymmetric error functions as performance measures for neural network training. The restriction on using squared error measures in neural network training may be motivated by analytical simplicity, but it leads to biased results regarding the final performance of forecasting methods if the true objective is not the estimation of the mean. Asymmetric cost functions may capture the actual decision problem directly and allow a robust minimization of relevant costs using standard MLP and training methods, finding optimum inventory levels. Our approach to train neural networks with asymmetric cost functions has a number of advantages. Minimising an asymmetric cost function allows the neural network not only to forecast, but instead to reach optimal business decisions directly, taking the model building process closer towards business reality. As demonstrated, considerations of finding optimal service levels in inventory management are incorporated within the NN training process, leading directly to the forecast of a cost minimum stock level without further computations.

As we attempted to exemplify a NN's ability to minimise LLC and produce valid predictions of a given quantile on a probability density function, we limited design complexity to three simple and homogeneous artificial time series, albeit minimising the ability to generalise from the results to other artificial or empirical time series as well as varying and inconsistent time series patterns. While length and form of the time series were selected to balance the tradeoff between empirical relevance and feasibility in our experiments, it holds only for the evaluated time series.

However, the limitations and promises of using asymmetric cost functions with neural networks justify systematic analysis. Future research may incorporate the modelling of dynamic carry-over-, spill-over-, threshold- and saturation-effects for exact asymmetric cost functions where applicable. In particular, verification on multiple time series, other network topologies and architectures is required, in order to evaluate current research results. As a consequence, the experiments particularly require extension to additional artificial time series and multiple step-ahead forecasts for multiple origins, in contrast to the multi-origin single step-ahead forecasts implemented to model newsboy decisions. Further experiments may also be extended to incorporate the estimation of multiple points on different, non Gaussian error distributions to facilitate generalization. In addition, the experiments need to be reevaluated using large scale corporate forecasting competition data as the M3-benchmark to evaluate the empirical relevance for corporate decision making.

6. REFERENCES

[1] R. Fildes and V. Kumar, "Telecommunications demand forecasting-a review," *International Journal of Forecasting*, vol. 18, pp. 489-522, 2002.

[2] B. Baesens, S. Viaene, D. Van den Poel, J. Vanthienen, and G. Dedene, "Bayesian neural network learning for repeat purchase modelling in direct marketing," *European Journal of Operational Research*, vol. 138, pp. 191-211, 2002.

[3] S. Viaene and G. Dedene, "Cost-sensitive learning and decision making revisited," *European Journal of Operational Research*, vol. 166, pp. 212-220, 2004.
[4] F. Provost, T. Fawcett, and R. Kohavi, "The Case Against Accuracy Estimation for Comparing Induction Algorithms," presented at Proc. of the 5th Intern. Conf. on Machine Learning, San Francisco, CA, USA, 1998.
[5] P. Domingos, "MetaCost: a general method for making classifiers cost-sensitive," presented at Proc. of the 5th ACM SIGKDD Intern. Conf. on Knowledge Discovery and Data Mining, San Diego, CA, USA, 1999.
[6] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan, "AdaCost: Misclassification Cost-Sensitive Boosting," presented at Proc. of the 16th Intern. Conf. on Machine Learning, Bled, Slovenia, 1999.
[7] F. Provost and T. Fawcett, "Robust classification for imprecise environments," *Machine Learning*, vol. 42, pp. 203-231, 2001.
[8] F. Provost and T. Fawcett, "Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distributions," presented at Proc. of the 3rd Intern. Conf. on Knowledge Discovery and Data Mining, Newport Beach, CA, USA, 1997.
[9] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, pp. 1145-1159, 1997.
[10] M. H. Dunham, *Data mining: introductory and advanced topics*. Upper Saddle River, NJ: Prentice Hall, 2003.
[11] S. M. Weiss and N. Indurkhya, *Predictive data mining: a practical guide*. San Francisco: Morgan Kaufmann Publishers, 1998.
[12] C. W. J. Granger, *Forecasting in business and economics*. New York: Academic Press, 1980.
[13] P. F. Christoffersen and F. X. Diebold, "Further results on forecasting and model selection under asymmetric loss," *Journal of Applied Econometrics*, vol. 11, pp. 561-571, 1996.
[14] S. G. Makridakis, S. C. Wheelwright, and R. J. Hyndman, *Forecasting: methods and applications*. New York: Wiley, 1998.
[15] R. D. Reed and R. J. Marks, *Neural smithing: supervised learning in feedforward artificial neural networks*. Cambridge, Mass.: The MIT Press, 1999.
[16] C. M. Bishop, *Neural networks for pattern recognition*. Oxford: Oxford University Press, 1995.
[17] G. Arminger and N. Götz, *Asymmetric loss functions for evaluating the quality of forecasts in time series for goods management systems*. Dortmund: Univ., 1999.
[18] P. F. Christoffersen and F. X. Diebold, "Optimal prediction under asymmetric loss," *Econometric Theory*, vol. 13, pp. 808-817, 1997.
[19] C. W. J. Granger, "Prediction With A Generalized Cost Of Error Function," *Operational Research Quarterly*, vol. 20, pp. 199-&, 1969.
[20] S. F. Crone, "Training Artificial Neural Networks using Asymmetric Cost Functions," in *Computational Intelligence for the E-Age*, L. Wang, J. C.Rajapakse, K.Fukushima, and X. Y. S.-Y.Lee, Eds. Singapore: IEEE, 2002, pp. 2374-2380.
[21] R. G. Brown, "Exponential Smoothing for predicting demand," in *Tenth national meeting of the Operations Research Society of America*. San Francisco, 1956.
[22] E. A. Silver and R. Peterson, *Decision systems for inventory management and production planning*, 2. ed. New York [u.a.]: Wiley, 1985.
[23] G. Dorffner, "Neural Networks for Time Series Processing," *Neural Network World*, vol. 6, pp. 447-468, 1996.
[24] S. S. Haykin, *Neural networks: a comprehensive foundation*, 2nd ed. Upper Saddle River, N.J.: Prentice Hall, 1999.
[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors (from Nature 1986)," *Spie Milestone Series Ms*, vol. 96, pp. 138, 1994.
[26] G. P. Zhang and M. Qi, "Neural network forecasting for seasonal and trend time series," *European Journal of Operational Research*, vol. 160, pp. 501, 2005.
[27] G. E. P. Box and G. M. Jenkins, *Time series analysis: forecasting and control*. San Francisco: Holden-Day, 1970.

Does Cost-Sensitive Learning Beat Sampling for Classifying Rare Classes?

Kate McCarthy, Bibi Zabar and Gary Weiss
Fordham University
441 East Fordham Road
Bronx, NY 10458
creed112@aol.com, zabar@fordham.edu, gweiss@cis.fordham.edu

ABSTRACT
A highly-skewed class distribution usually causes the learned classifier to predict the majority class much more often than the minority class. This is a consequence of the fact that most classifiers are designed to maximize accuracy. In many instances, such as for medical diagnosis, the minority class is the class of primary interest and hence this classification behavior is unacceptable. In this paper, we compare two basic strategies for dealing with data that has a skewed class distribution and non-uniform misclassification costs. One strategy is based on cost-sensitive learning while the other strategy employs sampling to create a more balanced class distribution in the training set. We compare two sampling techniques, up-sampling and down-sampling, to the cost-sensitive learning approach. The purpose of this paper is to determine which technique produces the best overall classifier—and under what circumstances.

Categories and Subject Descriptors
I.2.6 [Artificial Intelligence]: Learning – *Induction*
H.2.8 [Database Management]: Applications - *Data Mining*

General Terms
Algorithms

Keywords
Cost-sensitive learning, sampling, data mining, induction, decision trees, rare classes, class imbalance

1. INTRODUCTION
In many real-world domains, such as fraud detection and medical diagnosis, the class distribution of the data is skewed and the cost of misclassifying the minority class is substantially greater than the cost of misclassifying the majority class. In these cases, it is important to create a classifier that minimizes the overall misclassification cost. This tends to cause the classifiers to perform better on the minority class than if the misclassification costs were equal. For highly skewed class distribution, this also ensures that the classifier does not only predict the majority class.

The most direct method for dealing with highly skewed class distributions with unequal misclassification costs is to use cost-sensitive learning. An alternate strategy for dealing with skewed data with non-uniform misclassification costs is to use sampling to alter the class distribution of the training data so that the resulting training set is more balanced. There are two basic sampling methods for achieving a more balanced class distribution: up-sampling and down-sampling (also referred to as over-sampling and under-sampling). In this context, up-sampling replicates minority class examples and down-sampling discards majority class examples.

This paper compares cost-sensitive learning, up-sampling, and down-sampling to determine which method leads to the best overall classifier performance, where the best overall classifier is the one that minimizes total cost. Since sampling is often used instead of cost-sensitive learning in practice, we compare these methods to see which yields better results. Our conjecture is that cost-sensitive learning will outperform both up-sampling and down-sampling because of well-known problems (described in the next section) with these sampling methods. We evaluate this conjecture using C5.0 [18], a more advanced version of Quinlan's popular C4.5 program. We also evaluate this conjecture for data sets that are not skewed (but have non-uniform misclassification costs) to broaden the scope of our study. We compare cost-sensitive learning only to the basic up-sampling and down-sampling methods because these are the only methods available to most practitioners (some of the variants developed by researchers to address the weaknesses with sampling are discussed in Section 7).

2. BACKGROUND
In this section we provide basic background information on cost-sensitive learning, sampling, and the connection between the two. Some related work is also described.

2.1 Cost-Sensitive Learning
In this paper we focus our attention on two-class learning problems. The behavior of a classifier for such problems can be described by a confusion matrix. Figure 1 provides the terminology for such a confusion matrix. Holding with established practice, the positive class is the minority class and the negative class is the majority class.

UBDM '05, August 21, 2005, Chicago, Illinois, USA.

		ACTUAL	
		Positive class	Negative class
PREDICTED	Positive class	True positive (TP)	False positive (FP)
	Negative class	False negative (FN)	True negative (TN)

Figure 1: A Confusion Matrix

Corresponding to a confusion matrix is a cost matrix. The cost matrix will provide the costs associated with the four outcomes shown in the confusion matrix, which we refer to as C_{TP}, C_{FP}, C_{FN}, and C_{TN}. As is often the case in cost-sensitive learning, we assign no costs to correct classifications, so C_{TP} and C_{TN} are set to 0. Since the positive (minority) class is often more interesting than the negative (majority) class, typically $C_{FN} > C_{FP}$ (note that a false negative means that a positive example was misclassified).

A cost-sensitive learner can accept cost information from a user and assign different costs to different types of misclassification errors. Learners can implement cost-sensitive learning in a variety of ways. One common method is to alter the class probability thresholds used to assign the classification value. For example, in a decision tree learner the probability threshold associated with a terminal node is typically set to 0.5, so that the node is labeled with the most probable class. If the ratio of misclassification costs for a two-class problem is set to 2:1, then the class probability threshold would be 0.33 [9, 17]. Note that in this implementation of cost-sensitive learning no data is discarded or replicated.

When misclassification costs are known or can be assumed the best metric to evaluate overall classifier performance is total cost. Total cost is the only evaluation metric used in this paper and is used to evaluate the results for both cost-sensitive learning and sampling. The formula for total cost is shown below, in equation 1.

$$\text{Total Cost} = (FN \times C_{FN}) + (FP \times C_{FP}) \quad [1]$$

2.2 Sampling

Sampling can be used to alter the class distribution of the training data. As described earlier, this can be accomplished via up-sampling or down-sampling. Both sampling methods have been used to deal with skewed class distributions [1, 2, 3, 6, 10, 11]. The reason that altering the class distribution of the training data aids learning with highly-skewed data sets is that it effectively imposes non-uniform misclassification costs. For example, if one alters the class distribution of the training set so that the ratio of positive to negative examples goes from 1:1 to 2:1, then one has effectively assigned a misclassification cost ratio of 2:1. This equivalency between altering the class distribution of the training data and altering the misclassification cost ratio is well known and was formally established by Elkan [9].

Previous research on learning with skewed class distributions has altered the class distribution using up-sampling and down-sampling. There are disadvantages to using sampling to implement cost-sensitive learning, however. The disadvantage with down-sampling is that it discards potentially useful data. There are two disadvantages with up-sampling. First, it increases the size of the training set, which will increase the time necessary to learn the classifier. Second, since most up-sampling methods generate exact copies of existing examples, overfitting is likely to occur in that classification rules may be formed to cover a single, replicated example.

2.3 Why Use Sampling?

Given the disadvantages with sampling, it is worth asking why anyone would use sampling to deal with highly-skewed class distributions (with non-uniform misclassification costs) when cost-sensitive learning appears to be a more direct solution. In this section, we discuss several reasons for this. The most obvious reason is that many learning algorithms are not cost-sensitive and therefore a wrapper approach, like the one using sampling, is the only option. This is certainly less true today than in the past, but many of the older non-commercial learners still provide no mechanism for cost sensitive learning.

A second reason is that many highly skewed data sets are enormous and therefore require the size of the training set to be reduced. In this case, down-sampling seems to be a reasonable, and valid, strategy. In this paper, we do not consider the need to reduce the training set size. We would point out, however, that if one needs to discard some training data, it still might be beneficial to discard some of the majority class examples in order to reduce the training set size to the required size, and then also employ cost-sensitive learning, so that the amount of training data is not reduced beyond what is absolutely necessary.

A final reason one might give for using sampling instead of cost-sensitive learning is that the misclassification costs are often not known. This is not a valid reason for using sampling over cost-sensitive learning, however, since the same issue arises with sampling—what is the proper sampling rate? Ideally, the sampling rate should be based on the cost information. If that is not available, one might try various sampling rates and look at the performance of the induced classifier. However, the same strategy can be employed with cost-sensitive learning—various cost ratios can be evaluated and one can select the cost ratio based on the observed performance characteristics of the induced classifier. Alternatively, if misclassification costs are not known one can evaluate the performance of a classifier over a range of costs by using ROC analysis.

Overall, we feel that the only reason to use sampling to handle skewed class distributions is if the amount of available training data cannot be handled by the learning algorithm. Otherwise, our conjecture is that cost-sensitive learning should be used. We evaluate this conjecture in this paper.

3. DATA SETS

We used a total of fourteen data sets in our experiments. Twelve of the data sets were obtained from the UCI Repository and two of the data sets came from AT&T and were used in previously published work done by Weiss and Hirsh [16]. A summary of these data sets is provided in Table 1. The data sets are listed in descending order according to class imbalance (the most imbalanced data sets are listed first). The data sets marked with an asterisk (*) were originally multi-class data sets that were previously mapped into two classes for work done by Weiss and Provost [17]. The letter-a and letter-vowel data sets are derived from the letter recognition data set that is available from the UCI Repository.

The data sets were chosen on the basis of their class distributions and data set sizes. Although the main focus of our research is to compare cost-sensitive learning and sampling for classifying rare classes in imbalanced data sets, we also included a few data sets with more balanced class distributions to see if and how the overall results would differ. The boa1, promoters, and coding data sets each had an evenly balanced 50-50 distribution, so they were used for the sake of comparison. We used data sets of varying sizes to see how this would affect our results. One would expect that cost-sensitive learning would outperform down-sampling for small data sets, since throwing away any data in this situation should be harmful.

Since these data sets do not come with misclassification cost information, we evaluated the cost-sensitive and sampling strategies using a wide variety of costs. This is described in detail in the next section.

Table 1: Data Set Summary

Data Set	% Minority	Total Examples
Letter-a*	4%	20,000
Pendigits*	8%	13,821
Connect-4*	10%	11,258
Bridges1	15%	102
Letter-vowel*	19%	20,000
Hepatitis	21%	155
Contraceptive	23%	1,473
Adult	24%	21,281
Blackjack	36%	15,000
Weather	40%	5,597
Sonar	47%	208
Boa1	50%	11,000
Promoters	50%	106
Coding	50%	20,000

4. EXPERIMENTS

In this section we begin by describing C5.0, the learner used for our experiments. We then describe our experimental methodology for using cost-sensitive learning and sampling.

4.1 C5.0

All of our experiments utilize C5.0 [18], a commercial classifier induction program, which is a more advanced version of Quinlan's popular C4.5 and ID3 learners [14, 15]. Unlike these older programs, C5.0 supports cost-sensitive learning.

Both the cost-sensitive learning and sampling experiments used 75% of the data for training and 25% for testing. Each experiment was run ten times, using random sampling to create these two data sets. All results shown in this paper are the averages of these ten runs. Classifiers are evaluated using total cost, which was defined earlier in equation 1.

4.2 Cost-Sensitive Learning

In our experiments, we are interested in targeting the cases where the cost of incorrectly classifying a minority (positive) class example will have a higher cost than the cost of incorrectly classifying a majority (negative) class example. Hence we applied a higher misclassification cost to C_{FN}, the cost of a false negative misclassification. For our experiments, a false positive prediction, C_{FP}, was assigned a cost of 1, while C_{FN} was allowed to vary. For the majority of the experiments C_{FN} was evaluated for the values: 1, 2, 3, 4, 6, and 10, although for some experiments the costs were allowed to increase beyond this point.

4.3 Sampling

Up-sampling and down-sampling were used to implement the desired misclassification cost ratios, as described in Section 2.2. Since C5.0 does not provide the necessary support for sampling, the required sampling was done external to C5.0 and the resulting sampled training data was then passed to C5.0. No changes were made to the test data, but none were necessary since the resulting classifiers were evaluated using total cost, based on the cost information associated with each experiment. The misclassification cost ratios used for sampling were the same ones for cost-sensitive learning. Note that the greater cost ratio, the more training examples had to be discarded when down-sampling. The test set size was held fixed for all experiments.

5. RESULTS

Classifiers were generated for each data set using cost-sensitive learning, up-sampling and down-sampling for a variety of misclassification cost ratios. These classifiers were evaluated using total cost. We generated one figure for each of the fourteen data sets, showing how the total cost varies when cost-sensitive learning, up-sampling and down-sampling are used. Some of these figures are included in this section while the remaining figures can be found in the Appendix. After presenting these detailed results for each data set, summary statistics are provided which make it easier to compare and contrast the cost-sensitive learning method with the two sampling methods.

The results for the letter-a data set in Figure 2 show that cost-sensitive learning and up-sampling performed similarly whereas down-sampling performed much worse for all cost ratios (note that all methods will perform the same for 1:1). The letter-vowel data set, shown in Figure A1 in the Appendix, provides nearly identical results except that cost-sensitive learning performed slightly better than up-sampling for most cost ratios (both still outperform down-sampling).

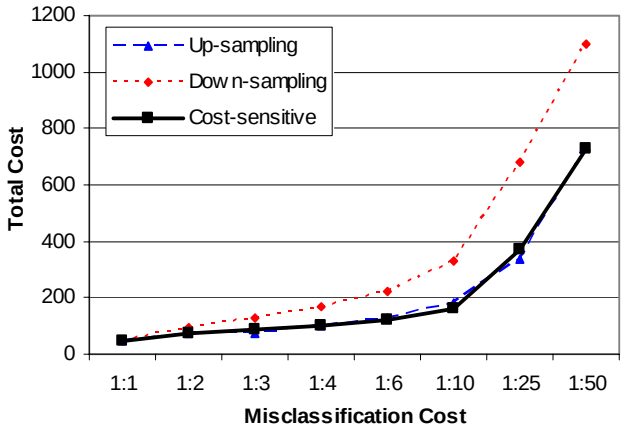


Figure 2: Results for Letter-a

The results for the weather data set, provided in Figure 3, show that up-sampling consistently performed much worse than down-sampling and cost-sensitive learning, both of which performed similarly. This *exact* same pattern also occurs in the results for the adult and boa1 data sets, which are provided in Figures A2 and A3, respectively, in the Appendix.

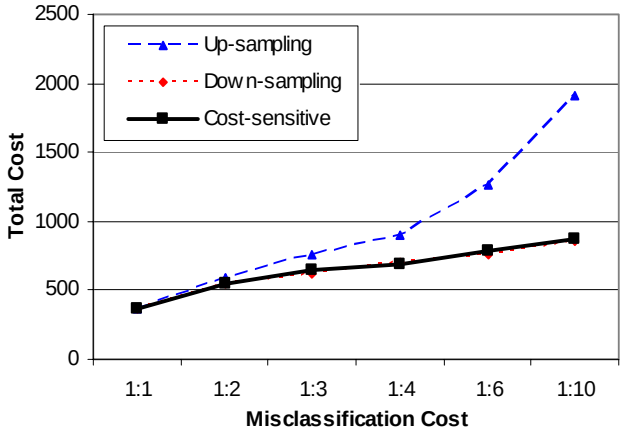


Figure 3: Results for Weather

The results for the coding data set in Figure 4 show that cost-sensitive learning outperformed both sampling methods, although the difference in total cost is much greater when compared to up-sampling. As we shall see shortly in Table 3, however, cost-sensitive learning still outperforms down-sampling by 9%, a substantial amount (it outperforms up-sampling by 20%).

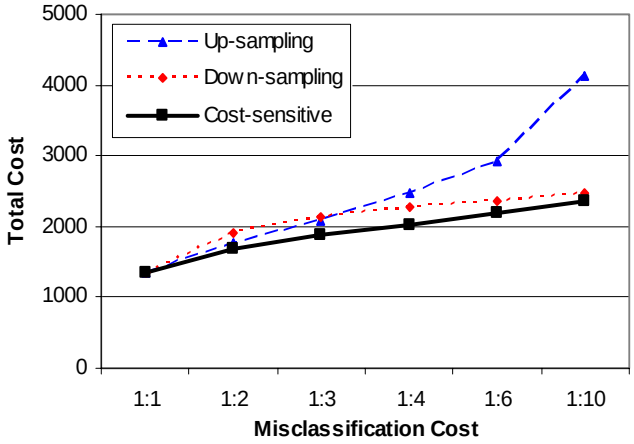


Figure 4: Results for Coding

The blackjack data set, shown in Figure 5, is the only data set for which all three methods yielded nearly identical performance for all cost ratios. The connect-4 data set (Figure A4) yielded nearly identical costs for all three methods as well, except for the highest cost ratio, 1:25, in which case up-sampling performed the worst.

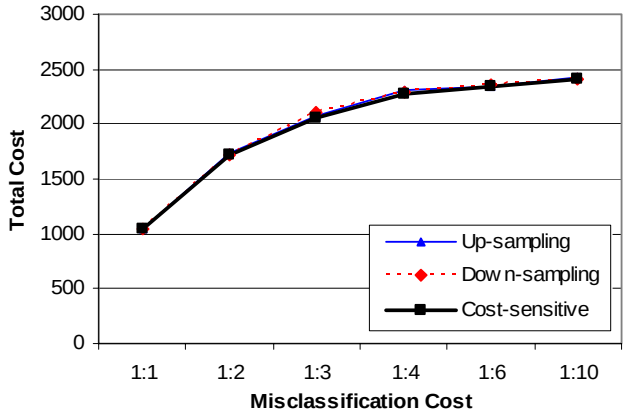


Figure 5: Results for Blackjack

There were three data sets for which the cost-sensitive method underperformed the two sampling methods for most cost ratios. This occurred for the contraceptive, hepatitis, and bridges1 data sets. The results for the contraceptive data set are shown in Figure 6, while the results for the hepatitis data set and bridges1 data set can be found in Figures A5 and A6 in the Appendix.

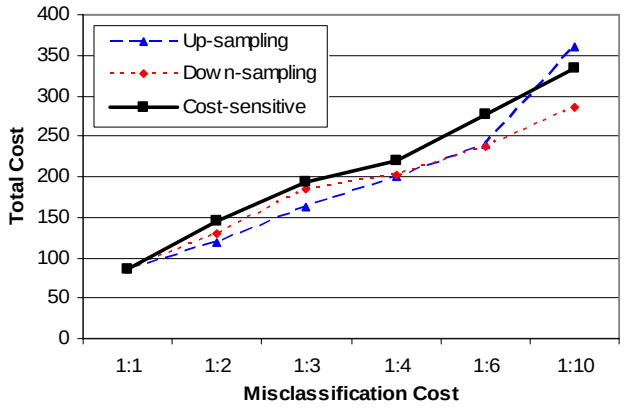


Figure 6: Results for Contraceptive

The sonar data set (Figure A7) is the only data set for which down-sampling consistently beats both the cost-sensitive and up-sampling method. The promoters data set (Figure A8) is the only data set for which up-sampling consistently beat the other two methods. We previously noted that the coding data set (Figure 4) is the only one in which the cost-sensitive method consistently beat the two sampling methods. Thus, we see that it is quite rare for any of the three methods to beat both of the other two methods—although it is common for each to beat one of the other methods. The only data set not yet discussed is the pendigits data set (Figure A9). Overall, the cost-sensitive learning method tends to beat both sampling methods for this data set, although the results vary by cost ratio.

Tables 2 and 3 summarize the performance of up-sampling, down-sampling, and cost-sensitive learning for all fourteen data sets. Table 2 specifies the first/second/third place finishes over the evaluated cost ratios for each data set and method. For example, Table 2 shows that for the letter-a data set up-sampling generates

the best results (i.e., lowest total cost) for 4 of the 7 evaluated cost ratios and the second best result for 3 of the 7 cost ratios.

Table 2: First/Second/Third Place Finishes

Data Set	Up-sampling	Down-sampling	Cost-Sensitive
Letter-a	4/3/0	0/0/7	3/4/0
Pendigits	3/1/3	1/2/4	3/4/0
Connect-4	2/0/3	0/3/2	3/2/0
Bridges1	5/0/0	0/5/0	0/3/2
Letter-vowel	4/1/0	0/0/5	1/4/0
Hepatitis	3/2/0	2/3/0	0/5/0
Contraceptive	3/2/0	2/3/0	0/1/4
Adult	2/3/0	3/1/1	0/4/1
Blackjack	1/1/3	2/1/2	3/2/0
Weather	0/0/5	4/1/0	1/4/0
Sonar	2/3/0	3/2/0	0/2/3
Boa1	0/0/5	4/1/0	2/3/0
Promoters	5/0/0	0/2/3	0/3/2
Coding	0/2/3	0/3/2	5/0/0
Total	33/18/22	21/27/26	21/41/12

The problem with Table 2 is that it does not quantify the improvements—the reduction in total cost. It treats all "wins" as equal even if the difference in costs between the methods is quite small. Table 3 remedies this by providing the relative reduction in cost for the strategies. The second and third columns compare cost-sensitive learning (abbreviated "Cost") versus up-sampling and down-sampling, respectively. The last column compares up-sampling to down-sampling. A negative value indicates an increase in cost rather than a reduction in cost. As an example, the results in Table 3 for the letter-a data set indicate that cost-sensitive learning performs slightly worse than up-sampling (-0.9%) but much better than down-sampling (37.9%) and that up-sampling performs much better than down-sampling (38.4%).

Table 3: Comparison of Relative Improvements

Data Set	Cost vs. Up-Sampling	Cost vs. Down-Sampling	Up- vs. Down-Sampling
Letter-a	-0.9%	37.9%	38.4%
Pendigits	3.5%	5.4%	0.9%
Connect-4	3.2%	-0.1%	-3.9%
Bridges1	-38.4%	-8.6%	21.2%
Letter-vowel	-7.7%	18.0%	23.7%
Hepatitis	-11.4%	-8.2%	2.3%
Contraceptive	-11.9%	-11.6%	-0.9%
Adult	8.7%	-0.8%	-12.0%
Blackjack	0.5%	0.5%	0.0%
Weather	27.9%	-1.3%	-50.0%
Sonar	-0.9%	-23.8%	-33.78%
Boa1	17.6%	-0.6%	-30.0%
Promoters	-40.6%	-1.2%	28.2%
Coding	20.0%	9.1%	-18.0%
Ave Savings	**-2.2%**	**1.1%**	**-2.4%**
Total Wins	**7**	**6**	**6**

The results from Table 2 and Table 3 show that cost-sensitive learning, as implemented in C5.0, does not consistently beat both or either of the sampling methods. Furthermore, none of three methods is a clear winner over all, or either, of the other methods. Overall, up-sampling seems to perform the best, by a relatively small margin, followed by cost-sensitive learning, with down-sampling doing the worst (based on total average savings). However, the results vary widely for each of the data sets. The best way to characterize the overall performance of the cost-sensitive approach based on Table 2 is that it rarely performs the worst. Even up-sampling, which performs the best overall, comes in last many more times (22 versus 12). Thus, one conclusion is that performance of cost-sensitive learning does not fluctuate quite as much as the sampling methods, over the different data sets.

6. DISCUSSION

Based on the results from all of the data sets, there was no definitive winner between cost-sensitive learning, up-sampling and down-sampling. Given that there is no clear and consistent winner, the logical question to ask is whether we can characterize under what circumstances each method performs best. We begin by analyzing the impact of data set size. Our study included four data sets (bridges1, hepatitis, sonar, and promoters) that are substantially smaller than the rest. If we compute the first/second/third place records for these four data sets from Table 2, we get the following results: up-sampling *15/5/0*, down-sampling *5/12/3* and cost-sensitive learning *0/13/7*. Based on this data, up-sampling clearly does much better than down-sampling and cost-sensitive learning. The data in Table 2 also supports this conclusion. The one exception is the sonar data set, where down-sampling beats up-sampling.

With the exception of the sonar results, the sampling results make sense. That is, we expect down-sampling, which throws away data, to perform more poorly than up-sampling for small data sets. The data also implies that up-sampling also outperforms cost-sensitive learning in these cases, however. One possible explanation for the failure of cost-sensitive learning in this situation is that when there is very little training data, it will be difficult to accurately estimate the class-membership probabilities—something that is required in order to get good results from cost-sensitive learning.

If we look at the eight data sets with over 10,000 examples each (letter-a, pendigits, connect-4, letter-vowel, adult, blackjack, boa, and coding), our results are as follows for first/second/third place finishes: up-sampling *16/11/17*, down-sampling *10/11/2*, and cost-sensitive *20/23/1*. The results from Table 3 show that over these eight data sets the average improvement between cost-sensitive learning and up-sampling is 5.5% and between cost-sensitive learning and down-sampling is 5.7%. Thus, for the large data sets, cost-sensitive learning does often yield the best results. Perhaps cost-sensitive learning does well in these cases because the larger amount of training data makes it easier to more accurately estimate the class-membership probabilities.

Another factor worth considering is the degree to which the class distribution of the data set is unbalanced. This will impact the extent to which sampling must be used to get the desired distribution. The results in Tables 2 and 3, which are ordered by decreasing class imbalance, show no obvious pattern, however.

Our results do not generally support our conjecture that cost-sensitive learning should outperform sampling for obtaining the best classifier performance. However, the results tend to indicate that the conjecture may hold for larger data sets. This suggests that perhaps cost-sensitive learning performs well only when there are sufficient data to generate accurate probability estimates (for c5.0 this translates to having many examples at each leaf node). We have found some supporting evidence to suggest why cost-sensitive learning is not a clear winner in all cases. Recent research [7] has shown that cost-sensitive learning, including C5.0's implementation of cost-sensitive learning, does not always produce the desired, and expected, results. Specifically, this research showed that one can achieve lower total cost by using a cost ratio for learning that is different from the actual cost information. This tends to indicate that there may be a problem with the cost-sensitive learning process.

7. RELATED WORK

Previous research has compared cost-sensitive learning and sampling. The experiments that we performed are similar to the work that was done by Chen, Liaw, and Breiman [6], who proposed two methods of dealing with highly-skewed class distributions based on the Random Forest algorithm. Balanced Random Forest (BRF) uses down-sampling of the majority class to create a training set with a more equal distribution between the two classes, whereas Weighted Random Forest (WRF) uses the idea of cost-sensitive learning. By assigning a higher misclassification cost to the minority class, WRF improves classification performance of the minority class and also reduces the total cost. However, although both BRF and WRF outperform existing methods, the authors found that neither one is consistently superior to the other. Thus, the cost-sensitive version of the Random Forest does not outperform the version than employs down-sampling.

Drummond and Holte [8] found that down-sampling outperforms up-sampling for skewed class distributions and non-uniform cost ratios. Their results indicate that this is because up-sampling shows little sensitivity to changes in misclassification cost, while down-sampling shows reasonable sensitivity to these changes. Breiman et al. [2] analyzed classifiers produced by sampling and by varying the cost matrix and found that these classifiers were indeed similar. Japkowicz and Stephen [10] found that cost-sensitive learning outperforms under-sampling and over-sampling, but only on artificially generated data sets. Maloof [12] also compared cost-sensitive learning to sampling but found that cost-sensitive learning, up-sampling and down-sampling performed nearly identically. However, because only a single data set was analyzed, one really could not draw any general conclusions from that data. Since we analyzed fourteen real-world data sets, we believe our research extends this earlier work and provides the most conclusive evidence that cost-sensitive learning does not clearly, or consistently, outperform up-sampling or down-sampling.

8. CONCLUSION

The results from our study indicate that between cost-sensitive learning, up-sampling, and down-sampling, there is no clear or consistent winner for maximizing classifier performance when cost information is known. If we focus exclusively on large data sets with more than 10,000 total examples, however, it appears that cost-sensitive learning often outperforms the sampling methods—although it still does not happen in every case. Note that in this study our focus was on using the cost information to improve the performance of the minority class, but in fact our results are much more general; they can be used to assess the relative performance of the three methods for implementing cost-sensitive learning. Our results also allow us to compare up-sampling to down-sampling. We found that up-sampling performed better than down-sampling overall, although the behavior varies widely for each data set.

There are a variety of enhancements that people have made to improve the effectiveness of sampling. While these techniques have been compared to up-sampling and down-sampling, they generally have not been compared to cost-sensitive learning. This would be worth studying in the future. Some of these enhancements include introducing new "synthetic" examples when up-sampling [5], deleting less useful majority-class examples when down-sampling [11] and using multiple sub-samples when down-sampling such than each example is used in at least one sub-sample [3].

In our research, we plotted classifier performance for different cost ratios and then summarized the results by recording the number of first/second/third place finishes for each method and also by averaging the results. We did this based on the assumption that the actual cost information will be known or can be estimated. This is not always the case and the reporting of our results could benefit by using other methods, such as ROC analysis or cost curves.

The implications of this research are significant. The fact that sampling, a wrapper approach, performs competitively—if not better—than a commercial tool that implements cost-sensitivity raises several important questions. These questions are: 1) why doesn't the cost-sensitive learner perform better given the known drawbacks with sampling, 2) are there ways we can improve cost-sensitive learners and 3) are we better off not using the cost-sensitivity features of a learner and using sampling instead. We hope to address these questions in future research.

9. REFERENCES

[1] Abe, N., Zadrozny, B., and Langford, J. An iterative method for multi-class cost-sensitive learning. *KDD '04*, August 22-25, 2004, Seattle, Washington, USA, 2004.

[2] Breiman, E., J Friedman, R. Olshen and C. Stone. *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group, 1984.

[3] Chan, P., and Stolfo, S. Toward scalable learning with non-uniform cost and class distributions: a case study in credit card fraud detection. *American Association for Artificial Intelligence*, 1998.

[4] Chawla, N. C4.5 and imbalanced datasets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure. *ICML 2003 Workshop on Imbalanced Datasets*.

[5] Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P. SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, Volume 16, 321-357, 2002.

[6] Chen C., Liaw, A., and Breiman, L. Using random forest to learn unbalanced data. Technical Report 666, Statistics Department, University of California at Berkeley, 2004. <http://www.stat.berkeley.edu/users/chenchao/666.pdf>

[7] Ciraco, M., Rogalewski, M., and Weiss, G. Improving classifier utility by altering the misclassification cost ratio. *Proceedings of the KDD-2005 Workshop on Utility-Based Data Mining*.

[8] Drummond, C., and Holte, R. C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. *Workshop on Learning from Imbalanced Data sets II, ICML*, Washington DC, 2003.

[9] Elkan, C. The foundations of cost-sensitive learning. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001.

[10] Japkowicz N, and Stephen, S. The class imbalance problem: a systematic study. *Intelligent Data Analysis Journal*, 6(5), 2002.

[11] Kubat, M and Matwin, S. Addressing the curse of imbalanced training sets: one-sided selection. *Proceedings of the Fourteenth International Conference on Machine Learning*, 179-186, 1997.

[12] Maloof, M. Learning when data sets are imbalanced and when costs are unequal and unknown. *ICML 2003 Workshop on Imbalanced Datasets*.

[13] Pednault, E., Rosen, B., and Apte, C. The importance of estimation errors in cost-sensitive learning. *IBM Research Report RC-21757*, May 30, 2000.

[14] Quinlan, J.R. C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, 1993.

[15] Quinlan, J.R. Induction of decision trees. Machine Learning 1: 81-106, 1986.

[16] Weiss, G., and Hirsh, H. A quantitative study of small disjuncts. *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, 2000.

[17] Weiss, G., and Provost, F. Learning when training data are costly: the effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 2003.

[18] "Data Mining Tools See5 and C5.0." RuleQuest, Nov. 2004. RuleQuest Research. May 13, 2005. <http://www.rulequest.com/see5-info.html>

APPENDIX

The results for the letter-vowel data set in Figure A1 show that up-sampling performed better than cost-sensitive learning for some cost ratios. Furthermore, both up-sampling and cost-sensitive learning perform better than down-sampling.

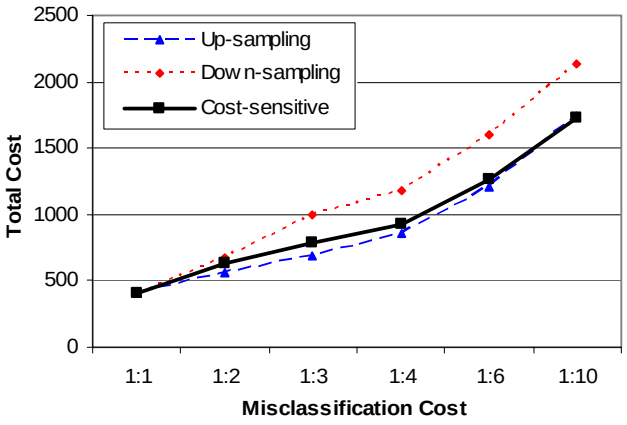


Figure A1: Results for Letter-vowel

The results for the adult data set in Figure A2 and the boa1 data set in Figure A3 both have up-sampling performing much worse than down-sampling and cost-sensitive learning, both of which perform similarly. These results mimic those of the weather data set in Figure 3 in the main body of this paper.

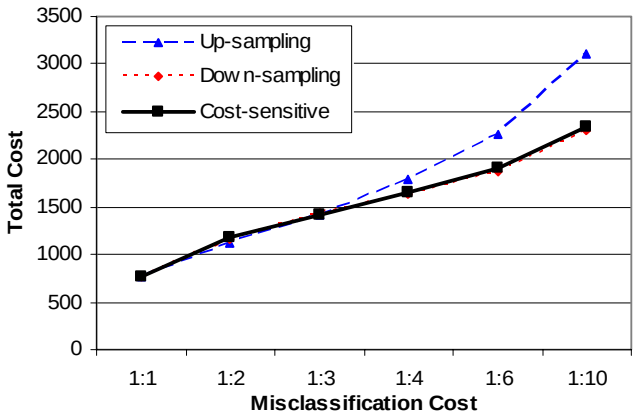


Figure A2: Results for Adult

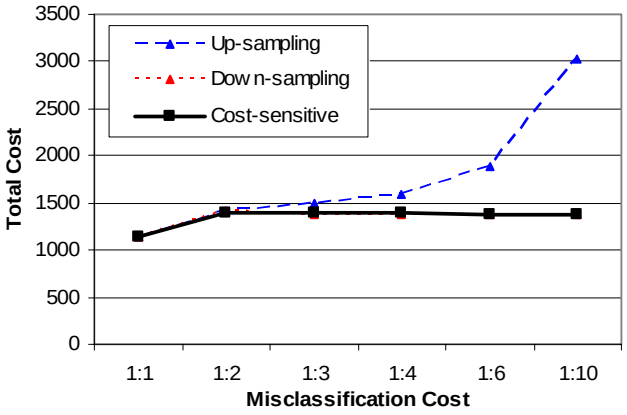


Figure A3: Results for Boa1

The connect-4 data set yields nearly identical performance for all three methods (like the blackjack data set in Figure 5), except for the 1:25 cost ratio.

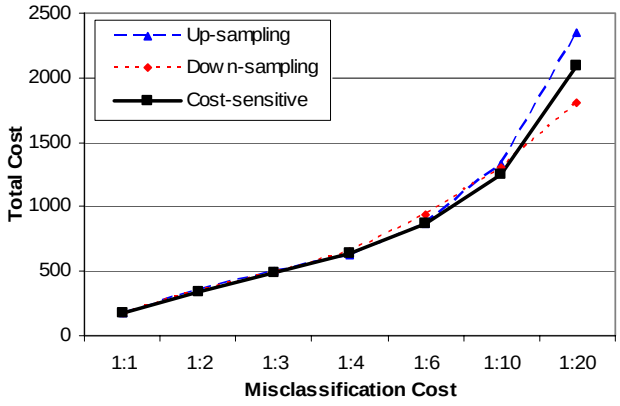


Figure A4: Results for Connect-4

The results for the hepatitis and bridges1 data sets in Figures A5 and A6 have the cost-sensitive method underperforming the two sampling methods for most cost ratios. The contraceptive data set in Figure 6 exhibited similar behavior.

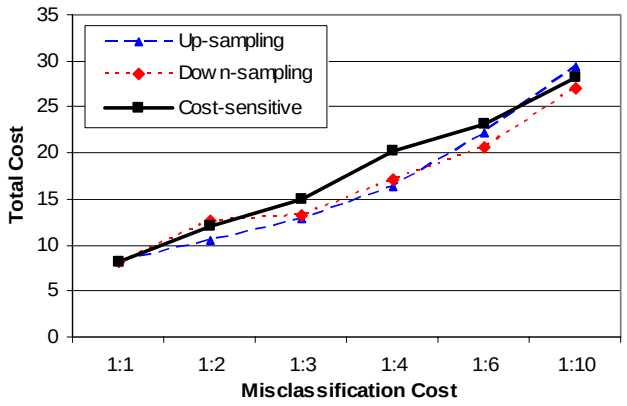


Figure A5: Results for Hepatitis

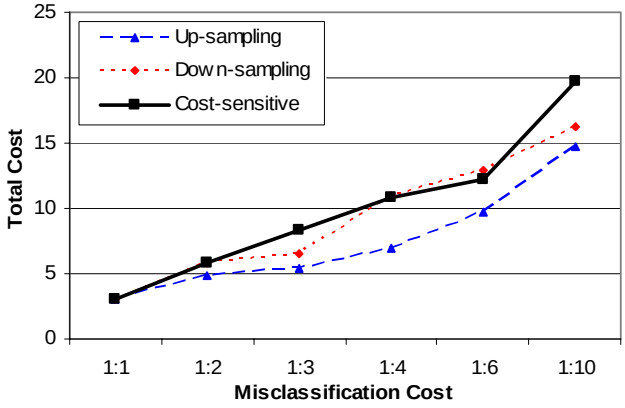


Figure A6: Results for Bridges1

The sonar data set is the only data set in which down-sampling substantially beat both cost-sensitive learning and up-sampling. This is unexpected since the sonar data set is quite small and one would expect down-sampling to perform worst in this situation (for other small data sets, down-sampling did in fact tend to perform poorly).

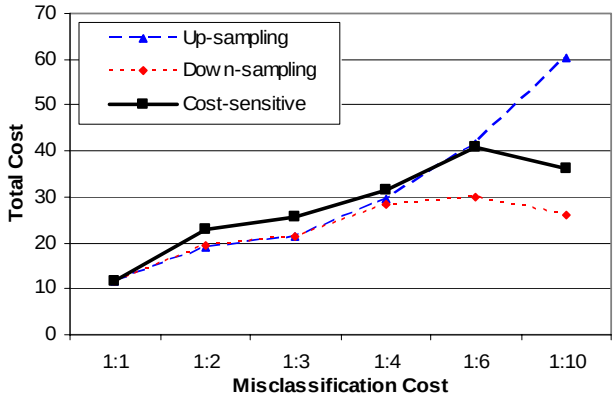


Figure A7: Results for Sonar

The promoters data set is the only data set for which up-sampling substantially beat both down-sampling and up-sampling.

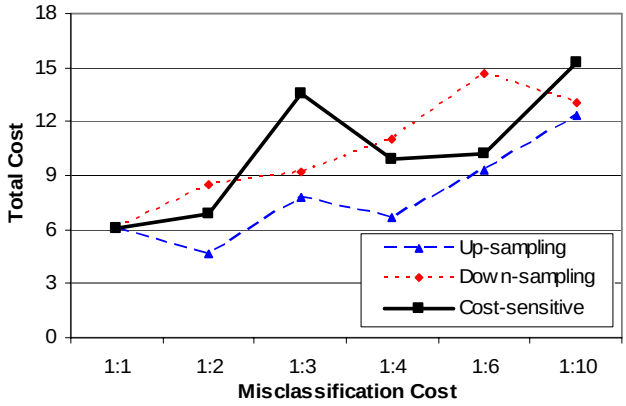


Figure A8: Results for Promoters

The results for the pendigits data set in Figure A9 vary for the different cost ratios, although the cost-sensitive learning method performs best overall.

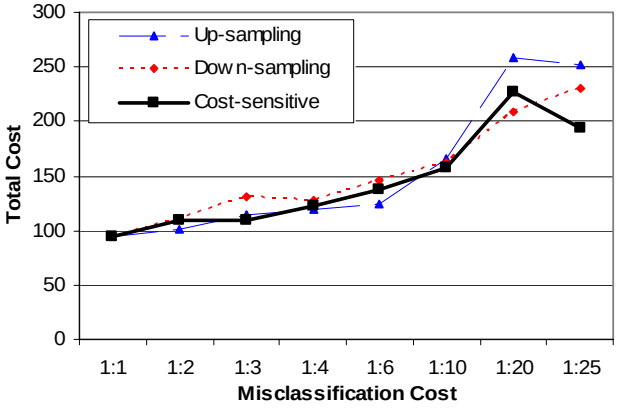


Figure A9: Results for Pendigits

Interruptible Anytime Algorithms for Iterative Improvement of Decision Trees

Saher Esmeir
Computer Science Department
Technion—IIT
Haifa 32000, Israel
esaher@cs.technion.ac.il

Shaul Markovitch
Computer Science Department
Technion—IIT
Haifa 32000, Israel
shaulm@cs.technion.ac.il

ABSTRACT

Finding a minimal decision tree consistent with the examples is an NP-complete problem. Therefore, most of the existing algorithms for decision tree induction use a greedy approach based on local heuristics. These algorithms usually require a fixed small amount of time and result in trees that are not globally optimal. Recently, the LSID3 contract anytime algorithm was introduced to allow using extra resources for building better decision trees. A contract anytime algorithm needs to get its resource allocation a priori. In many cases, however, the time allocation is not known in advance, disallowing the use of contract algorithms. To overcome this problem, in this work we present two interruptible anytime algorithms for inducing decision trees. Interruptible anytime algorithms do not require their resource allocation in advance and thus must be ready to be interrupted and return a valid solution at any moment. The first interruptible algorithm we propose is based on a general technique for converting a contract algorithm to an interruptible one by sequencing. The second is an iterative improvement algorithm that repeatedly selects a subtree whose reconstruction is estimated to yield the highest marginal utility and rebuilds it with higher resource allocation. Empirical evaluation shows a good anytime behavior for both algorithms. The iterative improvement algorithm shows smoother performance profiles which allow more refined control.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Concept-learning, Induction*; H.2.8 [**Database Management**]: Applications—*Data Mining*

General Terms

Algorithms, Experimentation

Keywords

Decision trees, Anytime algorithms, Anytime learning, Cost-quality tradeoff, Hard concepts

UBDM '05, August 21, 2005, Chicago, Illinois, USA.

1. INTRODUCTION

Despite the recent progress in developing advanced induction algorithms, such as Support Vector Machines [7], *decision trees* [6, 22] are still considered attractive for many real-life applications mostly due to their interpretability [13]. Craven and Shavlik [8] listed several reasons for the importance of the comprehensibility of learned classifiers. These reasons include, among others, the possibility to validate the induced model by human and to generate human-readable explanations for the classifier predictions.

Another model evaluation criterion that is mentioned in [8] is the flexibility of the model representation. In this manner, decision trees have a great advantage: they can be easily converted into logical rules. When classification cost is an important factor, decision trees are favored since they test only values of the features on the path from the root to the relevant decision leaf. In terms of accuracy, decision trees were shown to be competitive with other classifiers for several learning tasks [12, 28, 15, 32].

Based on the Occam's Razor principle [2], small decision trees that are consistent with the training examples have better predictive power than their larger counterparts. Finding the smallest consistent tree, however, was shown to be NP-complete [16, 19]. For this reason, most existing decision tree induction algorithms take a greedy approach and use local heuristics for choosing the best splitting attribute.

The greedy approach indeed performs quite well for many learning problems, and is able to generate decision trees very fast. In some cases, however, when the concept to learn is hard and the user is willing to allocate more time, the existing greedy algorithms are not able to exploit the additional resources for generating a better decision tree.

Algorithms that are able to trade resources for the quality of the output are called *anytime algorithms* [25, 3]. Recently, the LSID3 algorithm for anytime induction of decision trees has been introduced [9]. The algorithm performs repeated lookahead probes in order to evaluate candidate splitting attributes. The number of repetitions is determined in advance according to the allocated time.

LSID3 requires that the allocated time will be known ahead and does not guarantee any solution if the allocation is not honored, and hence is called a *contract anytime algorithm*. As such, LSID3 has two shortcomings. First, in many real-

life applications the time allocated for the learning phase is not known a priori. One example for such a setup is when the user is willing to allow the induction algorithm to run until the classifier is needed, and the time for this event is not known in advance. Another example is an application where the user wants the induction algorithm to run until it reaches some expected accuracy on a set-aside validation set. A second problem is that LSID3 assumes a mapping from the time allocation to the contract parameter, i.e., to the number of lookahead probes the algorithm can afford. In many cases, however, such a mapping is not possible.

To overcome the above problems, we need to come up with an *interruptible anytime algorithm*, which does not require the allocated time in advance and can therefore be interrupted anytime. In this work we present two interruptible anytime algorithms for decision tree induction, that can trade off the learning cost for the quality of the produced hypothesis and allow queries for solution at any moment. We start with a method that converts LSID3 to an interruptible algorithm using the general sequencing method described in [26]. This conversion, however, uses the contract algorithm as a black-box and hence cannot take into account specific aspects of decision tree induction. Next, we present a new repair-based algorithm, *IIDT*, that repeatedly replaces subtrees of the current hypothesis by subtrees generated with higher resource allocation and are therefore expected to be better. The two methods are empirically tested on datasets representing difficult concepts. Note that in this paper we assume a batch setup where all the training examples are given at the beginning of the learning process, unlike the incremental setup in which the induced tree is restructured when a new training instance becomes available [30].

2. CONTRACT INDUCTION OF DECISION TREES

The interruptible algorithms presented in the next sections both use a contract algorithm as a component. Specifically, in this paper we use the LSID3 algorithm. This section gives a short overview of this algorithm.

LSID3 adopts the top-down induction of decision trees (TDIDT) scheme. Under this framework, an attribute is chosen to partition the entire dataset into subsets, each of which is used to recursively build a subtree.

In ID3 each candidate split is evaluated by the information gain it yields and the attribute that maximizes this measure is selected. In LSID3 we measure the usefulness of a candidate split by the expected size of the subtree it results in. One can estimate this size by calling ID3 itself. Nevertheless, this results in a fixed time algorithm rather than in an anytime one. Moreover, ID3 might be insufficient to correctly predict the size. Therefor, in order to produce a better estimation of the tree size, instead of calling ID3 once, LSID3 samples the space of "good" trees by repeatedly invoking a stochastic version of ID3 (SID3). In SID3, instead of choosing the attribute that maximizes the information gain, the splitting attribute is drawn randomly with a likelihood that is proportional to the attribute's information gain. Since SID3 is not a deterministic algorithm, different runs of it might return different trees of different sizes. The size of each tree is an upper bound on the optimal tree size and hence we consider the minimal one as the estimator.

```
Procedure LSID3-Choose-Attribute(E, A, r)
  If r = 0
    Return ID3-Choose-Attribute(E, A)
  Foreach a ∈ A
    Foreach v_i ∈ domain(a)
      E_i ← {e ∈ E | a(e) = v_i}
      min_i ← ∞
      Repeat r times
        T ← SID3(E_i, A − {a})
        min_i ← min(min_i, Size(T))
    total_a ← Σ_{i=1}^{|domain(a)|} min_i
  Return a for which total_a is minimal
```

Figure 1: Attribute selection in LSID3.

```
Procedure Sequenced-LSID3(E, A)
    T ← ID3(E, A)
    i ← 0
    While not-interrupted
        r ← 2^i
        T ← LSID3(E, A, r)
        i ← i + 1
    Return T
```

Figure 2: Conversion of LSID3 to an interruptible algorithm by sequenced invocations.

Given an attribute a, LSID3 partitions the set of examples according to the different values a can take and calls SID3 several times for each subset. a is evaluated by summing up the estimated size of each subtree. For each subtree there are several estimations obtained from several calls to SID3. The algorithm considers the minimal one.

LSID3 is a contract algorithm parameterized by r, the number of times SID3 is called for each candidate. LSID3 with $r = 0$ is defined to be ID3. Figure 1 formalizes the choice of splitting attributes as made by LSID3. The runtime of LSID3 grows linearly with r. Let m be the number of examples and $n = |A|$ be the number of attributes. The worst-case time complexity of ID3 is $O(mn^2)$ [30]. It is easy to see that SID3 has the same worst-case complexity. LSID3(r) invokes SID3 r times for each candidate split. Recall the analysis in [30] for the time complexity of ID3, we can write the runtime of LSID3(r) as $\sum_{i=1}^{n} r \cdot i \cdot O(mi^2)$. An empirical based average-case analysis for ID3 showed that the complexity of ID3 is actually linear in n rather than quadratic i.e., $O(nm)$ [29]. Hence, we derive that the average case complexity of LSID3 is

$$\sum_{i=1}^{n} r \cdot i \cdot O(mi) = \sum_{i=1}^{n} O(rmi^2) = O(rmn^3). \qquad (1)$$

3. SEQUENCING CONTRACT ANYTIME ALGORITHMS

By definition, every interruptible algorithm can serve as a contract one. Russell and Zilberstein [26] showed that any

contract algorithm $\mathcal{A}$ can be converted into an interruptible algorithm $\mathcal{B}$ with a constant penalty. $\mathcal{B}$ is constructed by running $\mathcal{A}$ repeatedly with exponentially increasing time limits $\tau, 2\tau, \ldots, 2^i\tau, \ldots$ where τ is a free parameter that affects the granularity of the composed algorithm. Smaller values of τ will yield more frequent improvements, but there is no benefit in setting it to a too low value for which the improvement in quality is insignificant. It can be shown that the above sequence of runtimes is optimal when the different runs are scheduled on a single processor [26].

This general approach can be used to convert LSID3 into an interruptible algorithm. LSID3 gets its contract time in terms of r, the number of samplings per node. When $r = 0$, LSID3 is defined to be identical to ID3 which requires much less time than LSID3 with $r = 1$. Therefore, we slightly modify the sequencing method by first calling LSID3 with $r = 0$ and then continue according to the original method with exponentially increasing values of r, starting from $r = 1$. Figure 2 formalizes the resulting algorithm.

One problem with the sequencing approach is the exponential growth of the gaps between the points of time at which an improved result can be obtained. This is due to the generality of the algorithm that views the contract algorithm as a black-box. Thus, in the case of LSID3 at each iteration the whole decision tree is rebuilt. In Section 4 we present an interruptible anytime algorithm that instead of trying to rebuild the whole tree, iteratively improves subtrees.

4. INTERRUPTIBLE INDUCTION BY ITERATIVE IMPROVEMENT

In this section we present IIDT, an interruptible algorithm for decision-tree learning. As in LSID3, IIDT exploits additional resources in attempt to produce better trees. The key difference between the algorithms is that LSID3 uses the available resources to induce a decision tree top-down, where each decision made at a node is final and does not change. IIDT, on the contrary, does not get its resource allocation in advance and might be queried for a solution at any moment.

IIDT first performs a quick induction of an initial tree by calling ID3. It then iteratively attempts to improve the current tree by choosing a node, computing its next resource allocation and rebuilding the subtree below it. If the newly induced subtree is better than the existing one, a replacement takes place. We formalize IIDT in Figure 3.

Figure 4 illustrate the way IIDT works. The target concept is $a_1(x) \oplus a_2(x)$ with additional two irrelevant attributes a_3 and a_4. The leftmost tree was constructed using ID3. In the first iteration the subtree rooted at the bolded node is selected for improvement and replaced by a smaller tree (surrounded by a dashed line). Next, the root is selected for improvement and the whole tree is replaced by a tree that perfectly describes the concept.

IIDT is designed as a general framework for interruptible learning of decision trees that allows using different approaches for choosing the node to improve, for allocating resources for an improvement iteration, for rebuilding a subtree and for deciding whether an alternative subtree is better or not. In

```
Procedure IIDT(E, A)
    T ← ID3(E, A)
    While not-interrupted
        node ← CHOOSE-NODE(T, E, A)
        t ← subtree of T rooted at node
        A_node ← {a ∈ A | a ∉ ancestor of node}
        E_node ← {e ∈ E | e reaches node}
        r ← NEXT-R(node)
        t' ← REBUILD-TREE(E_node, A_node, r)
        If EVALUATE(t) > EVALUATE(t')
            replace t with t'
    Return T
```

Figure 3: Interruptible learning of decision trees.

the remainder of this section we focus on the components of IIDT and suggest a possible implementation that is based on LSID3.

4.1 Reconstructing a Subtree

After deciding upon the amount of resources allocated for the reconstruction process, the problem becomes a task for a contract algorithm. A good candidate for such an algorithm is LSID3 which exhibited good anytime performance in the empirical study reported in [9]. We expect that calling LSID3 with higher resource allocation will result in a better subtree.

4.2 Choosing a Subtree to Improve

Intuitively, the next node we would like to improve is the one with the highest expected marginal utility, i.e., the one with the highest ratio of expected benefit and expected cost [14, 24]. Estimating the expected cost and expected gain of rebuilding a subtree is a difficult problem. There is no apparent way for estimating the expected improvement either in terms of tree size or generalization accuracy. In addition, precise prediction of the resources to be consumed by LSID3 is not an easy task. In the remainder of this subsection we show how to approximate these values, and how to incorporate these approximations into the node selection algorithm.

4.2.1 Resource Allocation

The LSID3 algorithm receives its resource allocation in terms of r, the number of samplings devoted for each attribute. We adopt here the above mentioned strategy that doubles the amount of resource allocation at each iteration. Thus, if the resources allocated for the last improvement attempt of *node* were $r = \text{LAST-R}(node)$, the next allocation will be $2r$.[1]

4.2.2 Expected Cost

The expected cost can be approximated using the average time complexity of LSID3 as expressed by Equation 1. For each *node*, we estimate the expected runtime of LSID3(r) to rebuild the subtree below it by $\text{COST}(node) = \text{NEXT-R}(node) \cdot m \cdot n^3$.

[1]Note that LAST-R can be inherited from an ancestor of *node*, in case the subtree rooted at *node* was reproduced as a part of a containing tree.

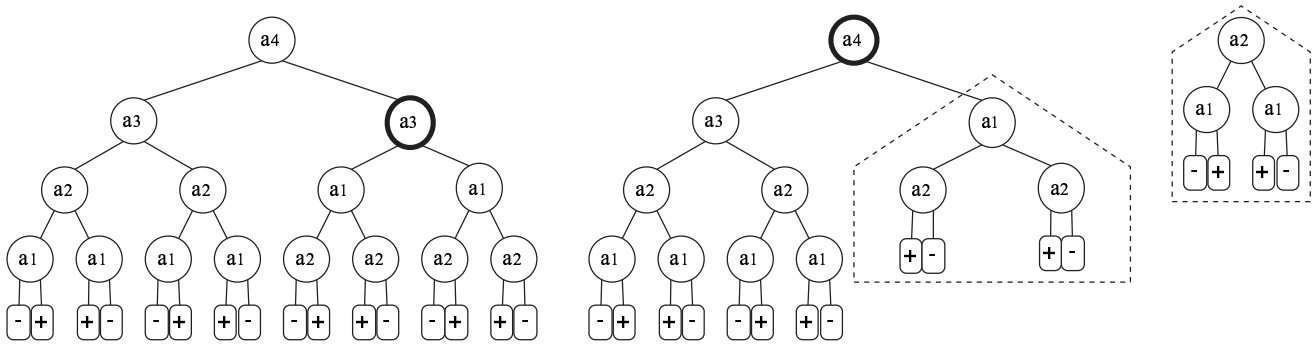


Figure 4: Iterative improvement of the decision tree produced for the 2-XOR concept $a_1(x) \oplus a_2(x)$ with additional two irrelevant attributes a_3 and a_4.

We observe that in terms of expected cost, subtrees rooted in deeper levels are preferred since they have less examples and attributes to consider and thus have shorter expected runtime. We also observe that since for each node the next time allocation doubles the previous one, nodes that were previously selected for improvement a large number of times, will have higher associated costs and thus are less likely to be chosen again.

4.2.3 *Expected benefit*

The whole framework of decision trees induction rests on the assumption that smaller consistent trees are better than large ones. Therefore the size of a subtree can serve as a measure for its quality. We cannot, however, know or estimate the size of the reconstructed subtree before actually building it. Therefore, we use instead an upper limit on the reduction in size that can be achieved.

The minimal size possible for a decision tree is obtained when all examples are labelled with the same class. Such cases are easily recognized by the greedy ID3 and by LSID3. Similarly if a subtree was replaceable by another subtree of depth 1, i.e., consists of a single split, ID3 (and LSID3) would have chosen the smaller subtree. Thus, the maximal reduction of the size of an existing subtree is to the size of a tree of depth 2. Assuming that the maximal number of values per attribute is b, the maximal size of such a tree (measured by the number of leaves) is b^2. Hence, an upper bound on the benefit from reconstructing a tree t that was previously induced is $\textsc{Size}(t) - b^2$.

Ignoring the expected costs, and relying solely on the expected benefit results in always giving the highest score to the root node. This makes sense: assuming we have infinite resources, we would attempt to improve the whole decision tree rather than parts of it.

4.2.4 *Granularity*

Considering the cost and benefit approximations described above, the selection procedure would prefer deep nodes (that are expected to have low costs) with large subtrees (that are expected to yield large benefits). When no such large subtrees exist, our algorithm may repeatedly attempt to improve smaller trees rooted at deep nodes due to their low associated costs. In the short term, such a behavior would indeed be beneficial but in the long term it can be harmful since when the algorithm later improves subtrees in upper levels, the resources spent on deeper nodes are wasted. On the other hand, if the algorithm would have first selected the upper level trees, this waste would be avoided, but the time gaps between potential improvements would have been increased.

To allow control of the tradeoff between the efficiency of resource usage and the flexibility of anytime control we add a granularity parameter $0 \leq g \leq 1$ that serves as a threshold for the minimal time allocation to an improvement phase. A node can be selected for improvement only if its normalized expected cost is above g. To compute the normalized expected cost, we divide the expected cost by the expected cost of the root node. Note that by this definition, it is possible to have nodes with cost which is higher than the cost of the root node, and therefore with relative cost higher than one. Such nodes, however, can never be selected for improvement since their expected benefit is necessarily lower than the expected benefit of the root node. Hence, when $g = 1$, IIDT is forced to choose the root node and its behavior becomes identical to the sequencing algorithm described in Section 3.

Figure 5 formalizes the procedure for choosing a node for reconstruction.

4.3 Evaluating a Subtree

Although LSID3 was shown to produce better trees when allocated more resources, an improved result is not guaranteed. Thus, to avoid a degradation in the quality of the induced tree, we replace an existing subtree only if the alternative is expected to improve the quality of the complete decision tree. Following Occam's Razor, we measure the usefulness of a subtree by its size. Only if the reconstructed subtree is of a smaller size, would it replace an existing subtree. This guarantees that the size of the complete decision tree is monotonically decreasing.

Another possible measure is the accuracy of the decision

Procedure CHOOSE-NODE(T, E, A)
  Foreach $node \in T$
    $A_{node} \leftarrow \{a \in A \mid a \notin$ ancestor of $node\}$
    $E_{node} \leftarrow \{e \in E \mid e$ reaches $node\}$
    $r_{node} \leftarrow$ NEXT-R($node$)
    $cost_{node} \leftarrow r_{node} \cdot |E_{node}| \cdot |A_{node}|^3$
    $max\text{-}cost \leftarrow$ NEXT-R($root$) $\cdot |E| \cdot |A|^3$
    If $(cost_{node}/max\text{-}cost) > g$
      $l\text{-}bound \leftarrow (\min_{a \in A_{node}} |\text{DOMAIN}(a)|)^2$
      $\Delta q \leftarrow$ LEAVES($node$) $- \ l\text{-}bound$
      $u_{node} \leftarrow \Delta q / cost_{node}$
  $best \leftarrow node$ that maximizes u_{node}
  Return $\langle best, r_{best} \rangle$

Procedure NEXT-R($node$)
  If LAST-R($node$) $= 0$
    Return 1
  Else
    Return $2 \cdot$ LAST-R($node$)

Figure 5: Choosing a node for reconstruction.

tree on a set-aside validation set of examples. Only if the accuracy on the validation set increases, the modification is applied. This measure suffers from two drawbacks. The first is that putting aside a set of examples for validation results in a smaller set of training examples and thus makes the learning process harder. The second is the bias towards overfitting the validation set, that might reduce the generalization abilities of the tree.

5. EXPERIMENTAL EVALUATION

A variety of experiments were conducted to test the performance and anytime behavior of IIDT. We compare two versions of IIDT to the fixed time algorithms ID3 and C4.5. Both versions of IIDT use the components described in Section 4. IIDT(1) is parameterized with a granularity factor 1 and thus behaves exactly as the sequencing method described in Section 3, while in IIDT(0.1) the granularity factor is set to 0.1. In addition, we tested a version of IIDT that evaluates subtrees by their accuracy on a validation set rather than their size. However, this modification did not help and in some cases a degradation in the performance was observed. Thus, in what follows we describe the results for the size-based estimation.

The behavior of anytime learners on easy concepts is not interesting since the greedy algorithms are able to produce good trees with small allocation of resources. Therefore, we present here the results for more complex concepts that can benefit from larger resource allocation: the *Glass* and the *Tic-Tac-Toe* UCI datasets [1], the *20-Multiplexer* dataset [23] and the 10-XOR dataset, generated with additional 10 irrelevant attributes. Table 1 summarizes the basic characteristics of the used datasets.

The performance of the different algorithms is compared both in terms of generalization accuracy and size of the induced trees, measured by the number of leaves. Following the recommendations of Bouckaert [4], 10 runs of 10-fold cross-validation experiment were conducted for each dataset.

Dataset	Instances	Attributes		Classes
		Nominal	Numeric	
Glass	214	10	9	7
Tic-Tac-Toe	958	9	0	2
Multiplexer-20	500	20	0	2
XOR-10	10000	20	0	2

Table 1: Characteristics of the datasets used.

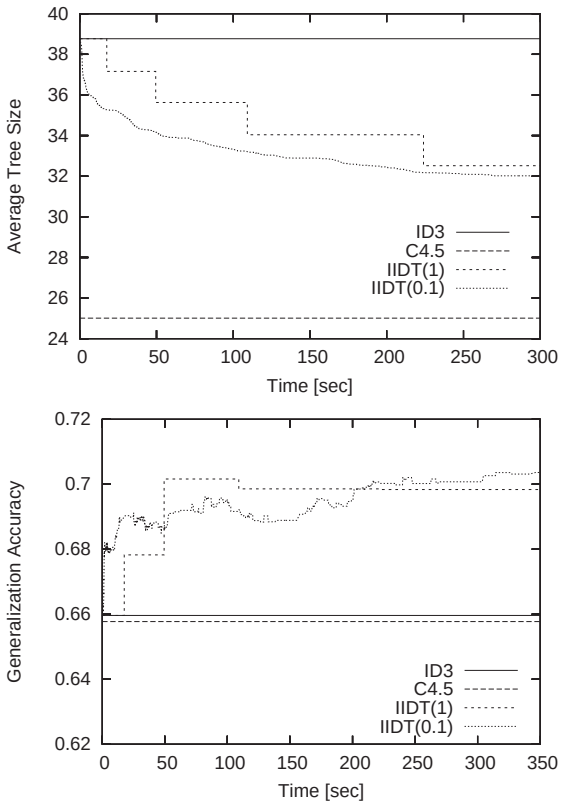


Figure 6: Anytime behavior on the Glass dataset.

Figures 6, 7, 8 and 9 show the anytime graphs for both tree size and accuracy for the 4 datasets. Each graph represents an average of 100 runs (for the 10×10 cross validation). In all cases the both anytime versions indeed exploit the additional resources and produce better trees, both in terms of size and accuracy.[2] Since our algorithm replaces a subtree only if the new one is smaller, all size graphs decrease monotonically. The most interesting anytime behavior is for the difficult 10-XOR problem. There, the tree size decreases from 4000 leaves to almost the optimal size 2^{10}, and the accuracy is increased from 50% (which is the accuracy achieved by ID3 and C4.5) to almost 100%. The shape of the graphs is typical to anytime algorithms with diminishing returns. The difference between the performance of the two anytime algorithms is interesting. IIDT(0.1) with the lower granu-

[2]Note that in some cases C4.5 produces smaller (yet less accurate) trees since it allows inconsistency with the training data by post-pruning the tree.

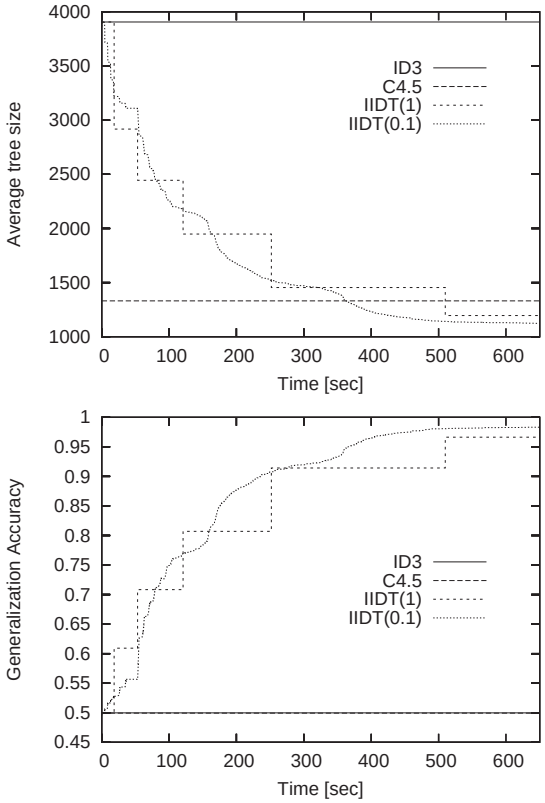


Figure 7: Anytime behavior on the 10-XOR dataset.

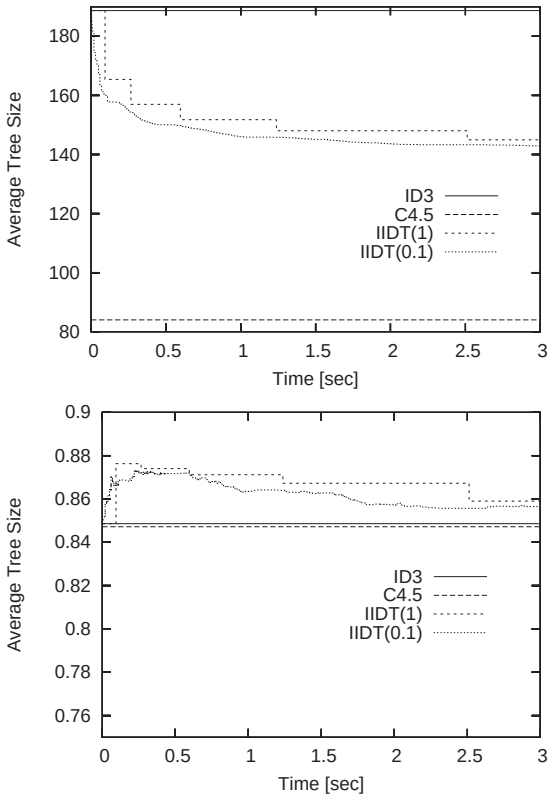


Figure 8: Anytime behavior on the Tic-Tac-Toe dataset.

larity parameter indeed produces smoother anytime graphs (with lower volatility) which allows for better control and better predictability of return. Moreover, in large portions of the time axis, the IIDT(0.1) graph dominates the one for IIDT(1) due to its more sophisticated node selection.

The smoothness of the IIDT(0.1) graph is somehow misleading since it represents an average of 100 step graphs with steps occurring in different time points (vs. the graph for IIDT(1) where the steps are roughly at the same time points). Figure 10 shows one anytime graph (out of the 100). We can see that although the IIDT(0.1) graph is less smooth than the average, it is still much smoother than the corresponding IIDT(1) graph.

6. RELATED WORK

While, to our knowledge, no other work tried specifically to design an anytime interruptible algorithm for decision tree induction, there are several related works that need to be discussed here. Opitz introduced an anytime approach for theory refinement [20]. This approach starts by generating a knowledge-base neural network from a set of rules, and then it uses the training data and the additional time resources in an attempt to improve the resulted hypothesis.

Lizotte et al. [18] presented a model for a budgeted learning task. In their work the term budgeted learning refers to the problem of collecting a data sample under budget constraints for the total cost of the tests that can be taken. Although this notation is equivalent to anytime contract algorithms the problem dealt by Lizotte et al. is different than this faced by our anytime approach: while the first attempts to find the best way to spend a budget for collecting a sample of data, we assume that the dataset has already been obtained and address the question of how to exploit our budget to learning a better hypothesis from this data.

Pruning techniques also attempt to obtain smaller decision trees, but their goals and their search space are different. The main goal of pruning is to avoid overfitting the data. Pruning techniques are orthogonal to our approach and tackle different problems. We intend to integrate pruning phases in IIDT and thus allow handling overfitting problems.

Ensemble-based methods can also be viewed as anytime algorithms. The boosting method [27] iteratively refines the constructed ensemble by increasing the weight of misclassified instances and adding a new hypothesis learned based on the updated weights. This process can continue as long as the time allocation allows. In bagging [5] a committee of trees is formed by making bootstarp replicates of the training set and using each such replication to learn a decision tree. Additional resources can be exploited to generate larger committees. Unlike the problem we face in this work, the classifiers constructed by the boosting and bagging algorithms consist of ensembles of decision trees rather than a single tree. A major problem with ensemble-based meth-

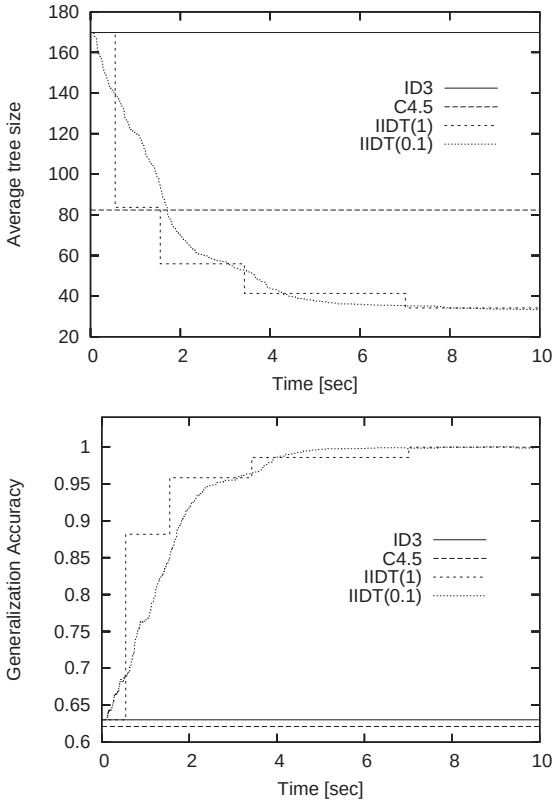


Figure 9: Anytime behavior on the 20-Multiplexer dataset.

ods is that in many cases the induced ensemble is large, complex and difficult to interpret [11]. Another problem is that when the concept to learn is hard, greedy trees are unable to discover any knowledge about the target concept and hence their combination cannot improve the performance. To experimentally test this, we examined the performance of Bagging on the XOR10 dataset. Our results indicate that the committee failed to learn the concept and performed no better than a random guesser, even for a large number of tree-members (up to 1000). In the future, we intend to empirically compare the anytime behavior of of other ensemble methods such as Boosting and Random Decision Tree [10] to IIDT, as well as examining committees of trees produced by more expensive algorithms such as LSID3.

Papagelis and Kalles [21] presented GATree, an algorithm that uses genetic algorithms to evolve decision trees. When tested on several UCI datasets, GATree was reported to produce trees as accurate as C4.5 but of significantly smaller size. GATree can be viewed as an anytime interruptible algorithm that uses additional time to produce more and more generations. We conducted several experiments with GATree, with its default parameters as reported in [21]. For this purpose we used the free GATree version available in `http://www.GATree.com`. The results indicate that the anytime behavior of GATree is problimatic. Although improvements were observed, they were not consistent and the results suffered from considerable fluctuations. In addition,

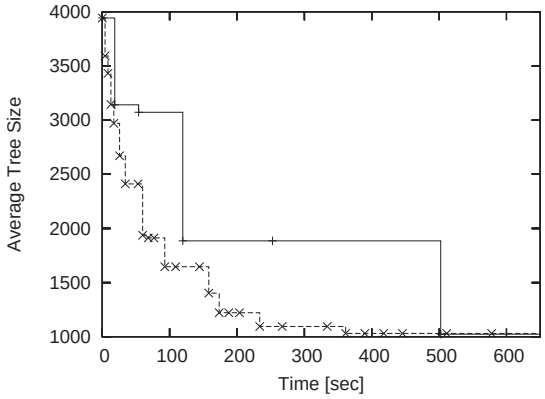


Figure 10: Time steps for an arbitrary run on 10-XOR.

we tested our IIDT on the parity concepts used to evaluate GATree. Although we could not use exactly the same datasets, we followed the same method to create them and the results show that IIDT achieved better results than those reported for GATree. For example, for the 4 attributes parity problem with 6 additional irrelevant attributes, IIDT was able to reach 99% accuracy while GATree was reported to have 85% average accuracy.

Utgoff [31] presented DMTI, an induction algorithm that uses a direct measure of tree quality instead of greedy heuristic to evaluate the possible splits. Several possible tree measure were examined and the MDL (Minimum Description Length) measure had the best performance. DMTI can use a fixed amount of additional resources and hence cannot serve as interruptible anytime algorithm. Further, DMTI uses the greedy approach to produce the lookahead trees and that might be insufficient to well-estimate the usefulness of a split.

Last et al. [17] introduced an interruptible anytime algorithm for feature selection. Their proposed method selects features by constructing an information-theoretic connectionist network, which represents interactions between the input attributes and the target class.

7. CONCLUSIONS

In this work explored the problem of how to produce better decision trees when more time resources are available. Unlike the contract setup that was addressed in a previous study, this work does not assume a priori knowledge of the amount of the resources available and allows the user to interrupt the learning phase at any moment.

The major contribution of this paper is the IIDT framework that can be adjusted to use any contract algorithm for reproducing a decision tree and any measure for choosing the subtree to rebuild. We studied an instantiation of this framework that bases the decision of what subtree to rebuild next on the expected cost and expected benefit and uses LSID3 for rebuilding subtrees.

The reported experimental study shows that IIDT exhibits

a good anytime behavior allowing a tradeoff between the cost of the learning process and the quality of the induced hypothesis. The smoothness of the performance profiles was shown to be flexibly controlled by the granularity parameter.

In the future we, intend to apply monitoring techniques for optimal scheduling of IIDT. In addition, we plan to integrate pruning phases in the IIDT framework as well as examining several different strategies for choosing nodes and improving subtrees.

8. REFERENCES

[1] C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998.

[2] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's Razor. *Information Processing Letters*, 24(6):377–380, 1987.

[3] M. Boddy and T. L. Dean. Deliberation scheduling for problem solving in time constrained environments. *Artificial Intelligence*, 67(2):245–285, 1994.

[4] R. R. Bouckaert. Choosing between two learning algorithms based on calibrated tests. In *ICML'03*, pages 51–58, Washington, DC, USA, 2003.

[5] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[6] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[7] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[8] M. W. Craven and J. W. Shavlik. *Extracting Comprehensible Models from Trained Neural Networks*. PhD thesis, Computer Science department, University of Wisconsin, Madison, 1996.

[9] S. Esmeir and S. Markovitch. Lookahead-based algorithms for anytime induction of decision trees. In *ICML'04*, pages 257–264, 2004.

[10] W. Fan, H. Wang, P. S. Yu, and S. Ma. Is random model better? on its accuracy and efficiency. In *ICDM'03*, pages 51–58, 2003.

[11] Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 124–133, Bled, Slovenia, 1999.

[12] E. Gabrilovich and S. Markovitch. Text categorization with many redundant features: Using aggressive feature selection to make svms competitive with c4.5. In *ICML'04*, pages 321–328, 2004.

[13] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. New York: Springer-Verlag, 2001.

[14] E. Hovitz. *Computation and Action under Bounded Resources*. PhD thesis, Computer Science Department, Stanford University, 1990.

[15] J. Huang, J. Lu, and C. Ling. Comparing naive bayes, decision trees, and svm using accuracy and auc. In *ICDM'03*, Melbourne, FL, 2003.

[16] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[17] M. Last, A. Kandel, O. Maimon, and E. Eberbach. Anytime algorithm for feature selection. In *RSCTC'01*, pages 532–539. Springer-Verlag, 2001.

[18] D. J. Lizotte, O. Madani, and R. Greiner. Budgeted learning of naive bayes classifiers. In *UAI'03*, Acapulco, Mexico, 2003.

[19] O. J. Murphy and R. L. McCraw. Designing storage efficient decision trees. *IEEE Transactions on Computers*, 40(3):315–320, 1991.

[20] D. W. Opitz. *An Anytime Approach to Connectionist Theory Refinement: Refining the Topologies of Knowledge-Based Neural Networks*. PhD thesis, Department of Computer Sciences, University of Wisconsin-Madison, 1995.

[21] A. Papagelis and D. Kalles. Breeding decision trees using evolutionary techniques. In *ICML'01*, pages 393–400, San Francisco, CA, USA, 2001.

[22] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[23] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[24] S. J. Russell and E. Wefald. Principles of metareasoning. In *KR'89*, pages 400–411, San Mateo, California, 1989.

[25] S. J. Russell and S. Zilberstein. Composing real-time systems. In *IJCAI'91*, pages 212–217, Sydney, Australia, 1991.

[26] S. J. Russell and S. Zilberstein. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1-2):181–213, 1996.

[27] R. Schapire. A brief introduction to boosting. In *IJCAI'99*, pages 1401–1406, Stockholm, Sweden, 1999.

[28] Y. Shan, E. Milios, A. Roger, C. Blouin, and E. Susko. Automatic recognition of regions of intrinsically poor multiple alignment using machine learning. In *CSB'03*, 2003.

[29] J. W. Shavlik, R. J. Mooney, and G. G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6(2):111–143, 1991.

[30] P. E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4(2):161–186, 1989.

[31] P. E. Utgoff, N. C. Berkman, and J. A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, 1997.

[32] X. Zhu, S. Sun, S. E. Cheng, and M. Bern. Classification of protein crystallization imagery. In *EMBS'04*, San Francisco, CA, 2004.

Contextual Recommender Problems

[Extended Abstract]

Omid Madani
madani@yahoo-inc.com

Dennis DeCoste
decosted@yahoo-inc.com

Yahoo! Research
74 N. Pasadena Ave, 3rd floor
Pasadena, CA 91103

ABSTRACT

The contextual recommender task is the problem of making useful offers, *e.g.*, placing ads or related links on a web page, based on the context information, *e.g.*, contents of the page and information about the user visiting, and information on the available alternatives, *i.e.*, the advertisements or relevant links. In the case of ads for example, the goal is to select ads that result in high click rates, where the (ad) click rate is some unknown function of the attributes of the context and ad. We describe the task and make connections to related problems including recommender and multi-armed bandit problems.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning - Induction

General Terms

Algorithms

Keywords

Recommenders, Multi-Armed Bandit, Personalization, Exploration-Exploitation, Regression, Reinforcement Learning, Data Mining, Utility

1. INTRODUCTION

Users (browsers) select pages to view and the task is to put one or more ads on such pages. The contextual (ad) problem consists of making such selection and placement decisions in order to maximize the expected return over some period of time, where expected return is a function of the likelihood of the ads being clicked – and possibly even a transaction or purchase taking place – and the prices of those clicked ads. Information that may significantly aid such decisions include page and user attributes, such as content and site information and users' recent behavior, and attributes of the available ads, such as ad content and bid prices. The horizon or time period for optimization will also implicitly or explicitly figure into the problem. However, a major challenge is *sparsity*: there may be many ads available (*e.g.*, millions), whereas the number of interactions we may get from a single typical user, in a time period of interest, may be very small in comparison (*e.g.*, a handful a day), and furthermore click rates for arbitrary ads are relatively small as well (*e.g.*, one percent).

We explore a number of different ways of viewing the problem. These viewpoints reveal the different aspects of the task, or may just reflect the type of available resources and data. While we focus on the task of selecting ads to show (*contextual adevertising*), we expect that the abstraction, the *contextual recommendation problem*, applies to other tasks such as (personalized) web search and web page organization (see Section 7). Our focus in this paper is on an informal exploration. We leave fomralizations and concrete solutions to future work.

The paper is organized as follows. Section 2 describes the information that should be useful. It identifies an important distinction: the system has some control over choice of ads but does not have control over choice of users or, in general, contexts. Section 3 discusses the contextual problem as a prediction problem, ignoring the controllable (decision theoretic) aspects of the problem. Section 4 discusses a simplified version of the contextual task as a standard n-armed (multi-armed) bandit problem. Section 5 extends the n-armed bandit viewpoint and explains connections with typical recommender problems. The number of users of the system can be in millions and the collaborative or the "community" aspects of the problem can help significantly in addressing sparsity and making better decisions. Section 6 in turn combines the problems of Sections 3 and 5 and describes perhaps the most general problem in which information about contexts and ads are represented as points in large dimensional spaces, but the decision theoretic, multi-armed bandit, and community aspects may all be taken into account for better performance. The contextual task is a challenging problem in which increasing utility is a direct function of improving algorithms.

2. THE AVAILABLE INFORMATION

In this paper, for simplicity, we assume that the objective is to maximize ad click rates. Two major types of informa-

UBDM '05 , August 21, 2005, Chicago, Illinois, USA.

tion that we may have to make optimal decisions are: (1) the *context attributes*, such as information about the user and site, and (2) the information about the arms, *e.g.*, information about the ads, which we refer to as *arm attributes*, such as ad topics and bid prices. The distinction is that we (the decision maker) do not have control over the choice of context attributes but we can choose from among the arms and therefore we have some control over the choice of arms and their attribute values.

Attributes (features) of the overall *context*, include page attributes, such as: page content, page topics, site, source, time and date. Context also includes attributes of the user, such as: user id, demographics, overall interests, and recent searches. Attributes of the arms (mostly ads) include: click price, overall click rate, ad contents, advertiser account id, ad topics, ad url and contents of the page pointed to. We may also have some control over the presentation of the page with the ads, placement of the ads, highlighting, and so on. Therefore, the list of potentially predictive features (both for contexts as well as arms) is long. These features may have different feature types (numeric, boolean, probabilistic, ..). Furthermore, not all the attribute values may be available at all times (missing values). The ideal system handles all these possibilities effectively.

Depending on the exact problem formulation, we may need other information. For example in the Bayesian formulation of the problem, we need priors over ad click rates (given context and arm attributes), and the horizon we want to optimize over [2]. Finally, while we don't have control over context attributes, we may know something about the distribution of what we will encounter.

3. THE PREDICTION PROBLEM

Assume we had access to a function that would predict the click rate well given the context and arm information. Such a function would be able to handle a large number of features, potentially with many missing values. Given a sufficiently large matrix of feature values and click outcomes, we could learn such a function via various algorithms (linear methods, decision trees, knn, ..). This problem is basically a typical machine learning problem (prediction, regression, etc). However, an issue is how the training data is produced or how to obtain such data. The context of feature values should be representative of the distribution of contexts we get.[1] But another potentially bigger issue is the choice of arm(s) for each context. This is under our control, and the question is how to make such decisions. There are several alternatives to choosing arms, including:

1. Uniformly at random (pure exploration). This may be fine if we don't have too many arms or possible arm feature values and/or we have a significant amount of time to explore, without much concern for exploitation.

2. We have strong beliefs/priors that only a limited number of arm values are relevant for each context. Then problem reduces to a series of smaller explore-exploit problems, and we can apply the appropriate strategy (*e.g.*, random sampling of 1) in this case.

3. (dynamic) Experiment design: in this case, we want to be more selective in the choice of arms, depending on the contexts, in order to increase our accumulated rewards while we are learning more about the world (exploring). A number of methods, ranging from standard multi-armed bandit to optimization algorithms such as genetic algorithms could be explored. See section 4.

Note that we may require not just the click rate (an expectation), but ideally a distribution to be output from such a function, in order to capture the level of uncertainty of the function over the predicted expectation. In practice, we expect that the training matrix will always be relatively limited compared to the space of possible feature values, and therefore significant uncertainties would remain.

Given that we have such a predictor function, we could use it to obtain the best action (choice of arm) given a context as well as where to explore. We may also use it to identify the patterns (combinations) of context and arm features that lead to relatively high click rates.

This pure learning approach, ignoring the controllable aspects of the problem, is very similar to the work of Joachims who explored learning better ranking functions using click data in the context of web search [4].

4. THE MULTI›ARMED BANDIT PROBLEM

As it is clear from previous discussion the overall problem involves exploration and exploitation at some level. Exploration means: to explore different arms (*e.g.*, ad types) to better estimate click rates and more effectively find winner arms, and exploitation means: to choose to pull those arms that are currently known to yield good rates, and exploit their good rate of returns. One could treat each context individually, without taking into account information about other contexts. With this simplification the problem becomes a standard multi-armed bandit problem [2]. There is significant literature on the problem: there are a number of ways of formulating the problem, and a number of algorithms and heuristics exist, with a substantial understanding of many theoretical and empirical properties of these technqiues [6, 1, 9, 3, 2].

Consider learning for a single user, *i.e.*, the user (user id) is our context. Also, assume the arms are ad topics. When a user visits a page, the systems task is to pick a certain ad topic, and from that ad topic pick a certain ad to show.[2] The objective is to maximize click rate over some period of time. We could then initialize the arm priors (say in a Bayesian formulation of the n-armed bandit problem), and figure out which arms work best for that person. A major problem we face with this approach is the problem of *sparsity*: there may be many ad topics available (thousands and beyond), whereas the number of interactions we may get from a single user, in a time period of interest, may be very small in comparison. The average baseline click rate (when showing an arbitrary ad) is very low (*e.g.*, below one percent). In standard n-armed bandit problems, the scale, *i.e.*, the number of arms is much more limited and/or the number of interactions or desired horizon needs to be significantly larger for descent optimization opportunity [3].

[1]However exceptions exist, for example when we are trying to focus on some subset of the feature space or when active learning.

[2]In general, we assume most individual ads appear and disappear too quickly to obtain sufficient statistics.

[3]A subtle difference with the typical n-armed bandit prob-

When there is no information differentiating the arms, there is no way around the sparsity problem. However, often we have much information that has the potential to better focus the experimentation and lead to better returns (click rates). Such information may be obtained from similar users and/or demographics information on that user. Additionally, the arms are not independent. Some ads are closer to one another than others along some dimensions (eg college football ads are closer to college basketball ads than to political ads with respect to the topic aspect). Therefore the arms are not independent and the information obtained for one arm can affect what we know about the other arms. In a Bayesian setting, such information influences the priors over the click rates. Still, as the population of users interact with the system we obtain more information about user behavior and potential user and arm similarities that can further help the choice of displayed arms. In the next section, we further develop the idea of using similar contexts and arms to affect decisions more dynamically.

5. THE RECOMMENDER PROBLEM

Consider a matrix where the rows are users (user ids) and the columns are ad topics. Whenever a user visits a page an ad topic is picked (via some mechanism) and shown to the user, and the outcome is recorded (whether or not there was a click). We expect that with a sufficiently large population of users and collection of ad topics, many users would behave similarly, and cluster into a smaller number of groups. Similar behavior, in our case, means similar patterns of likelihoods of clicking on certain topics. Such clustering also imposes clusters on the ads (columns). Thus click through rates that are known for some users can be used to infer similar click through rates for other similar users. The similarity among users may be defined in terms of click through rates themselves and/or inferred/predicted at least partly based on other user attributes such as their demographics and recent and past behavior. In the same vein, similarity among the items (columns) may be a function of the meta attributes of topics (*e.g.*, similarity in terms of subject) in addition to the click through rates the topics obtain from the users' activity.

Therefore, when we want to select ads to display to a single user, treating the problem as merely a single narmed bandit task misses much opportunity for faster optimization. This problem is very similar to recommendation problems [8], and involves many similar issues: users (in general, explicit contexts) with similar tastes, missing values, and our choice of the columns/items to show when a user (re)visits. Some difference from typical recommender problems are:

1. In recommendation problems, one does not recommend the same item again after it has been viewed or bought, but here, items/arms (*e.g.*, , a certain ad topic) can be repeatedly shown so better estimates of click rates or expected revenue for it is obtained.[4] The missing quantity is the rate (more generally a distribution) rather than grade of likability.
2. In many recommender problems, users have some control of what they see and rate, here the control is completely under the systems from the outset.[5]
3. Often in a recommendation problem there is no explicit notion of reward. Here, time lost is revenue lost, and exploration/exploitation and reward maximization take a markedly more relevant role.

Several recommender solutions methods, in particular collaborative filtering approaches, as well as techniques such as dimensionality reduction and clustering, nearest neighbors and other machine learning methods, apply to the contextual problem as well. Perhaps the most important distinguishing factor is the very direct connection of superior algorithms, *e.g.*, better clusterings or similarity metrics, to higher returns. In case of clustering, this could be contrasted with standard applications of clusterings in which the true objective may be ill defined or subjective. In contextual problems, the ultimate objective is the expected returns, and the performance of any context and arm clustering would be measured against that ideal of how effectively it increases expected returns. See for example the work of Kleinberg et. al. [5], which studies algorithms for clustering for similar economic end goals. The challenge here is how to do exploration and exploitation with the understanding that information obtained about a single user can help the whole community of users, and information about the community can help better serve a single user. Similar questions apply to the arms.

6. THE GENERAL PROBLEM

When the set of possible contexts and arms are enumerable, techniques developed for the problems of previous section are directly applicable, and sampling methods may address scalability. This "manageable" scenario occurs when, for example, contexts are restricted to user ids and arms are restricted to ad topics (see Section 5). This assumption may turn out to be too restrictive in practice. To allow for the full power of prediction, each context or arm can have a number of attributes "active" (*e.g.*, a page may belong to multiple topics), where the space of possible attributes can range in 100s and beyond. Thus, both a context and an arm may be viewed as points in their own large dimensional spaces.

In such scenarios, the set of possible contexts and/or arms is not enumerable and cannot be represented explicitly. And yet, all the aspects of the problem, the prediction problem, the exploration vs exploitation problem, and the recommender problem remain. Ignoring any aspect may lead to significant loss of opportunity for optimization. Obviously, techniques in learning and optimization in large dimensional spaces, such as dimensionality reduction and learning similarity metrics, is relevant. However, we are not aware of prior research that addresses exploration and exploitation in such large dimensional spaces, taking community (collaborative filtering effects) into account.

lem is the perspective of optimization: often the problem is cast from the user's point of view. Thus the user is presumably motivated to interact and experiment actively and intelligently to recognize and exploit the better arms. In the case of contextual problems, the system (the company) is primarily doing the optimization, though the users should benefit as well.

[4]However, in our problem there may also be a notion of decay in interest. This in part depends on how we define the context.

[5]Still it is possible to imagine scenarios where we allow the users to pick the type of ads they want to see.

7. SUMMARY

We described the contextual recommender problem, motivated by contextual advertising, and identified several related subproblems:

- A prediction problem involving a large number of features, possibly with missing values. The objective is to obtain a predictor that outputs a distribution preferably instead of a single numeric quantity.
- A generalization of the multi-armed bandit problem to a set of contexts, in order to address sparsity issues. This problem may also be considered a special recommender problem and, in particular, collaborative filtering techniques are relevant.
- Further extension to the case where the space of context or arms is not enumerable: A context or an arm is modeled as a point in a large dimensional feature space.

There may remain other challenges, for example nonstationarity: A user's needs and interests change over time. Contextual recommender problems are general. For instance, instead of ads, other items can be offered, for example navigational links [7]. Personalized web search may also be viewed as a special case in which knowledge of the query significantly reduces ambiguity and the need for extensive exploration [4]. Studying these problems in theory as well as developing experience and an understanding of effective practical algorithms should be of great value.

8. REFERENCES

[1] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The non-stochastic multi-armed bandit problem. *SIAM Journal on Computing*, 2002.

[2] D. A. Berry and B. Fristedt. *Bandit Problems*. Chapman and Hall, 1985.

[3] J. Gittins. *Multi-Armed Bandit Allocation Indices*. John Wiley and Sons, 1989.

[4] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*. ACM, 2002.

[5] J. Kleingberg, C. Papadimitriou, and P. Raghavan. Segmentation problems. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 1998.

[6] O. Madani, D. J. Lizotte, and R. Greiner. The budgeted multi-armed bandit problem. In *COLT*, 2004.

[7] M. Perkowitz and O. Etzioni. Towards adaptable web sites: Conceptual framework and case study. *Artificial Intelligence*, 2001.

[8] P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3), 1997.

[9] R. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 1998.

A Fast High Utility Itemsets Mining Algorithm

Ying Liu Wei-keng Liao Alok Choudhary

Electrical and Computer Engineering Department, Northwestern University
Evanston, IL, USA 60208

{yingliu, wkliao, choudhar}@ece.northwestern.edu

ABSTRACT

Association rule mining (ARM) identifies frequent itemsets from databases and generates association rules by considering each item in equal value. However, items are actually different in many aspects in a number of real applications, such as retail marketing, network log, etc. The difference between items makes a strong impact on the decision making in these applications. Therefore, traditional ARM cannot meet the demands arising from these applications. By considering the different values of individual items as utilities, utility mining focuses on identifying the itemsets with high utilities. As "downward closure property" doesn't apply to utility mining, the generation of candidate itemsets is the most costly in terms of time and memory space. In this paper, we present a Two-Phase algorithm to efficiently prune down the number of candidates and can precisely obtain the complete set of high utility itemsets. In the first phase, we propose a model that applies the "transaction-weighted downward closure property" on the search space to expedite the identification of candidates. In the second phase, one extra database scan is performed to identify the high utility itemsets. We also parallelize our algorithm on shared memory multi-process architecture using Common Count Partitioned Database (CCPD) strategy. We verify our algorithm by applying it to both synthetic and real databases. It performs very efficiently in terms of speed and memory cost, and shows good scalability on multiple processors, even on large databases that are difficult for existing algorithms to handle.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications – *data mining.*

General Terms

Algorithms, Design

Keywords

utility mining, association rules mining, downward closure property, transaction-weighted utilization

UBDM '05 , August 21, 2005, Chicago, Illinois, USA.

1. INTRODUCTION

Association rules mining (ARM) [1] is one of the most widely used techniques in data mining and knowledge discovery and has tremendous applications in business, science and other domains. For example, in the business, its applications include retail shelf management, inventory predictions, supply chain management, bundling products marketing. The main objective of ARM is to identify frequently occurring patterns of itemsets. It first finds all the itemsets whose co-occurrence frequency are beyond a minimum support threshold, and then generates rules from the frequent itemsets based on a minimum confidence threshold. Traditional ARM model treat all the items in the database equally by only considering if an item is present in a transaction or not.

The frequent itemsets identified by ARM does not reflect the impact of any other factor except frequency of the presence or absence of an item. Frequent itemsets may only contribute a small portion of the overall profit, whereas non-frequent itemsets may contribute a large portion of the profit. In reality, a retail business may be interested in identifying its most valuable customers (customers who contribute a major fraction of the profits to the company). These are the customers, who may buy full priced items, high margin items, or gourmet items, which may be absent from a large number of transactions because most customers do not buy these items. In a traditional frequency oriented ARM, these transactions representing highly profitable customers may be left out. For instance, {*milk, bread*} may be a frequent itemset with *support* 40%, contributing 4% of the total profit, and the corresponding consumers is Group A, whereas {*birthday cake, birthday card*} may be a non-frequent itemset with *support* 8% (assume support threshold is 10%), contributing 8% of the total profit, and the corresponding consumers is Group B. The marketing professionals must be more interested in promoting the sale of {*birthday cake, birthday card*} by designing promotion campaigns or coupons tailored for Group B (valuable customers), although this itemset is missed by ARM. Another example is web log data. A sequence of webpages visited by a user can be defined as a transaction. Since the number of visits to a webpage and the time spent on a particular webpage is different between different users, the total time spent on a page by a user can be viewed as utility. The website designers can catch the interests or behavior patterns of the customers by looking at the utilities of the page combinations and then consider re-organizing the link structure of their website to cater to the preference of users. Frequency is not sufficient to answer questions, such as whether an itemset is highly profitable, or whether an itemset has a strong impact. Utility mining is likely to be useful in a wide range of practical applications.

Recently, to address the limitation of AMR, a *utility mining* model was defined [2]. Intuitively, utility is a measure of how "useful" (i. e. "profitable") an itemset is. The *utility* of an item or itemset is based on *local transaction utility* and *external utility*. The local transaction utility of an item is defined according to the information stored in a transaction, like the quantity of the item sold in the transaction. The external utility of an item is based on information from resources besides transactions, like a profit table. The external utility can be a measure for describing user preferences. The definition of utility of an itemset X, $u(X)$, is the sum of the utilities of X in all the transactions containing X. The goal of utility mining is to identify high utility itemsets which drive a large portion of the total utility. Traditional ARM model assumes that the utility of each item is always 1 and the sales quantity is either 0 or 1, thus it is only a special case of utility mining, where the utility or the sales quantity of each item could be any number. If $u(X)$ is greater than a utility threshold, X is a high utility itemset, otherwise, it is a low utility itemset. Table 1 is an example of a transaction database where the total utility is 400. The number in each transaction in Table 1(a) is the sales volume of each item, and the external utility of each item is listed in Table 1(b). $u(\{B, D\}) = (6\times10+1\times6) + (10\times10+1\times6) = 172$. {B, D} is a high utility itemset if the utility threshold is set at 120.

Table 1. A transaction database and its utility table

(a) Transaction table. Each row is a transaction. The columns represent the number of items in a particular transaction. TID is the transaction identification number

TID \ ITEM	A	B	C	D	E
T_1	0	0	18	0	1
T_2	0	6	0	1	1
T_3	2	0	1	0	1
T_4	1	0	0	1	1
T_5	0	0	4	0	2
T_6	1	1	0	0	0
T_7	0	10	0	1	1
T_8	3	0	25	3	1
T_9	1	1	0	0	0
T_{10}	0	6	2	0	2

(b) The utility table. The right column displays the profit of each item per unit in dollars

ITEM	PROFIT ($)(per unit)
A	3
B	10
C	1
D	6
E	5

To the best of our knowledge, there is no efficient strategy to find all the high utility itemsets. A naïve attempt may be to eliminate the items that contribute a small portion of the total utility. However, a high utility itemset may consist of some low utility items. Another attempt is to adopt the level-wise searching schema that exists in fast AMR algorithms, such as Apriori [1]. The base of these traditional ARM algorithms is the "downward closure property" (anti-monotone property): any subset of a frequent itemset must also be frequent. That is, only the frequent k-itemsets are exploited to generate potential frequent (k+1)-itemsets. This approach is efficient since a great number of item combinations are pruned at each level. However, this property doesn't apply to the *utility mining* model. For example, $u(D) = 36 < 120$, D is a low utility item, but its superset {B, D} is a high utility itemset. Without this property, the number of candidates generated at each level quickly approaches all the combinations of all the items. For 10^5 items, more than 10^9 2-itemsets candidates may be generated. Moreover, to discover a long pattern, the number of candidates is exorbitantly large. The cost of either computation time or memory is intolerable, regardless of what implementation is applied. *The challenge of utility mining is in restricting the size of the candidate set and simplifying the computation for calculating the utility.*

Nowadays, in any real application, the size of the data set easily goes to hundreds of Mbytes or Gbytes. In order to tackle this challenge, we propose a Two-Phase algorithm to efficiently mine high utility itemsets. In Phase I, we define *transaction-weighted utilization* and propose a model ─ *transaction-weighted utilization mining*. *Transaction-weighted utilization of an itemset* X is estimated by the sum of the transaction utilities of all the transactions containing X. This model maintains a *Transaction-weighted Downward Closure Property*: any subset of a high transaction-weighted utilization itemset must also be high in transaction-weighted utilization. (Please note we use a new term *transaction-weighted utilization* to distinguish it from *utility*. The focus of this paper is not proposing this new term, but to utilize the property of *transaction-weighted utilization* to help solve the difficulties in utility mining.) Thus, only the combinations of high transaction-weighted utilization itemsets are added into the candidate set at each level. Therefore, the size of the candidate set is substantially reduced during the level-wise search. The memory cost as well as the computation cost is also efficiently reduced. Phase I may overestimate some low utility itemsets as high transaction-weighted utilization itemsets since we use the *transaction-weighted utilization mining* model, but it never underestimates any itemsets. In phase II, only one extra database scan is performed to filter out the overestimated itemsets. The savings provided by Phase I may compensate for the cost incurred by the extra scan during Phase II. As shared memory parallel machines are becoming the dominant type of supercomputers in industry, we parallelize our algorithm on shared memory multi-process architecture using Common Count Partitioned Database (CCPD) scheme. We verify our algorithm by applying it to both synthetic and real databases. It not only performs very efficiently in terms of speed and memory cost compared to the best existing utility mining algorithm [2] (to our best knowledge), but also shows good scalability on multiple processors. Our algorithm easily handles very large databases that existing algorithms cannot handle.

The rest of this paper is organized as follows. Section 2 overviews the related work. Section 3 formally describes the utility mining model. In Section 4, we propose the Two-Phase algorithm. Section 5 presents our parallelization scheme. The experimental

results are presented in section 6 and we summarize our work in section 7.

2. RELATED WORK

In the past ten years, a number of traditional ARM algorithms and optimizations have been proposed. The common assumption of them is that each item in a database is equal in weight and the sales quantity is 0 or 1. All of these algorithms exploit the "downward closure property" as proposed in Apriori [1] (all subsets of a frequent itemset must be frequent), such as DHP [3], DIC [4], ECLAT [5], FP-growth [6].

Quantitative association rules mining is introduced in [7], which associates an antecedent with an impact on a target numeric variable. A behavior of a subset is interesting if the statistical distribution of the targeted quantitative variable stands out from the rest. A data-driven algorithm, called "Window", is developed. OPUS [8] is an efficient algorithm to discover quantitative associations rules in dense data sets. The focus of these two algorithms (identifying rules where the antecedent strongly impacts the targeting numeric attribute) is different from ours (identifying those valuable item combinations and the implied valuable customers). The discovery from our work can guide the layout of goods in stores, or promotion campaigns to valuable customers.

Researches that assign different weights to items have been proposed. MINWAL [9] mines the weighted association rules in binary transaction databases based on the k-support bound property. An efficient association rules generation method, WAR [10], focuses on the generation of rules from the available frequent itemsets instead of searching for weighted frequent itemsets. WARM [11] proposes a weighted ARM model where itemset weight is defined as the average weight value of the items comprising this itemset. [12] proposes a scheme that uniformly weights all the transactions without considering the differences among the items. These weighted ARM models are special cases of utility mining.

One of the problems in the field of knowledge discovery is of studying good measures of interestingness of discovered patterns. Some interestingness measures have been proposed [19]. Actionability and unexpectedness are two important subjective measures. According to these measures, a pattern is interesting if the user can take some action by knowing this pattern or it is surprising to the user. Concepts are defined in probabilistic terms.

A concept, *itemset share*, is proposed in [13]. It can be regarded as a utility because it reflects the impact of the sales quantities of items on the cost or profit of an itemset. Itemset share is defined as a fraction of some numerical value, such as total quantity of items sold or total profit. Several heuristics have been proposed and their effectiveness is well evaluated.

A utility mining algorithm is proposed in [14], where the concept of "useful" is defined as an itemset that supports a specific objective that people want to achieve. It focuses on mining the top-K high utility closed patterns that directly support a given business objective. Since the "downward closure property" no longer holds, it develops a new pruning strategy based on a weaker but anti-monotonic condition. Although this work is contributed to utility mining, the definition of utility and the goal of this algorithm in his work are different from those in our work.

An alternative formal definition of utility mining and theoretical model was proposed in [2], where the utility is defined as the combination of utility information in each transaction and additional resources. Since this model cannot rely on "downward closure property" to restrict the number of itemsets to be examined, a heuristics is used to predict whether an itemset should be added to the candidate set. However, the prediction usually overestimates, especially at the beginning stages, where the number of candidates approaches the number of all the combinations of items. The examination of all the combinations is impractical, either in computation cost or in memory space cost, whenever the number of items is large or the utility threshold is low. Although this algorithm is not efficient or scalable, it is by far the best to solve this specific problem. Our work is based on this definition and achieves a significant breakthrough of this problem in terms of computational cost, memory cost and accuracy.

3. UTILITY MINING

The goal of utility mining is to discover all the itemsets whose utility values are beyond a user specified threshold in a transaction database. We start with the definition of a set of terms that leads to the formal definition of utility mining problem. The same terms are given in [2].

- $I = \{i_1, i_2, \ldots, i_m\}$ is a set of items.
- $D = \{T_1, T_2, \ldots, T_n\}$ be a transaction database where each transaction $T_i \in D$ is a subset of I.
- $o(i_p, T_q)$, *local transaction utility value*, represents the quantity of item i_p in transaction T_q. For example, $o(\text{A}, T_8) = 3$, in Table 1(a).
- $s(i_p)$, *external utility,* is the value associated with item i_p in the Utility Table. This value reflects the importance of an item, which is independent of transactions. For example, in Table 1(b), the external utility of item A, $s(\text{A})$, is 3.
- $u(i_p, T_q)$, *utility*, the quantitative measure of utility for item i_p in transaction T_q, is defined as $o(ip,Tq) \times s(ip)$. For example, $u(A, T_8) = 3 \times 3$, in Table 1.
- $u(X, T_q)$, *utility of an itemset X in transaction* T_q, is defined as $\sum_{ip \in X} u(ip,Tq)$, where $X = \{i_1, i_2, \ldots, i_k\}$ is a k-itemset, $X \subseteq T_q$ and $1 \leq k \leq m$.
- *u(X), utility of an itemset X*, is defined as $\sum_{T_q \in D \wedge X \subseteq T_q} u(X,Tq)$.

(3.1)

Utility mining is to find all the high utility itemsets. An itemset X is a *high utility itemset* if $u(X) \geq \varepsilon$, where $X \subseteq I$ and ε is the minimum utility threshold, otherwise, it is a *low utility itemset*. For example, in Table 1, $u(\text{A}, T_8) = 3\times3 = 9$, $u(\{\text{A, D, E}\}, T_8) = u(\text{A}, T_8) + u(\text{D}, T_8) + u(\text{E}, T_8) = 3\times3 + 3\times6 + 1\times5 = 32$, and $u(\{\text{A, D, E}\}) = u(\{\text{A, D, E}\}, T_4) + u(\{\text{A, D, E}\}, T_8) = 14 + 32 = 46$. If $\varepsilon = 120$, {A, D, E} is a low utility itemset.

3.1 Computational Model and Complexity

We now examine the computational model proposed in [2]. We refer this algorithm as MEU (Mining using Expected Utility) for

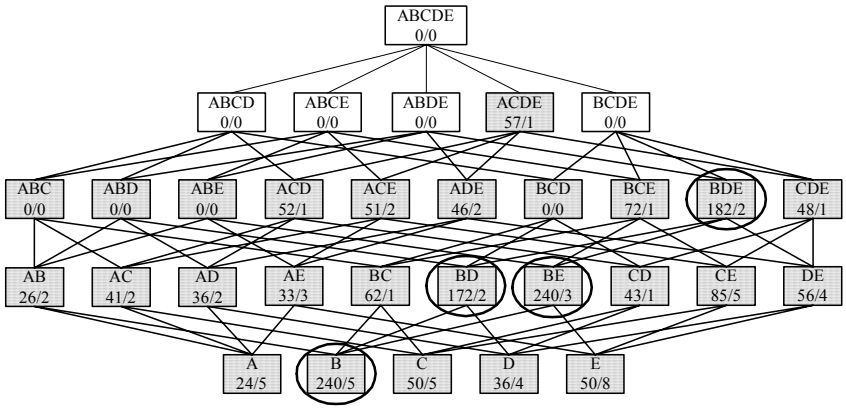


Figure 1. Itemsets lattice related to the example in Table 1. ε = 120. Itemsets in circles are the high utility itemsets. Numbers in each box are utility / number of occurrences. Gray-shaded boxes denote the search space.

the rest of this paper. MEU prunes the search space by predicting the high utility *k*-itemset, I^k, with the *expected utility value*, denoted as $u'(I^k)$. $u'(I^k)$ is calculated from the utility values of all its (*k*-1) subsets. If $u'(I^k)$ is greater than ε, I^k is added to the candidate set for *k*-itemsets, otherwise, I^k is pruned. $u'(I^k)$ is calculated as

$$u'(I^k) = \frac{\sup_{\min}(I^k)}{k-1} \sum_{i=1}^{m} \frac{u(I_i^{k-1})}{\sup(I_i^{k-1})} + \frac{k-m}{k-1} \times \varepsilon \qquad (3.2)$$

I_i^{k-1} is a (*k*-1)-itemset such that $I_i^{k-1} = I^k - \{i\}$, i.e. I_i^{k-1} includes all the items except item *i*. sup(*I*) is the support of itemset *I*, which is the percentage of all the transactions that contain itemset *I*. The minimum support among all the (*k*-1) subsets of I^k is given as

$$\sup_{\min}(I^k) = \min_{\forall I_i^{k-1} \subset I^k, (1 \le i \le m)} \{\sup(I_i^{k-1})\} \qquad (3.3)$$

For each I^k, there are *k* (*k*-1)-subsets. In (3.2) and (3.3), *m* is the number of high utility itemsets among the (*k*-1) subsets where I_i^{k-1} ($1 \le i \le m$) are high utility itemsets, and I_i^{k-1} ($m+1 \le i \le k$) are low utility itemsets. This prediction is based on the *support boundary property* [2] which states that the support of an itemset always decreases as its size increases. Thus, if the support of an itemset is zero, its superset will not appear in the database at all. This approach uses the high utility itemsets at level (*k*-1) to calculate the expected utility value for level *k* and the utility threshold ε to substitute the low utility itemsets.

Let us use Table 1 as an example. Figure 1 shows the search space of the database, given ε = 120. Since $u(\{D, E\}) = 56 < \varepsilon$, $\sup_{\min}(\{B, D, E\}) = \min\{\sup(\{B, D\}), \sup(\{B, E\})\} = \min\{0.2, 0.3\} = 0.2$. The expected utility value of {B, D, E} is:

$$\begin{aligned} u'(\{B,C,D\}) &= \frac{0.2}{3-1} \times \left(\frac{u(\{B,D\})}{\sup(\{B,D\})} + \frac{u(\{B,E\})}{\sup(\{B,E\})}\right) + \frac{3-2}{3-1} \times \varepsilon \\ &= \frac{0.2}{2} \times \left(\frac{172}{0.2} + \frac{240}{0.3}\right) + \frac{1}{2} \times 120 \\ &= 226 > \varepsilon \end{aligned}$$

Hence, {B, C, D} is a candidate for 3-itemset. Observed from Figure 1, four out of 31 potential candidates are high utility itemsets, which are marked in circles. Using MEU, 26 itemsets (in gray-shaded boxes) have been added into the candidate sets by prediction.

This model somehow reduces the number of candidates; however, it has drawbacks in the following aspects:

1) **Pruning the candidates** – When *m* is small, the term $\frac{k-m}{k-1}$ is close to 1 or even greater than 1. In this situation, $u'(I^k)$ is most likely greater than ε. When $k = 2$, $u'(I^2)$ is always greater than ε, no matter *m* is 0, 1, or 2. Similarly, when $k = 3$, $u'(I^3) \ge \varepsilon$ if *m* is 0, 1, or 2. Therefore, this estimation does not prune the candidates effectively at the beginning stages. As shown in Figure 1, all the 2-itemsets and 3-itemsets are included in the candidate sets. If there are 10^5 different items in the database, the number of 2-itemsets is approximately 5×10^9. Such requirements can easily overwhelm the available memory space and computation power of most of the machines.

2) **Accuracy –** This model may miss some high utility itemsets when the variation of the itemset supports is large. For example, if ε = 40 in our example, the expected utility of itemset {C, D, E} is

$$\begin{aligned} u'(\{C,D,E\}) &= \frac{\sup_{\min}(\{C,D,E\})}{3-1} \times \left(\frac{u(\{C,D\})}{\sup(\{B,C\})} + \frac{u(\{C,E\})}{\sup(\{C,E\})} + \frac{u(\{D,E\})}{\sup(\{D,E\})}\right) \\ &= \frac{\min\{0.1,0.5,0.4\}}{2} \times \left(\frac{43}{0.1} + \frac{85}{0.5} + \frac{56}{0.4}\right) \\ &= 37 < \varepsilon \end{aligned}$$

Therefore, {C, D, E} is predicted to be a low utility itemset and then pruned from the candidate set for 3-itemsets. However, {C, D, E} is indeed a high utility itemset because $u(\{C, D, E\}) = 48 > \varepsilon$.

4. TWO-PHASE ALGORITHM

To address the drawbacks in MEU, we propose a novel Two-Phase algorithm that can highly effectively prune candidate itemsets and simplify the calculation of utility. It substantially reduces the search space and the memory cost and requires less computation. In Phase I, we define a *transaction-weighted*

utilization mining model that holds a "*Transaction-weighted Downward Closure Property*". (The purpose of introducing this new concept is not to define a new problem, but to utilize its property to prune the search space.) *High transaction-weighted utilization itemsets* are identified in this phase. The size of candidate set is reduced by only considering the supersets of high transaction-weighted utilization itemsets. In Phase II, one database scan is performed to filter out the high transaction-weighted utilization itemsets that are indeed low utility itemsets. This algorithm guarantees that the complete set of high utility itemsets will be identified.

4.1 Phase I

Definition 1. (Transaction Utility) The *transaction utility of transaction* T_q, denoted as $tu(T_q)$, is the sum of the utilities of all items in T_q: $tu(T_q) = \sum_{i_p \in T_q} u(i_p, T_q)$, where $u(i_p, T_q)$ is the same as in Section 3. Table 2 gives the transaction utility for each transaction in Table 1.

Table 2. Transaction utility of the transaction database

TID	Transaction Utility	TID	Transaction Utility
T_1	23	T_6	13
T_2	71	T_7	111
T_3	12	T_8	57
T_4	14	T_9	13
T_5	14	T_{10}	72

Definition 2. (Transaction-weighted Utilization) The *transaction-weighted utilization of an itemset X*, denoted as $twu(X)$, is the sum of the transaction utilities of all the transactions containing X:

$$twu(X) = \sum_{X \subseteq T_q \in D} tu(T_q) \qquad (4.1)$$

For the example in Table 1, $twu(\mathrm{A}) = tu(T_3) + tu(T_4) + tu(T_6) + tu(T_8) + tu(T_9) = 12 + 14 + 13 + 57 + 13 = 109$ and $twu(\{\mathrm{A, D}\}) = tu(T_4) + tu(T_8) = 14 + 57 = 71$.

Definition 3. (High Transaction-weighted Utilization Itemset) For a given itemset X, X is a *high transaction-weighted utilization itemset* if $twu(X) \geq \varepsilon$', where ε' is the user specified threshold.

Theorem 1. (Transaction-weighted Downward Closure Property) Let I^k be a k-itemset and I^{k-1} be a (k-1)-itemset such that $I^{k-1} \subset I^k$. If I^k is a high transaction-weighted utilization itemset, I^{k-1} is a high transaction-weighted utilization itemset.

Proof: Let T_{I^k} be the collection of the transactions containing I^k and $T_{I^{k-1}}$ be the collection of transactions containing I^{k-1}. Since $I^{k-1} \subset I^k$, $T_{I^{k-1}}$ is a superset of T_{I^k}. According to Definition 2,

$$twu(I^{k-1}) = \sum_{I^{k-1} \subseteq T_q \in D} tu(T_q) \geq \sum_{I^k \subseteq T_p \in D} tu(T_p) = twu(I^k) \geq \varepsilon' \qquad \square$$

The *Transaction-weighted Downward Closure Property* indicates that any superset of a low transaction-weighted utilization itemset is low in transaction-weighted utilization. That is, only the combinations of high transaction-weighted utilization (k-1)-itemsets could be added into the candidate set C_k at each level.

Theorem 2. Let *HTWU* be the collection of all high transaction-weighted utilization itemsets in a transaction database D, and *HU* be the collection of high utility itemsets in D. If ε'= ε, then $HU \subseteq HTWU$.

Proof: $\forall X \in HU$, if X is a high utility itemset, then

$$\varepsilon' = \varepsilon \leq u(X) = \sum_{X \subseteq T_q} u(X, T_q) = \sum_{X \subseteq T_q} \sum_{i_p \in X} u(i_p, T_q)$$
$$\leq \sum_{X \subseteq T_q} \sum_{i_p \in T_q} u(i_p, T_q) = \sum_{X \subseteq T_q} tu(T_q) = twu(X)$$

Thus, X is a high transaction-weighted utilization itemset and $X \in HTWU$. $\square$

According to Theorem 2, we can utilize the *Transaction-weighted Downward Closure Property* in our *transaction-weighted utilization mining* in Phase I by assuming ε' = ε and prune those overestimated itemsets in Phase II.

Figure 2 shows the search space of Phase I. Twelve out of 31 itemsets (in gray-shaded boxes) are generated as candidates (including the single items), and 9 out of 31 itemsets (in circles) are the high transaction-weighted utilization itemsets. The level-wise search stops at the third level, one level less than MEU in Figure 1. (For larger databases, the savings should be more evident.) By holding the *Transaction-weighted Downward Closure Property*, the search space in our algorithm is small. *Transaction-weighted utilization mining* model outperforms MEU in several aspects:

1) **Less candidates** ─ When ε' is large, the search space can be significantly reduced at the second level and higher levels. As shown in Figure 2, four out of 10 itemsets are pruned because they all contain item A (A is not a high transaction-weighted utilization item). However, in MEU, the prediction hardly prunes any itemset at the beginning stages. In Figure 1, all the 10 2-itemsets are added into the candidate set C_2 because their expected utility values are all greater than ε.

2) **Accuracy** ─ Based on Theorem 2, if we let ε'=ε, the complete set of high utility itemsets is a subset of the high transaction-weighted utilization itemsets discovered by our *transaction-weighted utilization mining* model. However, the prediction in MEU may miss some high utility itemsets when the variation of itemset supports is large.

3) **Arithmetic complexity** ─ One of the kernel operations in the Two-Phase algorithm is the calculation for each itemset's transaction-weighted utilization as in equation 4.1. Compared to the calculation for each itemset's expected utility value (equation 3.2) in MEU, equation 4.1 only incurs add operations rather than a number of multiplications. Thus, since the kernel calculation may occur a huge number of times, the overall computation is much less complex.

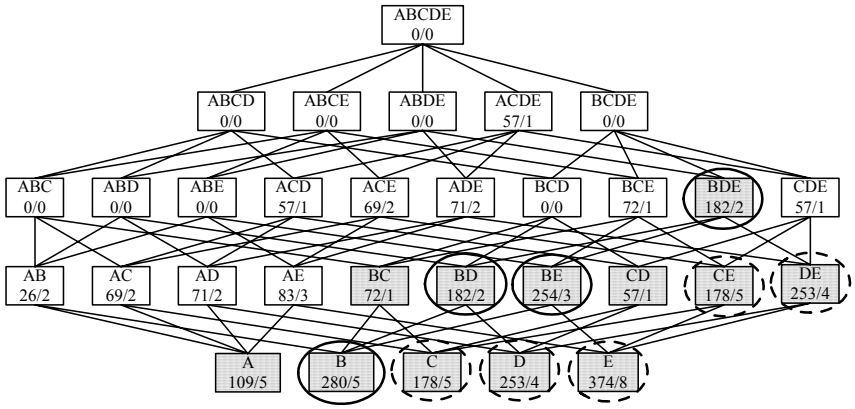


Figure 2. Itemsets lattice related to the example in Table 1. ε' = 120. Itemsets in circles (solid and dashed) are the high transaction-weighted utilization itemsets in *transaction-weighted utilization mining model*. Gray-shaded boxes denote the search space. Itemsets in solid circles are high utility itemsets found by MEU. Numbers in each box are transaction-weighted utilization / number of occurrence.

4.2 Phase II

In Phase II, one database scan is required to select the high utility itemsets from high transaction-weighted utilization itemsets identified in Phase I. The number of the high transaction-weighted utilization itemsets is small when ε' is high. Hence, the time saved in Phase I may compensate for the cost incurred by the extra scan during Phase II. The total computational cost of Phase II is the cost of equation (3.1) × the total number of high transaction-weighted utilization itemsets.

In Figure 2, the high utility itemsets ({B}, {B, D}, {B, E} and {B, D, E}) (in solid black circles) are covered by the high transaction-weighted utilization itemsets (in solid and dashed black circles). Nine itemsets in circles are maintained after Phase I, and one database scan is performed in Phase II to prune 5 of the 9 itemsets since they are not high utility itemsets.

5. PARALLEL IMPLEMENTATION

Since the size of the databases in commercial activities is usually in Gbytes, the computation time or the memory consumption may be intolerable on a single processor. Therefore, high performance parallel computing is highly desired. Due to the fact that shared memory parallel machines are becoming the dominant type of supercomputers in industry because of its simplicity, we design our utility mining parallel implementation on shared-memory architecture.

In shared-memory multi-process architecture (SMPs), each processor has its direct and equal access to all the system's memory as well as its own local caches. To achieve a good scalability on SMPs, each processor must maximize access to local cache and avoid or reduce false sharing. That is, we need to minimize the Ping-Pong effect, where multiple processors might be trying to modify different variables that coincidently reside on the same cache line.

The nature of our Two-Phase utility mining algorithm is a level-wised search method, which is the same as Apriori [1]. [16, 17] has proved that Common Count Partitioned Database (CCPD) is the best strategy to parallel it. In CCPD, data is evenly partitioned on each processor, and each processor traverses its local database partition for incrementing the transaction-weighted utilization of each itemset. All the processors share a single common hash tree, which stores the candidate itemsets at each level of search as well as their transaction-weight utilization. To build the hash tree in parallel, CCPD associated a lock with each leaf node. When a processor wants to insert a candidate into the tree, it starts at root, and successively hashes on the items until it reaches a leaf. It then acquires the lock and inserts the candidate. With this locking mechanism, each processor can insert itemsets in different parts of the hash tree in parallel. For transaction-weight utilization calculation, each processor computes the value from its local database partition.

6. EXPERIMENTAL EVALUATION AND PERFORMANCE STUDY

We evaluate the performance of our Two-Phase algorithm by varying the size of the search space. We also analyze the scalability and result accuracy. All the experiments were performed on a 700-MHz Xeon 8-way shared memory parallel machine with a 4 Gbytes memory, running the Red Hat Linux Advanced Server 2.1 operating system. The program is implemented in C. In parallel implementation, we use OpenMP pragmas [18]. OpenMP is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism. Due to its simplicity, OpenMP is quickly becoming one of the most widely used programming styles for SMPs. In SMPs, processors communicate through shared variables in the single memory space. Synchronization is used to coordinate processes. In order to give a fair comparison, we also implement MEU so that both of the implementations can run on the same machine. We use synthetic data and real world data for our evaluation purpose. The details of these databases are described in the following subsections.

6.1 Synthetic Data from IBM Quest Data Generator

We use two sets of synthetic databases from IBM Quest data generator [15]. One is a dense database, T10.I6.DX000K, where the average transaction size is 10; the other is a sparse database,

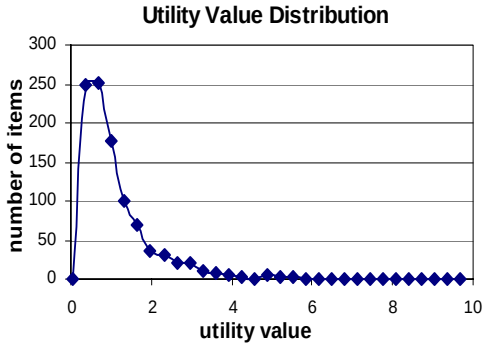


Figure 3. Utility value distribution in utility table.

T20.I6.DX000K, where the average transaction size is 20. The average size of the maximal potentially frequent itemsets is 6 in both sets of databases. In both sets of databases, we vary the number of transactions from 1000K to 8000K, and the number of items from 1K to 8K. However, the IBM Quest data generator only generates the quantity of 0 or 1 for each item in a transaction. In order to fit them into the scenario of utility mining, we randomly generate the quantity of each item in each transaction, ranging from 1 to 5. Utility tables are also synthetically created by assigning a utility value to each item randomly, ranging from 0.01 to 10.00. Observed from real world databases that most items are in the low profit range, we generate the utility values using a log normal distribution. Figure 3 shows the histogram of the utility values of 1000 items.

Table 3. The number of candidate itemsets generated by Phase I of Two-Phase algorithm vs. MEU in the first two database scans

Threshold \ Databases		T10.I6.D1000K		T20.I6.D1000K	
		Phase I	MEU	Phase I	MEU
0.5%	1st *scan*	226128	499500	315615	499500
	2nd *scan*	17	-	18653	-
0.75%	1st *scan*	153181	499500	253116	499500
	2nd *scan*	0	-	1531	-
1%	1st *scan*	98790	499500	203841	499500
	2nd *scan*	0	-	183	-
1.25%	1st *scan*	68265	499500	159330	499500
	2nd *scan*	0	-	33	-
1.5%	1st *scan*	44850	499500	135460	499500
	2nd *scan*	0	-	8	-
1.75%	1st *scan*	27730	499500	104653	499500
	2nd *scan*	0	-	4	-
2%	1st *scan*	16836	499500	84666	499500
	2nd *scan*	0	-	1	-

6.1.1 Number of Candidates

Table 3 presents the number of candidate itemsets generated by Phase I of our Two-Phase algorithm vs. MEU. We only provide the numbers in the first two database scans. The number of items is set at 1000, and the minimum utility threshold varies from 0.5% to 2%. The number of candidate itemsets generated by Phase I at the first database scan decreases dramatically as the threshold goes up. However, the number of candidates generated by MEU is always 499500, which is all the combinations of 1000 items. Phase I generates much fewer candidates compared to MEU. For example, 16836 by Phase I vs. 499500 by MEU at a utility threshold 2% in database T10.I6.D1000K. After the second database scan, the number of candidates generated by our algorithm decreases substantially, mostly decreasing more than 99%. We don't provide the exact numbers for MEU because it actually takes an inordinate amount of time (longer than 10 hours) to complete the second scan. In the case of T20.I6.D1000K, more candidates are generated, because each transaction is longer than that in T10.I6.D1000K. Observed from Table 3, the *Transaction-weighted Downward Closure Property* in transaction-weighted utilization mining model can help prune candidates very effectively.

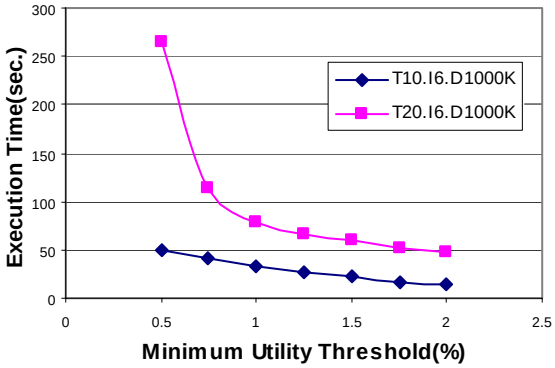


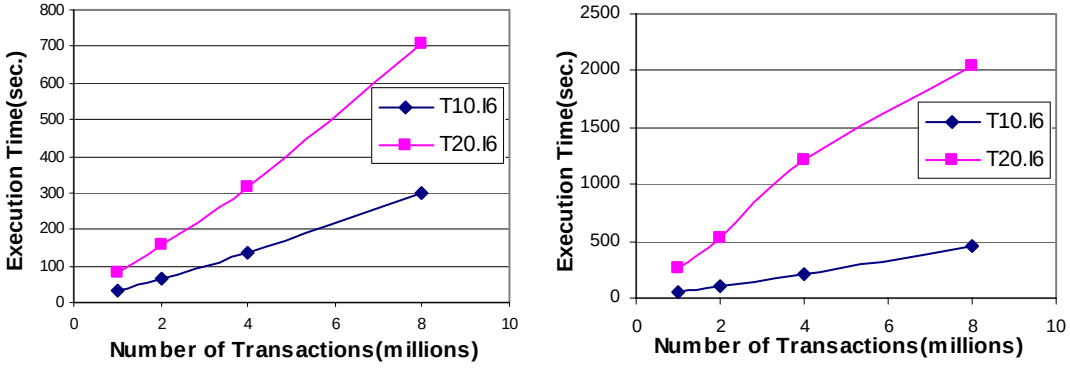
Figure 4. Execution time with varying minimum utility threshold.

6.1.2 Scalability

Figure 4 shows the execution time (including both Phase I and Phase II) of the Two-Phase algorithm using T20.I6.D1000K and T10.I6.D1000K. Since the number of candidate itemsets decreases as the minimum utility threshold increases, the execution time decreases, correspondingly. When threshold is 0.5%, the execution time of T20.I6.D1000K is somewhat longer, because it takes 3 more scans over the database in this case compared to other cases with higher threshold values.

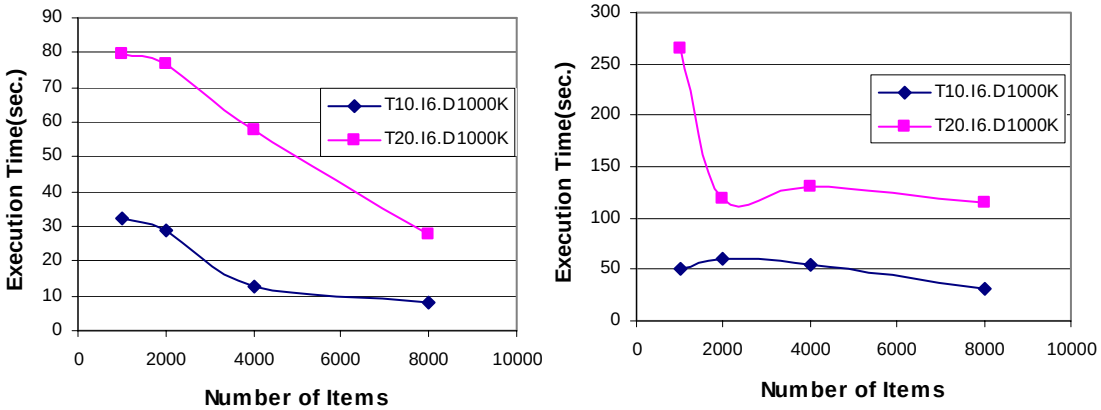
Figure 5 presents the scalability of the Two-Phase algorithm by increasing the number of transactions in the database. The number of transactions varies from 1000K to 8000K. The minimum utility threshold is set at 1% to 0.5% in Figure 5(a) and Figure 5(b), respectively. The execution times for either database increase approximately linearly as the data size increases. The execution times for T20.I6.DX000K are longer than that of T10.I6.DX000K, because more computation is required for longer transactions. In addition, the size of database T20.I6.DX000K is larger and therefore takes a longer time to scan.

Figure 6 presents the performance when varying the numbers of items. The number of items varies from 1K to 8K. The minimum utility threshold is set at 1% and 0.5% in Figure 6(a) and Figure 6(b), respectively. When the threshold is 1% as in Figure 6(a), the time decreases as the number of items increases. However, in Figure 6(b), the time for database T10.I6.D1000K with 2K items



(a) minimum utility threshold = 1% (b) minimum utility threshold = 0.5%

Figure 5. Execution time for databases with different sizes.



(a) minimum utility threshold = 1% (b) minimum utility threshold = 0.5%

Figure 6. Execution time for databases with different number of items.

is longer than that with 1K items. This is because the total number of candidates is 403651 in the former case, greater than 226145 in the latter case. Similarly, the time for database T20.I6.D1000K with 4K items is longer than that with 2K items, since the total numbers of candidates for the two cases are 1274406 and 779413, respectively. Thus, we can see that the execution time is proportional to the number of candidates generated during the entire process.

6.1.3 Relationship Between Effectiveness vs. Average Transaction Size

As discussed in Section 4 that high transaction-weighted utilization itemsets identified by Phase I (referred as HTWUI in Table 4) cover high utility itemsets (refered as HUI), we would like to investigate the relationship between them. As shown in Table 4, each number of HUI is smaller than the number of HTWUI, correspondingly. Another observation is that the number of HUI is closer to that of HTWUI in database T10.I6.D1000K than in T20.I6.D1000K. This is because in Phase I, the transaction-weighted utilization of any itemset X is defined as the sum of the *transaction utilities* of all the transactions containing X (equation 4.1). This overestimation gets worse when transactions are longer, because more unrelated items tend to be included in longer transactions. Despite the overestimation, the efficiency of Phase I is still evident. Hence, our proposed algorithm performs more efficiently, especially in dense databases.

Table 4. Comparison of the number of candidate itemsets (CI), high transaction-weighted utilization itemsets (HTWUI), and high utility itemsets (HUI)

Threshold	T10.I6.D1000K			T20.I6.D1000K		
	CI	HTWUI	HUI	CI	HTWUI	HUI
0.5%	226145	711	25	334268	3311	25
0.75%	153181	557	9	254647	1269	9
1%	98790	445	6	204024	799	6
1.25%	68265	370	2	159363	615	2
1.5%	44850	300	0	135468	542	0
1.75%	27730	236	0	104657	465	0
2%	16836	184	0	84667	416	0

6.2 Real-World Market Data

We also evaluated the Two-Phase algorithm using a real world data from a major grocery chain store in California. It contains

products from various categories, such as food, health care, gifts, and others. There are 1,112,949 transactions and 46,086 items in the database. Each transaction consists of the products and the sales volume of each product purchased by a customer at a time point. The size of this database is 73MByte. The average transaction length is 7.2. The utility table describes the profit of each product.

In order to evaluate if the utility mining results is useful to the grocery store, we compare the high utility itemsets with the frequent itemsets mined by traditional ARM. We do observe a number of interesting items/itemsets. For example, a kind of bagged fresh vegetable is a frequent item (the support is over 3%), however, its contribution the total profit is less than 0.25%. A combination of two kinds of canned vegetable is also a good example, which occurs in more than 1% of the transactions, but contributes less than 0.25% of the overall profit. Therefore, utility mining can help the marketing professionals in this grocery store make better decisions, such as highlight their highly profitable items/itemsets and reduce the inventory cost for frequent but less profitable items/itemsets.

We evaluate the scalability of our algorithm by varying the threshold. As shown in Table 5, it is fast and scales well. MEU doesn't work with this dataset unless out-of-core technique is designed and implemented, because the number of 2-itemset candidates is so large (approximate 2 billion) that it overwhelms the memory space available to us. Actually, very few machines can afford such a huge memory cost.

Table 5. Experiment summary of the real-world market data

Minimum utility threshold	Running time (seconds)	# Candidates	# High transaction-weighted utilization (Phase I)	# High utility (Phase II)
1%	25.76	11936	9	2
0.75%	33.3	23229	26	3
0.5%	53.09	69425	80	5
0.25%	170.49	627506	457	17
0.1%	1074.94	7332326	3292	80

Result accuracy is a very important feature of utility mining, because the mining results can be used to guide the marketing decisions. Therefore, the accuracy comparison between our Two-Phase algorithm and MEU is given in Table 6. The miss rate is defined as (the number of high utility itemsets – the number of high utility itemsets discovered) ÷ the number of high utility itemsets. To control the execution time of MEU, we set the minimum support and the utility threshold to the same value, i.e. 1%, 0.75%, 0.5%, 0.25% and 0.1%. With this support constraint, MEU works with this data set. However, it may lose some high utility itemsets whose support values are below the support threshold. For example, when the utility threshold is set at 0.1%, the Two-Phase algorithm discovers 80 high utility itemsets whereas MEU (support is set at 0.1%) only gets 66 and misses 14 high utility 2-itemsets. Our algorithm guarantees that all the high utility itemsets will be discovered.

Table 6. Accuracy comparison between Two-Phase algorithm and MEU (with support constraint) on the real-world market data

Threshold	# High utility (Two-Phase)	# High utility (MEU with support constraint)	MEU Miss rate
1%	2	1	50%
0.75%	3	2	33.3%
0.5%	5	3	40%
0.25%	17	17	0%
0.1%	80	66	17.5%

6.3 Parallel Performance

We vary the number of processors from 1 to 8 to study the scalability of our parallel implementation on the real grocery store dataset. Figure 7(a) presents the measured total execution time. The corresponding speedups are presented in Figure 7(b). As the minimum utility threshold decreasing, the search space is increasing dramatically. We observed that it scales better when the searching space increasing. The best case is 4.5 times speedup on 8 processors in the case of minimum threshold = 0.25%. The performance limitation stems from the significant amount of atomic access to the shared hash tree structure. Overall speaking, the parallel scalability in our experiment is good.

7. CONCLUSIONS

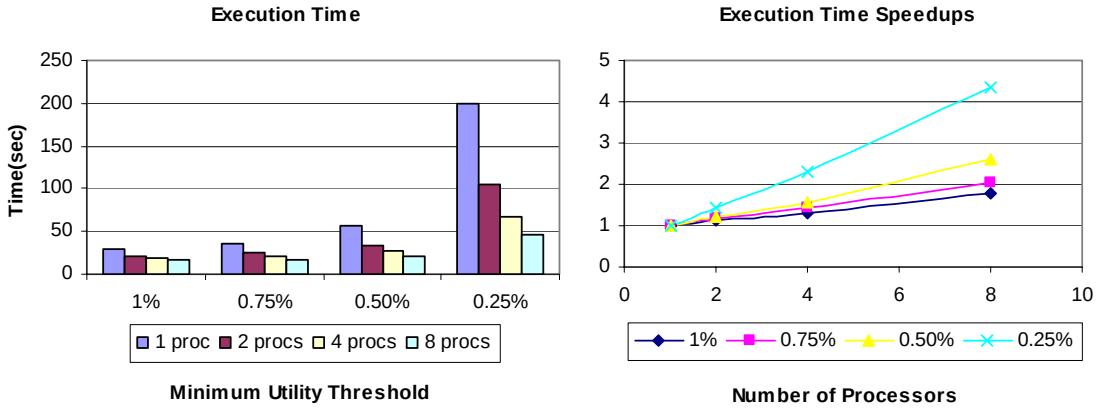
This paper proposed a Two-Phase algorithm that can discover high utility itemsets highly efficiently. Utility mining problem is at the heart of several domains, including retailing business, web log techniques, etc. In Phase I of our algorithm, we defined a term *transaction-weighted utilization,* and proposed the *transaction-weighted utilization mining* model that holds *Transaction-weighted Downward Closure Property*. That is, if a *k*-itemset is a low transaction-weighted utilization itemset, none of its supersets can be a high transaction-weighted utilization itemset. The transaction-weighted utilization mining not only effectively restricts the search space, but also covers all the high utility itemsets. Although Phase I may overestimate some itemsets due to the different definitions, only one extra database scan is needed in Phase II to filter out the overestimated itemsets. Our algorithm requires fewer database scans, less memory space and less computational cost. The accuracy, effectiveness and scalability of the proposed algorithm are demonstrated using both real and synthetic data on shared memory parallel machines. Another important feature is that Two-Phase algorithm can easily handle very large databases for which other existing algorithms are infeasible.

8. ACKNOWLEDGMENTS

This work was supported in part by NSF grants CCF-0444405, CNS-0406341, CCR-0325207, DOE grant DE-FC02-01ER25485 and Intel Corp.

9. REFERENCES

[1] Agrawal, R., and Srikant, R. Fast algorithms for mining association rules. *20th VLDB Conference*, 1994.



(a) Execution time

(b) Speedups of the execution time

Figure 7. Execution time of the real world database and speedups on Xeon 8-way SMPs. The minimum threshold is varied from 0.25% to 1%.

[2] Yao, H., Hamilton, H. J., and Butz, C. J. A Foundational Approach to Mining Itemset Utilities from Databases. *Proc. of the 4th SIAM International Conference on Data Mining*, Florida, USA, 2004.

[3] Park, J. S., Chen, M., and Yu, P.S. An Effective hash Based Algorithm for Mining Association Rules. *Proc. of ACM SIGMOD Conference*, New York, 1995.

[4] Brin, S., Motwani, R., UIIman J. D., and Tsur, S. Dynamic itemset counting and implication rules for market basket data. *Proc. of ACM SIGMOD Conference on Management of Data*, Arizona, 1997.

[5] Zaki, M. J., Parthasarathy, S., Ogihara, M., and Li, W. New Algorithms for Fast Discovery of Association Rules. *Proc. of 3rd Conference on Knowledge Discovery and Data mining*, California, 1997.

[6] Han, J., Pei, J., and Yin, Y. Mining Frequent Patterns without Candidate Generation. *Proc. of SIGMOD*, 2000.

[7] Aumann, Y., and Lindell, Y. A statistical theory for quantitative association rules. *Proc. of the 5th KDD*, 1999.

[8] Webb, G. I. Discovering associations with numeric variables. *Proc. of the 7th KDD,* 2001.

[9] Cai, C. H., Fu, Ada W. C., Cheng, C. H., and Kwong, W. W. Mining Association Rules with Weighted Items. *Proc. of International Database Engineering and Applications Symposium (IDEAS)*, 1998.

[10] Wang, W., Yang, J., and Yu, P. S. Efficient Mining of Weighted Association Rules (WAR). *Proc. of 6th KDD*, 2000.

[11] Tao, F., Murtagh, F., and Farid, M. Weighted Association Rule Mining using Weighted Support and Significance Framework. *Proc. of International Conference on Knowledge Discovery and Data mining*, 2003.

[12] Lu, S., Hu, H., and Li, F. Mining weighted association rules. *Intelligent Data Analysis*, 5(3) (2001), 211-225.

[13] Barber, B., and Hamilton, H. J. Extracting share frequent itemsets with infrequent subsets. *Data Mining and Knowledge Discovery*, 7(2) (2003), 153-185.

[14] Chan, R., Yang, Q., Shen, Y. Mining high utility Itemsets. *Proc. of IEEE ICDM*, Florida, 2003.

[15] IBM, IBM synthetic data generation code. *http://www.almaden.ibm.com/software/quest/Resources/index.shtml.*

[16] Zaki, M. J., Parallel and Distributed Association Mining: A Survey. *IEEE Concurrency, special issue on Parallel Mechanisms for Data Mining*, Vol. 7, No. 4, December 1999, 4-25.

[17] Zaki, M. J., Ogihara, M., Parthasarathy, S., and Li, W. Parallel Data Mining for Association Rules on Shared-memory Multi-processors. *Supercomputing*, Pittsburg, PA, November 1996.

[18] OpenMP. OpenMP: Simple, Portable, Scalable SMP Programming. *http://www.openmp.org/*.

[19] Silberschatz, A., Tuzhilin, A. What Makes Patterns Interesting in Knowledge Discovery Systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(6), December 1996.