

UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

A comparison of methods for learning cost-sensitive classifiers

Permalink

<https://escholarship.org/uc/item/7cx363kc>

Author

Green, Michael Todd

Publication Date

2010

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA, SAN DIEGO

A Comparison of Methods For Learning Cost-Sensitive Classifiers

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science

by

Michael T. Green

Committee in charge:

Professor Charles Elkan, Chair
Professor Garrison Cottrell
Professor Sanjoy Dasgupta

2010

Copyright
Michael T. Green, 2010
All rights reserved.

The thesis of Michael T. Green is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2010

DEDICATION

To Krissy, Jordan, and Evan

the two most powerful warriors
are patience and time
—Leo Nikolaevich Tolstoy

TABLE OF CONTENTS

Signature Page	iii
Dedication	iv
Table of Contents	v
List of Figures	vii
List of Tables	viii
Acknowledgements	ix
Abstract of the Thesis	x
Chapter 1 Introduction	1
1.1 Statement of the Problem	2
1.2 Definition of Terms	3
1.3 Description of Remaining Chapters	3
Chapter 2 Previous Work	4
2.1 Review of Relevant Work	4
2.2 Estimating Class Membership Probabilities	6
2.3 MetaCost	9
2.4 Costing: Cost-Sensitive Problem Translation	12
2.5 Direct Cost-Sensitive Decision Making	14
2.6 Applicability to Problem Domains And Novelty of this Research	15
Chapter 3 Empirical Research Methodology	17
3.1 Experiment Approach	17
3.2 Measuring Performance	18
3.3 Data Sets	19
3.3.1 KDD98 Data Set	19
3.3.2 DMEF2 Data Set	20
Chapter 4 Experimental Environment	21
4.1 RapidMiner Community Edition v4.4	21
4.2 Implementation Choices and Coding	21
4.2.1 Implementing MetaCost	23
4.2.2 Implementing Costing	24
4.2.3 Implementing Direct Cost-Sensitive Classification	26
4.3 Reusable Elements of Implemented Code	29

Chapter 5	Results and Analysis	31
	5.1 Validation Results	31
	5.2 Empirical Comparison	34
Chapter 6	Limitations of Work	38
Chapter 7	Applications of Thesis and Conclusion	39
	7.1 Results	40
	7.2 Problems Left Unsolved	40
Bibliography		42
Appendix A	RapidMiner XML Implementations	45

LIST OF FIGURES

Figure 2.1: Lift chart curves for bagging (top) and logistic regression (bottom). The non-inferior percentage of positive examples for logistic regression at any arbitrary fraction of the ordered data set is consistent with a learner that is producing better calibrated class probability estimates than bagging.	10
Figure 4.1: MetaCost RapidMiner Process Diagram	25
Figure 4.2: Costing RapidMiner Process Diagram	27
Figure 4.3: Direct Cost RapidMiner Process Diagram	28
Figure A.1: RapidMiner Process for MetaCost Experiment (KDD98 example)	46
Figure A.2: RapidMiner Process for Costing Experiment (KDD98 example)	47
Figure A.3: Rapid Miner Process for Direct Cost Experiment (KDD98 example)	48

LIST OF TABLES

Table 2.1:	Mean squared errors observed when applying the bagging and logistic regression algorithms to the Blood Transfusion data set. The MSE is lower for logistic regression. This can indicate that the algorithm produces better calibrated class probabilities than bagging.	9
Table 2.2:	General Methods of Cost-Sensitive Problem Domain Translation	15
Table 4.1:	New Rapidminer Operators. These new operators were created for the thesis and are included in the plug-in.	29
Table 4.2:	New Rapidminer Operators. These new processes were created for the thesis and are included in the plug-in.	30
Table 5.1:	KDD-98 Validation Results. The results of the implemented algorithms closely approximate the results produced in prior research.	32
Table 5.2:	DMEF-2 Validation Results. The results of the costing algorithm implemented closely approximates the results produced in prior research. Comparisons for MetaCost and Direct Cost are not available.	32
Table 5.3:	Probability Calibration Results. The mean squared error (MSE) is reported for each decision tree algorithm against the KDD98 and DMEF2 data sets. Against both test data sets, the CalibratedDecisionTree had a lower corresponding MSE than the DecisionTree algorithm. In addition, the absolute difference between the MSE when the algorithms are applied to the training data set and the MSE when the algorithms are applied to the test data set is smaller for the calibrated operator.	33
Table 5.4:	KDD-98 Results. The performance is similar across all algorithms. Increasing the number of re-samples for MetaCost and costing does not have a significant impact.	37
Table 5.5:	DMEF-22 Results. The performance is similar across all algorithms. Costing appears to benefit from increasing the number of re-samples suggesting that it may outperform other methods.	37

ACKNOWLEDGEMENTS

I have depended on the help and support of many people, to whom I owe far more gratitude than I can possibly express here.

ABSTRACT OF THE THESIS

A Comparison of Methods For Learning Cost-Sensitive Classifiers

by

Michael T. Green

Master of Science in Computer Science

University of California, San Diego, 2010

Professor Charles Elkan, Chair

There is a significant body of research in machine learning addressing techniques for performing classification problems where the sole objective is to minimize the error rate (i.e., the costs of misclassification are assumed to be symmetric). More recent research has proposed a variety of approaches to attacking classification problem domains where the costs of misclassification are not uniform. Many of these approaches make algorithm-specific modifications to algorithms that previously focused only on minimizing the error rate. Other approaches have resulted in general methods that transform an arbitrary error-rate focused classifier into a cost-sensitive classifier. While the research has demonstrated the success of many of these *general* approaches in improving the performance of arbitrary algorithms compared to their cost-insensitive contemporaries, there has been relatively little examination of how well they perform relative to one another. We describe and categorize three general methods of converting a cost-sensitive method into the cost-insensitive problem domain. Each method is capable of of example-based cost-sensitive classification. We then present an empirical comparison of their performance when applied to the KDD98 and DMEF2 data sets. We present results

showing that *costing*, a technique that uses the misclassification cost of individual examples to create re-weighted training data subsets, appears to outperform alternative methods when applied to DMEF2 data using increased number of re-sampled subsets. However, the performance of all methods is not statistically differentiable across either data set.

Chapter 1

Introduction

There is a large body of research on classifier learning algorithms that reflects a large variety of approaches to classification learning. In much of the research, the performance of the algorithms is reported in terms of a minimized *error rate* where all misclassification errors are assumed to be equal. However, in a number of real-world classification problems, the costs of misclassification are not equally costly. In these cases, the difference in the misclassification costs can have a dramatic impact on the ultimate decision made.

Integrating misclassification costs is particularly important for classification when one class is observed rarely but can be quite costly when mis-classified. If the “minority” class is observed very rarely, a cost-insensitive classifier can achieve a very low error rate simply by labeling all examples as belonging to the class that is observed frequently. In many real world problem domains, this approach would eliminate any utility of the classifier. For example, in the medical diagnosis realm, there is a cost for mis-classifying a normal screening mammogram as “abnormal” as well as a clear cost of classifying an abnormal mammogram as “normal”. The two misclassification costs are not uniform. Ignoring the non-uniformity of these costs would produce a classifier that would reduce its overall error rate by classifying most examples as “normal”- even if it meant misclassifying some “abnormal” examples. These types of classification problems are present in a number of real world problem domains.

As a result of this issue, more recent research has proposed numerous ap-

proaches with the objective of minimizing the overall cost of misclassification. One approach to the problem is to make specific adjustments to an *error rate* focused algorithm in order to account for the non uniform costs of misclassification. The benefit of this approach is that a classifier most appropriate to the problem domain may be chosen for modification. In addition, the modifications can be tailor made to provide the optimum sensitivity and specificity for the classification problem. However, there are drawbacks to this approach. For many classification algorithms, the best method to improve its cost-sensitivity may not be readily apparent (particularly if the user is not the original author). Even if a method could be devised to transform a cost-insensitive classifier, the process could be time consuming. Amongst a substantial and ever increasing body of research on classifiers, this approach is likely to limit the available options to addressing problems in the cost-sensitive domain.

A different approach to developing cost-sensitive algorithms has been to devise general methods of transforming any arbitrary classifier to be applied to problems in the cost-sensitive problem domain. These general methods seek to treat the arbitrary classifier as a “black box” by requiring no direct knowledge or manipulation of the underlying classifier. Instead, these general approaches focus on transforming the cost-sensitive classification problem into the cost-insensitive problem domain. These approaches have demonstrated success in improving the performance of a cost-insensitive classifier when applied to a cost-sensitive problem. However, the research on these general methods has not studied how well they perform relative to each other on standard, publicly available data sets.

1.1 Statement of the Problem

Recent research has produced a number of novel general approaches to transforming a cost-insensitive classifier into one that is cost-sensitive. The empirical results presented have demonstrated that these approaches can dramatically improve the performance of previously cost-insensitive classifiers when applied to a cost-sensitive problem. However, there has been limited research which would

direct a user of these methods as to when one method should be preferred over another. By using empirical tests against publicly available data sets, we will demonstrate how well these methods perform relative to one another.

1.2 Definition of Terms

Cost-Insensitive: An approach where the objective is to minimize the error rate.

Cost-Sensitive: An approach where the objective is to minimize the total costs of misclassifying examples. That is to say, the total misclassification costs may be minimized while the error rate is not minimized.

Class-Dependent Misclassification Costs: The costs of misclassification of an example from class A as class B is uniform for all examples in class A . The costs of misclassifying an example from class A as class C is also uniform for all examples, but may be different than the misclassification costs for an example in class A misclassified as class B . Cost-sensitive methods that incorporate misclassification costs typically employ a misclassification matrix to specify those costs.

Example-Dependent Misclassification Costs: The costs of misclassification of an example from class A as class B varies depending on features of the example.

1.3 Description of Remaining Chapters

We provide a brief overview of previous work in cost-sensitive classification in Chapter 2. In particular, we detail methods representing various categories of approaches to cost-sensitive transformation. In Chapter 3 we discuss the research methodology and the standardized data sets employed. In Chapter 4 we present and analyze the results of the empirical study. In Chapter 5 we discuss the limitations of our analysis. In Chapter 6 we identify applications of this thesis and discuss conclusions and possible further work on this topic.

Chapter 2

Previous Work

In this chapter we discuss general approaches to applying cost-insensitive algorithms to cost-sensitive problem domains. After providing a high level overview of these approaches, we present details on those algorithms chosen for empirical study.

2.1 Review of Relevant Work

As mentioned previously, the body of research on cost-sensitive learning algorithms can be divided into two domains. The first domain includes approaches that seek to directly modify a previously cost-insensitive algorithm such that it incorporates costs. The second domain includes approaches that largely treat existing algorithms as a black box and instead transform the problem by making other changes to the problem space. Approaches in the second domain can be placed into one of three distinct categories pertaining to the general steps of supervised learning.

The first category contains approaches that rely on modifying the data input space (instead of any part of the classifier) such that a new set of examples is created and sensitivity has been shifted towards cost-sensitive outcomes. The modifications applied to the data input space may involve over/under samples of training examples and/or modifying the labels of input examples. By modifying the data input space to promote greater emphasis on costlier examples, algorithms

in this category essentially translate a problem from the cost-sensitive problem domain into the cost-insensitive problem domain. Because the problem has been translated to the cost-insensitive domain, previously inappropriate cost-insensitive algorithms can be applied. This approach allows cost-sensitive domain problems to leverage advances with cost-insensitive domain approaches. In addition, this approach requires little (or no) detailed knowledge of the cost-insensitive learner that will process the training set. Elkan [11] proves a theorem the use of a standard non-cost-sensitive classifier to make optimal cost-sensitive classification decisions by changing the proportion of negative examples in a training set. Examples of other approaches which fit into this category are *MetaCost* [9] and *costing* [26], which will be discussed in further detail in sections to follow.

A second category of approaches contains methods that focus on modifications to the training stage of existing classification methods. These algorithms use a form of risk theory to incorporate misclassification costs during the classification stage. Classifiers following this approach assign each example to its lowest risk class. Approaches fitting into this category avoid the need to have detailed knowledge of an underlying base learner. Ensemble or meta-learner methods use such an approach [18]. These methods effectively “wrap” an arbitrary inner learner and employ a cost-sensitive feedback loop in order to increase the importance of specific examples. Examples of this approach are Fan and Stolfo’s *AdaCost* [12] and Masnadi-Shirazi and Vasconcelos’ [19] *asymmetric boosting*, which both modify the well known *AdaBoost* [21] algorithm to improve cost-sensitive performance. As demonstrated by the results for *asymmetric boosting*, these approaches can successfully improve the cost-sensitive performance.

Finally, a third approach avoids modifying the input data or the models of existing classifiers and simply applies Bayes risk theory directly to input data in order to derive estimates of conditional risks and then directly use those estimates at “Decision Time” to choose the least risky (i.e., costly) label for an example. Elkan [11] proposes the use of such a method when using Bayesian or decision tree learning methods.

In addition to the general approach towards cost-sensitive classification,

methodologies in this realm can also differ along two other dimensions. The first dimension is the ability of the cost-sensitive classifier to support multi-class problem solving domains versus only the 2-class classification case. The second dimension is the degree of freedom allowed in accommodating misclassification costs. Much of the early research focused on incorporating uniform misclassification costs, where a constant “cost” of misclassifying an example of class a as an example of class b affects the learner, into the learning process. More recently, there has been an increasing focus on cost-sensitive learning problems where the costs of misclassification are non-uniform (i.e. they vary from one example to the next for the same type of misclassification). There are numerous real world problems where the misclassification costs fit this pattern. In credit card fraud detection systems, fraudulent charges for higher dollar amounts are clearly more costly than those of smaller dollar amounts. Other situations where fine-grained cost-sensitivity can be important are business marketing and operational decision making as well as medical diagnosis problems.

In this paper, we present empirical results analyzing the performance of approaches that are general and incorporate example-dependent cost-sensitive learning problems where the costs of misclassification are non-uniform. In the following sections, we discuss considerations for class membership probability estimation and discuss details of the specific approaches researched.

2.2 Estimating Class Membership Probabilities

Two of the approaches studied in this research employ a form of risk theory to shift decisions during training or decision time. Stated simply, the general form of the risk theory seeks to label examples according to the label that has the lowest expected cost. Clearly the expected cost is a function of the misclassification cost and the likelihood of misclassification. For this reason, estimating class membership probabilities is a critical component of approaches that employ risk theory. While there are numerous well documented examples of using classifiers successfully for discrete class prediction, the ability of such classifiers to produce

reliable probabilistic output should not be taken for granted. In this section, we discuss methods for estimating class membership probabilities and why certain methods do not produce well-calibrated class probabilities. At the end of this section, we also present empirical results establishing that bagging does not produce well calibrated probabilities.

Bagging [5]) creates an ensemble of classifiers by training each on one of a series of bootstrap samples of a larger set of training data. The classifier is produced by the same “base” learning algorithm. Each trained classifier produces a class prediction of either 0 or 1. The class probability $P(Y_i|X)$ produced by the ensemble is the fraction of trained classifiers that predict that particular class. Margineantu [17] observed that the votes upon which class probabilities produced by bagging are based are a measure of the variance of “base” learner on a particular example. The higher the variance, the less stable the base learner is a particular example. This variance is not the same as the class membership probability. For example, if a base learner has learned to classify a particular example that has a true probability of being in class 1 of 75%, each classifier in the ensemble may predict class 1 resulting in a class membership probability estimate of 100%. For this reason, bagging is not a sound choice for estimating class membership probabilities.

Decision trees are an alternative method for producing class membership probabilities. A class membership probability estimate $p = k/n$ for a particular class is the number of training examples k among all examples n at the terminal leaf after applying the rules to a test example. While these estimates avoid the same issue as bagging, class probabilities produced in this way should still be expected to exhibit a bias towards zero or one estimations. By design, decision tree methods produce leaves that tend toward homogeneity. In addition, leaves representing a small number of training examples will suffer from high variance and are not be statistically reliable. For these two reasons, class probability estimates based on the “raw” scores at decision tree leafs will result in compromised risk theory decisions.

Zadrozny et al. [25] proposes an approach to obtain well-calibrated class probabilities using decision trees. To counteract the bias towards a zero or one

decision when using decision trees for probability estimates, they discuss three alternatives. The first is a technique, called *m*-estimation or *smoothing* [6] involves correcting the derived class probability towards an assumed underlying probability much like a Laplace correction method. However, rather than “correct” the probability estimates towards 50%, *smoothing* corrects estimates towards a specific target unique to a given data set by replacing $p = \frac{k}{n}$ with $p' = \frac{k+b \times m}{n+m}$ where b is the base rate of the positive class and m is a factor that determines how much the probability estimate is shifted towards it. A second approach, called *curtailment* ensures that the node used to make the probability estimation represents a sufficient number of examples. *Curtailment* differs from *pruning* in that the full tree is maintained and a single node may serve as a leaf in some decision instances and may serve as a branch in others.

Zadrozny and Elkan discuss another method for calibrating probability estimates based on a naive Bayesian classifier. Although naive Bayesian classifiers are based on the often inaccurate assumption of feature independence, such classifiers still tend to rank examples well. The method, called “binning”, is a non-parametric method that takes advantage of the naive Bayesian classifier’s ability to rank examples. With binning, the examples are sorted according to their ranks and then placed into one of an arbitrary number of equal sized subsets (bins). For each bin, the upper and lower boundary is calculated based on the classifier score. The class membership probability is then estimated by calculating the fraction of examples of class j in the bin.

It is important to note that not all standard classification learners require modification in order to produce well calibrated probabilities. Logistic regression is an example of a classification method that can produce reliable class membership probabilities. To demonstrate the difference in reliability of class membership probabilities, we present the results of an experiment where we apply two different learners to the Blood Transfusion data set available at the UCI KDD Archive [1]. The Blood Transfusion data set contains 748 examples each with four numeric attributes and a binomial attribute indicating whether or not the person represented by the example donated blood during a particular blood drive. We applied

Table 2.1: Mean squared errors observed when applying the bagging and logistic regression algorithms to the Blood Transfusion data set. The MSE is lower for logistic regression. This can indicate that the algorithm produces better calibrated class probabilities than bagging.

	Bagging	Logistic Regression
Mean Squared Error	0.20171	0.16350

the bagging algorithm using a naive Bayesian classifier as the inner learner. For comparison, we applied a learner employing logistic regression (using RapidMiner’s FastLargeMargin operator). For each algorithm, we calculated the mean squared error (MSE) using the predicted class membership probabilities and assigning a value of 0 or 1 for the true class probability for non-responders and responders, respectively. Table 2.1 displays the mean squared error observed for both algorithms. A lower MSE can indicate better calibrated probabilities. Since it could also indicate better ranking, it is possible that logistic regression is estimating true class membership probabilities more accurately and/or it is providing better class probability rankings. The Figure 2.1 shows the resulting lift charts for each experimental run. The lift charts provide a graphical representation of the set of test examples sorted in descending order of estimated class membership probability. The first instance is the one that the learning scheme estimates is most likely to be positive, the second is the next most likely to be positive, and so on. The charts show the number of true positive examples observed (y-axis) versus the fraction of the test set used (x-axis).

2.3 MetaCost

The MetaCost algorithm focuses on relabeling the training data examples such that each new label reflects the least costly prediction for the example, even if the new label is different from the true label for the example. For an example x , we need to know the probability of each class j as $P(j|x)$. If we also know the potential cost of misclassifying the example as class i for each possible j $C(i|j)$, we

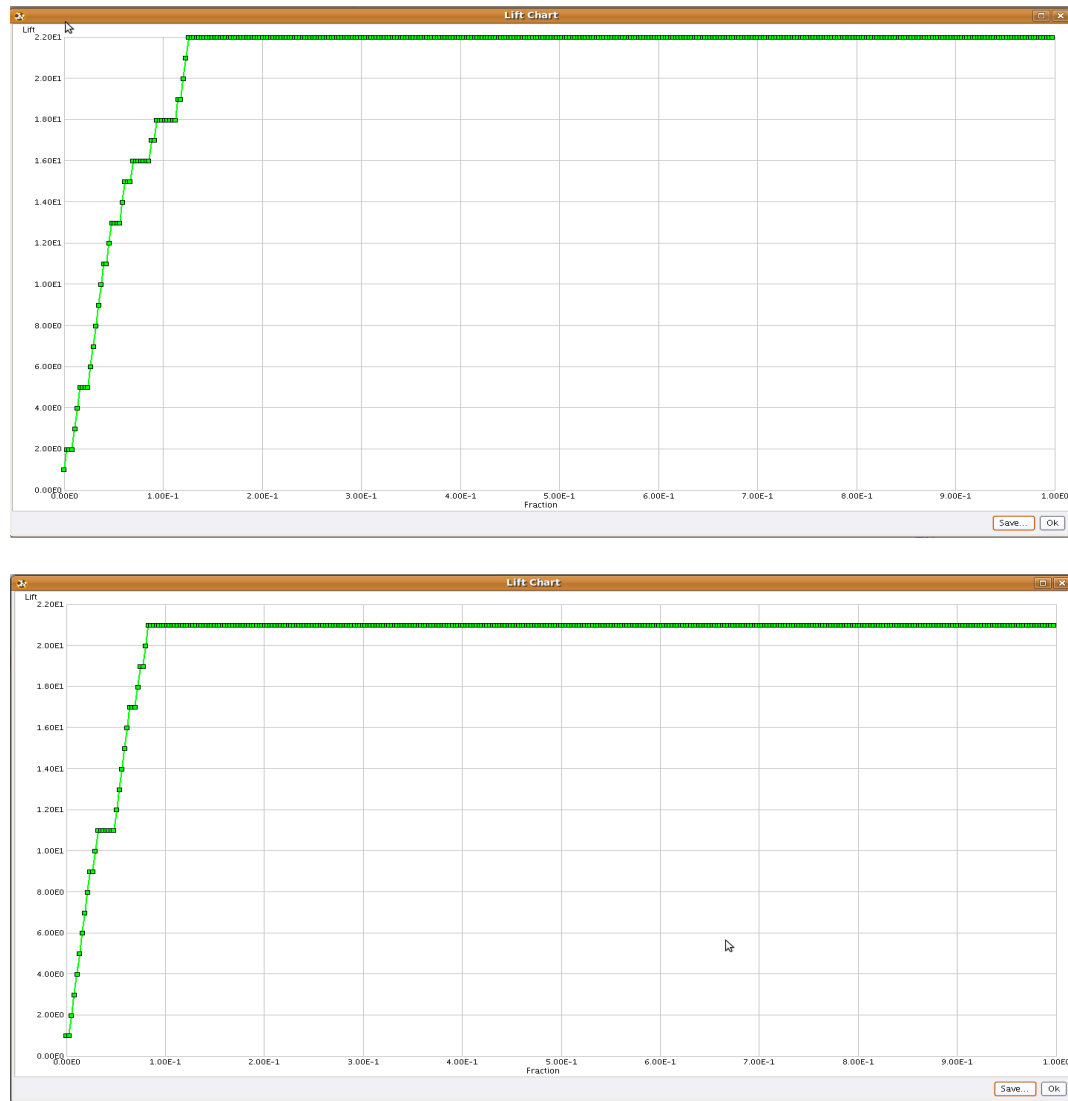


Figure 2.1: Lift chart curves for bagging (top) and logistic regression (bottom). The non-inferior percentage of positive examples for logistic regression at any arbitrary fraction of the ordered data set is consistent with a learner that is producing better calibrated class probability estimates than bagging.

can relabel the example to its least costly prediction. The new label reflects the *Bayes optimal* prediction which seeks to minimize the *conditional risk* $R(i|x)$ [10]:

$$R(i|x) = \sum_j P(j|x)C(i|j) \quad (2.1)$$

The conditional risk $R(i|x)$ is the expected cost of predicting that an example x belongs to class i . Assuming the probability and costs estimates are accurate, the *Bayes optimal* prediction is guaranteed to achieve the lowest possible overall cost. In the MetaCost algorithm, the examples in the training set are relabeled with their optimal predictions given the estimated probabilities and misclassification costs. The relabeled training set is provided to the otherwise cost-insensitive “core” classifier. The predictions produced by the base classifier should then be sensitive to the cost of misclassifying examples. Domingos presents empirical evidence demonstrating improved cost-sensitive performance of the C4.5R algorithm.

To estimate class probabilities, the MetaCost algorithm uses bagging on a specified number of re-samples of the training data. With bagging, a re-sample of the training data is created based on sampling *with replacement* from the training set. In the new training set, each example in the original training set may appear once, more than once, or not at all. A model is learned by applying the core classifier to the re-sampled training set and this procedure is repeated m times to produce m models. Class probabilities are determined by uniform voting of each model. If the base learner produces class probabilities rather than an outright prediction, MetaCost uses the average of the resulting probabilities. As discussed above, bagging does not produce well calibrated class membership probabilities. Prior work by Zadrozny and Elkan has established that MetaCost’s use bagging, or use of bagging for probability estimates in general, will not produce well-calibrated probability estimates.

The pseudo-code for the MetaCost algorithm is listed in Algorithm 1. Please note that L can be any arbitrary classification learning algorithm capable of producing either a predicted class or class membership probabilities based on examples from S as inputs.

Algorithm 1 The MetaCost Procedure (S, L, C, m, n, p, q)

Require: S is the training set, L is a classification learning algorithm, C is a cost matrix, m is the number of re-samples to generate, p is the class probability indicator, q is a re-sampling indicator, p is *True* iff L produces class probabilities, q is *True* iff all re-samples are to be used for each example

```

1: for  $i = 1$  to  $m$  do
2:   Let  $S_i$  be a re-sample of  $S$  with  $n$  examples
3:   Let  $M_i$  be a Model produced by applying  $L$  to  $S_i$ 
4: end for
5:
6: for each example  $x$  in  $S$  do
7:   for each class  $j$  do
8:     Let  $P(j|x) = \frac{1}{\sum_i 1} \sum_i P(j|x, M_i)$ 
9:     if  $p$  then
10:       $P(j|x, M_i)$  is produced by  $M_i$ 
11:     else
12:       $P(j|x, M_i) = 1$  for the class predicted by  $M_i$  for  $x$ , and 0 for all others
13:     end if
14:     if  $q$  then
15:        $i$  ranges over all  $M_i$ 
16:     else
17:        $i$  ranges over all  $M_i$  such that  $x \notin S_i$ 
18:     end if
19:   end for
20:   Let  $x$ 's class =  $\operatorname{argmin}_i \sum_j P(j|x) C(i, j)$ 
21: end for
22:  $M$  = Model produced by applying  $L$  to  $S$ 
23: Return  $M$ 

```

2.4 Costing: Cost-Sensitive Problem Translation

In this section we review an alternative approach to cost-sensitization called *costing* proposed by Zadrozny et al. [26]. This algorithm enables the application of

a cost-insensitive algorithm by transforming the training set to account for the cost of misclassifying individual examples. Costing uses sub-sampling to transform a data set $D(x, y, c)$, where c is the cost of misclassifying an example (x, y) , to create a new dataset $\hat{D}(x, y)$. The sub-sampling method ensures that $\hat{D}(x, y) \equiv D(x, y, c)$ in the sense that optimal error rate classifiers drawn from $\hat{D}$ are the same optimal error rate classifiers that would be drawn from D . The sub-sampling methodology is called “rejection sampling” [23]. Rejection sampling “accepts” examples into a new sub-sample dataset based on the proportionate misclassification cost of each example and avoids over fitting issues typically encountered with sampling-based schemes. With rejection sampling, each example is considered for inclusion in the subsample and is rejected with a probability inversely proportionate to the cost of misclassifying that example. Costing’s approach relies on well known results from decision theory which state that if examples are drawn from a distribution

$$\hat{D}(x, y, c) \equiv \frac{c}{E_{x,y,c \sim D}[c]} D(x, y, c) \quad (2.2)$$

then the optimal error rate classifiers for $\hat{D}$ would also be optimal error rate classifiers for D [26]. In cost-proportionate rejection sampling, samples in $\hat{D}$ are obtained by drawing an example from D and then accepting the example with the probability proportional to the relative weight of the example’s cost c/Z , where Z is a constant chosen so that $\max_{(x,y,c) \in D} c \leq Z$. In practice, Z is chosen to be the cost of the most costly example in D .

The central limit theorem implies that this rejection sampling scheme would reduce the original data set by a factor of $\Theta(c_{avg}/Z)$, where c_{avg} is $\frac{1}{N} \sum_{i=1}^n Cost(n)$ and $Cost(n)$ is the cost c for the n th example in the data set. However, Zadrozny demonstrates that the PAC-learning theorems [22] [15] prove that optimizing data sets derived from cost-proportionate rejection sampling will likely produce results no worse than using no sampling. In other words, cost-proportionate rejection sampling produces smaller data sets that are probabilistically as informative as the original data set.

The costing algorithm is further improved by creating an ensemble of hypotheses each derived from different rejection-sampled data sets. The costing al-

gorithm is shown in shown in Algorithm 2.4.

Algorithm 2 The Costing Algorithm (S, L, T)

Require: S is the training set, L is a classification learning algorithm, t is the number of rejection samples to generate, Z is a constant, Z is chosen to be $\geq$ the most costly example in S

- 1: **for** $i = 1$ to t **do**
 - 2: Let S_i = a re-sample of S with acceptance probability c/Z
 - 3: Let h_i be the model produced by applying L to S_i
 - 4: **end for**
 - 5: Output: $h(x) = \text{sign}(\sum_{i=1}^t h_i(x))$
-

2.5 Direct Cost-Sensitive Decision Making

As discussed above, the MetaCost algorithm transforms a cost-insensitive classification problem into a cost-sensitive problem by submitting examples re-labeled according to their lowest (cost-sensitive) risk to a base learner. Zadrozny and Elkan [25] propose a more direct method for cost-sensitive decision making. Rather than applying conditional risk estimations to the training set for a base learner during the training phase, the estimates of conditional risk are applied at “decision time” directly to the test examples in order to classify them to the class with the lowest conditional risk. Zadrozny and Elkan use decision trees, with the *smoothing* and *curtailment* modifications discussed above, for producing class membership probability estimates. In addition, they propose the use of binned Naive Bayes for class membership probability estimates. Since the results of each of these approaches are independent, Zadrozny and Elkan suggest determining all three and averaging the results. Given an estimate of the class membership probability, the other component of conditional risk is the (estimate of) misclassification cost. Zadrozny and Elkan propose a method to estimate the misclassification cost using least squares multiple linear regression. To account for sample selection bias where the estimated feature is only known for one of the classes and the feature is

Table 2.2: General Methods of Cost-Sensitive Problem Domain Translation

General Method	Classification Domain		Misclassification Costs	
	2 Class	> 2 Class	Class Dependent	Example Dependent
MetaCost	Yes	Yes	Yes	No
Costing	Yes	No	Yes	Yes
Direct Cost	Yes	Yes	Yes	Yes

not orthogonal to the class membership probability, Zadrozny and Elkan employ a procedure proposed by Heckman of the University of Chicago [14]. The decisions are made directly using the test examples rather than using a classification model based on the predictions as an intermediary.

2.6 Applicability to Problem Domains And Novelty of this Research

As presented in their original research, each of the methods above have been shown to be successful in improving the sensitivity of classification methods to misclassification costs. However, the methods evaluated are not applicable towards completely overlapping problem domains. For instance, the MetaCost algorithm can be applied to both two class and multi-class classification problem domains while the costing algorithm as described works for only two class classification problems. In addition to the number of classes to be identified, the general methods vary in their ability to accommodate example dependent misclassification costs versus only class dependent misclassification costs. Table 2.2 summarizes the general methods discussed above along two important problem domain dimensions.

Approaches from all three categories have proven capable of producing classifiers better at minimizing the cumulative cost of misclassification when compared to cost-sensitive analogs. However, there has been limited research comparing these approaches to each other with regard to how well they minimize cumulative misclassification costs when applied to the same data set.

In this research we chose to evaluate the algorithms in classification problem

domains where there are only two possible classes and the costs of misclassification may vary based on individual examples. This problem domain encompasses many real world problems. For example, with credit card fraud detection systems, fraudulent charges for higher dollar amounts are more costly than those of smaller dollar amounts. Also, with medical diagnostics, the risk of stroke (i.e., costs) due to an undiagnosed atrial fibrillation is much greater for a patient with co-morbidities such as diabetes and obesity.

Chapter 3

Empirical Research Methodology

3.1 Experiment Approach

This chapter focuses on an empirical evaluation and comparison of three general methods for making otherwise cost-insensitive classifiers into cost-sensitive classifiers. At the outset of this work, the goal was to evaluate each approach described in the original research, with little or no modifications. To do this, the methods were implemented and executed within a common platform and applied against data sets within the 2 class classification problem domain. However, as described in their original research, the general methods studied could not be applied to a completely overlapping set of problems. The true intent of this research was to understand differences in the performance of three general approaches to this problem and not necessarily evaluate the specific implementations detailed in the original research. Where techniques employed by one approach could improve the results achieved by a second approach, those techniques were introduced. This was done to make the results more attributable to the fundamental differences in the approaches and not merely attributable to specific implementation choices. Given the large number of real world classification problems where misclassification costs are example dependent, we focused the evaluation on problems that are in that domain. In the knowledge discovery realm, there are numerous real-world problems that match this description. Specifically, the experiments focused on the two class problem domain where the costs of misclassification vary at the example

level (as opposed to the class level).

The experiments were conducted by applying each of the three methods against two data sets, resulting in six separate experimental runs. Each experimental run was conducted by averaging the results of a “static” 10-fold cross validation. The 10-fold cross validations used the supplied data set to create 10 new training/testing data set pairs based on stratified sampling with 90% of the data randomly selected into each training set and 10% selected into the test set. The term “stratified sampling” means that the resulting data set has the same ratio of positive to negative responders as the original data set. We use the term “static” because the cross validation data set pairs were maintained such that each method evaluated would run against the same exact sets.

3.2 Measuring Performance

Most research on cost-sensitive classification reports algorithm performance in terms of a “total cost”. In this context, total cost is the summation of the misclassification costs of each test example of the entire test set. For comparison purposes in this research, we report the more generalizable metric of “total benefit”. Elkan [11] discusses the rationale behind favoring “benefit” in that it allows for a more clear reporting of a change (positive or negative) from any baseline state. The “net benefit” of any example was calculated by taking the benefit derived by acting based on the classification of the example and subtracting the costs of acting based on the classification of the example. The total benefit was considered to be the sum of net benefits over the entire test set. In each experiment, the net benefit directly related to monetary performance so we report the net benefit as “profit”. In order to gain further insight into the decision making employed by the resulting algorithms, statistics were tracked on the win/loss records for each solicitation or catalog mailed.

3.3 Data Sets

The data sets for the evaluation were selected because they were publicly available and because they contain non-uniform misclassification cost information for each example. Despite the existence of many other problem domains that are cost-sensitive, both data sets used are derived from the marketing domain. The lack of publicly available data sets with specific misclassification costs prevented us from using data from other domains such as the finance and medical diagnosis domains.

3.3.1 KDD98 Data Set

The KDD'98 data set is available from the UCI data repository [1]. The data set contains a training and a test set each consisting of approximately 96,000 records. For this research, the two data sets were combined into a single data set. The records pertain to potential donors for a charity solicited by a mail campaign. In addition to potential donor biographic data, each record indicates whether or not the person made a donation and how much the person donated, if a donation was made. The percentage of donors in both sets is about 5%. Each solicitation mailed costs the charity approximately \$0.68. The amount donated by responders to the mailing varied from \$1 to \$200. The profit obtained by soliciting every individual in the test set would be \$10,560. The winning learner for the KDD-98 competition obtained a profit of \$14,712.

The cost of misclassifying each example is derived by taking the difference in profit between mailing and not mailing to an individual. The profit from mailing a solicitation is the donation amount (e.g., \$0 to \$200) minus the cost of the mailing of \$0.68. Not mailing to an individual results in zero profit. For example, misclassifying a would-be respondent would cost somewhere between \$0.32 and \$199.32 in profits. Misclassifying a non-respondent would cost \$0.68 in profits. As discussed above, the results for experiments using this data set are reported as the resulting profit based on classifying the test data set.

The *smoothing* technique applied to the MetaCost and direct classification

methods requires an input based on the assumed underlying base rate. For both *MetaCost* and the direct classification method, this base rate of 5% was used. To employ the Heckman performance estimation procedure, the variables *lastgift* (the amount of the most recent gift) and *ampergift* (the average dollar amount in response to the last 22 solicitations) were provided as the independent variables to the mean least squares (MLR) operator.

3.3.2 DMEF2 Data Set

The DMEF2 data set is the “Data Set 2” available from the Direct Marketing Educational Foundation dataset library for \$25.00 [2]. The data set contains approximately 96000 records that pertain to the purchase history for customers of a mail order catalog campaign. In addition to customer biographic data, each record indicates whether or not the person made a purchase and how much the person purchased, if one was made. The percentage of purchasers in the set is about 2.5%. Because it was not specified, we assumed that each catalog mailed costs the company \$2.00 in postage. The amount purchased by responders to the campaign varied from \$3 to \$6,247.

The cost of misclassifying each example is derived by taking the absolute difference in profit between mailing and not mailing to an individual. The profit from mailing a catalog is proportional to the purchase amount (e.g., \$0 to \$6,247) minus the cost of the mailing of \$2.00. Not mailing to an individual results in zero profit. For example, misclassifying a would-be respondent would cost somewhere between \$1 and \$6,245 in profits. Misclassifying a non-respondent would cost \$0.68 in profits.

For both *MetaCost* and the direct classification method, this base rate of 2.5% was used. In addition, to employ the Heckman performance estimation procedure, the variables based on the lifetime average purchase amount and the most recent purchase amount (in addition to the class probability estimate) were provided as the independent variables to the Multiple Linear Regression (MLR) function.

Chapter 4

Experimental Environment

4.1 RapidMiner Community Edition v4.4

All of the experiments were conducted using the Community Edition version 4.4 of RapidMiner (www.rapidminer.com). RapidMiner is an open-source data mining platform that supports the creation and execution of complex nested data mining tasks via a graphical user interface (GUI). The underlying source code is written in Java. The Community Edition of RapidMiner comes with over 300 data mining tasks already implemented, and includes the entire suite of Weka [24] algorithms (also written in Java). RapidMiner supports complex routines via XML-based scripts that can be built and “stepped through” within the RapidMiner GUI. As an open source platform that is written in a highly portable language and supports script based complex routines, RapidMiner was an ideal choice for this research.

4.2 Implementation Choices and Coding

Due to the extensible nature of RapidMiner, each new classification method could be created either directly as an operator written in native Java source code using RapidMiner’s class library or as a RapidMiner process consisting of one or more existing operators implemented in RapidMiner. In order to increase the reliability of our experiments, we attempted to implement classification methods

as RapidMiner processes where possible. Extensive custom coding of new Java classes was necessary in order to ensure that the methods evaluated adhered to our intended algorithms. However, all new classes were incorporated into newly created RapidMiner processes which allowed for repeatability, runtime modification of parameters (e.g., the number of re-samples for MetaCost and costing), and leveraging RapidMiner’s support for logging experiment results.

Custom coding in Java was required in order to accommodate our intention to conduct example-based cost-sensitive experiments. To account for the variety of ways in which example based costs could be incorporated across different data sets, an abstract `ExampleBasedPerformanceCalculator` class was implemented along with a specific implementation of the class for each of the two data sets. Each data set specific `ExampleBasedPerformanceCalculator` implements a set of methods which return misclassification costs for an example labeled with its predicted and actual class membership. The actual class that is used for a specific experiment can be specified at run time by providing the class name as an input parameter to the RapidMiner process. Custom Java coding was also required in order to incorporate new techniques to improve the reliability of class membership probabilities from decision trees. The “static” cross validation data sets discussed earlier were maintained through the use of a constant `RandomSeed` within the RapidMiner processes.

Ultimately, we created six stand alone RapidMiner processes. One process constituted an experiment run for a single classification method and a single data set. Lesion studies to understand the impact of increasing the number of re-samples for MetaCost and costing were conducted by modifying a parameter of the appropriate RapidMiner process. From these test runs, we report the average performance for each algorithm, the standard deviation, and the area under ROC curve (AUC). Unlike algorithm performance and standard deviation, AUC is reported based solely on categorization accuracy (i.e., AUC does not incorporate misclassification costs). We report AUC as it provides additional insight into the overall quality of the algorithms as classifiers per ranking.

The underlying classification algorithm applied was the C4.5 decision tree

method. RapidMiner’s DecisionTree operator was used as the C4.5 decision tree method. Each experiment used an unpruned tree with a maximum depth of 4 levels. In practice, the tree depth for each experiment was exactly 4. Each node was split based on the information gain with no minimum gain required for a split. All experiments were run using 10 fold cross validation against a superset consisting of the combined training and testing datasets. The subsets were created based upon stratified sampling to promote consistent class base rate across the samples. Through the use of a consistent random seed variable, the subsets created were consistent across all experiments. Additional implementation details are specified in the following sub sections.

4.2.1 Implementing MetaCost

RapidMiner comes with two implementations of MetaCost. The first is written directly as a RapidMiner operation and the second is a “wrapped version” of the Weka implementation. Although the original description of Domingos’ MetaCost does not prohibit example-based misclassification costs, most implementations of MetaCost are written to support only class level misclassification costs. This was also the case with the RapidMiner and Weka implementations of MetaCost. For this research, the RapidMiner version of MetaCost served as the basis for the general approach and was modified to incorporate for example-based misclassification costs. Further, in order to evaluate differences in the core approaches of the three methods, we took steps to mitigate the issues introduced by MetaCost’s use of bagging to estimate class membership probabilities. First, we specified a classifier known to produce well calibrated probabilities as the inner learner. For the inner learner, we created a CalibratedDecisionTree learner that produced probabilities based on the average of the *smoothing* and *curtailment* methods described previously. Second, the class membership probabilities produced were the average of the CalibratedDecisionTree learners. Other than improving the implementation’s ability to produce well calibrated class membership probabilities and enabling example-based misclassification costs, the modifications should not affect the integrity of the approach as discussed previously. The experiments with Meta-

Cost were run using a bagged collection of models resulting from 25 re-samples of the training set. An interesting side note is that the MetaCost implementation included with RapidMiner was not written to follow Domingos' algorithm explicitly. Like Domingos' original specification, RapidMiner's MetaCost uses bagging based on an arbitrary number of re-samples using an arbitrary inner learner capable of producing class membership probabilities. However, unlike Domingos' specification, RapidMiner's MetaCost used the estimated class membership probabilities directly as part of the resulting classification model.

Figure 4.2.1 displays the RapidMiner process diagram used for MetaCost. The `ArffExampleSource` operators load the examples from the training and test data sets, respectively. The `ExampleSetMerge` operator merges the training and test data sets into a single data set. The `IdTagging` operator creates a unique identification number for each example. This enables the subsequent filtering and re-merging of examples. The `ChangeAttributeRole` specifies which attribute is to be treated as the cost for the example. The `XValidation` operator contains the specification for the cross validation. `MyMetaCostOperator` implements the MetaCost algorithm. The `CalibratedDecisionTree` operator implements the DecisionTree capable of producing well-calibrated probability estimates. The `OperatorChain` operator contains the functionality necessary to apply the learning model and determine performance. The `ModelApplier` operator simply applies the learning model created above to the test data set created by the `XValidation` operator. The `ThesisAlgorithmPerformanceOperator` calculates the resulting performance of the learning model. The `ProcessLog` operator tracks the performance of each individual iteration within the cross validation. The XML version of the RapidMiner Process, which specifies the values used as input parameters can be found in the Appendix.

4.2.2 Implementing Costing

Zadrozny's Costing algorithm or a similar method was not included with RapidMiner. This method was implemented largely from scratch based on the description in Zadrozny's original paper. To evaluate the Costing methodology, we

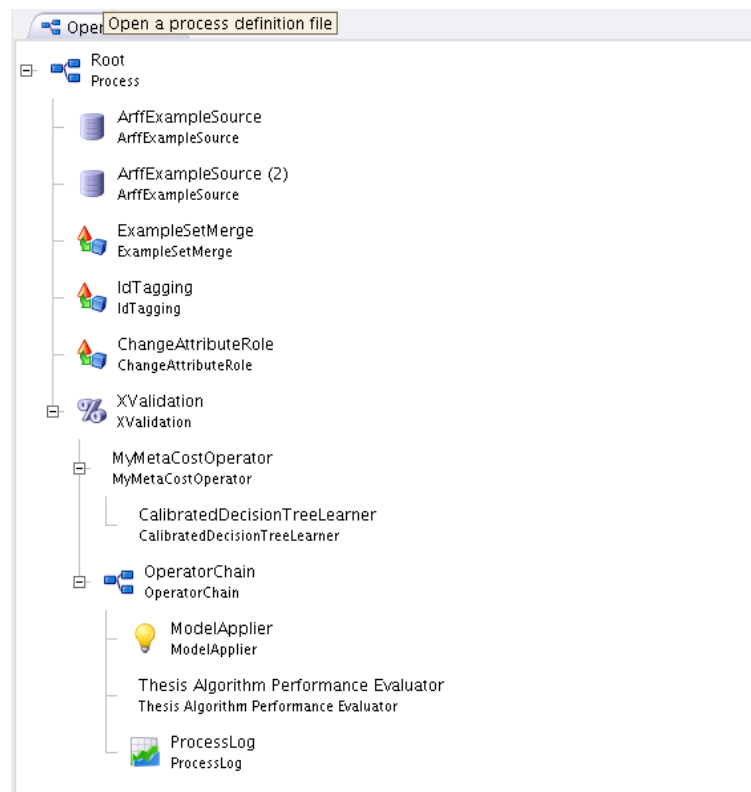


Figure 4.1: MetaCost RapidMiner Process Diagram

implemented the algorithm discussed earlier as a meta classifier using the RapidMiner tool set. Beyond the choice of the core classifier (C4.5), the costing algorithm requires two additional choices. The first choice is the specification for the constant Z used in the rejection sampling step. We used the recommended choice of $Z = \max_{(x,y,c) \in SC} c$, where c represents the potential profit. For the second choice, the number of re-samples, we chose 50 in order to stay consistent with our evaluation of the other methods. In accordance with the algorithm, the exact size of the sampled data set varied according to the rejection sampling results during each iteration.

One important and notable difference between the costing process and the other implemented algorithms is the usage of the DecisionTree operator instead of the CalibratedDecisionTreeLearner operator as the “inner learner”. We made this implementation choice for two reasons. First, the costing approach is the only algorithm evaluated that does not explicitly utilize class membership probabilities. Secondly, since the costing approach sends much smaller sub-samples of the data set to its inner learner, the use of *curtailment* by the CalibratedDecisionTreeLearner could render most decision tree models into decision stumps. Figure 4.2.2 displays the RapidMiner process diagram used for the costing algorithm. The XML version of the RapidMiner Process, which specifies the values used as input parameters can be found in the Appendix.

4.2.3 Implementing Direct Cost-Sensitive Classification

Zadrozny and Elkan’s [25] direct cost-sensitive classification method was also implemented from scratch based on the description provided with the original research. Although Zadrozny discusses three techniques to improve the estimation of class membership probabilities, only the *smoothing* technique and the *curtailment* technique were implemented. The value of m used with the *smoothing* technique was set at 200 approximating the heuristic that $bm = 10$ where b is the underlying base rate of the more rare class. The base rate of the more rare class (i.e., positive responders) was similar across both data sets. So the value of 200 for m was used for all experiments. Zadrozny references experiments that

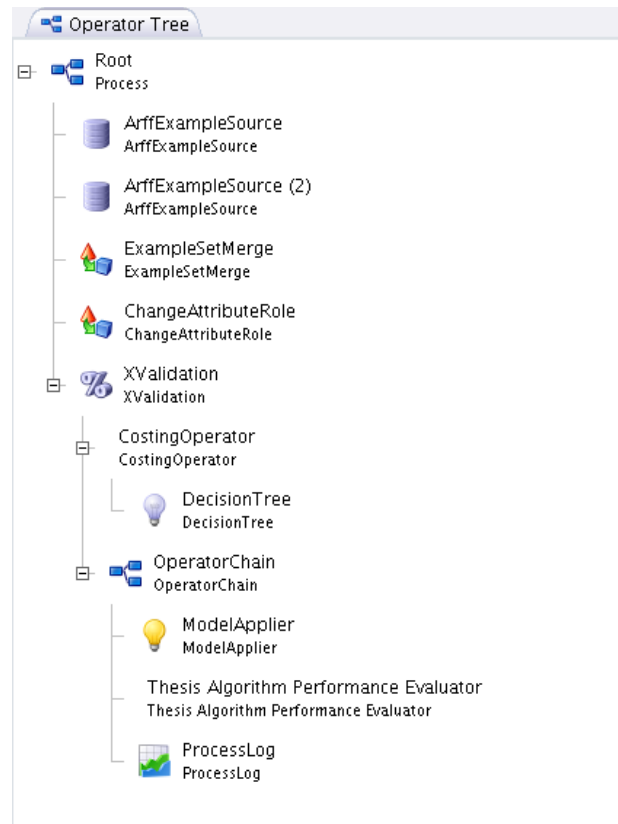


Figure 4.2: Costing RapidMiner Process Diagram

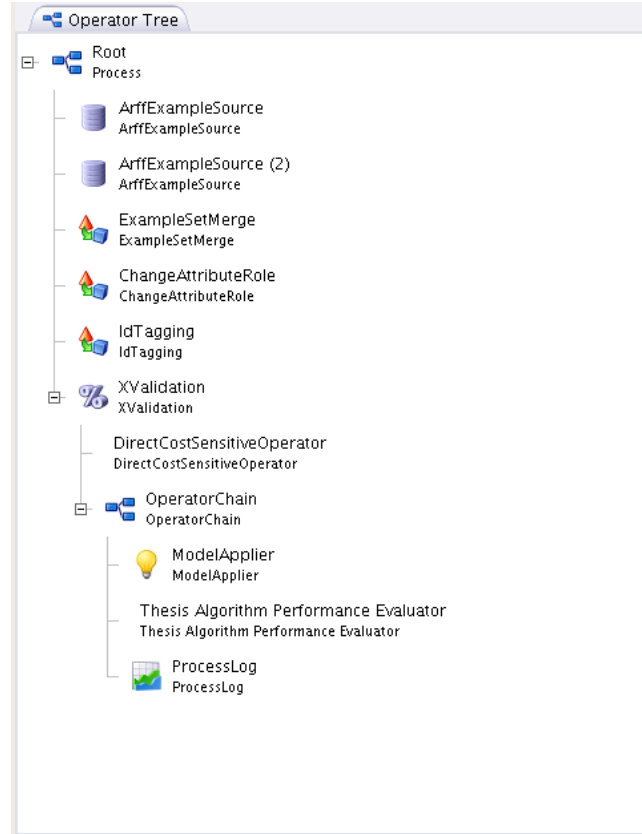


Figure 4.3: Direct Cost RapidMiner Process Diagram

demonstrate that the technique is qualitatively similar across a wide range of values for m . Therefore the precise value of m is unimportant. Class membership probabilities were based on the average of estimates using those two approaches. As discussed in the original work, the Heckman performance estimation procedure was implemented to account for non-responders when estimating the conditional risk. Figure 4.2.3 displays the RapidMiner process diagram used for the direct cost-sensitive classification algorithm. The XML version of the RapidMiner Process, which specifies the values used as input parameters can be found in the Appendix.

Table 4.1: New Rapidminer Operators. These new operators were created for the thesis and are included in the plug-in.

Operator	Description
CalibratedDecistionTreeLearner	DecisionTree using average of <i>smoothing</i> and <i>curtailment</i> for class probabilities
AlgorithmPerformanceEvaluator	Evaluates example-based operators
MetaCost	MetaCost using example-based misclassification costs
Costing	Implements the costing algorithm
DirectCost	Implements the direct cost-sensitive algorithm

4.3 Reusable Elements of Implemented Code

As a result of this work, a number of new classes were created which may have utility for other RapidMiner users. Each of the methods evaluated has been implemented in a class. In addition, the utility for example based cost-sensitive classification, using the abstract ExampleBasedPerformanceCalculator construct, would enable others to begin evaluating algorithms with costs that vary by example. All classes and their associated processes were packed as a RapidMiner plug-in and will be submitted to the RapidMiner website for inclusion in their plug-in library. A RapidMiner plug-in may be incorporated easily into any existing version of RapidMiner v4.4 or later by copying the associated JAR file into the appropriate plug-in directory. Table 4.1 summarizes the new RapidMiner **operators** created for this research that are included in the plug-in. Table 4.2 summarizes the new RapidMiner **processes** created for this research that are included in the plug-in.

Table 4.2: New Rapidminer Operators. These new processes were created for the thesis and are included in the plug-in.

Process	Description
MetaCostEvaluationProcessKDD98	MetaCost on KDD98 data set
MetaCostEvaluationProcessDMEF2	MetaCost on DMEF2 data set
CostingEvaluationProcessKDD98	Costing on KDD98 data set
CostingEvaluationProcessDMEF2	Costing on DMEF2 data set
DirectCostEvaluationProcessKDD98	DirectCost on KDD98 data set
DirectCostEvaluationProcessDMEF2	DirectCost on DMEF2 data set

Chapter 5

Results and Analysis

We present an empirical evaluation of the three general approaches to applying a cost-insensitive algorithm, all based on the C4.5 decision tree algorithm, to a cost-sensitive problem domain. Before commenting on the relative performance of each method against the test data sets, we discuss previously reported results of experiments using these methods against the same data sets. This information is helpful in order to understand how closely the algorithms implemented in this research reflect the performance of the original algorithms. We first discuss the results of experiments to validate the implementation of the algorithms and then discuss the results of the empirical comparison using 10 fold cross validation.

5.1 Validation Results

Table 5.1 and Table 5.2 show the experimental results of the reliability evaluation using the KDD'98 data set and DMEF2 data set, respectively. The results are based on fixed train-test splits used in prior reported research. The results from prior research are based on performance reported by Domingos [9] for MetaCost, Zadrozny et al. [26] for costing, and Zadrozny and Elkan [25] for the DirectCost method. For both MetaCost and the costing algorithms, the number of re-samples was set to 100. All achieved a dramatically improved performance relative to simply classifying all customers as *responders*, which would have resulted in a profit of \$10,560. Not to lose sight of the improvements brought on by these

Table 5.1: KDD-98 Validation Results. The results of the implemented algorithms closely approximate the results produced in prior research.

Algorithm	Correct Sends	Incorrect Sends	Total Sends	% Correct	Profit	Prior Results
MetaCost-100	3,030	66,555	69,585	4.35%	\$12,894	\$13,284
Costing-100	3,114	59,281	62,395	4.99%	\$14,393	\$14,935
Direct Cost	2,745	52,816	55,561	4.94%	\$14,541	\$14,879

Table 5.2: DMEF-2 Validation Results. The results of the costing algorithm implemented closely approximates the results produced in prior research. Comparisons for MetaCost and Direct Cost are not available.

Algorithm	Correct Sends	Incorrectly Sends	Total Sends	% Correct	Profit	Prior Results
MetaCost-100	978	33,085	34,043	2.81%	\$35,836	(none)
Costing-100	945	31,251	32,196	2.94%	\$35,977	\$36,992
Direct Cost	793	24,678	25,471	3.11%	\$34,585	(none)

techniques, it is interesting to note that all achieved their improved performance with a success rate very similar or less than the underlying proportion of positive examples in the test data set (5%). The implication is that all methods were simply better at identifying donors with more significant profit potential as opposed to simply improving the likelihood of identifying a donor.

We note that the costing and direct cost-sensitive decision making performed less well in this research than previously reported. In all three cases, some of the differences may be explained by differences in the settings for the core C4.5 algorithm. The C4.5 algorithm requires the user to specify criteria for the minimum information gained as well as a maximum tree depth. For the validation results, the minimum information gained was set to 0.0. For the validation experiments of all three algorithms, the maximum tree depth was set to five for the KDD-98 results. Note that this is different than the maximum tree depth of three used for the empirical comparison experiments. Memory limitations in the experimental environment prevented the use of a consistent maximum tree depth greater

Table 5.3: Probability Calibration Results. The mean squared error (MSE) is reported for each decision tree algorithm against the KDD98 and DMEF2 data sets. Against both test data sets, the CalibratedDecisionTree had a lower corresponding MSE than the DecisionTree algorithm. In addition, the absolute difference between the MSE when the algorithms are applied to the training data set and the MSE when the algorithms are applied to the test data set is smaller for the calibrated operator.

Method	KDD98		DMEF2	
	Training	Test	Training	Test
DecisionTree	0.04045	0.05607	0.02122	0.02684
CalibratedDecisionTree	0.04746	0.04775	0.02345	0.02399

than 10. As these parameters were not reported in prior research, some differences may be explained due to differences in these settings. For example, an experiment where the costing algorithm implemented in this research run at a maximum tree depth of 20 on the DMEF-2 data set produced a result of \$36,507.

The differences with the previously reported results may suggest that there is an unidentified flaw in the implementations written in this research. One final potential source of the difference is variations in the performance of the underlying base learner. For instance, while the performance of the costing algorithm using RapidMiner’s DecisionTree operator was \$14,740, the performance using the same settings with Weka’s J48 algorithm was \$14,508. Although the difference in this case is in favor of RapidMiner’s implementation, it is conceivable that the results could go another way with another data set.

To validate the CalibratedDecisionTreeLearner operator we evaluated it along with the DecisionTree operator upon which it was based. To compare the accuracy of the probability estimates, we report the mean squared error (MSE) of the generated class probability model when applied against the training and test data sets. The experiments were conducted against the standard training / test data sets available for the KDD98 data set and the training / test set used in prior research [26]. With both data sets, the true labels are known but not the underlying probabilities. We define the true probability to be 1 if the label is positive and 0 if the label is negative. Table 5.3 shows the results of the experiment to validate

the improved calibration capabilities of the implemented `CalibratedDecisionTree` operator.

5.2 Empirical Comparison

Table 5.4 and Table 5.5 show the experimental results of the evaluation using the KDD’98 data set and DMEF2 data set, respectively. The results are reported as the average profit “per fold” and thusly differ in magnitude from those reported earlier. The profit is determined by adding the amount that was donated (KDD98) or purchased (DMEF2) by each test example chosen for solicitation. We subtract a fixed mailing fee for each example chosen for solicitation. As both MetaCost and costing allow the user to specify an arbitrary number of re-samples upon which to base predictions, we report the results of each algorithm using an increasing number of re-samples. As each additional iteration consumed additional RAM, memory restrictions in the experimental environment limited the level to which we could increase the number of re-samples. Given the differences in the underlying approach, one striking result is the relative similarity of the results using 10 fold cross validation against both data sets. Against the KDD-98 data set, the costing algorithm and the Direct Cost algorithm appeared to perform somewhat better than the MetaCost approach. Against the DMEF-2 data set, the results were similar although the costing algorithm appeared to show an advantage at increased numbers of re-samples. MetaCost performed less well on the KDD’98 data set than either costing or direct cost-sensitive decision making. This is similar to Elkan’s finding in prior research. Also, similar to prior research (reported separately), costing and the direct cost-sensitive classification method performed similarly well against the KDD’98 data set.

Despite MetaCost’s relative poorer performance on the KDD’98 data set, the algorithm achieved the best results on the DMEF2 data set. Given the performance of \$27,584 that would have resulted from classifying all of the customers as *responders*, all three methods achieved dramatic improvements. Unlike the results for the KDD’98 data set, all three methods achieved their results by improving

on the percentage “hit rate” of identified responders relative to the test set as a whole.

The lesion study examining the role of increasing sub-sampling also yielded mixed results. Using the KDD-98 data set, Zadrozny showed a clear positive correlation to the number of rejection samples and performance of the algorithm against both the KDD’98 data set and the DMEF2 data set. In our experiment using the KDD-98 data set, neither MetaCost nor costing exhibited significant improved performance as the number of subsamples was increased. Unlike Zadrozny’s work, where the number of rejection samples was evaluated at up to 200 rejection samples, the maximum number used for the empirical comparison was 50. At the maximum number of re-samples tested, neither showed significant improvement. However, against the DMEF-2 data set, both algorithms continued to improve as the number of re-samples increased. Against the DMEF2 data set, the costing algorithm’s performance continued to improve with increasing re-samples and did not appear to plateau until it was run with 400 re-samples. At 200 re-samples, the costing algorithm demonstrated a nearly 20% improvement over its performance with only 10 re-samples.

A key confounding factor impacting the interpretation of the results is the standard error of the profit results. Much of this variability is likely due to underlying differences in the “profit potential” in each of the cross folds. As the cross folds were consistent across experimental runs, higher than average results across all algorithms for a single cross fold indicate that the underlying potential is different. For example, the average profit obtained across all three algorithms for the third fold was \$8,009 while average across the tenth fold was \$11,048. Two possible remedies to this factor exist that were not performed. The first is to modify the stratification performed for the 10 fold cross validation such that each set had a similar proportion of positive examples AND a similar level of profit potential. A second remedy is to reduce the number of cross folds which would increase the number of observations and reduce the size (and potential variance) of the profit potential of each test set.

Finally, one potential source of the difference between the performance of

the costing algorithm and the performance of MetaCost and the direct methods is the usage of RapidMiner’s native DecisionTree instead of the newly implemented CalibratedDecisionTreeLearner. When evaluated using the CalibratedDecisionTreeLearner, the performance of the costing algorithm fell substantially to \$5,000. As mentioned earlier, the explanation for this could be the material impact of *curtailment* on the decision tree models representing much smaller data sets. However, it should be noted that both MetaCost and the direct method also performed less well when using RapidMiner’s native DecisionTree instead of the CalibratedDecisionTreeLearner. So, in effect, we are reporting the results for all algorithms using the best available decision tree learner for that algorithm.

Table 5.4: KDD-98 Results. The performance is similar across all algorithms. Increasing the number of re-samples for MetaCost and costing does not have a significant impact.

Algorithm	Correct Sends	Incorrect Sends	% Correct	AUC	Profit
MetaCost-10	704±43	14890±529	4.51%	0.29±0.10	\$2,492.97±365
MetaCost-25	709±28	14987±272	4.52%	0.25±0.09	\$2,493.54±387
MetaCost-50	n/a	n/a	n/a	n/a	n/a
Costing-10	656±61	13132±1015	4.76%	0.40±0.01	\$2,663.12±340
Costing-25	669±61	13598±724	4.69%	0.43±0.01	\$2,638.19±355
Costing-50	699±57	14081±961	4.73%	0.44±0.01	\$2,670.26±420
Direct Cost	605±14	12082±264	4.77%	0.52±0.01	\$2,659.58±437

Table 5.5: DMEF-22 Results. The performance is similar across all algorithms. Costing appears to benefit from increasing the number of re-samples suggesting that it may outperform other methods.

Algorithm	Correct Sends	Incorrect Sends	% Correct	AUC	Profit
MetaCost-10	196±29	6870±1444	2.77%	0.35±0.07	\$6,762.50±2434
MetaCost-25	205±15	7286±895	2.73%	0.37±0.09	\$6,642.90±2124
MetaCost-50	197±17	6780±909	2.83%	0.39±0.07	\$6,932.30±2096
Costing-10	155±36	4912±1626	3.06%	0.53±0.03	\$6,374.70±1795
Costing-25	182±23	6014±1389	2.94%	0.58±0.02	\$6,896.50±2409
Costing-50	167±16	5144±855	3.14%	0.60±0.02	\$7,265.50±2020
Costing-100	169±14	5076±557	3.18%	0.62±0.01	\$7,378.50±2340
Costing-200	169±6	5278±198	3.17%	0.62±0.01	\$7,635.60±2334
Costing-400	172±9	5337±232	3.13%	0.63±0.01	\$7,395.40±2231
Direct Cost	158±13	4995±537	3.07%	0.42±0.03	\$6,731.30±2173

Chapter 6

Limitations of Work

While we report the results of experiments and their standard errors from the cross validations, we are unable to draw a statistical conclusion as to the superiority of one algorithm over another. Some of the difficulties of drawing a conclusion stem from the relatively large standard deviations discussed in the results section. However, we point out a limitation of the research in the form of the challenge when making statistical comparisons of classifiers over multiple data sets. While numerous methods for statistical comparison of classifier methods on a single data set have been proposed, there is no established procedure for the statistical comparison of multiple classifiers on multiple data sets [7]. Even for comparing multiple classifiers on a single data set, Dietterich [8] warns against t-test as statistically unsafe after repetitive random sampling and cross validation after examining five statistical tests. Demsar recommends the Wilcoxon Signed-Ranks test, a non-parametric alternative to the paired t-test, for the comparison of two classifiers across data sets. For the comparison of multiple models numerous papers use the Friedman test, which is the non-parametric equivalent of the repeated-measures ANOVA. The empirical comparison done in this research could be improved by using the Friedman test, which would require a minimum of five data sets in order to draw a statistical distinction between algorithms.

Chapter 7

Applications of Thesis and Conclusion

The work done with this thesis could lay the groundwork for a future comparative evaluation of cost-sensitive problem transformation algorithms. Using a standard platform and publicly available data sets will enable more robust determinations of the true impact of future approaches over existing methodologies. For researchers considering any of the approaches evaluated in this thesis, this work should serve as some validation that the differences across these methods may be relatively insignificant compared to the choice of the underlying base learner.

Specific contributions of this paper include:

1. The thesis presents the approaches of three techniques to transform a cost-insensitive problem to the cost-sensitive problem domain.
2. The thesis identifies an appropriate platform to evaluate different approaches and improved upon available open-source code to accommodate problem domains where the misclassification costs are example dependent.
3. The thesis compares and contrasts the results of the three techniques on two real-world data sets that are publicly available and complex

Despite the reported differences in performance on the test data sets, there are not enough data to support that any approach performs significantly better

than another. The available data are not enough to reject the null hypothesis that all three methods are non inferior to one another. The results do confirm that general methods of transforming cost-sensitive learning problems into cost-insensitive problems can outperform their cost-insensitive contemporaries. The fact that the relative performance can differ across data sets demonstrate the importance of conducting empirical evaluations against multiple data sets in order to validate true performance improvements.

7.1 Results

The results and analysis was discussed in a prior section of this paper. For the KDD'98 and DMEF2 data sets, all methodologies performed better than the alternative of classifying all examples or no examples as a responder. This validates the cost-sensitive improvements intended by the original researchers. While the costing algorithm achieved the best results against both data sets, the variance in the performance results prevents rejecting the null hypothesis that no algorithm is non-inferior to another.

7.2 Problems Left Unsolved

This research leaves open many possible avenues for further study. The results are derived from only two data sets. In addition to both coming from the marketing problem domain, the database share other similarities such as the very low response rates. In addition, the costs for misclassification are dramatically asymmetric in that it is never very costly to mis-classify a non-responder while it is occasionally extremely costly to mis-classify a responder. While there are numerous real-world problems that also exhibit this behavior, there are others (e.g., loan approval decisions) that do not. Therefore further research involving more and different types of cost-sensitive labeling problems would be beneficial to further validate the results of this study. This research did not include an algorithm fitting the risk-based approach to cost-sensitization. Further study could

also include algorithms from this family.

As a wealth of work has already been completed on making individual classification algorithms cost-sensitive, the primary utility of a general method is that it perform reasonably well when used on multiple classification algorithms (and KDD problem domains). The work completed for this thesis compared generalizable approaches to change an arbitrary classification algorithm to become cost-sensitive. To further substantiate the value of the studied approaches, additional studies should be performed applying them to additional classification algorithms.

In addition, only two of the cost-sensitization methods (MetaCost and the direct cost-sensitive method) pertain to multi-class classification problems. Further work may identify and evaluate additional generalized cost-sensitive methods that are well-suited for a multi-class classification problem domains.

Bibliography

- [1] D.J. Newman A. Asuncion. UCI machine learning repository, 2009.
- [2] S Anifantis. The dmef data set library. Direct Marketing Association website: <http://www.thedma.org/dmef/dmefdataset.shtml>, 2002.
- [3] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants, 1998.
- [4] James M. Bower and David Beeman. *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SIMulation System*. Springer-Verlag, New York, NY, 1998.
- [5] Leo Breiman and Leo Breiman. Bagging predictors. In *Machine Learning*, pages 123–140, 1996.
- [6] James Cussens. Bayes and pseudo-bayes estimates of conditional probabilities and their reliability. In *Proceedings of the European Conference on Machine Learning*, pages 136–152. Springer Verlag, 1993.
- [7] Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.
- [8] Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10:1895–1923, 1998.
- [9] Pedro Domingos. Metacost: A general method for making classifiers cost-sensitive. In *In Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 155–164. ACM Press, 1999.
- [10] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition, November 2000.
- [11] Charles Elkan. Foundations of cost sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI’01)*, pages 973–978, 2001.

- [12] Wei Fan and Salvatore J. Stolfo. Adacost: misclassification cost-sensitive boosting. In *In Proceedings of the 16th International Conference on Machine Learning*, pages 97–105. Morgan Kaufmann, 1999.
- [13] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer Systems and Sciences*, 55(1):119–139, 1997.
- [14] James Heckman. Sample selection bias as a specification error. In *Econometrica*, pages 153–161. ACM Press, 1979.
- [15] Michael Kearns, Robert E. Schapire, Linda M. Sellie, and Lisa Hellerstein. Toward efficient agnostic learning. In *Machine Learning*, pages 341–352. ACM Press, 1994.
- [16] Pat Langley. Crafting papers on machine learning. In *In Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pages 1207–1212, 2000.
- [17] Dragos D. Margineantu. On class-probability estimates and cost-sensitive evaluation of classifiers. In *In Workshop on Cost-Sensitive Learning at the 17th International Conference on Machine Learning (WCSL at ICML2000)*, 2000.
- [18] Dragos D. Margineantu. Class probability estimation and cost-sensitive classification decisions. In *In Proceedings of the 13th European Conference on Machine Learning*, pages 270–281, 2002.
- [19] Hamed Masnadi-Shirazi and Nuno Vasconcelos. Asymmetric boosting. In *In Proceedings of the 24th International Conference on Machine Learning*, pages 609–619, 2007.
- [20] Foster Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. In *In Proceedings of the 15th International Conference on Machine Learning*, pages 445–453. Morgan Kaufmann, 1997.
- [21] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Machine Learning*, pages 80–91, 1999.
- [22] Leslie G. Valiant. A theory of the learnable. In *Annual ACM Symposium on Theory of Computing. Archive of Proceedings of the sixteenth annual ACM symposium on Theory*, pages 436–445, 1984.
- [23] John von Neumann. *Monte Carlo Method*, chapter Various Techniques Used in Connection With Random Digits, pages 36–38. 12. National Bureau of Standards Applied Mathematics, U.S. Government Printing Office, 1951.

- [24] Ian H. Witten and Frank Eibe. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufman Publishers, San Francisco, 2000.
- [25] Bianca Zadrozny and Charles 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.
- [26] Bianca Zadrozny, John Langford, and Naoki Abe. Cost-sensitive learning by cost-proportionate example weighting. In *Proceedings of the Third IEEE International Conference on Data Mining*, pages 435–443, 2003.

Appendix A

RapidMiner XML Implementations

```

<operator name="Root" class="Process" expanded="yes">
  <parameter key="logverbosity" value="all"/>
  <operator name="ArffExampleSource" class="ArffExampleSource">
    <parameter key="data_file" value="kdd98.train.r.arff"/>
    <parameter key="label_attribute" value="responded"/>
  </operator>
  <operator name="ArffExampleSource (2)" class="ArffExampleSource">
    <parameter key="data_file" value="kdd98.test.r.arff"/>
    <parameter key="label_attribute" value="responded"/>
  </operator>
  <operator name="ExampleSetMerge" class="ExampleSetMerge">
  </operator>
  <operator name="IdTagging" class="IdTagging">
  </operator>
  <operator name="ChangeAttributeRole" class="ChangeAttributeRole">
    <parameter key="name" value="cost"/>
    <parameter key="target_role" value="cost"/>
  </operator>
  <operator name="XValidation" class="XValidation" expanded="yes">
    <operator name="MyMetaCostOperator" class="MyMetaCostOperator" expanded="yes">
      <parameter key="use_subset_for_training" value="0.05"/>
      <parameter key="example_performance_calculator"
        value="KDD98ExamplePerformanceCalculator"/>
      <operator name="CalibratedDecisionTreeLearner"
        class="CalibratedDecisionTreeLearner">
        <parameter key="criterion" value="information_gain"/>
        <parameter key="minimal_gain" value="0.0"/>
        <parameter key="maximal_depth" value="3"/>
        <parameter key="no_pre_pruning" value="true"/>
      </operator>
    </operator>
    <operator name="OperatorChain" class="OperatorChain" expanded="yes">
      <operator name="ModelApplier" class="ModelApplier">
        <list key="application_parameters">
        </list>
      </operator>
      <operator name="Thesis Algorithm Performance Evaluator"
        class="Thesis Algorithm Performance Evaluator">
        <list key="additional_performance_criteria">
          <parameter key="KDD98ExamplePerformanceCalculator"
            value="1"/>
        </list>
      </operator>
      <operator name="ProcessLog" class="ProcessLog">
        <parameter key="filename" value="metaCost.KDD98.log"/>
        <list key="log">
          <parameter key="Iteration" value="operator.XValidation.value.iteration"/>
          <parameter key="Performance"
            value="...Thesis Algorithm Performance Evaluator.value.user1"/>
          <parameter key="AUC"
            value="...Thesis Algorithm Performance Evaluator.value.performance"/>
          <parameter key="Sent Correctly"
            value="...Thesis Algorithm Performance Evaluator.value.true_positive"/>
          <parameter key="Sent Incorrectly"
            value="...Thesis Algorithm Performance Evaluator.value.false_positive"/>
          <parameter key="Unsent Correctly"
            value="...Thesis Algorithm Performance Evaluator.value.true_negative"/>
          <parameter key="Unsent Incorrectly"
            value="...Thesis Algorithm Performance Evaluator.value.false_negative"/>
        </list>
      </operator>
    </operator>
  </operator>
</operator>

```

Figure A.1: RapidMiner Process for MetaCost Experiment (KDD98 example)

```

<operator name="Root" class="Process" expanded="yes">
  <operator name="ArffExampleSource" class="ArffExampleSource">
    <parameter key="data_file" value=".../kdd98.train.r.arff"/>
    <parameter key="label_attribute" value="responded"/>
  </operator>
  <operator name="ArffExampleSource (2)" class="ArffExampleSource">
    <parameter key="data_file" value=".../kdd98.test.r.arff"/>
    <parameter key="label_attribute" value="responded"/>
  </operator>
  <operator name="ExampleSetMerge" class="ExampleSetMerge">
  </operator>
  <operator name="ChangeAttributeRole" class="ChangeAttributeRole">
    <parameter key="name" value="cost"/>
    <parameter key="target_role" value="cost"/>
  </operator>
  <operator name="XValidation" class="XValidation" expanded="yes">
    <operator name="CostingOperator" class="CostingOperator" expanded="yes">
      <parameter key="keep_example_set" value="true"/>
      <parameter key="iterations" value="25"/>
      <parameter key="example_performance_calculator"
        value="...KDD98ExamplePerformanceCalculator"/>
      <operator name="DecisionTree" class="DecisionTree">
        <parameter key="criterion" value="information_gain"/>
        <parameter key="minimal_leaf_size" value="1"/>
        <parameter key="minimal_gain" value="0.0"/>
        <parameter key="maximal_depth" value="3"/>
        <parameter key="no_pre_pruning" value="true"/>
        <parameter key="no_pruning" value="true"/>
      </operator>
    </operator>
    <operator name="OperatorChain" class="OperatorChain" expanded="yes">
      <operator name="ModelApplier" class="ModelApplier">
        <list key="application_parameters">
        </list>
      </operator>
      <operator name="Thesis Algorithm Performance Evaluator"
        class="Thesis Algorithm Performance Evaluator">
        <list key="additional_performance_criteria">
          <parameter key="...KDD98ExamplePerformanceCalculator" value="1"/>
        </list>
      </operator>
      <operator name="ProcessLog" class="ProcessLog">
        <parameter key="filename" value=".../costing.KDD98.log"/>
        <list key="log">
          <parameter key="Iteration" value="operator.XValidation.value.iteration"/>
          <parameter key="Performance"
            value="...Thesis Algorithm Performance Evaluator.value.user1"/>
          <parameter key="AUC"
            value="...Thesis Algorithm Performance Evaluator.value.AUC"/>
          <parameter key="Sent Correctly"
            value="...Thesis Algorithm Performance Evaluator.value.true_positive"/>
          <parameter key="Sent Incorrectly"
            value="...Thesis Algorithm Performance Evaluator.value.false_positive"/>
          <parameter key="Unsent Correctly"
            value="...Thesis Algorithm Performance Evaluator.value.true_negative"/>
          <parameter key="Unsent Incorrectly"
            value="...Thesis Algorithm Performance Evaluator.value.false_negative"/>
        </list>
      </operator>
    </operator>
  </operator>
</operator>

```

Figure A.2: RapidMiner Process for Costing Experiment (KDD98 example)

```

<operator name="Root" class="Process" expanded="yes">
  <operator name="ArffExampleSource" class="ArffExampleSource">
    <parameter key="data_file" value=".../kdd98.train.r.arff"/>
    <parameter key="label_attribute" value="responded"/>
  </operator>
  <operator name="ArffExampleSource (2)" class="ArffExampleSource">
    <parameter key="data_file" value=".../KDD98/kdd98.test.r.arff"/>
    <parameter key="label_attribute" value="responded"/>
  </operator>
  <operator name="ExampleSetMerge" class="ExampleSetMerge">
  </operator>
  <operator name="ChangeAttributeRole" class="ChangeAttributeRole">
    <parameter key="name" value="cost"/>
    <parameter key="target_role" value="cost"/>
  </operator>
  <operator name="IdTagging" class="IdTagging">
  </operator>
  <operator name="XValidation" class="XValidation" expanded="yes">
    <operator name="DirectCostOperator" class="DirectCostOperator" expanded="yes">
      <parameter key="example_performance_calculator"
        value="...KDD98ExamplePerformanceCalculator"/>
      <operator name="CalibratedDecisionTreeLearner"
        class="CalibratedDecisionTreeLearner">
        <parameter key="criterion" value="information_gain"/>
        <parameter key="minimal_gain" value="0.0"/>
        <parameter key="maximal_depth" value="3"/>
        <parameter key="no_pre_pruning" value="true"/>
      </operator>
    </operator>
    <operator name="OperatorChain" class="OperatorChain" expanded="yes">
      <operator name="ModelApplier" class="ModelApplier">
        <list key="application_parameters">
        </list>
      </operator>
      <operator name="Thesis Algorithm Performance Evaluator"
        class="Thesis Algorithm Performance Evaluator">
        <list key="additional_performance_criteria">
          <parameter key="...KDD98ExamplePerformanceCalculator" value="1"/>
        </list>
      </operator>
    <operator name="ProcessLog" class="ProcessLog">
      <parameter key="filename"
        value=".../directCost.KDD98.log"/>
      <list key="log">
        <parameter key="Iteration"
          value="...XValidation.value.iteration"/>
        <parameter key="Performance"
          value="...Thesis Algorithm Performance Evaluator.value.user1"/>
        <parameter key="AUC"
          value="...Thesis Algorithm Performance Evaluator.value.AUC"/>
        <parameter key="Sent Correctly"
          value="...Thesis Algorithm Performance Evaluator.value.true_positive"/>
        <parameter key="Sent Incorrectly"
          value="...Thesis Algorithm Performance Evaluator.value.false_positive"/>
        <parameter key="Unsent Correctly"
          value="...Thesis Algorithm Performance Evaluator.value.true_negative"/>
        <parameter key="Unsent Incorrectly"
          value="...Thesis Algorithm Performance Evaluator.value.false_negative"/>
      </list>
    </operator>
  </operator>
</operator>

```

Figure A.3: Rapid Miner Process for Direct Cost Experiment (KDD98 example)