Efficient Techniques for Cost-Sensitive Learning with Multiple Cost Considerations

By

Tao Wang

Submitted in fulfilment of the requirement for the degree of

Doctor of Philosophy

University of Technology, Sydney

April 2013

CERTIFICATE OF AUTHORSHIP/ORIGINALITY

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Candidate

. . . .

To my wife Yanhui and Our Children,

Alice and Jessica

Abstract

Cost-sensitive learning is one of the active research topics in data mining and machine learning, designed for dealing with the non-uniform cost of misclassification errors. In the last ten to fifteen years, diverse learning methods and techniques were proposed to minimize the total cost of misclassification, test and other types. This thesis studies the up-to-date prevailing cost-sensitive learning methods and techniques, and proposes some new and efficient cost-sensitive learning methods and techniques in the following three areas:

First, we focus on the data over-fitting issue. In an applied context of cost-sensitive learning, many existing data mining algorithms can generate good results on training data but normally do not produce an optimal model when applied to unseen data in real world applications. We deal with this issue by developing three simple and efficient strategies - feature selection, smoothing and threshold pruning to overcome data over-fitting in cost-sensitive learning. This work sets up a solid foundation for our further research and analysis in this thesis in the other areas of cost-sensitive learning.

Second, we design and develop an innovative and practical objective-resource cost-sensitive learning framework for addressing a real world issue where multiple cost units are involved. A lazy cost-sensitive decision tree is built to minimize the objective cost subjecting to given budgets of other resource costs.

Finally, we study semi-supervised learning approach in the context of cost-sensitive learning. Two new classification algorithms are proposed to learn cost-sensitive classifier from training datasets with a small amount of labelled data and plenty unlabelled data. We also analyse the impact of the different input parameters to the performance of our new algorithms.

Acknowledgements

I am indebted to many people for helping me and guiding me, to complete this thesis.

First and foremost is my supervisor, Prof. Chengqi Zhang. Without his support and encouragement, I might not have the courage to even start my PhD study. Over the years, his wisdom, dedication and leadership influenced me deeply, not only on my PhD research, but also my attitude towards study, work and other aspects of my life.

I would like to thank to my co-supervisor, Prof. Shichao Zhang. I cannot remember how many times he stayed very late helping me improve my papers, again and again, until they were good enough. As a father with two young daughters, studying PhD part time is always difficult and challenging. Shichao fully understands my situation. He always guided me, supported me, and was so patient with me, especially when the time my progress was slow. Without his help and support, I cannot imagine that I could've gone through past six long years to complete this thesis!

I give warm thank to my colleague and friend, Dr. Zhexning Qin, for working together on most of my papers and sharing his ideas. His insightful comments had been very influential in many occasions for my thinking.

I am grateful to Kamal Nigam for providing the source code of his semi-supervised EM algorithm. His work of mining on unlabelled data is the foundation of my research on semi-supervised cost-sensitive learning, and providing the source code of his original algorithm saved me heaps of time.

I won't be here today without the love and support of my mother and my sister, Wei. They were my first teachers and they helped and inspired me dearly from my early childhood to most period of my adult life.

Last and most importantly, I want to thank my wife, Yanhui, for her enduring patience and understanding. She provided me with the faith and confidence to go through the past six long years of PhD study. She was always there when I needed a shoulder to lean on.

The text of Chapter 3, in part, is a reprint of the material as it appears in the "Handling over-fitting in test cost-sensitive decision tree learning by feature selection,

smoothing and pruning. *Journal of Systems and Software*". The thesis author was the primary author, and the co-authors listed in this publication directed and supervised the research which forms the basis for the Chapter.

The text of Chapter 6, in part, is a reprint of the material as it appears in the "Cost-sensitive classification with inadequate labelled data. *Information Systems*". The thesis author was the primary author, and the co-authors listed in this publication directed and supervised the research which forms the basis for the Chapter.

Table of Contents

List of Figures

Chapter 1 Introduction

1.1 Motivation and Objectives

Life is about making decisions. Different decisions have different consequences. When we make the decision very often we know the possible outcomes. Our aim is to maximize the potential benefits and minimize the losses. For example, we choose to buy insurance policies because we are willing to pay the cost of the insurance instead of having the risk of losing the entire property. Different insurance companies have different types of insurance products. Their costs are different and their protection levels are different too. Before we decide what insurance policies we are going to buy, we also consider the probabilities of losing our property. How much we would like to pay for the insurance policies and how much benefit we can get?

In order to make our decisions as rational as possible, we have to answer these questions more accurately. Today, with the rapid development of information technology and computer systems, on one hand, we have a large amount of data (labelled or unlabelled) available to help us make better decisions. On the other hand, we are able to construct and evaluate systems that learn from these historical data to make the decisions that not only minimize the expected number of errors but also minimize the total cost associated with those decisions.

There are a variety of reasons for studying cost and its impact to our decisions, and many cost-sensitive learning methods and algorithms were developed in the last 10 to 15 years. The goal of this thesis is to present a set of new and efficient techniques to further improve the decision making process and minimize the total cost involved in cost-sensitive classification. More specifically, below are the three main objectives of this thesis:

- **Objective 1:** Research the causes of data over-fitting issue in cost-sensitive learning, develop efficient strategies to reduce the impact of data over-fitting in cost-sensitive learning algorithms, finally make these algorithms more stable and improve their overall performance.

- **Objective 2:** Propose an innovative Objective-Resource framework for resolving a real world issue in cost-sensitive learning where multiple costs are involved. These costs are often cannot be measured by the same cost unit. Our new framework minimizes the misclassification cost (referred to objective cost) subjecting to given budgets of other resource costs.

- **Objective 3:** Propose new semi-supervised classification strategies to learn cost-sensitive classifier from training datasets with both labelled and unlabelled data. This addresses many real world cost-sensitive classification issues in which there are few labelled data and plentiful unlabelled data.

1.2 Thesis Outline and Contributions

The remainder of this dissertation is organized as follows:

First, in Chapter 2, we review the previous cost-sensitive learning research which is related to the work in this thesis. Chapter 3 proposes several efficient techniques such as smoothing, feature selection and threshold pruning which address a common but very challenging issue in cost-sensitive classification, data over-fitting. In Chapter 4, we modify the popular K-Nearest Neighbour (KNN) algorithm to make it cost sensitive. Two new cost-sensitive KNN algorithms and several additional methods are developed to minimize the misclassification cost. Chapter 5 proposes an objective-resource framework to overcome the difficulty of handling multiple cost units in cost-sensitive learning, a novel multiple-unit lazy Cost-sensitive decision tree algorithm is developed to learn from real world medical data under the proposed new cost-sensitive learning framework. In Chapter 6, we attempt to utilize unlabelled training data to build cost-sensitive classifiers, this addresses a real world issue in which labelled data is very difficult, time consuming and expensive to obtain, but unlabelled data is often freely available. At the end, Chapter 7 concludes the research in this thesis and points out our future work and directions.

1.3 My Related Publications

This thesis is supported by the following publications:

- **Refereed Journal Papers**

1. Wang, T., Qin, Z., Jin, Z. and Zhang, S. (2009) Handling over-fitting in test cost-sensitive decision tree learning by feature selection, smoothing and pruning. *Journal of Systems and Software.* Vol. 83, pp. 1137-1147.
2. Wang, T., Qin, Z., Zhang, S. and Zhang, C. (2011) Cost-sensitive classification with inadequate labelled data. *Information Systems*. Vol. 37, pp. 508-516.

- **Refereed Conference Papers**

3. Qin, Z., Zhang, S., Liu, L. and Wang, T. (2008): Cost-sensitive Semi-supervised Classification using CS-EM. *IEEE 8th International Conference on Computer and Information Technology*, 8 - 11 July, 2008: 131-136.
4. Qin, Z., Wang, T. and Zhang, S. (2010) Incorporating medical history to cost sensitive classification with lazy learning strategy. *Proceedings of the International Conference on Progress in Informatics and Computing conference.* IEEE. Vol. 1, pp. 19-23.
5. Qin, Z., Zhang, C., Wang, T. and Zhang, S. (2011) Cost sensitive classification in data mining. *Advanced Data Mining and Applications*, pp. 1-11.

Chapter 2 Background and Related Work

Supervised learning is an important subarea in data mining and machine learning. In supervised learning, the examples provided to the learning systems already have class labels. The systems need learn to predict class labels for unseen examples based on the existing labelled examples. When the set of possible predictions is discrete, the learning task is called *classification*. In traditional classification systems, the task is to build a classifier aimed at improving predication accuracy. This implicitly assumes that all classification errors involve the same cost.

However, in most data mining and machine learning applications, different misclassification errors often involve different costs. Traditional data mining methods that aim at minimizing error rate will perform poorly in these areas, as they assume equal misclassification cost and relatively balanced class distributions. This leads to the development of domain-driven learning techniques, referred to *cost-sensitive learning* (Turney 1995), for addressing classification problems with non-uniform costs.

2.1 Cost-sensitive Learning

Cost-sensitive learning is an extension of traditional non-cost-sensitive data mining. It is an important research area with many real world applications. For example, in medical diagnosis domain, diseases are not only very expensive but also rare; for a bank, an error of approving a home loan to a bad customer is more costly than an error of rejecting a home loan to a good customer. Given a naturally much skewed class distribution and costly faulty predictions for the rare class, an error based classifier may very likely ends up building a useless model. Cost-sensitive learning is an advanced form of data mining that satisfies these special needs. Research in cost-sensitive learning is still in an early stage and there are different methods to it. Most cost-sensitive learning methods are developed based on the existing non-cost-sensitive data mining methods. To make an error-based classifier cost-sensitive, a common method is to introduce biases into an error based classification system in three ways: 1) by changing the class distribution of the training data, 2) by modifying the learning algorithms, 3) and by taking the boosting approach (Li et al. 2005). An alternative method is called direct cost-sensitive learning which uses the conditional probability estimates provided by error based classifiers to directly compute the optimal class label for each test example using cost function (Zadrozny and Elkan 2001).

Other cost-sensitive learning methods such as cost-sensitive specification and cost-sensitive genetic programming are also included in this Chapter. The details of these methods are discussed in section 2.3.

2.2 Settings and Definitions of Cost-sensitive Learning Problem

- **Types of Cost**

According to Turney (2000)'s paper, there are nine major types of cost involved in cost-sensitive learning. Some of these costs are listed below:

Misclassification Costs: In data mining, different types of misclassification errors usually involve different costs. These costs are the most important costs in cost-sensitive learning. They can either be stationary (represented as a cost matrix) or example dependent.

Test Costs: In some domains, such as medical diagnosis, many tests involve costs. Some tests are more costly than other. If the misclassification costs surpass the test costs greatly, then all tests should be performed. If the test costs are much more than the misclassification costs, then it is rationale not to do any tests.

Teacher Costs: Sometimes it is expensive to decide the correct class label of an example. In this situation, a learning algorithm should consider the cost of teaching, and one way of doing it is actively selecting instances for the teacher, i.e., active learning.

Computation Costs: Size and structural complexity, time and space requirements of a classification algorithm both in training and test phases are considered under this category.

In addition to the above costs, there are other types of costs - **intervention costs, unwanted achievement costs, human-computer interaction costs, costs of cases and costs of instability**. In this research, we concentrate on the cost-sensitive learning methods which minimize the misclassification costs and the test costs.

▪ Cost Matrix and Cost Function

Most of the cost-sensitive learning methods surveyed in this paper assume that for an M-class problem, an *M* by *M* cost matrix *C* is available at learning time. The value of *C(i, j)* is the cost involved when a test case is predicted to be class *i* but actually it belongs to class *j*.

In reality, *C(i, j)* can be example dependent, can be represented by a function, but in this survey, most cost-sensitive learning methods assume that *C(i, j)* does not change during the learning or decision making process. So the cost matrix *C* is static.

A static cost matrix always has the following structure when there are only two classes:

Table 1. Two-Class Cost Matrix

	Actual negative	**Actual positive**
Predict negative	$C(0, 0) = C_{00}$	$C(0, 1) = C_{01}$
Predict positive	$C(1, 0) = C_{10}$	$C(1, 1) = C_{11}$

As per above cost matrix, the cost of a false positive is C_{10} while the cost of a false negative is C_{01}. Conceptually, the cost of labelling an example incorrectly should always be greater than the cost of labelling it correctly. Mathematically, it should always be the case that $C_{10} > C_{00}$ and $C_{01} > C_{11}$ (Elkan 2001).

As we just mentioned, the cost values of a static cost matrix are always fixed, usually defined by dollar values associated with the correct or incorrect decisions. The learning task is not altered if all the cost values are scaled by a constant factor.

Cost values can represent either costs or benefits, or both, and a careful analysis is needed prior to designing the matrix so that all potential costs that are incurred by a decision are captured. Costs are represented by positive values, whereas benefits are represented by negative values (Margineantu 2001).

As per Elkan (2001), if a cost matrix C is known in advance, let the (i, j) entry in C be the cost of predicting class i when the true class is j. If $i = j$ then the prediction is correct, while if the prediction is incorrect. The optimal prediction for an example x is the class i that minimizes:

$$L(x,i) = \sum_{j=1}^{n} p(j \mid x)C(i,j) \tag{2.1}$$

In this framework, a test example should always be predicted to have the class that leads to the lowest expected cost, where the expectation is computed using the conditional probability of each class given the example. The role of a learning algorithm is to produce a classifier that for any example can estimate the probability $P(j|x)$ of each class j being the true class of x. For an example x, making the prediction i means acting as if i is the true class of x. The essence of cost-sensitive decision-making is that it can be optimal to act as if one class is true even when some other class is more probable.

2.3 Cost-sensitive Learning Methods

2.3.1 By Changing the Class Distribution of the Training Data

The first approach to cost-sensitive classification is to change the class distribution of the training data. Re-sampling and instance weighting are the most common approaches. MetaCost (Domingos 1999) is also a popular algorithm in this category which employs a “meta-learning” procedure, bagging, to re-label training examples with their estimated minimal cost classes, and then apply the error based learner to the new training set to generate a final model.

- **Re-sampling and instance weighting**

  One of the most common practical approaches to cost-sensitive classification is to change the class distribution of the training data with respect to the cost function and then present an error-based learning algorithm with those modified data. A simple approach is stratification, which is, changing class distribution in the training data in proportion to their cost (Li et al. 2005). This is achievable either by over sampling the examples from the more costly class or by under sampling the examples from the less costly class.

An alternative approach is the instance weighting method. It assigns a weight to each instance of the training data. The weight reflects the influence of misclassifying the case in terms of the cost incurred. Whilst the re-balancing method is applicable to any error-based classifiers, the re-weighting method is generally used by the classifiers that can handle instance weights, such as decision tree induction algorithm C4.5 (Quinlan 1993) and Bayesian classifiers.

In the past, many researchers have studied the effects of re-sampling and instance weighting methods in cost-sensitive learning, especially on imbalanced data sets. In most of these studies, re-sampling (or instance weighting) methods are used as a comparison to other cost-sensitive learning approaches. Some examples of these studies include: Kubat and Watwin (1997), Japkowicz (2000), Ting (2002), Maloof (2003), Zadrozny et al. (2003), Drummond and Holte (2003), Chawla (2003), Abe et al. (2004) and Zhou and Liu (2006).

Stratification is simple and easy to implement for any two class problems. However, Elkan (2001) argues that changing the balance of negative and positive training examples has little effect on the classifier learned by standard decision tree learning methods. He recommends that in cost-sensitive learning which considers various kinds of cost, a classifier should be learned from the training set as given, and then computes the optimal decision explicitly using the probability estimates given by the classifier. We will discuss this approach in section 2.3.5.

- **MetaCost**

Domingos (1999) also found that stratification has several shortcomings: First, it distorts the distribution of examples, which may seriously affect the performance of some algorithms. Second, it reduces the data available for learning, if stratification is carried out by under sampling. It also increases learning time, if it is done by over sampling. Most seriously, it is only applicable to two class problems and to multi-class problems with a particular type of cost matrix. Domingos (1999) proposes a more sophisticated method called MetaCost that employs a “meta-learning” procedure, bagging, to re-label training examples with their estimated minimal cost classes, and then apply the error based learner to the new training set to generate a final model. His experimental results show that MetaCost systematically reduces cost compared to error-based classification and to stratification, often by large amounts, and that MetaCost can be efficiently applied to large databases and multi-class problems.

All above methods make error-based classifiers cost-sensitive by changing the class distribution of training data. This makes reduction of overall cost more likely, but not necessarily as there is a trade-off among errors with different costs. The major advantages of these methods are the simplicity without any change in algorithms and the applicability to any existing error-based classification learning algorithms. They generate better models in terms of overall costs, compared to the models derived from original sample data (Li et al. 2005).

2.3.2 By Modifying the Learning Algorithms

The second approach to cost-sensitive classification is to change classification algorithms directly. They are implemented either by inducing various biases in the process of building models, or by adjusting thresholds or ordering rules generated. Decision tree and Naive Bayes are well known algorithms that have been mostly modified to make them cost-sensitive (Pazzani et al. 1994; Gama et al. 2000; Vadera 2005; Abrahams et al. 2005). Other popular algorithms that have been modified to be cost-sensitive include Neural Networks and Support Vector Machine (Kukar and Kononenko 1998; Fumera and Roli 2002; Brefeld et al. 2003).

- **Modifying Decision Tree Algorithm**

  A standard decision tree algorithm normally has two phases: decision tree growing and pruning. It can be modified to be cost-sensitive in both phases. In the decision tree growing phase, a typical approach is to change the split criterion which takes costs into account. Pazzani et al. (1994) studied several methods of considering misclassification cost when selecting tests in decision tree splits. One method uses the GINI criterion with the altered priors (suggested by Brieman et al. (1986)) methods, and another simply uses the cost of misclassification as a test selection metric. They then compared these "cost-sensitive" decision tree algorithms to the original CARTS and C4.5 and found the new "cost-sensitive" decision tree algorithms actually perform worse in terms of minimizing the misclassification cost. Drummond and Holte (2000) also investigated how the split criterion of decision tree algorithms are influenced by misclassification cost. Their experiment results show too that split criterion that are relatively insensitive to costs, and perform as well as or even better than split criterion that are cost sensitive.

However, recently researchers moved away from traditional ways of building decision trees and presented some new decision tree algorithms which take misclassification costs into account when making splits:

Vadera (2005) presented a new non-liner decision tree learning algorithm which takes account of misclassification cost. The algorithm is based on the hypothesis that non-linear decision nodes provide a better basis for cost-sensitive induction than axis-parallel decision nodes and utilizes discriminant analysis to construct non-linear cost-sensitive decision trees. The new algorithm, called CSNL (Cost-sensitive Non-linear Decision Tree), takes advantage of existing work by statisticians who have developed a theory of multivariate discriminate analysis to make non-linear splits when building decision trees.

Abrahams et al. (2005) proposed a framework for cost-sensitive classification under a generalized cost function. By combining decision trees with sequential binary programming, the new framework can handle non-uniformed misclassification cost, constrained classification, and complex objective functions that other methods cannot. In their paper, Abrahams et al. use the term generalized partitioning optimization problem (GPOP) for problems in which the benefit of the generalized cost function is maximized. A novel method called Sequential Binary Programming (SBP) is introduced. Using SBP to solve GPOPs begins with the original set S and then iteratively solves this binary program for each available one-dimensional split on each existing partition. The split that yields the highest objective function value is then accepted, and further partitioning is evaluated until additional partitions fail to improve the objective function or an exit criterion is reached. With each iteration, SBP allows us to select the partition that yields the optimal reclassification. In this way, the true objective function is used throughout the growing process. The real power of SBP over other direct cost-sensitive methods is its ability to accommodate any objective function, not just a cost matrix.

Standard methods for pruning decision trees are highly sensitive to the prevalence of different classes among training examples. If all classes except one are rare, C4.5 often prunes the decision tree down to a single note. Such a classifier is useless in cost-sensitive learning because the misclassification of a rare class normally incurs very high cost (Elkan 2001). Bradford et al. (1998) have studied decision tree pruning for minimizing misclassification cost through improved probability estimation. They have extended existing pruning methods to involve cost-complexity characteristics and formed two variants of pruning based on Laplace corrections. Their experimental results show that no method dominates the others in all datasets, and different pruning techniques are better for different cost matrices. They also show that Laplace correction performs well compared to others, for some cost matrices. However, in Elkan (2001)'s paper, he showed that Laplace pruning is similar to no pruning and argued that in cost-sensitive learning, growing a decision tree can be done in a cost-insensitive way. When using a decision tree to estimate class probabilities, it is preferable to do no pruning, but use smoothed probabilities instead.

- **Modifying Naive Bayes Algorithm**

  Cost-sensitivity learning techniques have also been researched extensively on Naive Bayes Classifier. Pazzani et al. (1994) studied cost-sensitive decision making with Naive Bayesian algorithm among their decision tree approaches. They call this cost-sensitive Bayes classifier Bayes-Cost. Bayes-Cost simply assigns a test example to the least expected cost class which is determined by function of the probability estimates returned by the classifier. Their experiment results showed that on some data sets, the Bayes-Cost algorithm has costs that are much lower than the other algorithms, but on other data sets, its costs are considerably higher than the other algorithms. They suggested that the reason behind this is due to feature interaction and irrelevant features in different data sets.

  Gama et al. (2000) presented an iterative approach to Naive Bayes classifier which is also sensitive to misclassification cost. This approach builds distribution tables using Naive Bayesian algorithm at first, and then applies an optimization process. The optimization process updates the contingency tables iteratively and aims to improve the probability class distribution associated with each training example. When there are unequal error costs in the domain, this iterative update can be guided by misclassification cost.

- **Modifying Neural Network Algorithm**

Artificial neural networks are very popular and useful tools in data mining and machine learning, where they are often used for classifications.

Kukar and Kononenko (1998) conducted a comparative study of different approaches to cost-sensitive learning with neural networks, including cost-sensitive classification, adaptive output, adaptive learning rate and minimization of misclassification cost. Their results showed that neural networks trained using a back-propagation version that minimizes the misclassification cost performs significantly better than the other methods.

Normally, the back-propagation algorithm minimizes the squared error of the neural network. Therefore, the original back-propagation learning procedure is not suitable for cost-sensitive learning. In order to minimize the costs of the errors made, the misclassification cost method takes cost into account by changing the error function and introducing the cost factor C[i, j] (i = desired class, j = actual class). Instead of minimizing the squared error, the modified back-propagation learning procedure minimizes the misclassification cost.

Over-fitting of training data is a serious problem in the process of back-propagation learning. It is caused by an oversized network and results in loss of its generalization abilities. The approach Kukar and Kononenko used to tackle this problem is to stop the learning process in the moment when the generalization abilities of the network ceased to increase.

- **Modifying Support Vector Machine Algorithm**

The Support Vector Machine (SVM) learning method is based on the structural risk minimisation (SRM) induction principle, which was derived from the statistical learning theory. SVMs have proven to be effective in many real-world applications. However, SVMs are not cost-sensitive.

Fumera and Roli (2002) proposed a cost-sensitive SVM classifier that directly embeds the reject option. This extension was derived by taking into account a theoretical implication of the SRM induction principle when applied to classification with reject option, and by following Vapnik (1982)'s maximum margin approach to the derivation of standard SVMs. They devised a novel formulation of the training task as a non-convex optimisation problem, and developed a specific learning algorithm to solve it. Their cost-sensitive learning method allows for a greater flexibility in defining the decision boundaries, with respect to the rejection technique used for standard SVMs.

Their experimental results show that on the most of data sets, the cost-sensitive SVM with embedded reject option achieved a better error-reject trade-off than standard non cost-sensitive SVMs. However, to allow the cost-sensitive learning SVM algorithm to scale up for larger data sets, its high computational cost must be addressed.

Brefeld et al. (2003) presented a natural cost-sensitive extension of the SVM. They considered the extension of SVMs by example dependent cost, and discussed its relationship to the cost- sensitive Bayes rule. They showed that the Bayes rule only depends on differences between costs for correct and incorrect classification. This allows them simplify the learning problem by assigning the cost for correct classification to zero. For the simplified problem, they stated a bound for the cost-sensitive risk. A bound for the original problem with costs for correct classification can be obtained in a similar manner.

Masnadi-Shirazi et al. (2010) proposed a new procedure for learning cost-sensitive SVM classifiers in which the SVM hinge loss is extended to the cost sensitive setting, and the cost-sensitive SVM is derived as the minimizer of the associated risk. The extension of the hinge loss draws on recent connections between risk minimization and probability elicitation. These connections are generalized to cost-sensitive classification, in a manner that guarantees consistency with the cost-sensitive Bayes risk, and associated Bayes decision rule. This ensures that optimal decision rules, under the new hinge loss, implement the Bayes-optimal cost-sensitive classification boundary. Minimization of the new hinge loss is shown to be a generalization of the classic SVM optimization problem, and can be solved by identical procedures. It enforces cost sensitivity for both separable and non-separable training data, enforcing a larger margin for the preferred class, independent of the choice of slack penalty. It also offers guarantees of optimality, namely classifiers that implement the cost-sensitive Bayes decision rule and approximate the cost-sensitive Bayes risk.

Other papers regarding learning cost-sensitive SVM methods include Lin et al. (2002), Geibel and Wysotzki (2003) and Xu et al. (2006).

2.3.3 By Taking the Boosting Approach

The third approach to cost-sensitive classification is to employ the boosting method. The algorithm generates a set of different weak classifiers in sequential trials due to each of the training data being re-weighted, and then constructs a composite classifier by voting them in terms of the accuracy of each weak classifier. The boosting method is applicable to any kinds of base classifier such as decision trees, Bayesian networks or neural networks (Li et al. 2005).

- **AdaBoost and AdaCost**

  AdaBoost (AdaptiveBoosting) is the most popular boosting algorithm used in practice and the terms Boosting and AdaBoost are therefore often used interchangeably. AdaCost (Fan et al. 1999) is a two-class cost-sensitive version of AdaBoost. It introduced a misclassification cost adjustment function into the re-writing function of AdaBoost. The function is capable of increasing the weights of costly misclassification instances more aggressively, but decreasing the weights of costly correct classification instances more conservatively. In this way, each weak classifier correctly classifies more expensive examples more likely and the final voted ensemble will also correctly predict more costly instances with the hope of reduction in overall cost. This method has been reported to reduce cost significantly.

- **Cost-sensitive Boosting**

  Ting and Zheng (1998) proposed two other cost-sensitive boosting methods. Their first approach is very similar to AdaBoost. Only at the classification stage, it used the minimum expected cost criterion to select the predicted class. The second approach called cost-boosting, entirely modifies the weight updating rule of AdaBoost. According to the new rule, if an instance is misclassified, its weight is replaced with its misclassification cost; otherwise its weight remains unchanged. In his further studies, Ting (2002) improved his boosting approaches by presenting two new variants. The new approaches relearn their models when misclassification cost changes.

Masnadi-Shirazi et al. (2011) proposed a new framework for the design of cost-sensitive boosting algorithms. The framework is based on the identification of two necessary conditions for optimal cost-sensitive learning that 1) expected losses must be minimized by optimal cost-sensitive decision rules and 2) empirical loss minimization must emphasize the neighbourhood of the target cost-sensitive boundary. It is shown that these conditions enable the derivation of cost-sensitive losses that can be minimized by gradient descent, in the functional space of convex combinations of weak learners, to produce novel boosting algorithms. The proposed framework is applied to the derivation of cost-sensitive extensions of AdaBoost, RealBoost, and LogitBoost.

Concept drift is a phenomenon typically experienced when data distributions change continuously over a period of time. Venkatesan et al. (2010) proposed a cost-sensitive boosting approach for learning under concept drift. The proposed methodology estimates relevance costs of ‘old’ data samples with regard to ‘newer’ samples and integrates it into the boosting process. Their experiment results demonstrate that the cost-sensitive boosting approach significantly improves classification performance over existing algorithms.

- **Asymmetric Boosting**

  Masnadi-Shirazi and Vasconcelos (2007) proposed a new cost-sensitive boosting algorithm, asymmetric boosting which is derived from sound decision-theoretic principles. It exploits the statistical interpretation of boosting to determine a principled extension of the boosting cost. Similar to AdaBoost, the new asymmetric boosting algorithm minimizes the cost by gradient descent on the functional space of convex combinations of weak classifiers, and generate large margin detectors. The resulting asymmetric boosting algorithm provides a proper combination of cost-sensitive weight update rule, and cost-sensitive method for finding the gradient direction.

  In their paper, Masnadi-Shirazi and Vasconcelos showed that similar to AdaBoost, asymmetric boosting is a margin maximization method. It increases the margin of the detector even after the training error is exhausted. The only difference is that the margins in asymmetric boosting are unbalanced, reflecting the costs assigned to the different error types, as per the cost function. Their experimental results showed that asymmetric boosting outperforms or comparable to those of various previous cost-sensitive boosting methods.

Other cost-sensitive boosting methods presented recently by researchers are Khoshgoftaar et al. (2002), Viola and Jones (2002), Merler et al. (2003), Abe et al. (2004) and Sun et al. (2005).

2.3.4 Cost-sensitive Stacking

Stacking is a method of combining the outputs of multiple independent classifiers for multi-label classification. Lo et al. (2011) have proposed formulating the audio tagging task as a cost-sensitive multi-label classification problem and extended a multi-label classification method, namely stacking, to its cost-sensitive version to solve the problem. Inspired by the idea of stacking, they improved their MIREX 2009 classifier ensemble by using cost-sensitive stacking. They first train K SVM-based and K AdaBoost-based cost-sensitive binary tag classifiers by using the tag counts as costs independently. Then, they use stacking SVM to respectively process the outputs of the two sets of binary tag classifiers. Finally, the stacked SVM and AdaBoost scores are merged by using either a probability ensemble to classify test examples.

2.3.5 Direct Cost-sensitive Learning Approach

Any learned classifier that can provide conditional probability estimates for training examples can also provide conditional probability estimates for test examples. In case of decision trees, the class probability distribution of an example can be estimated by the class distribution of the examples in the leaf that is reached by that example. In case of neural networks, class ranking is usually done based on the activation values of the output units. If the learned model does not explicitly compute these values, most classifiers can be modified to output some values that reflect the internal class probability estimate (Margineantu 2001). Using these probability estimates we can directly compute the optimal class label for each test example using the cost function. This cost-sensitive learning method is called direct cost-sensitive learning (Zadrozny and Elkan 2001).

All direct cost-sensitive learning algorithms have one thing in common: They do not manipulate the internal behaviour of the classifier nor do they manipulate the training data in any way. They are based on the optimal cost-sensitive decision criterion that directly use the output produced by the classifiers in order to make an optimal cost-sensitive prediction.

An advantage of the direct cost-sensitive learning is that it does not require the models to be retrained every time the costs change, as costs are only introduced in the post learning phase. However, the probability estimates output by classifiers from many error based learners are neither unbiased, nor well calibrated. Decision tree learners such as C4.5 and ID3 are well known examples. They focus primarily on discrimination between the classes and only produce posterior class membership probability estimates as by-product of classification. Naive Bayes classifiers are based on the assumption that within each class, the values of the attributes of examples are independent. It is well known that these classifiers tend to give inaccurate probability estimates. Several methods have been proposed for obtaining better probability estimates from decision tree, Naive Bayes and SVM classifiers (Viaene and Dedene 2004; Margineantu 2002, Zadrozny and Elkan 2002, 2001; Provost and Domingos 2003, 1998; Platt 1999). These probability estimation methods are listed here:

- **Laplace correction**

  For decision trees in general and C4.5 in specific, there are two reasons why they can not provide accurate posterior probability estimation, as pointed out by Zadrozny and Elkan (2001): Firstly, there is a high bias because decision trees always try to create homogeneous leaves and the probabilities of classes are always close to zero or one. Secondly, the reliability of class probabilities is low if there are only a few instances in a leaf. The later problem is usually eliminated by pruning algorithms, but those pruning algorithms are focused on accuracy maximization and are therefore not suited to estimate class probabilities, especially for datasets with an unbalanced class distribution. Using C4.5, Provost and Domingos (2003) evaluated different pruning methods and recommended not pruning the tree, instead, using Laplace correction to calculate class probabilities at leaves. The Laplace correction method basically corrects the probabilities by shifting them towards 0.5, in a two-class problem.

- **M-Smoothing**

Zadrozny and Elkan (2001) proposed to use an un-pruned decision tree and transform the scores of the leaves by smoothing them. They point out that the Laplace correction method doesn't work well for datasets with a skewed class distribution. They suggest using a smoothing method called m-estimation. According to that method, the class probabilities are calculated as follow.

$$\Pr(i|x) = \frac{N_i + b*m}{N+m},$$

where b is the base rate of the class distribution and m is a parameter that controls the impact of this correction. The base rate is the relative frequency of the minority class. They recommended choosing the constant m so that $b \times m = 10$. The experiments conducted by Zadrozny and Elkan (2001) show that using C4.5 decision tree as the base learner, the direct cost-sensitive learning approach with m-estimation achieved less misclassification cost than that of MetaCost (Domingos 1999) on the KDD-98 data set. They call this method m-smoothing. For example, if a leaf contains three training examples, one is positive and the other two are negative, the raw C4.5 decision tree score of any test example assigned to this leaf is 0.33. The smoothed score with $m = 200$ and $b = 0.05$ (the base rate of KDD-98 data set) is:

$$P' = (1 + 0.05 \times 200) / (3 + 200) = 11 / 203 = 0.0542.$$

Therefore, the smoothed score is effectively shifted towards the base rate of KDD-98 data set.

Furthermore, Niculescu-Mizil and Caruana (2005) experimented two other ways of correcting the poor probability estimates predicted by decision tree, SVM and other error based classifiers: Platt Scaling (Platt 1999) and Isotonic Regression. These methods can also be used in directly cost-sensitive learning algorithms.

- **Curtailment**

Zadrozny and Elkan (2001) pointed out that without pruning decision tree learning methods tend to overfit training data and create leaves in which the number of examples is too small to induce conditional probability estimates that are statistically reliable. Smoothing attempts to correct these estimates by shifting them towards the overall average probability. However, if the parent of a small leaf, i.e. a leaf with few training examples, contains enough examples to induce a statistically reliable probability estimate, then assigning this estimate to a test example associated with the small leaf may be more accurate than assigning it a combination of the base rate and the observed leaf frequency, as done by smoothing. If the parent of a small leaf still contains too few examples, the score of the grandparent of the leaf can be used, and so on until the root of the tree is reached. At the root, of course, the observed frequency is the training set base rate. They call this method of improving conditional probability estimates curtailment because when classifying an example, the searching stops as soon as the node reached has less than i examples, where i is a parameter of the method. The score of the parent of this node is then assigned to the example in question.

- **Binning NB**

Binning is a method suggested by Zadrozny and Elkan (2001). It aims to make class probabilities computed by Naive Bayes classifier more accurate. As mentioned above, Naive Bayes classifier builds on the assumption that attributes are independent. Zadrozny and Elkan pointed out that this assumption leads to too drastic probabilities because usually attributes do somehow correlate positively. They argue that therefore the probability score computed by Naive Bayes classifier, is either too close to zero or too close to one. Despite of that, Naive Bayes classifier has a high accuracy in ranking instances, so that instances with a higher score indeed have a higher probability to belong to that specific class than instances with a lower score. The binning algorithm, makes use of this "accuracy of ranking" to get calibrated probability estimation scores.

- **Platt Calibration**

Platt (1999) proposed a calibration method for transforming SVM predictions to posterior probabilities by passing them through a sigmoid. A sigmoid transformation is also used for boosted decision trees and other classifiers. Let the output of a learning method be $f(x)$. To get calibrated probabilities, pass the output through a sigmoid:

$$P\,(y = 1 \mid f) = 1 / (1 + exp\,(Af + B)),$$

where the parameters A and B are fitted using maximum likelihood estimation from a fitting training set $(f_i\,;\,y_i)$. Gradient descent is used to find A and B. Platt Scaling is most effective when the distortion in the predicted probabilities is sigmoid-shaped (Niculescu-Mizil and Caruana 2005).

- **Isotonic Regression**

Compared to Platt Calibration, Isotonic Regression is a more powerful calibration method which can correct any monotonic distortion (Niculescu-Mizil and Caruana, 2005). Zadrozny and Elkan (2002; 2001) successfully use this method to calibrate probability estimates from SVM, Naive Bayes and decision tree classifiers. Isotonic Regression is more general than other calibration methods we discussed above. The only restriction is that the mapping function must be isotonic (monotonically increasing). This means that given the predictions f_i from a model and the true targets y_i, the basic assumption in Isotonic Regression is that:

$$y_i = m\ (f_i) + \epsilon_{i,}$$

where m is an isotonic function. Then given a train set (f_i, y_i), the Isotonic Regression problem is to find the isotonic function m' that:

$$m' = \arg\min z \sum (yi - z(fi))^2$$

2.3.6 Other Cost-sensitive Learning Methods

- **Cost-sensitive specification**

Cost-sensitive specialisation (Webb 1996) involves specializing aspects of a classifier associated with high misclassification cost and generalizing those associated with low misclassification cost, the aim is to reduce the total cost. It is inspired by the theorem of decreasing inductive power which suggests that elements of a classifier having high misclassification cost should be specialized in order to minimize the proportion of false positives to true positives. In terms of decision trees, the leaves associated with classes of high cost are specialized, leaves having lower cost are generalized. One advantage of cost-sensitive specialization is that it does not require accurate misclassification cost. Only the relative ordering of them is required.

In his paper, Webb suggested that cost-sensitive specialisation is widely applicable and simple to implement. It could be used to augment the effect of standard cost-sensitive induction techniques and directly extended to test application cost sensitive induction tasks. It could also be applied in conjunction with alternative approaches to classification cost sensitive learning in the expectation of further boosting the effect of those approaches.

- **Cost-sensitive CBR System**

Wilke et al. (1996) presented a new algorithm, called KNNcost, for learning feature weights for CBR (case based reasoning) systems used for classification. Unlike other CBR algorithms, KNNcost considers the profit of a correct and the cost of an incorrect decision. Their method is based on conjugate gradient and it uses an integrated decision value matrix within the error function. They have shown that the method based on cost minimization is much more effective than the method based on accuracy, namely KNNacc and both provide improvements over initial CBR systems. However, their evaluation has only covered one application domain of very limited size, and they did not compare their algorithm to other existing methods.

- **Cost-sensitive Genetic Programming**

Kwedlo and Kretowski (2001) proposed a new cost-sensitive approach consists in modifying the existing system called EDRL-MD (Evolutionary Decision Rule Learner with Multivariate Discretization). EDRL-MD learns decision rules using a genetic algorithm. It learns rules by simultaneously searching for threshold values for all continuous-valued attributes. This approach is called multi-variate discretization.

To convert EDRL-MD for cost-sensitive learning they modified the fitness function used to guide the search process. The changes allow the genetic algorithm to minimize the expected misclassification cost rather than the error rate. They also presented the cost-sensitive methods for resolving conflicts between rules and for choosing a default class.

Li et al. (2005) presented a constrained genetic programming method (CGP), which is a GP based cost-sensitive classifier. Similar to EDRL-MD, CGP is capable of building decision trees to minimize not only the expected number of errors, but also the expected misclassification cost through a novel constraint fitness function.

2.3.7 Cost-sensitive Learning with Multiple Costs

The cost-sensitive learning methods mentioned previously mainly focus on reducing the misclassification cost and ignore the test cost, which is not true in many real world applications. Recently, researchers started to consider both test cost and misclassification cost (Turney 1995, 2000; Zubek and Dietterich 2002; Greiner et al. 2002; Ling et al. 2004). The objective is to minimize the expected total costs of test and misclassification.

Turney (1995) developed a learning system, called ICET, a cost-sensitive algorithm that employs genetic search to tune parameters used to construct decision trees. Each decision tree is built using Nunez' ICF criterion (described at the end of this section), which selects attributes greedily, based on their information gain and costs. Turney's method adjusts the test costs to change the behavior of Nunez' heuristic so that it builds different trees. These trees are evaluated on an internal holdout data set using the real test cost and misclassification cost. After several trials, the best set of test cost found by the genetic search is used by the Nunez' heuristic to build the final decision tree on the entire training data set. Because Turney simply modifies the attribute selection in C4.5 to add attribute costs when implementing the Nunez' criterion, his algorithm can deal with continuous attributes and with missing attribute values. Turney (1995) is also a seminal work laying the foundations of cost-sensitive learning with both attribute cost and misclassification cost. Turney compares his algorithm with C4.5 and with algorithms sensitive only to attribute cost (Norton, Nunez and Tan). He does not compare ICET with algorithms sensitive to misclassification cost only, because in his experiments he used simple misclassification cost matrices (equal costs on diagonal, equal costs off diagonal) which make algorithms sensitive only to misclassification cost equivalent to minimizing 0/1 loss. ICET outperformed the simpler greedy algorithms on several medical domains from the UCI repository.

In Zubek and Dietterich (2002), the cost-sensitive learning problem is casted as a Markov Decision Process (MDP), and an optimal solution is given as a search in a state space for optimal policies. For a given new case, depending on the values obtained so far, the optimal policy can suggest a best action to perform in order to minimize the misclassification cost and the test cost. While related to other work, their research adopts an optimal search strategy, which may incur very high computational cost to conduct the search.

Similar in the interest in constructing an optimal learner, Greiner et al. (2002) studied the theoretical aspects of active learning with test cost using a PAC learning framework. It is a theoretical work on a dynamic programming algorithm (value iteration) searching for best diagnostic policies measuring at most a constant number of attributes. Their theoretical bound is not applicable in practice, because it requires a specified amount of training data in order to obtain close-to-optimal policies.

Ling et al. (2004) proposed a new method for building and testing decision trees involving misclassification cost and test cost. The task is to minimize the expected total cost of test and misclassification. It assumes a static cost structure where the cost is not a function of time or cases. It also assumes the test cost and the misclassification cost have been defined on the same cost scale, such as the dollar cost incurred in a medical diagnosis. At the end of section 2.3.7, we will provide more details of this work.

Following the work in Ling et al. (2004), Qin et al. (2004) proposed a general framework for involving multiple costs in different cost scales. The task is to minimize one cost scale and control other cost scales as per specified budgets. Chai et al. (2004) proposed a test cost sensitive Naive Bayes network. Ling and Yang worked on test strategies in test cost sensitive learning (Ling et al. 2005; Yang et al. 2006), and they aimed to find the best test attribute set for decision making. Zhang et al. (2005) considered the cost-sensitive learning in data with missing values and concluded that some data are left as unknown in domain of test cost-sensitive learning and could be useful for better decisions.

Some of these test cost-sensitive learning methods are discussed below:

- **EG2**

EG2 (Núñez 1991) is a decision tree induction algorithm that uses the Information Cost Function (ICF) for selection of attributes. It is a modified version of ID3 (Quinlan 1989), the predecessor of a popular decision tree induction algorithm C4.5. ICF selects attributes based on both their information gain and their cost. The ICF for the *i*-th attribute, ICF*i*, is defined as follow:

$$ICFi = \frac{2^{Ii}-1}{(Ci+1)^{w}},$$

where Ii is the information gain associated with the i-th attribute at a given stage in the construction of the decision tree and Ci is the cost of measuring the i-th attribute, and w is an adjustable parameter between 0 and 1.

EG2 is able to reduce the overall cost by selecting attributes with less test cost and larger information gain to split.

- **CS-ID3**

CS-ID3 (Tan 1993) is a decision tree algorithm that selects the split attribute which maximizes the following function:

$$C = \frac{Ii^2}{Ci}$$

It is very similar to EG2, where Ii is the information gain associated with the i-th attribute at a given stage in the construction of the decision tree and Ci is the cost of measuring the i-th attribute. However, CS-ID3 does not build a full decision tree then classify examples. Instead, it only constructs a lazy ,ree (a decision path) for each test example to classify them.

- **IDX**

IDX (Norton, 1989) is also a decision tree algorithm that selects the split attribute that maximizes the following function:

$$C = \frac{Ii}{Ci}$$

Same as EG2 and CS-ID3, in the above function, Ii is the information gain associated with the i-th attribute at a given stage in the construction of the decision tree and Ci is the cost of measuring the i-th attribute. In C4.5, at each step, a greedy search strategy is used to choose the attribute with the highest information gain ratio. IDX uses a look-ahead strategy that looks n tests ahead, where n is a parameter that may be set by the user.

- **ICET**

ICET is a hybrid of a genetic algorithm and a decision tree induction algorithm. The genetic algorithm evolves a population of biases for the decision tree induction algorithm. The genetic algorithm is GENESIS (Grefenstette 1986). The decision tree induction algorithm is EG2. ICET manipulates the bias of EG2 by adjusting the parameters Ci and w. In the original design of EG2, Ci is the attribute test cost. But in ICET, it is treated as a bias parameter.

In ICET, the genetic algorithm GENESIS begins with a population of 50 individuals. EG2 is run on each one of them to build a corresponding decision tree. The "fitness" of the individual is the total of test and misclassifications costs averaged over the number of cases. In the next generation, the population is replaced with new individuals generated from the previous generation. The fittest individuals in the first generation have the most offspring in the second generation. After a fixed number of generations, ICET stops and outputs the decision tree generated by the fittest individual.

- **MDP**

In Zubek and Dietterich (2002)'s paper, cost-sensitive learning problem is casted as a Markov Decision Process (MDP), and solutions are given as searching in a state space for optimal policies. For a given new case, depending on the values obtained, the resulting policy can suggest an optimal action to perform in order to minimize both the misclassification and the test costs. Their admissible search heuristic is shown to reduce the problem search space remarkably. However, it may take very high computational cost to conduct the search process. In addition, to reduce over-fitting, they have introduced a supplementary pruning heuristic named statistical pruning.

- **Cost-sensitive Naive Bayes**

Chai et al. (2005) presented a test Cost-sensitive Naive Bayes (CSNB) classifier which modifies the Naive Bayes classifier by including a test strategy which determines how unknown attributes are selected to perform test on in order to minimize the sum of misclassification cost and test cost. In the framework of CSNB, attributes are intelligently selected for testing to get both sequential test strategies and batch test strategies.

- **Test Cost-sensitive Decision Tree**

Ling et al. (2004) proposed a new method for building and testing decision trees involving misclassification cost and test cost, called the TCSDT classification. The task is to minimize the expected total cost of test and misclassification. It assumes a static cost structure where the cost is not a function of time or cases. The TCSDT classification is based on C4.5. When building a decision tree, at each step, instead of choosing an attribute that minimizes the entropy (as in C4.5), the TCSDT classification chooses an attribute that reduces and minimizes the total of misclassification cost and test cost, for the split. Similar to C4.5, the TCSDT classification chooses a locally optimal attribute without backtracking. To make a decision tree cost-sensitive, the decision on which attribute to split on is determined by calculating the sum of misclassification and test cost for every possible split, and, of course, choosing the lowest.

TCSDT can be applied to many real world applications. For example, in medical diagnosis, it costs money to request a blood test, X-ray, or other types of test, some tests can be quite expensive. Doctors often have to balance the cost of a test and the accuracy of the diagnosis (prediction) to decide what tests should be performed. If a test is too expensive compared to the potential reduction of misclassification cost, it is desirable to skip the test. TCSDT can automate this decision process and minimize the total cost of test and misclassification.

Elkan (2001) points out that this decision tree approach may lead to a classification model that minimizes the cost of misclassification of the training set, but does not produce an optimal model when applied to unseen data, mainly because of over-fitting. That is, when a decision tree is built, some branches may be built reflecting anomalies in the training data due to noise or outliers. This will lead to the algorithms perform well only on the training data but poor on the test data.

Recently, two pre-pruning methods have been proposed by Du et al. 2007 to avoid data over-fitting for the TCSDT classification. The first method is to build a decision tree with maximum of two levels. The second method is when building the decision trees, setting a threshold of cost reduction (threshold = FP + FN) for the potential splits. If the cost reduction is less than the threshold, a leaf node is formed; no further split will be performed. Their experimental results show that the two pre-pruning methods perform better than the original TCSDT as well as C4.5 on the selected UCI data sets.

However, the data over-fitting problem in the TCSDT classification is still not well studied. Although the TCSDT classification extends C4.5 by using minimal cost as the splitting criterion, it has the same deficiencies as other decision tree algorithms when producing class probabilities. For decision trees in general and TCSDT in specific, there are two reasons why they cannot provide accurate posterior probability estimation, as pointed out by Zadrozny and Elkan (2001). Firstly, there is a high bias because decision trees always try to create homogeneous leaves and the probabilities of classes are always close to zero or one. Secondly, the reliability of class probabilities is low if there are only a few instances in a leaf.

- **Multiple Scale Cost-sensitive Decision Tree**

Qin et al. (2004, 2005) argue that cost sensitive decision tree algorithm must consider the resource budget when building trees and classifying test examples. Based on the decision tree built by Ling et al. (2004), they propose a new decision tree algorithm which considers multiple cost scales in the tree building and testing process. In their algorithm, misclassification cost and test cost can be on the same scale if both of them can be converted to dollar values. They can be on different scales if one of them cannot be converted. Other resource costs such as time cost are always on different scales. When building decision trees, instead of using total cost as the split criterion, they use cost gain ratio (cost gain / resource cost) to split. In their paper, a resource budget is set for each resource. During the testing, if an example is run out of resource, it has to stop at the internal node which represents the attribute. Missing values are handled in the same way as that in Ling et al. (2004).

The aim of Qin et al. (2004)'s decision tree algorithm is to minimize the misclassification cost in respect to limited resource budget. In their framework, misclassification cost does not have a budget. All other resources have limited budget. Multiple resource costs, such as test cost, time cost and computation cost, can be involved in the decision process. The decision tree built by Ling et al. (2004) becomes a special case of this more general cost sensitive decision tree building framework.

2.4 Other Related Works

2.4.1 Feature Selection

Feature selection (also called attribute selection) is an important research area of data mining and machine learning. It is a kind of data pre-processing strategy. Many classification algorithms such as nearest neighbour and decision tree can be benefited from an effective feature selection process. The reason is in practice, the real world data sets often contain noisy data and irrelevant/distracting/correlated attributes which often "confuse" classification algorithms, and results in data over-fitting and poor predication accuracy on unseen data.

Many feature selection methods were developed over the years in practical data mining and machine learning research, such in Statistics and Pattern Recognition. The two commonly used approaches are the filter approach and the wrapper approach. The filter approach selects features using a pre-processing step which is independent of the induction algorithm. The main disadvantage of this approach is that it totally ignores the effects of the selected feature subset on the performance of the induction algorithm. On the contrary, in the wrapper approach, the feature subset selection algorithm conducts a search for a good subset using the induction algorithm itself as a part of the evaluation function. The accuracy of the induced classifiers is estimated using accuracy estimation techniques (Kohavi and John 1997).

Most previous feature selection research focuses on improving predication accuracy. To the best of our knowledge, the impact of using feature selection to improve cost-sensitive classifier performance is not well studied. We believe, due to the fact that if properly designed, the feature selection approach can effectively remove the noisy data and irrelevant/distracting/correlated attributes from training set so that our cost-sensitive KNN algorithms can find "better" neighbours with the most relevant attributes which minimizes misclassification cost.

2.4.2 Ensemble Method

Ensemble data mining method, also known as Committee Method or Model Combiner, is a data mining method that leverages the power of multiple models to achieve better prediction accuracy than any of the individual models could on their own. The algorithm works as below:

Firstly, base models are built using many different data mining algorithms.

Then a construction strategy such as forward stepwise selection, guided by some scoring function, extracts a well performing subset of all models. The simple forward model selection works as follows:

- Start with an empty ensemble;
- Add to the ensemble the model in the library that maximizes the ensemble's performance to the error (or cost) metric on a hill-climb set;
- Repeat Step 2 until all models have been examined;
- Return that subset of models that yields maximum performance on the hill-climb set.

Ensemble learning methods generate multiple models. Given a new example, the ensemble passes it to each of its multiple base models, obtains their predictions, and then combines them in some appropriate manner (e.g., averaging or voting). Usually, compared with individual classifiers, ensemble methods are more accurate and stable. Some of the most popular ensemble learning algorithms are Bagging, Boosting and Stacking.

Evaluating the prediction of an ensemble typically requires more computation than evaluating the prediction of a single model, so ensembles may be thought of as a way to compensate for poor learning algorithms by performing a lot of extra computation.

2.5 Experiment Settings

- **Weka Workbench**

Weka workbench (Witten and Frank 2000) is a collection of data mining and machine learning algorithms and data processing tools. It is an open source, java based software. It is designed so that you can quickly try out existing data mining methods on new data sets in flexible ways. It provides extensive support for the end to end process of experimental data mining.

All the experiments in this research will be conducted in Weka to validate the proposed methods and algorithms. The experimental results are used in the next step to evaluate and improve the proposed methods and algorithms. The methods and algorithms are evaluated from the aspects of both academic research and their usefulness in real world applications.

- **Data Sets**

Below is the list of data sets used in this thesis:

Dataset	**No. of attributes**	**No. of examples**	**Class distribution (P/N)**
KDD-98	481	3000	164/2836
Statlog (heart)	14	270	120/150
Australia	16	653	296/357
Breast	9	683	239/444
Ecoli	6	332	102/230
Heart	8	261	98/163
Credit-g	21	1000	300/700
Car	7	1728	134/1594
Hypothyroid	30	3772	291/3481
Sick	30	3772	231/3541
Credit-a	15	690	307/383
Diabetes	9	768	268/500
kr-vs-kp	37	3196	1527/1699
mushroom	23	8124	3916/4208
nursery	9	12960	4320/8640
optdigits	65	1143	571/572
page-blocks	11	5473	560/4913
spambase	58	4601	1813/2788
splice	62	2423	768/1655
vowel	14	990	90/900

waveform-5000	41	3347	1655/1692
pendigits	17	2288	1144/1144

These data sets are chosen based on the following criteria:

- Data sets should be two-class because the new algorithms developed in this PhD study currently can only handle two-class data sets. This condition is hard to satisfy and we resorted to converting several multi-class data sets into two-class data sets by choosing the least prevalent class as the positive class and union all other classes as the negative class. For two-class data sets, we always assign the minority class as the positive class and the majority class as the negative class.
- The experiments in this thesis do not focus on the data sets with many missing values, so all the data sets we selected do not have many missing values. If any examples have missing values, we either remove them from the data sets (for all the UCI data sets) or replace them using Weka's "ReplacingMissingValues" filter (for the KDD-98 data set).
- These data sets include both balanced and unbalanced class distributions. The imbalance level (the ratio of major class size to minor class size) in these data sets varies from 1.0 (Pendigits) to 17.3 (KDD-98).

- ## New Algorithms

The main new algorithms we developed in this PhD study are described in the below table:

#	Name	Category	Purpose of Design	Platform Coded
1	TCSDT-MS	TCSDT	Test Cost Sensitive Decision Tree with M-Smoothing	Weka
2	TCSDT-FS	TCSDT	Test Cost Sensitive Decision Tree with Feature Selection	Weka
3	TCSDT-TP	TCSDT	Test Cost Sensitive Decision Tree with	Weka

			Threshold Pruning	
4	Direct-CS-KNN	KNN	Direct Cost Sensitive KNN Classification	Weka
5	Distance-CS-KNN	KNN	KNN Classification with Cost Sensitive Distance Function	Weka
6	MSCSDT-PF	Multiple Cost Units	Multiple-Unit Cost Sensitive Decision Tree with Test Resource Budget Constraint	Weka
7	MSCSDT-PFAS	Multiple Cost Units	Attribute Selection Using Performance First Strategy And Test Resource Budgets, Then Use The Selected Attributes To Build Cost Sensitive Decision Tree Directly	Weka
8	MSCSDT-PFAS-Lazy	Multiple Cost Units	Attribute Selection Using Performance First Strategy And Test Resource Budgets For Each Test Example, Then Use The Selected Attributes To Build Cost Sensitive Decision Tree Directly To Classify The Example	Weka
9	Direct-EM	Semi-supervised	Direct Classification with Expectation Maximization	Weka
10	CS-EM	Semi-supervised	Cost Sensitive Classification with Expectation Maximization	Weka

2.6 Summary

In this Chapter, we brought an up-to-date review of the existing cost-sensitive learning framework and prevailing cost-sensitive learning methods. There are three major categories of these cost-sensitive learning methods. Each of them is implemented by changing one of the three components of the learning framework: the training data, the learning algorithm and the output of the learned model.

By changing the training data or the output of the learned model, any error based classifier can be used to make cost-sensitive decisions. This is the major benefit of the learning methods in these categories. Compared to changing the output of the learned model, changing the training data is more straightforward and easier to implement. Some researchers found that on some data sets, normally with relatively balanced class distributions, simple methods such as resample perform surprisingly well. However, the major drawback of changing the training data is that it can only apply to 2-class data sets or multi-class data sets with a particular type of cost matrix. Most of methods of changing the output of the learned model involve calculating the probability of a test example which belongs to a certain class. As we discussed in section 2.3.5, popular data mining algorithms such as Decision tree and Naive Bayes tend to produce inaccurate class probability estimates, therefore cannot be used directly to generate cost-sensitive learning models. Several methods such as smoothing and binning are proposed by researchers to tackle this problem. Some convincing results were found on certain test data sets.

Changing individual learning algorithms to be cost-sensitive is also very popular and was explored by many researchers in recent years. Decision tree, Naive Bayes, Neural Networks and SVM are among the most popular algorithms that were modified cost-sensitive. However, since these modern algorithms are already very sensitive to different data sets and class distributions, adding an extra level of complexity, cost, makes them even more sensitive, especially when the misclassification costs or the class distributions are very unbalanced. It is common that these methods perform very well on some data sets and poor on others. Data over-fitting is also an issue with these modified algorithms.

Furthermore, we discussed the latest progress in test cost-sensitive learning which aims to reduce the total cost of the test and the misclassification. C4.5 Decision tree is the mostly modified algorithm to consider both test cost and misclassification cost when making splits and classifying examples. As we just mentioned above, modern algorithms such as C4.5 are very sensitive to different data sets and class distributions. Adding misclassification cost, test cost and other types of cost for the algorithms to handle makes them too sensitive to be used in real world applications. They may generate good result on training data but normally does not produce an optimal model when applied to unseen data, mainly because of over-fitting. It is still a challenging task to combine multiple types of costs in cost sensitive learning research.

At the end, we specified the experiment settings of the tests we want to do as part of this PhD research, including the new methods and algorithms we are going to develop and all the data sets used in the experiment.

Chapter 3 Handling Over-fitting in Test Cost-sensitive Decision Tree Learning

Like other learning algorithms, cost-sensitive learning algorithms must face a significant challenge, over-fitting, in an applied context of cost-sensitive learning. Specifically speaking, they can generate good results on training data but normally do not produce an optimal model when applied to unseen data in real world applications. It is called *data over-fitting*. In this Chapter we deal with the issue of data over-fitting by designing three simple and efficient strategies, feature selection, smoothing and threshold pruning, against the TCSDT (test cost-sensitive decision tree) method. The feature selection approach is used to pre-process the data set before applying the TCSDT algorithm. The smoothing and threshold pruning are used in a TCSDT algorithm before calculating the class probability estimate for each decision tree leaf. To evaluate our approaches, we conduct extensive experiments on the selected UCI data sets across different cost ratios, and on a real world data set, KDD-98 with real misclassification cost. The experimental results show that our algorithms outperform both the original TCSDT and other competing algorithms on reducing data over-fitting.

3.1 Introduction

Existing cost-sensitive classification approaches incorporate the misclassification cost and other costs in the classification technique, thus provide practical classification result in a multiple-cost environment. However, they face a significant challenge, over-fitting, in an applied context of cost-sensitive learning. Specifically speaking, they can generate good results on training data but normally do not produce an optimal model when applied to unseen data in real world applications.

If a decision tree induction algorithm generates a decision tree that depends too much on irrelevant features of the training instances, and performs well only on training data but poorly on unseen data, the data over-fitting happens. Many previous works have been done to tackle the over-fitting problem in traditional decision tree learning. These works fall into the following three main categories:

- Pre-pruning – this involves a "termination condition" to determine when it is desirable to terminate some of the branches prematurely when a decision tree is generated.
- Post-pruning – this approach generates a complete decision tree and then removes some of the branches with an aim of improving the classification accuracy on unseen data.
- Data pre-processing – this approach does not try to simplify the resulting decision tree directly. It indirectly tries to reach a simplification through the training data used by the learning algorithm. The aim of the data pre-processing is to find an optimal number of characteristics in order to build a simpler decision tree.

Among them, post-pruning is the most commonly used method. However, standard error based pruning methods are not suitable for cost-sensitive learning, because they are based on accuracy maximization. As mentioned in Elkan (2001), the overall conclusion of Bradford et al. (1998) is that the best option is either no pruning or what they called "Laplace pruning". Domingos and Provost (2003) showed that decision tree class probability estimates can be improved by skipping the pruning phase and smoothing the distributions by applying Laplace correction. Zadrozny and Elkan (2001) proposed to use an un-pruned decision tree and transform the scores of the leaves by smoothing them. They pointed out that the Laplace correction method doesn't work well for datasets with a skewed class distribution. They suggested using a smoothing method called m-estimation. The above efforts are efficient for dealing with the data over-fitting in direct cost-sensitive learning (only considering the misclassification cost) applications.

In this Chapter, we tackle the data over-fitting issue where multiple costs must be taken into account in a learning application. It is well-known that multiple-costs-sensitive learning is different from direct cost-sensitive learning, and the above efforts cannot be simply applied to solve the data over-fitting for multiple-costs-sensitive learning applications. Therefore, Du et al. (2007) proposed two pre-pruning methods to reduce data over-fitting for the TCSDT classification. Different from the pre-pruning approach, in this research we deal with the issue of data over-fitting by designing three new, simple and efficient strategies, feature selection, smoothing and threshold pruning, against the TCSDT method. Briefly described, our strategies reduce data over-fitting by simplifying the final decision tree and improving the probability estimates produced by the TCSDT classification on the unseen data. The feature selection method removes the noisy and less relevant features from training data, The smoothing method can provide much better, more calibrated posterior probability estimation on the unseen data, and the threshold pruning method makes the estimation statistically more reliable (by pruning a decision tree leaf with number of examples or cost reduction is less than a predefined threshold). The feature selection method is used to pre-process the training data before the TCSDT algorithm is applied. The smoothing and the threshold pruning are used in the TCSDT algorithm before calculating the class probability estimate for each decision tree leaf.

Among the three new methods we've just proposed, feature selection is a major research area in data mining and machine learning. Previous research mainly focuses on using the feature selection to improve classification accuracy (Kohavi and John, 1997). In this research, we explore the effectiveness of using feature selection to tackle TCSDT data over-fitting. Our expectation is that by removing the noisy and less relevant features from the training data, we can build a much simpler decision tree which not only reduces misclassification cost but also test cost on the unseen data. Different from the m-estimation that Zadrozny and Elkan (2001) proposed, in the TCSDT classification, our smoothing method can also involve in the tree-building process. The smoothed probability estimates are then used to calculate the potential misclassification cost for each split. Some unnecessary splits will be avoided; hence we reduce test cost as well. Finally, we use cross validation to determine the m value in our smoothing approach. The threshold pruning method is different from any error based pruning or the "Laplace pruning" proposed by Bradford et al. (1998), it uses a set of predefined thresholds, some are cost related, to determine if a decision tree leaf is to be pruned or not. These three methods can be applied to the TCSDT classification independently or together.

Our experimental results show that when test cost is considered, compared to the original TCSDT, Cost-sensitive C4.5, and the two TCSDT pre-pruning methods proposed by Du et al. (2007), our new methods, feature selection, smoothing and threshold pruning can reduce the total of misclassification and test cost on most of the UCI data sets we tested. In addition, we evaluate our new methods on a real world data set, KDD-98, with real misclassification cost. We compare our new methods with other competitors using AUC (Area under ROC curve), which is widely used by researchers to evaluate the effectiveness of cost-sensitive classification algorithms. The test results also in favour of our new methods especially feature selection and smoothing.

3.2 TCSDT with Feature Selection, Smoothing and Threshold Pruning

3.2.1 TCSDT Classification by Smoothing

As we described in Chapter 2, several methods have been proposed for obtaining better probability estimates from decision tree classifiers in direct cost-sensitive learning. Zadrozny and Elkan (2001) proposed to use an un-pruned decision tree and transform the scores of the leaves by smoothing them. They call this method m-estimation. In order to reduce the total of misclassification and test cost, we propose two changes to the original m-estimation: First we use cross valuation to determine the *m* value for different data sets. It is more proper than a fixed value. Second, we use the smoothed probability estimate together with cost matrix to calculate the misclassification cost for each potential split.

Now let's look at the m-estimation formula we specified in Chapter 2 again. In order to explain the impact of m-estimation to the TCSDT data over-fitting issue, we represent the formula in a slightly different way:

$$\Pr(i \mid x) = \left(\frac{N}{N+m}\right) \times \left(\frac{N_i}{N}\right) + \left(\frac{m}{N+m}\right) \times b$$

As we can see from this formula, the value *m* controls the balance between relative frequency and prior probability. It has the following impacts to TCSDT:

- In many real world data sets, noise is often expected in the training examples, and the noisy data is the main cause of the data over-fitting issue. With m-estimation, *m* can be set higher so that the noisy value for *Ni/N* plays less important role in the final estimation and data over-fitting is reduced.

- In a decision tree leaf, if the number of examples (*N*) is small, without smoothing, the probability estimation provided by this leaf (*Ni/N*) is statistically unstable. This is another cause of the data over-fitting issue. However, with m-estimation, *N/(N+m) is* close to 0 and *m/(N+m) is* close to 1, so that the probability estimation is shifted towards the base rate (*b*). It works particularly well on data sets with skew class distribution.

In the experiment section, we apply this smoothing method to the TCSDT classification, and compare this approach to the original TCSDT, Cost-sensitive C4.5 and other cost-sensitive decision tree algorithms, as well as the threshold pruning and feature selection approach we described below.

3.2.2 TCSDT Classification with Threshold Pruning

For decision tree algorithms, a most common way to avoid data over-fitting is pruning technique. However, standard methods for pruning decision trees are highly sensitive to the prevalence of different classes among training examples. If all classes except one are rare, then C4.5 often prunes the decision tree down to a single node and classifies all examples as members of the common class. Such a classifier is useless for decision-making if failing to recognize an example in a rare class is an expensive error (Elkan 2001). Therefore, traditional error based pruning methods are inadequate for the TCSDT classification. In this Chapter, we plan to use a new method, named threshold pruning to prune a TCSDT. It uses the following thresholds to determine whether or not a decision leaf needs to be pruned.

- Minimum number of examples in a leaf. This can be arbitrary number *N* (for example, 50) or a proportion *P* (for example, 10 percent) of the total examples in the data sets. In this paper, we set this threshold to *min* (*N, P*). This means we prune a decision tree leaf when it contains less than N examples and less than P percent of the total examples.
- Cost reduction. We set a threshold of cost reduction (ThresholdCR) to the sum of the FP cost, the FN cost and the test cost (TC1+TC2+…+TCn):

$$ThresholdCR = FP + FN + (TC1+TC2+\ldots+TCn)$$

The first threshold makes sure that there are enough examples in each decision tree leaf so that it can produce a reliable probability estimate. The second threshold utilizes the misclassification cost and the test cost to make the cost reduction of each split more significant in order to reduce data over-fitting.

3.2.3 TCSDT Classification with Cost-sensitive Feature Selection

Most of real world data sets contain noisy data and irrelevant/disturbing features. For top down decision tree building algorithms, when you process further down the tree, less and less data is available to help make the selection decision. The impact of noisy data increases. At the some point, the irrelevant/disturbing attribute will look good and are selected. Then the data over-fitting happens and causes the poor classification performance on unseen data. Feature selection strategy can certainly help in this situation.

As mentioned in Chapter 2, there are many different feature selection methods developed. They fall into two categories: the filter approach and the wrapper approach. As per the study of Kohavi and John (1997), the wrapper approach generally perform better than the filter approach, and some significant improvement in accuracy was achieved on some data sets for decision tree algorithm and naive bayes algorithm using the wrapper approach.

The wrapper approach proposed by Kohavi and John (1997) conducts a search in the space of possible parameters. Their search requires a state space, an initial state, a termination condition and a search engine. The goal of the search is to find the state with the highest evaluation, using a heuristic function to guide it. They use prediction accuracy estimation as both the heuristic function and the evaluation function. They've compared two search engines, hill-climbing and best-first, and found that the best-first search engine is more robust, and generally performs better, both in accuracy and in comprehensibility as measured by the number of features selected.

In this Chapter, we apply this wrapper approach to TCSDT algorithm to reduce its data over-fitting issue. TCSDT is a cost-sensitive learning algorithm. The ultimate goal is to minimize the total cost. We cannot simply apply Kohavi and John's wrapper approach directly on TCSDT. Therefore, we propose two variations of Kohavi and

John's feature selection wrapper to overcome TCSDT data over-fitting issue. The main difference is using cost-sensitive estimation as both the heuristic function and the evaluation function.

The first approach is using Total Cost:

Total Cost = Misclassification Cost + Test Cost,

where the Test Cost is defined in section 2.2.

The second approach is using F-measure:

F-measure = (2 x *recall* x *precision) / (recall + precision),*

where precision is the fraction of retrieved instances that are relevant, while recall is the fraction of relevant instances that are retrieved.

Our preliminary test shows that the "Total Cost" works well when the test cost is also considered. If only misclassification cost is considered, "F-measure" is a better choice. So for our feature selection method, we use "F-measure" in our first experiment (all test cost is set to 0) and "Total Cost" in our second experiment (test cost is set to between 0 and 100).

The other settings in our experiment are similar: The search space we chose is that each state represents a feature subset. So there are *n* bits in each state for a data set with *n* features. Each bit indicates whether a feature is selected (1) or not (0). We always start with an empty set of features. The main reason for this setup is computational. It is much faster to build a decision tree using only a few features in a data set. We chose the best-first search algorithm as our search engine. The following summary shows the setup of our TCSDT feature selection problem for a simple data set with three features:

TCSDT feature selection setup for data set with three features:

State:	A Boolean vector, one bit per feature
Initial state:	A empty set of features (0,0,0)
Search space:	(0,0,0) (0,1,0) (1,0,0) (0,0,1)
	(1,1,0) (0,1,1) (1,0,1) (1,1,1)
Search engine:	Best first
Evaluation function:	Total Cost or F-measure

3.3 Experimental Evaluation

3.3.1 Experiment Setup

The main purpose of this experiment is to evaluate the performance of the TCSDT classification with feature selection, smoothing and threshold pruning, named as TCSDT-FS, TCSDT-SM and TCSDT-TP, discussed above by comparing their average total (misclassification and test) cost and other key performance measurements such as AUC across different cost ratios (FN/FP) against the original TCSDT, C4.5 with Minimum Expected Cost and the two pre-pruning methods proposed by Du et al. (2007). The last method, TCSDT-Train, is the same as the original TCSDT, but utilizing full data set to build decision tree and classify examples. All these algorithms are implemented in Weka (Witten and Frank 2000), and they are listed in Table 2.

Table 3.1 List of Decision Tree Algorithms and Abbreviations

#	Method	Abbreviation	Base Classifier
1	Original TCSDT	TCSDT	TCSDT
2	TCSDT M-Smoothing	TCSDT-MS	TCSDT
3	TCSDT Threshold Pruning	TCSDT-TP	TCSDT
4	TCSDT Feature Selection	TCSDT-FS	TCSDT
5	C4.5 with Minimum Expected Cost	C4.5-CS	C4.5
6	TCSDT 2 Level Tree - Du et al. (2007)	TCSDT-2L	TCSDT
7	TCSDT Pruning - Du et al. (2007)	TCSDT-Prune	TCSDT
8	Original TCSDT – Full Data Set as Training Set	TCSDT-Train	TCSDT

Please note that we will be conducting three experiments using the above algorithms. As we mentioned in the previous section (section 3.2.3), There are two variations of our TCSDT Feature Selection strategy, the first is using the “Total Cost” (the sum of misclassification cost and test cost) to guide the feature selection process, the second is using the “F-measure”.

We then conduct experiment on ten data sets chosen from UCI repository, and one real world data set, KDD-98. The details of these data sets are listed in Table 3.2. The imbalance level (the ratio of major class size to minor class size) in these data sets varies from 1.2 (Australia) to 17.3 (KDD-98).

Table 3.2. Summary of the Data set Characteristics

Dataset	**No. of attributes**	**No. of examples**	**Class distribution (P/N)**
KDD-98	481	3000	164/2836
Statlog (heart)	14	270	120/150
Australia	16	653	296/357
Breast	9	683	239/444
Ecoli	6	332	102/230
Heart	8	261	98/163
Credit-g	21	1000	300/700
Car	7	1728	134/1594
Hypothyroid	30	3772	291/3481
Sick	30	3772	231/3541
Vowel	14	990	90/900

We conduct three experiments on above datasets with different cost matrixes:

In the first experiment, we use the real world data set, KDD-98 and the UCI Statlog(heart) data set. Only misclassification cost is considered, all test costs are set to 0. We then evaluate the classifier performance by calculating their misclassification cost, AUC and the size of the decision tree generated by different algorithms.

- The Statlog(heart) data set is one of a few data sets in UCI library with recommended cost matrix. The cost matrix is normalized and the cost ratio (FN/FP) is set to 5. The cost of TP and TN are both set to 0. This data set has been used extensively in cost-sensitive learning research previously.
- The KDD-98 data set is a real world data set from the direct marketing domain. It is a large and challenging data set which was first used in the 1998 KDD data mining contest, and later on it became very popular as a benchmark for the evaluation of cost-sensitive learning algorithms (Zadrozny and Elkan 2001). This data set contains information about people who donated to charity. The classification task is to choose which person to mail a request for a new donation. The misclassification cost information in this data set is available for each example. The original data set consists of 95412 training examples and

96367 test examples. The cost of sending out a request (including the mail cost) is \$0.68. The overall response rate is 5.1%. The average donation amount is \$15.6. In our experiment, we randomly select a subset of 3000 examples from the entire data set. In the subset, the response rate is 5.57%, and the average donation amount is \$15.8. The initial cost matrix can be setup as below:

Table 3.3 Initial Cost Matrix for KDD-98 (positive: made donation, negative: no donation)

	Actual positive	**Actual negative**
Predict positive	0.68 – 15.8 = -15.12	0.68
Predict negative	0	0

Then we can transform this cost matrix to the one below so that it is in the same format as all the other cost matrixes we used in this paper. This means the cost of FN is 15.12 and the cost of FP is 0.68. The cost ratio is FN/FP = 22.2.

Table 3.4 Transformed Cost Matrix for KDD-98 (positive: made donation, negative: no donation)

	Actual positive	**Actual negative**
Predict positive	0	0.68
Predict negative	15.12	0

In the second experiment, nine UCI data sets are used to perform the test. The misclassification cost FP is always set to 100, and FN is set to an integer varying from 200 to 2000 (200, 500, 1000, 2000 respectively). We assume that the misclassification of the minority class always incurs a higher cost. This is to simulate real-world scenarios in which the less frequent class is the more important class. The cost of TP and TN are both set to 0. Both misclassification cost and test cost are considered in this experiment. All test costs are randomly set between 0 and 100. We then evaluate the classifier performance by calculating their average total cost (misclassification cost and test cost).

Both the first and the second experiment are repeated for 10 times and ten-folder cross validation method is used in all tests to prevent over-fitting data.

As you will see in the next section, our new methods perform really well on the selected real world data sets and many UCI data sets. However, compared to the original TCSDT algorithm, they did not always win. On some UCI data sets, such as Ecoli, the new methods actually increased the misclassification and the test cost. In the third experiment, we try to find out in what circumstance, our new methods do not perform well, and what is the reason behind this.

3.3.2 Experiment Results and Discussion

In this section, we present an experimental comparison of the decision tree algorithms presented in the previous section.

The first experiment results are listed in Table 3.5. It lists the key performance measurements such as average misclassification cost and AUC for the KDD-98 and Statlog(heart) data sets when test cost is set to 0. Since test cost is not considered in this experiment, we use "F-measure" to guide the feature selection process of our TCSDT-FS algorithm. The aim of this experiment is to show that on real world data sets and with real misclassification cost, compared to the original TCSDT algorithm, our three new methods can successfully reduce data over-fitting.

Table 3.5 Key performance measurements on KDD-98 and Statlog(heart) when test cost is set to 0

Table 3.5.1 Average Misclassification Cost

Data Set	TCSDT	TCSDT-SM	TCSDT-TP	TCSDT-FS
KDD-98	0.8558	0.6428	0.7889	0.6569
Statlog(heart)	1.0593	0.444	0.5185	0.3556

Table 3.5.2 Area Under ROC (AUC)

Data Set	TCSDT	TCSDT-SM	TCSDT-TP	TCSDT-FS
KDD-98	0.503	0.535	0.515	0.527
Statlog(heart)	0.674	0.717	0.699	0.768

Table 3.5.3 No. of Leaves / Tree Size

Data Set	TCSDT	TCSDT-SM	TCSDT-TP	TCSDT-FS
KDD-98	1653/1862	1653/1862	275/302	459/505
Statlog(heart)	252/288	252/288	19/21	19/23

Table 3.5.4 Average Model Training Time (seconds)

**** The test was performed on a PC with Intel Pentium Dual Core 1.66G Hz CPU and 1GB memory***

Data Set	TCSDT	TCSDT-SM	TCSDT-TP	TCSDT-FS
KDD-98	0.31	0.31	0.32	0.05 (plus 327 seconds feature selection time)
Statlog(heart)	0.006	0.006	0.006	0.003 (plus 5 seconds feature selection time)

From the first experiment, we can draw some conclusions. First, in term of reducing misclassification cost, all our three new methods have achieved lower cost than that of the original TCSDT algorithm. On the KDD-98 data set, TCSDT-SM achieved the lowest cost; TCSDT-FS comes closely the second. Compared to original TCSDT, both of them reduced the misclassification cost by more than 20%. On the Statlog(heart) data set, TCSDT-FS achieved the lowest cost, TCSDT-SM comes the second. Both of them reduced the misclassification cost by more than 50%. Second, by using AUC, a well-recognized measurement in cost-sensitive learning, we can see our three new methods always achieved higher AUC than the original TCSDT algorithm. Third, our two methods, TCSDT-FS and TCSDT-TP have significantly reduced the decision tree size on both data sets. On KDD-98 data set, TCSDT-FS reduced the tree size by more than 70% and TCSDT-TP reduced the tree size by more than 80%. On Statlog(heart) data set, both TCSDT-FS and TCSDT-TP reduced the tree size by more than 90%. Fourth, in terms of the model training time, the proposed new methods, TCSDT-SM and TCSDT-TP are comparable to the original TCSDT algorithm, and TCSDT-FS is much faster because it builds decision trees based on a reduced number of attributes. However, TCSDT-FS takes a lot longer time on feature selection which is not required by other methods. Overall, TCSDT-FS uses more CPU time than other competing methods.

The test results of our second experiment are shown in Table 3.6 and 3.7. Table 3.8 lists the average total cost on selected nine UCI data sets when test cost is set to between 0 and 100. Table 3.7 lists the corresponding results on the t-test. Each *w/t/l* in the table means our two new algorithms at each row wins in *w* data sets, ties in *t* data sets and loses in *l* data sets.

Table 3.6 Average total cost on selected UCI data sets when test cost is set to between 0 and 100

Table 3.6.1 Cost Ratio (FP=100, FN=200)

Data Set	TCSDT	C4.5-CS	TCSDT-2L	TCSDT-Prune	TCSDT-SM	TCSDT-TP	TCSDT-FS
australia	46.77	113.84	48.06	44.63	43.87	44.62	43.98
breast	27.43	75.21	25.94	42.44	24.9	41	23.76
ecoli	33.54	57.89	39.81	38.24	35.84	37.59	34.71
heart	53.14	149.61	54.49	54.68	52.94	50.64	49.87
german_credit	60	180.83	60	60	60	60	60
car	15.51	107.63	15.51	15.51	15.51	15.51	15.51
hypothyroid	14.63	75.84	13.86	15.22	14.08	14.65	14.32
sick	11.85	86.41	11.05	11.52	11.8	11.91	11.42
vowel	18.18	105.62	18.18	18.18	18.18	18.18	18.18

Table 3.6.2 Cost Ratio (FP=100, FN=500)

Data Set	TCSDT	C4.5-CS	TCSDT-2L	TCSDT-Prune	TCSDT-SM	TCSDT-TP	TCSDT-FS
australia	50.3	131.33	51.73	51.17	50	50.01	51.05
breast	38.84	92.14	35.70	41.13	33.98	40.93	33.2
ecoli	39.34	56.4	38.23	39.42	37.64	38.38	38.12
heart	56.13	171.61	55.56	55.56	56.07	55.56	55.56
german_credit	70	209.06	70	70	70	70	70
car	38.42	105.1	38.77	38.77	38.66	37.56	37.14
hypothyroid	32	76.88	28.26	31.54	29.87	30.2	29.98
sick	27.2	85.87	27.43	27.33	26.49	27.2	26.32
vowel	45.45	107.78	45.45	45.45	45.45	45.45	45.45

Table 3.6.3 Cost Ratio (FP=100, FN=1000)

Data Set	TCSDT	C4.5-CS	TCSDT-2L	TCSDT-Prune	TCSDT-SM	TCSDT-TP	TCSDT-FS
australia	54.85	144.82	54.69	54.71	54.33	54.59	54.22
breast	49.1	96.65	47.91	46.95	41.14	46.16	43.15
ecoli	35.9	59.89	44.64	47.04	39.31	44.14	41.57
heart	55.56	189.56	55.56	55.56	55.56	55.56	55.56
german_credit	70	225.31	70	70	70	70	70
car	61.31	101.13	57.1	60.2	57.2	60.51	58.83
hypothyroid	46.73	73.38	42.5	45.22	43.93	45.75	45.88
sick	42.41	87.33	41.51	46.24	40.68	48.11	39.78
vowel	90.91	110.33	90.91	90.91	90.91	90.91	90.91

Table 3.6.4 Cost Ratio (FP=100, FN=2000)

Data Set	**TCSDT**	**C4.5-CS**	**TCSDT-2L**	**TCSDT-Prune**	**TCSDT-SM**	**TCSDT-TP**	**TCSDT-FS**
australia	54.67	171.99	54.67	54.67	54.67	54.67	54.67
breast	70.59	119.29	54.25	57.94	51.02	51.24	50.3
ecoli	43.95	61.59	45.62	49.67	42.2	42.75	43.64
heart	56.98	209.86	55.56	55.56	55.68	55.56	54.24
german_credit	70	236.35	70	70	70	70	70
car	69.38	104.28	66.34	68.1	67.29	67.83	67.45
hypothyroid	57.49	75.54	52.6	56.5	53.98	60.08	55.37
sick	68.44	104.54	64.65	66.32	58.04	66.8	57.63
vowel	90.91	114.85	90.91	90.91	90.91	90.91	90.91

Table 3.7 Summary of t-test when test cost is set to between 0 and 100

Cost Ratio (FP:FN)		**TCSDT**	**C4.5-CS**	**TCSDT-2L**	**TCSDT-Prune**
100:200	TCSDT-SM	5/3/1	9/0/0	4/3/2	5/3/1
	TCSDT-TP	4/3/2	9/0/0	3/3/3	5/3/1
	TCSDT-FS	5/3/1	9/0/0	4/3/2	6/3/0
100:500	TCSDT-SM	7/2/0	9/0/0	4/2/3	6/2/1
	TCSDT-TP	5/3/1	9/0/0	3/3/3	5/3/1
	TCSDT-FS	6/3/0	9/0/0	4/3/1	6/3/0
100:1000	TCSDT-SM	5/3/1	9/0/0	4/3/2	6/3/0
	TCSDT-TP	4/3/2	9/0/0	3/3/3	3/3/3
	TCSDT-FS	5/3/1	9/0/0	4/3/2	5/3/1
100:2000	TCSDT-SM	6/3/0	9/0/0	3/3/3	5/3/1
	TCSDT-TP	5/3/1	9/0/0	2/4/3	3/4/2
	TCSDT-FS	7/2/0	9/0/0	4/3/2	6/3/0

From the second experiment, we can also draw several conclusions. First, for all the data sets we have tested, C4.5-CS always incurs higher cost. This is because when building decision trees, C4.5 totally ignores the test cost for each split, and is very likely to pick up an attribute with high test cost to split. This also proves that error based pruning is not suitable for cost-sensitive classification which requires more accurate probability estimate. Second, our three new methods, TCSDT classification with feature selection, smoothing or threshold pruning outperform the original TCSDT classification on most of the selected UCI data sets across different cost ratios. Third, comparing our methods with Du et al. (2007)'s two pre-pruning methods, our methods outperform TCSDT-Prune, and comparable to TCSDT-2L. Forth, among our three new methods, feature selection and smoothing method generally perform better than threshold pruning on most of the data sets across different cost ratios in terms of reducing the total cost, especially when cost ratios are high (for example, 20) and on data sets with more skewed class distribution (for example, sick), but no algorithm win all the time on all data sets.

Why did not our new methods perform well on some UCI data sets? We try to answer this question in the next experiment. First we examine the test result on the UCI data sets. We take the test in Table 3.6.1 as an example. In this test, our two best performing methods, TCSDT-SM and TCSDT-FS won on 5 data sets, drew on 3 data sets (German_credit, Car and Vowel) and lost on 1 data set (Ecoli).

On all the 3 data sets, German_credit, Car and Vowel, the original TCSDT algorithm generates a decision tree with a single node. This is caused by the limitation of the original TCSDT algorithm. Our new methods are not designed to fix this limitation.

Ecoli is a small, low-dimension data set. It contains 6 attributes and 332 examples. In this experiment, we select this data set, together with KDD-98, Statlog(heart) and Sick. We also include a new test method, TCSDT-Train, which is the original TCSDT algorithm but using the full data set (contrast to the other methods tested in this paper, which all use 10-folder cross validation) to build classification model and classify examples. The difference of test result between the original TCSDT and the TCSDT-Train shows how much data over-fitting of the original TCSDT on a test data set. To make it consistent, we use the same test setting as the previous experiments. The cost ratio for Sick and Ecoli is set to FP=100, FN=200. We then measure the misclassification errors, the average total cost and the tree size. The full test results are shown in Table 3.8.

Table 3.8 Performance measurements on selected real world and UCI data sets

Table 3.8.1 Misclassification Errors (Correct/Incorrect)

Data Set	**TCSDT**	**TCSDT-Train**
KDD-98	2537/463	2999/1
Statlog(heart)	171/99	267/3
Sick	2437/1335	2776/996
Ecoli	320/12	321/11

Table 3.8.2 Average Total Cost

Data Set	TCSDT	TCSDT-Train
KDD-98	0.8558	0.0002
Statlog(heart)	1.0593	0.0111
Sick	11.85	10.97
Ecoli	33.54	33.53

Table 3.8.3 No. of Leaves / Tree Size / Tree Level

Data Set	TCSDT and TCSDT-Train
KDD-98	1653/1862/6
Statlog(heart)	252/288/7
Sick	8/12/2
Ecoli	6/8/2

From this experiment, we can clearly see that on the real world data set, KDD-98 and the Statlog(heart), the issue of TCSDT data over-fitting is very serious. On KDD-98, TCSDT generates a decision tree which misclassifies 463 test examples, and its average misclassification cost is 0.8558, while the TCSDT-Train generates a decision tree which only misclassifies 1 test example, and its average misclassification cost is 0.0002. On Statlog(heart), TCSDT generates a decision tree which misclassifies 99 test examples, and its average misclassification cost is 1.0593, while the TCSDT-Train generates a decision tree which only misclassifies 3 test examples, and its average misclassification cost is 0.0111. Since these two data sets contain large amount of noisy data and irrelevant attributes (for example, KDD-98 has 481 attributes!), the original TCSDT algorithm generates very large decision trees for these two data sets and they perform well on training data but badly on unseen instances. In this situation, our new methods, TCSDT-SM and TCSDT-FS can effectively reduce the data over-fitting by providing better, smoother probability estimation, removing irrelevant attributes and noisy data, and generating smaller, more stable decision trees.

On KDD-98 data set, TCSDT generates a decision tree with 1653 leaves, on average there are only less than 2 examples in each leaf. Since the number of examples in each leaf is so small, the probability estimation provided by this decision tree on unseen data is very poor. TCSDT-SM can make a significant improvement in such a situation by providing smoother and much stable probability estimation. TCSDT-FS on the other hand, removes most of the irrelevant/disturbing attributes and generates a much smaller and more stable decision tree which improves the classifier performance on unseen data.

On Statlog(heart) data set, exactly the same scenario. TCSDT generates a decision tree with 252 leaves, on average there is only 1 example in each leaf. Of course the probability estimation provided by this decision tree on unseen data is very poor too. Compared to the original TCSDT algorithm, our new methods made a significant improvement on this data set. Both TCSDT-SM and TCSDT-FS reduced the average misclassification cost by more than 50%, and TCSDT-FS reduced the tree size by more than 90%.

However, most of the UCI data sets are small, low-dimension and relatively clean. On these data sets, the TCSDT data over-fitting issue is not that obvious. Therefore, the space of improvement that our new methods can make is not as big as that we have achieved on KDD-98 and Statlog(heart). Sometime, our new methods actually downgrade the performance. For example, on Ecoli data set, TCSDT generates a decision tree which misclassifies 12 test examples, and its average misclassification cost is 33.54, while the TCSDT-Train generates a decision tree which misclassifies 11 test examples, and its average misclassification cost is 33.53. In this case, the performance of the TCSDT and the TCSDT-Train are almost the same. The TCSDT data over-fitting issue on Ecoli data set can be ignored. The size of the decision tree generated by TCSDT is quite small too. It has only 2 levels and 6 leaves. Therefore, our new methods did not make any improvements on this data set. However, on the other UCI data sets such as Sick, our new methods can make noticeable improvements in terms of reducing the total cost of misclassification and test.

3.3.3 Experimental Analysis

TCSDT, like other top down decision tree algorithms, suffers from the problem of over-fitting to the training data. When applying it to some real world data sets which contain noisy data and a large number of attributes, the resulting trees are often very large and over-specified, with very low predictive power for unseen data. In addition, traditional error based pruning is not suitable to TCSDT. The two pre-pruning methods developed by Du et al. (2007) work well on some data sets but suffer from some common issues with the pre-pruning approach: the stopping threshold is not easy to get right for different data sets. If it is set too high, the decision tree may terminate division before the benefits of subsequent splits become evident. If it is set too low, little simplification can be achieved.

In our experiments, we've demonstrated that our new methods can reduce TCSDT data over-fitting from the following perspectives:

- **Remove noisy data and irrelevant/disturbing attributes** from the training data. For example, the KDD-98 data set has 481 attributes. Most of them are irrelevant or useless to making predictions. Our TCSDT-FS algorithm removes most of these attributes and results in a much smaller and simpler tree with lower misclassification cost. The tree size has been reduced by more than 70% and the misclassification cost has been reduced by more than 20%.
- **Improve class membership probability estimation**. Our TCSDT-SM algorithm performs extremely well on both KDD-98 data sets and other UCI data sets across different cost ratios. Our other two methods, TCSDT-FS and TCSDT-TP also indirectly improve the class membership probability estimation by reducing decision tree levels and number of leaves. One thing worth mentioning is the TCSDT-2L algorithm, developed by Du et al. (2007), can also effectively simplify the decision tree and improve the class membership probability estimation. It performs well on most of the selected UCI data sets.
- **Cost reduction**. The major difference between TCSDT and other traditional decision tree algorithms is that TCSDT is cost-sensitive. Our new methods, specifically TCSDT-FS and TCSDT-TP, are designed to consider the cost of misclassification and test when selecting features and pruning tree. They are more effective than other cost-insensitive methods such as error based pruning when being applied on TCSDT.

However, our experiments also show that on some small, low-dimension and relatively clean UCI data sets, the proposed new methods do not always make improvement. Further tests and inspection uncover that the difference is caused by the following factors:

- **Data sets**. Real world data sets always contain noisy data and a large number of attributes. Most of these attributes are irrelevant and distractive to making a good decision. TCSDT algorithm is easily affected by the noise and tends to build decision trees which are over-specified. They usually perform extremely well on training data but poorly on unseen data. Our new methods can effectively tackle this issue and reduce data over-fitting. On the other hand, some UCI data sets are small, low-dimension and clean. On these data sets, the TCSDT data over-fitting issue is not that serious. The original TCSDT algorithm can build decision trees which perform really well on unseen data too (for example, on Ecoli data set). In this case, the space of improvement that our new methods can make is limited.
- **Resulting decision trees**. For data sets with noise and many attributes, TCSDT tends to build very large decision trees which give a perfect fit for the training data, but lack of ability to generalize. For example, as we demonstrated in the last experiment, on Statlog(heart) data set, the original TCSDT algorithm builds a decision tree with 7 levels and 252 leaves, on average each leaf only contains 1 example. The probability estimation provided by this decision tree on unseen data is poor and not reliable. Compared to the original TCSDT algorithm, our new methods made some significant improvements in terms of providing better probability estimation, reducing tree size and misclassification cost. However, on another data set, Ecoli, the original TCSDT algorithm builds a small and effective decision tree which only has 2 levels and 6 leaves. When applied to unseen data, this decision tree only misclassifies 12 examples and its average total cost is 33.54. The result is very close to the test when applying this decision tree on the training data, in which it misclassifies 11 examples and its average total cost is 33.53. In this case, it is very hard for our new methods to make further improvement.
- **TCSDT algorithm**. On some UCI data sets (for example, German_credit, Car and Vowel), the TCSDT algorithm generates decision trees which contains only a single node. This is caused by the limitation of the TCSDT

algorithm itself. Like other top down decision tree build algorithms, for each potential split, TCSDT chooses a locally optimal attribute without backtracking. It is very efficient but the resulting tree may not be globally optimal. When data set is small and test cost is high (compared to misclassification cost), quite often it cannot find any suitable attributes to split and only generates a decision tree with a single node. The proposed new methods are not designed to fix this issue. In our future work, we plan to further develop the TCSDT algorithm to make it more robust. For example, adding backtracking function to the tree building process.

3.4 Conclusions

In this Chapter, we applied three simple and efficient methods, feature selection, smoothing and threshold pruning, on the TCSDT algorithm to reduce data over-fitting. Our methods modify the TCSDT algorithm by introducing feature selection process before building decision tree, smoothing and pruning process before calculating the class probability estimate for each decision tree leaf. Our experiments show that the modified algorithms outperform the original TCSDT and other competing algorithms on the selected data sets (including a real world data set, KDD-98 and ten UCI data sets) across different cost ratios.

Chapter 4 Cost-sensitive K-Nearest Neighbours Classification

K-Nearest Neighbors (KNN) classification finds the *K* nearest neighbours of a test example in training data and then predicts the class of the example as the most frequent one among the neighbours. KNN is one of the most popular and well-studied classification algorithms in data mining and machine learning. Although in recent years cost-sensitive learning attracted significant attention from data mining researchers, and lots of research had been done to make many traditional error based classifiers cost-sensitive. Most of these research focused on model based classifiers, such as Decision Tree, Naïve Bayes and Support Vector Machine (SVM), little research was conducted to thoroughly investigate the effective approaches which make instance based classifiers, such as KNN, cost-sensitive. In this Chapter we propose two simple and effective approaches, Direct-CS-KNN and Distance-CS-KNN to build cost-sensitive KNN classifiers. In order to deal with some practical issues on Direct-CS-KNN and Distance-CS-KNN, we also propose several additional enhancements (smoothing, minimum-cost *K* value selection, feature selection and ensemble selection) to improve the performance of the new algorithms. Our experiment results show that the proposed new cost-sensitive KNN algorithms can effectively reduce misclassification cost, often by a large margin. And they consistently outperform CS-4.5 (cost-sensitive C4.5) on the UCI data sets we tested.

4.1 Introduction

KNN classification is one of the most popular and widely used instance-based learning algorithms. Different from model-based classification algorithms (i.e. training models from a given dataset and then predicting test examples with the models), it needs to store the training data in memory in order to find the “nearest neighbours” to answer a given query. To make KNN cost-sensitive, in this Chapter, we propose two simple but effective approaches, Direct-CS-KNN and Distance-CS-KNN aimed at minimising the misclassification cost. In order to handle some challenges and difficulties encountered in our new algorithms, we also propose several additional methods - smoothing, minimum-cost *K* value selection, feature selection and ensemble selection, to further improve the performance of our new cost-sensitive KNN algorithms. These improvements can apply to both Direct-CS-KNN and Distance-CS-KNN.

Our experiment results show that the proposed new cost-sensitive KNN algorithms can effectively reduce misclassification cost, often by a large margin. And they consistently outperform CS-4.5 (cost-sensitive C4.5) on the selected UCI data sets.

4.2 KNN Classification

KNN classification is an instance based learning algorithm which stores the whole training data in memory to compute the most relevant data to answer a given query. The answer to the query is the class represented by a majority of the *K* nearest neighbours. There are three key elements of this approach: a set of labelled examples, a distance function for computing the distance between examples, and the value of *K* - the number of nearest neighbours. To classify an unlabelled example, the distance of this example to the labelled examples is calculated, its *K*-nearest neighbours are identified, and the class labels of these nearest neighbours are then used to classify the unlabelled example (Wu et al. 2008).

The choice of the distance function is an important consideration in KNN. Although there are other options, most instance based learners use Euclidean distance which is defined as below:

$$Dist(X,Y) = \sqrt{\sum_{i=1}^{D} (Xi - Yi)^2}$$

, where X and Y are the two examples in data set, and Xi and Yi (i = 1 .. D) are their attributes.

Once the nearest-neighbour list is selected, the test example can be classified based on the following two voting methods:

1. Majority voting:

$$y' = \arg\max_v \sum_{(xi,yi)\in Dz} \delta(v, yi)$$

2. Distance-Weighted Voting:

$$y' = \arg\max_v \sum_{(xi,yi)\in Dz} w_i \delta(v, yi),$$

where $w_i = 1/d(x', xi)^2$

There are several key issues that affect the performance of KNN. One is the choice of *K*. If *K* is too small, then the result could be sensitive to noisy data. If *K* is too large, then the selected neighbours might include too many examples from other classes. Another issue is how to determine the class labels, the simplest method is to take a majority vote (method 1), but this could be an issue if the nearest neighbours vary widely in their distance and the closer neighbours more reliably indicate the class of the test example. The other issue of method 1 is that it is hard to deal with cost-sensitive learning and imbalanced data sets. A more sophisticated approach, which is less sensitive to the choice of *K*, weights each example's vote by its distance (method 2), where the weight factor is often taken to be the reciprocal of the squared distance ($w_i = 1/d(x', xi)^2$).

KNN classifiers are lazy learners, unlike eager learners (e.g. Decision Tree and SVM), KNN models are not built explicitly. Thus, building the model is cheap, but classifying unknown data is relatively expensive since it requires the computation of the *K*-nearest neighbours of the examples to be labelled. This, in general, requires computing the distance of the test examples to all the examples in the labelled set, which can be expensive particularly for large training sets.

4.3 Making KNN Cost-sensitive - the Proposed Approach

In this Chapter, we focus on binary classification problems. We propose two approaches to make KNN classifier sensitive to misclassification cost, and several additional methods to further improve the cost-sensitive KNN classifier performance.

4.3.1 Direct Cost-sensitive KNN

Our first approach is quite simple. We use KNN algorithm to train a traditional non-cost-sensitive classifier. After the *K* nearest neighbours is selected, we calculate class probability estimate using the below formula:

$$\mathrm{Pr}(i \mid x) = \frac{Ki}{K},$$

where Ki is the number of selected neighbours whose class label is i. Using the above probability estimate and formula (2.1) specified in Chapter 2, we can directly compute the optimal class label for each test example. We call this approach **DirectCS-KNN**.

In traditional cost-blind KNN classification, the *K* value is either fixed or selected using cross validation. When the *K* value is fixed, most of times it is quite small, such as 3, 5, 7 etc. The KNN classifier performance is not impacted a lot by the variation of the *K* value. However, the aim of our DirectCS-KNN is minimizing misclassification cost, the probability estimate (not the error rate) generated by the original KNN classifier is more important. In this case, the selection of an appropriate *K* value plays a critical role in terms of building a statistically stable cost-sensitive KNN classifier which can produce better probability estimate and reduce misclassification cost.

In this Chapter, we will test the following three ways of selecting the K value:

- Fixed value
- Cross validation
- Choose the *K* value which minimizes the misclassification cost in training set

Although our DirectCS-KNN approach is straightforward and easy to implement, it has the following shortcomings:

- If the *K* value selected is too small, the probability estimation provided by the KNN (*Ki/K*) is statistically unstable, it will cause data over-fitting and increase the misclassification cost to the test examples

- In many real world data sets, noise is often expected in the training examples, and KNN algorithm is particularly sensitive to the noisy data, therefore generates very poor probability estimate

As we described in Chapter 2, several methods have been proposed for obtaining better probability estimate from traditional cost-blind classifiers in direct cost-sensitive learning. Zadrozny and Elkan (2001) proposed to use an un-pruned decision tree and transform the scores of the leaves by smoothing them. They call this method m-estimation. The similar method can be applied to our DirectCS-KNN approach. In order to make DirectCS-KNN more stable and further reduce misclassification cost, in this Chapter, we propose two changes to the original m-estimation: First we use cross valuation to determine the *m* value for different data sets. It is more proper than a fixed value. Second, we use the smoothed probability estimate (together with cost matrix) at the step of determining the *K* value.

Now let's look at the m-estimation formula we specified in Chapter 2 again. In order to explain the impact of m-estimation to the performance of DirectCS-KNN approach, we represent the formula in a slightly different way:

$$\Pr(i|x) = \left(\frac{K}{K+m}\right) \times \left(\frac{K_i}{K}\right) + \left(\frac{m}{K+m}\right) \times b$$

As we can see from this formula, the value *m* controls the balance between relative frequency and prior probability. It has the following impacts:

- To the noisy data, with m-estimation, *m* can be set higher so that the noisy value for *Ki/K* plays a less important role in the final estimation and the impact of noisy data is reduced.
- When the *K* value is small, without smoothing, the probability estimation provided by the selected neighbors (*Ki/K*) is statistically unstable. However, with m-estimation, *K/(K+m)* closes to 0 and *m/(K+m)* closes to 1, so that the probability estimation is shifted towards the base rate *b*. It works particularly well on data sets with skew class distribution.

In the experiment section, we apply this smoothing method to the Direct-CS-KNN and Distance-CS-KNN (specified in section 4.3.2), and compare this approach to the other proposed variations of Cost-sensitive KNN algorithms.

4.3.2 KNN with Cost-sensitive Distance Function

The second approach involves modifying the distance function of the KNN algorithm. Let's review the distance-weighted voting function in section 4.2:

$$y' = \arg\max_v \sum_{(xi,yi)\in Dz} w_i \delta(v, yi),$$

where $w_i = 1/d(x', xi)^2$

Based on the second formula, in a binary decision case, we assume that the distance-weight of a test example to a positive training example is *Wp,* and the distance-weight of the same test example to a negative training example is *Wn.*

When misclassification cost is not considered, the training examples with the highest *W* values will be selected as the nearest neighbors regardless of their class labels. However, in a cost-sensitive situation, the cost of false positive (FP) and the cost of false negative (FN) might be very different. Selecting a nearest neighbor with different class labels incurs different potential cost. To simplify the case, we assume that the cost of true positive (TP) and true negative (TN) are both 0. In cost-sensitive learning, the purpose is to minimize the misclassification cost instead of errors. Now we calculate the potential cost (C_p) of selecting a positive nearest neighbor is *FP * Wn*. And the potential cost (C_n) of selecting a negative nearest neighbor is *FN * Wp*. The training examples with the lowest potential cost should be selected as the nearest neighbors.

We replace the distance-weighted voting function in the original KNN algorithm with this new cost-sensitive approach. The last step is to use the modified classifier to predict class labels for all test examples. Be aware that for each test example, after all its nearest neighbors are selected, the above cost-sensitive distance-weighted voting approach and the cost matrix will still be used to calculate the class label which minimizes the misclassification cost. The detail of the algorithm is described below:

For each test example, the total distance-weight of all positive neighbors is *Wpa,* and the total distance-weight of all negative neighbors is *Wna*, where:

Wpa = *W1* + *W2* + *W3* + …… + *Wk* (k is the number of positive neighbors)
Wna = *W1* + *W2* + *W3* + …… + *Wj* (j is the number of negative neighbors)

In this case we can define the probability of labeling an unlabelled example to *P* is:

$P_p = Wpa/(Wpa+Wna)$

And the probability of labeling an unlabelled example to *N* is:

$P_n = Wna/(Wpa+Wna)$,

where $P_p + P_n = 1$. Now we calculate the potential cost (C_p) of labeling this test example to *P* is $FP * P_n$. And the potential cost (C_n) of labeling this test example to *N* is $FN * P_p$.

If $C_p > C_n$, the unlabelled example is classified as N, and the probability of this prediction is $C_p/(C_p + C_n)$. Otherwise, the unlabelled example is classified as *P*, and the probability of this prediction is $C_n/(C_p + C_n)$. We call this approach **Distance-CS-KNN**.

4.3.3 KNN with Cost-sensitive Feature Selection

Most of real world data sets contain noisy data and irrelevant/disturbing features. One of the shortcomings of instance based learning algorithm such as KNN is that they are quite sensitive to noisy data and irrelevant/disturbing features, especially when the training data set is small. This issue can cause poor classification performance on unseen data. Feature selection strategy can certainly help in this situation.

As mentioned in Chapter 2, there are many different feature selection methods developed. They fall into two categories: the filter approach and the wrapper approach. As per the study of Kohavi and John (1997), the wrapper approach generally perform better than the filter approach, and some significant improvement in accuracy was achieved on some data sets for Decision Tree algorithm and Naïve Bayes algorithm using the wrapper approach.

The wrapper approach proposed by Kohavi and John (1997) conducts a search in the space of possible parameters. Their search requires a state space, an initial state, a termination condition and a search engine. The goal of the search is to find the state with the highest evaluation, using a heuristic function to guide it. They use prediction accuracy estimation as both the heuristic function and the evaluation function. They've compared two search engines, hill-climbing and best-first, and found that the best-first search engine is more robust, and generally performs better, both in accuracy and in comprehensibility as measured by the number of features selected.

In this Chapter, we apply this wrapper approach to both of our Direct-CS-KNN and Distance-CS-KNN algorithm to improve classifier performance. In cost-sensitive learning, the ultimate goal is to minimize the misclassification cost. We cannot simply apply Kohavi and John's wrapper approach directly to our cost-sensitive KNN algorithms. Therefore, we propose a variation of Kohavi and John's feature selection wrapper. The main difference is using misclassification cost instead of error rate as both the heuristic function and the evaluation function.

The other settings in our experiment are similar: The search space we chose is that each state represents a feature subset. So there are n bits in each state for a data set with n features. Each bit indicates whether a feature is selected (1) or not (0). We always start with an empty set of features. The main reason for this setup is computational. It is much faster to find nearest neighbours using only a few features in a data set. We chose the best-first search algorithm as our search engine. The following summary shows the setup of our cost-sensitive feature selection problem for a simple data set with three features:

Cost-sensitive KNN feature selection setup for data set with three features:

State:	A Boolean vector, one bit per feature
Initial state:	A empty set of features (0,0,0)
Search space:	(0,0,0) (0,1,0) (1,0,0) (0,0,1)
	(1,1,0) (0,1,1) (1,0,1) (1,1,1)

Search engine:	Best first
Evaluation function:	Misclassification Cost

4.3.4 KNN with Cost-sensitive Stacking

KNN classifier is very popular in many real world applications. The main reason is that the idea is straightforward and easy to implement. It is simple to define a dissimilarity measure on the set of observations. However, handling the parameter *K* could be tricky and difficult, especially in cost-sensitive learning. In this Chapter, we propose an ensemble based method, more specific, **Cost-sensitive Stacking**, to handle the parameter *K*.

As we mentioned in Chapter 2, ensemble selection is a well-developed and very popular meta learning algorithm. It tends to produce a better result when there is a significant diversity among the classification models and parameters. Stacking is an ensemble technique whose purpose is to achieve a generalization accuracy (as opposed to learning accuracy) , and make it as high as possible. The central idea is that one can do better than simply list all predictions as to the parent functions which are consistent with a learning set. One can also use in-sample/out-of-sample techniques to find a best guesser of parent functions. There are many different ways to implement stacking. Its primary implementation is as a technique for combining generaliser, but it can also be used when one has only a single generaliser, as a technique to improve that single generaliser (Wolpert 1992).

Our proposed Cost-sensitive Stacking algorithm works as below:

Adding multiple cost-sensitive KNN classifiers with different *K* values to the stack, and learning a classification model on each; estimating class probability for each example by the fraction of votes that it receives from the ensemble; using formula (2.1) to re-label each training example with the estimated optimal class; and reapplying the classifier to the relabeled training set. The idea is similar to bagging approach used in MetaCost (Domingos 1999).

4.4 Experimental Evaluation

4.4.1 Experiment Setup

The main purpose of this experiment is to evaluate the performance of the proposed cost-sensitive KNN classification algorithms with feature selection and stacking, by comparing their misclassification cost and other key performance measurements such as AUC across different cost ratios (FN/FP) against another popular classification algorithm, C4.5 with Minimum Expected Cost. All these algorithms are implemented in Weka (Witten and Frank 2000), and they are listed in Table 4.1 below.

Table 4.1 List of Cost-sensitive Algorithms and Abbreviations

#	Method	Abbreviation	Base Classifier
1	Direct Cost-sensitive KNN	DirectCS-KNN	KNN
2	Direct Cost-sensitive KNN with Smoothing	DirectCS-KNN-SM	KNN
3	Direct Cost-sensitive KNN with K value selection	DirectCS-KNN-CSK	KNN
4	Distance Cost-sensitive KNN	DistanceCS-KNN	KNN
5	Distance Cost-sensitive KNN with Feature Selection	DistanceCS-KNN-FS	KNN
6	Distance Cost-sensitive KNN with Stacking	DistanceCS-KNN-STK	KNN
7	C4.5 with Minimum Expected Cost	CS-C4.5	C4.5

Please note that we will be conducting three experiments using the above algorithms and six data sets chosen from UCI repository. The details of these data sets are listed in Table 4.2. The imbalance level (the ratio of major class size to minor class size) in these data sets varies from 1.02 (Waveform-5000) to 8.8 (Page-blocks).

Table 4.2 Summary of the Data set Characteristics

Dataset	No. of attributes	No. of examples	Class distribution (P/N)
Statlog (heart)	14	270	120/150
Credit-g	21	1000	300/700
Diabetes	9	768	268/500
Page-blocks	11	5473	560/4913
Spambase	58	4601	1813/2788
Waveform-5000	41	3347	1655/1692

We conduct three experiments on above datasets with different cost matrixes:

In the first experiment, we use the UCI Statlog(heart) data set. We evaluate the classifier performance by calculating the misclassification cost and AUC generated by the different variations of the Direct Cost-sensitive KNN algorithm and CS-C4.5. Below is a brief description of the Statlog(heart) data set:

- The Statlog(heart) data set is one of a few data sets in UCI library with recommended cost matrix. The cost matrix is normalized and the cost ratio (FN/FP) is set to 5. The cost of TP and TN are both set to 0. This data set has been used extensively in cost-sensitive learning research previously.

In the second experiment, we still use the Statlog(heart) data set to conduct the test. We evaluate the performance of our two new cost-sensitive algorithms, DirectCS-KNN and DistanceCS-KNN by calculating their misclassification cost and AUC.

In the third experiment, five UCI data sets are used to perform the test. The misclassification cost FP is always set to 1, and FN is set to an integer varying from 2 to 20 (2, 5, 10, 20 respectively). We assume that the misclassification of the minority class always incurs a higher cost. This is to simulate real-world scenarios in which the less frequent class is the more important class. The cost of TP and TN are both set to 0. We evaluate our DistanceCS-KNN classifier (and its variations) performance against CS-C4.5 by comparing their average misclassification cost.

All of the three experiments are repeated for 10 times and ten-folder cross validation method is used in all tests to prevent over-fitting data.

4.4.2 Experiment Results and Discussion

In this section, we present an experimental comparison of the cost-sensitive KNN and other competing algorithms presented in the previous section.

The first experiment results are listed in Table 4.3. It lists the key performance measurements such as average misclassification cost and AUC for the Statlog(heart) data set. The aim of this experiment is to show that on this popular UCI data set and with the recommended misclassification cost, we can achieve better performance on DirectCS-KNN algorithm through smoothing (DirectCS-KNN-SM) and *K*-value selection with minimum-cost (DirectCS-KNN-CSK) approach. In this experiment, we chose a fixed *K* value (*K*=5) for both DirectCS-KNN and DirectCS-KNN-SM, and automatically select *K* value with minimum-cost (on training data) for DirectCS-KNN-CSK.

Table 4.3 Key performance measurements on Statlog(heart)

Table 4.3.1 Average Misclassification Cost

Data Set	**CS-C4.5**	**DirectCS-KNN**	**DirectCS-KNN-SM**	**DirectCS-KNN-CSK**
Statlog(heart)	0.6704	0.3815	0.3605	0.3556

Table 4.3.2 Area Under ROC (AUC)

Data Set	**CS-C4.5**	**DirectCS-KNN**	**DirectCS-KNN-SM**	**DirectCS-KNN-CSK**
Statlog(heart)	0.759	0.744	0.763	0.768

From the first experiment, we can draw some conclusions. First, in term of reducing misclassification cost, our three new methods have achieved lower cost than CS-C4.5, all by a large margin. Compared to CS-C4.5, DirectCS-KNN reduced the misclassification cost by 43%, and both DirectCS-KNN-SM and DirectCS-KNN-CSK reduced the misclassification cost by more than 46%. Second, by using AUC, a well-recognized measurement in cost-sensitive learning, we can see our two new methods, DirectCS-KNN-SM and DirectCS-KNN-CSK achieved higher AUC than CS-C4.5, but the AUC of our DirectCS-KNN is slightly lower than CS-C4.5. Third, among the four algorithms we tested, DirectCS-KNN-SM and DirectCS-KNN-CSK always perform better in terms of achieving lower misclassification cost and higher AUC. Overall, DirectCS-KNN-CSK is the best of the four algorithms.

In the second experiment, we still use Statlog(heart) data set with the recommend cost matrix. This time we focus on our two new algorithms, DirectCS-KNN and

DistanceCS-KNN. We evaluate their performance by comparing their misclassification cost and AUC. Since the first experiment showed that smoothing and *K*-value selection with minimum-cost methods can reduce the misclassification cost of the DirectCS-KNN classifier, we apply both methods to the new cost-sensitive KNN algorithms, DirectCS-KNN and DistanceCS-KNN to achieve better performance. The test results are shown in Table 4.4 below.

Table 4.4 Key performance measurements on Statlog(heart)

Table 4.4.1 Average Misclassification Cost

Data Set	**DirectCS-KNN**	**DistanceCS-KNN**
Statlog(heart)	0.3512	0.344

Table 4.4.2 Area under ROC (AUC)

Data Set	**DirectCS-KNN**	**DistanceCS-KNN**
Statlog(heart)	0.7642	0.7688

The second experiment is simple and straightforward, it shows that our modified cost-sensitive KNN algorithm, DistanceCS-KNN performs better than the more naïve, straightforward DirectCS-KNN algorithm. It reduces misclassification cost and increases AUC. This experiment sets up a good foundation for the next experiment in this chapter. In our last experiment, we will mainly focus on the DistanceCS-KNN algorithm and its variations.

The test results of our last experiment are shown in Table 4.5 and 5.6. Table 4.5 lists the average misclassification cost on selected five UCI data sets. Table 4.6 lists the corresponding results on the t-test. Each *w/t/l* in the table means our new algorithm, DistanceCS-KNN and its variations, at each row wins in *w* data sets, ties in *t* data sets and loses in *l* data sets, against CS-C4.5. Similar to the second experiment, we applied both smoothing and *K*-value selection with minimum-cost methods on our DistanceCS-KNN algorithm and its variations.

Table 4.5 Average misclassification cost on selected UCI data sets

Table 4.5.1 Cost Ratio (FP=1, FN=2)

Data Set	DistanceCS-KNN	DistanceCS-KNN-FS	DistanceCS-KNN-STK	CS-C4.5
Diabetes	0.3758	0.3596	0.3633	0.3828
Credit-g	0.428	0.417	0.402	0.435
Page-blocks	0.0607	0.0592	0.0585	0.0422
Spambase	0.1091	0.1044	0.0993	0.1052
Waveform-5000	0.1341	0.1315	0.1298	0.1951

Table 4.5.2 Cost Ratio (FP=1, FN=5)

Data Set	DistanceCS-KNN	DistanceCS-KNN-FS	DistanceCS-KNN-STK	CS-C4.5
Diabetes	0.5573	0.536	0.5352	0.6003
Credit-g	0.598	0.5815	0.582	0.77
Page-blocks	0.0965	0.0896	0.0838	0.0846
Spambase	0.2006	0.1937	0.1864	0.2121
Waveform-5000	0.1637	0.1596	0.1562	0.3756

Table 4.5.3 Cost Ratio (FP=1, FN=10)

Data Set	DistanceCS-KNN	DistanceCS-KNN-FS	DistanceCS-KNN-STK	CS-C4.5
Diabetes	0.5898	0.5832	0.5869	0.8268
Credit-g	0.717	0.6756	0.62	1.043
Page-blocks	0.1214	0.1163	0.1031	0.1297
Spambase	0.3019	0.2836	0.2712	0.3512
Waveform-5000	0.1933	0.1896	0.1815	0.611

Table 4.5.4 Cost Ratio (FP=1, FN=20)

Data Set	DistanceCS-KNN	DistanceCS-KNN-FS	DistanceCS-KNN-STK	CS-C4.5
Diabetes	0.7591	0.725	0.717	0.9635
Credit-g	0.939	0.822	0.8161	1.258
Page-blocks	0.1782	0.1665	0.1546	0.1838
Spambase	0.4027	0.3817	0.3552	0.5781
Waveform-5000	0.1963	0.1915	0.1848	1.0678

Table 4.6 Summary of t-test

Cost Ratio (FP:FN)		CS-4.5-CS
1:2	DistanceCS-KNN	3/0/2
	DistanceCS-KNN-FS	4/0/1
	DistanceCS-KNN-STK	4/0/1
1:5	DistanceCS-KNN	4/0/1
	DistanceCS-KNN-FS	4/0/1
	DistanceCS-KNN-STK	5/0/0
1:10	DistanceCS-KNN	5/0/0
	DistanceCS-KNN-FS	5/0/0
	DistanceCS-KNN-STK	5/0/0
1:20	DistanceCS-KNN	5/0/0
	DistanceCS-KNN-FS	5/0/0
	DistanceCS-KNN-STK	5/0/0

From the last experiment, we can also draw several conclusions. First, for all the data sets we have tested, our cost-sensitive KNN algorithms generally perform better than CS-C4.5, the higher the cost ratio, the better our new algorithms perform. This is because CS-C4.5 ignores misclassification cost when building decision tree, it only considers cost at classification stage, while our cost-sensitive KNN algorithms consider misclassification cost at both classification stage and the stage of calculating distance weight. Second, our two new improvements, DistanceCS-KNN-FS and DistanceCS-KNN-STK outperform the original DistanceCS-KNN algorithm on most of the selected UCI data sets across different cost ratios. Third, DistanceCS-KNN-STK is the best among all the four algorithms we tested, it is very stable and performs better than other competing algorithms across different cost ratios.

4.5 Conclusions

In this Chapter, we studied the KNN classification algorithm in the context of cost-sensitive learning. We proposed two approaches, DirectCS-KNN and DistanceCS-KNN, to make KNN classifier sensitive to misclassification cost. We also proposed several methods (smoothing, minimum-cost *K* value selection, cost-sensitive feature selection and cost-sensitive stacking) to further improve the performance of our cost-sensitive KNN classifiers. We designed three experiments to demonstrate the effectiveness and performance of our new approaches step by step. The experimental results show that compared to CS-C4.5, our new cost-sensitive KNN algorithms can effectively reduce the misclassification cost on the selected UCI data across different cost ratios.

Chapter 5 Cost-Sensitive Classification with Multiple Cost Units

Existing test cost-sensitive learning algorithms simply minimize the sum of misclassification cost and test cost. This makes the test cost always a dominant factor if it is not well set with suitable units. In this Chapter we propose an objective-resource framework for minimizing the misclassification cost (referred to objective cost) subjecting to given budgets for other costs (referred to resource costs), and apply the objective-resource framework to train a cost-sensitive decision tree by utilizing historical data. We conduct experiments for evaluating our algorithms with six benchmark UCI datasets, and demonstrate that our proposed approach outperform a group of existing cost-sensitive learning algorithms in a cost/budget-changing environment which mirrors the pressing demand of minimizing the objective cost while controlling the resource costs in medical diagnosis domain.

5.1 Introduction

Cost-sensitive learning is powerful for capturing the importance of the minority class in, such as medical diagnosis, modelling imbalanced/skewed data, resource-bounded problem solving, and enhancing the distinguishability of minority classes from majority ones.

From an applied context of clinical diagnosis domain, the goal is to obtain a satisfied accuracy with minimal test cost. However, existing cost-sensitive learning algorithms simply minimize the sum of misclassification cost and test cost. It brings two problems: (1) How to convert those diverted costs into a unified unit/scale? (2) The test costs can always dominate the minimization cost if the test costs are not well set with a unified unit. Qin et al. (2004) raised these problems and proposed a general target-resource framework involving multiple-unit costs, but their approach still aims to minimize the sum of misclassification cost and test cost. Here we redesign the framework which minimizes the misclassification cost (in a unique cost unit, referred to *objective cost*) subjecting to given budgets of the test costs (referred to *resource costs*). Specifically, we train a cost-sensitive decision tree by distinguishing misclassification cost from resource costs. The new framework is useful in resource-bounded classification problem solving, and enhance the distinguishability of minority classes from the majority ones.

As per Turney (2000), there are nine major types of cost involved in cost-sensitive learning, including misclassification cost, test cost, teacher cost, computation cost, intervention cost, unwanted achievement cost, human-computer interaction cost, cost of cases, and cost of instability. With the objective-resource framework, the objective cost is the misclassification cost and others are the resource costs. In real world applications, resource costs may include other costs such as the test cost, the teacher cost, the computation cost etc.

In addition to the objective-resource framework, we also develop a cost-sensitive decision tree algorithm which utilizes historical data. In many real world applications, historical data can assist on minimizing different resource costs. However, it is difficult to incorporate the historical data into an existing cost-sensitive learning method.

When dealing with complex and versatile medical data, different attributes may have different measurements and usage requirements. The scenario in Example 1 below highlights a new setting of cost-sensitive learning.

Example 1: Assuming there are some medical tests for a medical diagnosis: test X needs 1 day and $3000; and Y needs 5 days and $1000. The test costs are valued in distinct units (test fee and test time). If the patient needs an urgent decision within 3 days, then test X should be more suitable for him/her. On the other hand, if the patient holds a valid test result of X in his/her medical history, the doctor can certainly reset X's test costs to zero before considering further tests.

This setting in cost-sensitive learning brings us two new challenging issues: (1) The constraint of multiple-unit cost should be satisfied. (2) How to efficiently combine test data and medical history to the learning process. To address these issues, in this Chapter, we propose an objective-resource framework and a new lazy decision tree learning algorithm. The proposed framework has some new concepts, such as objective cost and resource costs. With the new concepts, an attribute selection strategy is incorporated into a lazy decision tree algorithm to utilize the different resource costs with multiple cost units more efficiently when medical history is dynamically utilized in the tests.

5.2 Preliminary

5.2.1 Test-Cost-Sensitive Learning Framework with Unified Cost Unit

For example, in the medical diagnosis domain, some medical tests, such as blood and urine tests, are quick and simple, but may not provide sufficient information for patients in complex conditions; some other tests, are crucial for the quality of diagnosis but cost much more. Latest research effort (Turney 1995; Greiner, et al. 2002; Zubek and Dietterich 2002; Ling, et al. 2004; Sheng and Ling 2006; Sheng, Ling, Ni, and Zhang 2006; Ling, Sheng, Yang 2006) aims to minimize the total cost of test and misclassification by assuming the two costs are measured with a unified unit, and any tests could be chosen with a predefined cost for the diagnosis.

Test cost is the cost of obtaining the value of an attribute. Current test-cost-sensitive learning framework combines both the misclassification cost and the test cost in the learning process. It aims at minimizing the sum of the two costs. Table 5.1 is an example which shows the test costs for 6 test candidates.

Table 5.1 Test costs for 6 test candidates

Test candidate	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**
Test cost	50	20	10	10	5	5

When combining the test cost and the misclassification cost in the classification framework above, we often add test cost using Formula (2.1) in Chapter 2, i.e. total cost is the sum of all tested attributes for making a decision. Assuming there are m attributes for a test and each attribute k has a test cost t_k, the cost set is set to $T = \{t_1, t_{2, \ldots} t_m\}$. Following Formula (2.1), the optimal prediction for an example x in test-cost-sensitive learning is class i that is a procedure of minimizing $L'(x, i)$ in Formula (5.1) as follows.

$$L'(x,i) = L(x,i) + C(x,T) = \sum_{j=1}^{n} p(j \mid x)C(i,j) + \sum_{k=1}^{m} p(k) \times t_k \tag{5.1}$$

, where $L(x, i)$ is the misclassification cost as same as it is in Formula (2.1); $p(k)$ is the probability of performing k-th test while estimating the true class distribution probability $P(j|x)$; and $C(x, T)$ is the sum of total test cost.

A test-cost-sensitive learning framework is an extension of the classical cost-sensitive learning framework, and it takes into account the test cost in learning process. Therefore, we have a property as follows:

Property 1. $L(x, i)$ in classical cost-sensitive learning framework is a special case of $L'(x, i)$ in test-cost-sensitive learning framework.

Proof: This property is obvious. We need only to set all the attributes' test costs to zero and obtain the following result:

$$\begin{aligned} L'(x,i) &= \sum_{j} p(j \mid x)C(i,j) + \sum_{k=1}^{m} p(k) \times 0 \\ &= \sum_{j} p(j \mid x)C(i,j) \\ &= L(x,i) \end{aligned}$$

In other words, the objective of test-cost-sensitive learning framework is the same as classical cost-sensitive learning when all the test costs are set to 0. And classical cost-sensitive learning framework is only a special case of test-cost-sensitive learning framework.

While the previous research is useful for many real world applications, the above assumption failed to model the complex and versatile medical environment. This is because medical tests may require multiple costs on many distinct cost units and they might have to comply with specific constraints on each unit. As shown in Example 1, test X needs \$3000 and 3 days and test Y needs \$1000 and 10 days. It is hard to treat the two cost units (time and money) in a unified unit. Therefore, in this research we design an objective-resource framework for distinguishing misclassification cost from resource costs which is described in Section 5.3.

5.2.2 Lazy strategy for decision tree building

The previous test cost-sensitive learning research we described in the last section assumes that all the patients do not have any history records, i.e. all test results are unknown before performing relative tests. To handle the issue of existing history records, we introduce a lazy strategy which is designed to handle this situation. LazyDT (Friedman, Yun and Kohavi 1996) is a decision tree algorithm with a lazy building strategy which doesn't build a general tree for all test instances to be classified, but build an ad hoc decision tree for each unlabelled instance. It constructs the best decision tree for each test instance. In practice, only a path needs to be constructed. A caching scheme makes the algorithm run fast (Rokach and Maimon 2008).

We briefly describe the general steps of the LazyDT algorithm below (Friedman, Yun and Kohavi 1996). Given a unlabelled instance *y*, the core part of the algorithm is to get the test instance as part of input, follows a separate-and-classify methodology: a test is selected, only those instances with same test outcome as given instance is then solved recursively.

The generic lazyDT algorithm:

Inputs: *S* is the training set;
y is the test instance to be classified;
Output: class label for the test instance *y*;
1. If all instances in S are from a single class l, return l;.
2. If all instances in S have same features, return the majority class of *S*;
3. Otherwise, select a test *T* and let *t* be the value of the test on instance *y*, Let S_0 be the set of training instances satisfying $T = t$ and apply the algorithm to S_0 and *y*.

The LazyDT has some distinctive merits in comparison with regular decision tree algorithms. First, the decision paths built by LazyDT are often shorter and therefore more comprehensible than paths of regular decision trees. Second, it is well known that given limited training data, regular decision tree algorithms may suffer from the data fragmentation problem (Pagallo and Haussler 1990). Traditional decision tree algorithms select a test for the root of each sub-tree based on the average improvement of the test selection criterion (such as entropy). Because the choice is based on average improvement, a particular child branch of the test may see a decrease in the value of the criterion or remain the same. For instances that take such a branch, the test may be detrimental since it fragments the data unnecessarily. This can cause the resulting path to be less accurate because the remaining tests are selected based on fewer training instances. In contrast, the LazyDT constructs a customized "tree" for each test instance, which consists of only a single path from the root to a labelled leaf node. Given a test instance, the LazyDT selects a test by focusing on the branch that will be taken by the test instance. In this way, it can avoid unnecessary data fragmentation and produce a more accurate classifier for the specific instance.

Given the above strengths of the LazyDT, we are interested in further extending it to build a dynamic cost-sensitive decision tree for a test instance with partial known information (medical history). This extension is difficult because the setting addressed here is new and complicated. It needs a new strategy to well utilize the known information. In our setting, the unknown values in test instances could be obtained after spending a resource cost, whereas the LazyDT just ignores the attributes with unknown values during tree building phase. In next section, we will study how to utilize the known information when building a lazy CSDT for each test example to reduce cost.

5.3 Lazy Cost-Sensitive Learning Based on Objective-Resource Framework

5.3.1 Objective-resource framework

In our objective-resource framework for cost-sensitive learning, misclassification cost is measured by a unique unit, referred to *objective cost*. And other costs, such as test cost, are referred to *resource costs*. As showed in Example 1, different tests may be measured with costs of different units. For an unclassified instance, a positive constant value is assigned to each type of test resource cost, referred to test *resource budget*.

Example 2: For test X and Y in Example 1, the objective cost, misclassification cost, is measured by its own unit, and resource costs, including test prices and testing time, are measured by "dollar" and "day" respectively. For a new patient, David, we could define his three resource budgets as \$5000 and 3 days respectively. This means that he can pay \$5000 and wait for 3 days to get the test results for supporting a decision-making by his doctor.

We could extend the test costs in Table 5.2 to multiple-unit costs as shown in Table 5.3. For test A1, it costs 50 dollars and 5 days to obtain the class value. To make a decision, a doctor can select any tests from the 6 test candidates as long as the test costs are within the resource budgets.

Table 5.2 Resource costs for 6 test candidates

Test candidate	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**
Class Value	?	?	?	?	?	?
Test fee (*dollar*)	50	20	10	10	5	5
Test time (*days*)	5	2	3	2	1	2

In real medical examinations, some tests could have been carried out earlier and recorded in patients' medical history. Very often, these test results are directly applied to their current diagnoses if they are still valid. In other words, we only need to reassign the costs of the known attributes to 0 while others remain unchanged. Consequently, this small change can significantly reduce the misclassification cost and the test cost.

Example 3: In Table 5.2, assume the values of test candidate A4 and A5 are known before this diagnosis and are still valid. Accordingly, Table 5.2 can be modified to Table 5.3 below. The resource costs of the known attributes are set to 0. In addition, the resource budget of each test attribute is included in this table.

Table 5.3 Resource costs and budgets of test candidates with some known values

Test candidate	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**	**RESOURCE BUDGET**
Class Value	**?**	**?**	**?**	**2**	**2**	**?**	
Test fee (*dollar*)	50	20	10	0	0	5	60 (60%)
Test time (*days*)	5	2	3	0	0	2	4 (40%)

With Table 5.3, we can design a new LazyDT to utilize historical data as much as possible, to save test costs significantly. This means that different decision trees are built with different known test attributes. Our new LazyDT is based on objective-resource framework, aiming to minimize the misclassification cost and control the resource costs within the specified resource budgets. We now formally state the objective-resource framework as follows:

Assuming there are *n* test candidates, to obtain the value of each test result, we need to spend *m* kinds of costs on different test attributes with cost units $\{s_l \mid l = 1,\dots, m\}$, where the objective cost unit is s_0. For i$_{th}$ test candidate, $t_i(s_0)$ is the objective cost and $t_i(s_i)$ is the resource cost. For the *m* test attributes, we assume $B(s_i)$ are their resource budgets, where $B(s_i) \geq 0$.

Similar to Formula (2.1) in Chapter 2, the optimal prediction, the class *i* for an example *x*, is to minimize the objective cost in Formula (5.3) and subject to test resource constraints in Formula (5.4).

$$MC(x,i) = \sum_{j=1}^{n} p(j \mid x) C(i,j) \tag{5.3}$$

Subject to

$$\begin{cases} t_i(s_l) \;\leq\; B(s_l), & l = 1,\ \dots,\ m \\ Min_{t_i(s_k) < B(s_k)}\ RC(x) = \sum_{k=1}^{m} p(k) \times t_i(s_k) \end{cases} \tag{5.4}$$

, where $MC(x, i)$ is the total cost of misclassification with a test candidate, $RC(x, i)$ is the total cost of candidate test resources, $p(k)$ is the probability of performing k_{th} test to obtain the value before making a decision.

5.3.2 Lazy Decision Tree Sensitive to Multiple-Unit Costs

We now present a lazy algorithm for building test-cost-sensitive decision trees based on the objective-resource framework in Formula (5.2) and (5.3) as follows:

Lazy cost-sensitive decision tree algorithm:

MS-CSDT-Lazy

Inputs:

D —a data set of samples $\{x_1; x_2; \ldots; x_n\}$,

A —a set of attributes $\{A_1; A_2; \ldots; A_m\}$,

CL —predefined classes $\{c_1; c_2; \ldots; c_p\}$,

R —misclassification cost matrix,

C_T —a test cost vector on objective unit,

$C_{R1}..C_{Rn}$ —test cost vectors on resource unit,

$AS(D, A, CL, R, C_T, C_{R1}..C_{Rn})$ — splitting formula for internal nodes, result is an attribute

$LM(D, A, CL, R, C_T, C_{R1}..C_{Rn})$ — leaf marking formula, result is a class,

Output: class label for the test instance y

1. If there is no attribute satisfy the $AS(D, A, CL, R, C_T, C_{R1}..C_{Rn})$
   return $LM(D, A, CL, R, C_T, C_{R1}..C_{Rn})$
2. Otherwise, select attribute $A_i = AS(D, A, CL, R, C_T, C_{R1}..C_{Rn})$, and let t be the value of the attribute on instance y

Let D_0 be the set of training instances satisfying $T = t$ and recursively apply the Algorithm **MS-CSDT-Lazy** on D_0

We will use the above MS-CSDT-Lazy algorithm to build cost-sensitive decision trees and classify test examples.

5.3.3 Building Cost-sensitive Decision Trees Based on Resource Budgets

In Chapter 2, we reviewed a novel cost-sensitive decision tree building and testing algorithm, CSDT. Here we extend CSDT to work with our new multiple-unit cost-sensitive learning framework. The first step is to extend the test cost table. For easy description and implementation, we only consider 3 resource cost units. Our task is to build a cost-sensitive decision which utilizes the limited test resources and minimizes the misclassification cost. There are two basic operations in decision tree building: leaf marking and node attribute selection.

Leaf marking criteria

To minimize the objective cost, we use the same leaf marking criteria proposed in (Ling, et al. 2004). A leaf y which contains m examples is marked as class j which minimizes the objective cost T:

$$T = \arg\min \sum^{m} \sum_{j=1}^{n} p(j \mid y) C(i, j) \qquad (5.4)$$

Considering **Example 2** in a binary decision problem, we assume a candidate note contains P positive and N negative examples. If there are no further splits, we will label the node as a leaf. We calculate the objective cost using Formula (5.3):

$$L(x, P) = P \times 0 + N \times FP = N \times FP$$

$$L(x, N) = P \times FN + N \times 0 = P \times FN$$

$$T = Min\{L(x, P), L(x, N)\}$$

$$= Min\{N \times FP, P \times FN\}$$

We will mark the leaf as positive if $P \times FN > N \times FP$, otherwise it would be marked as negative.

Attribute selection criteria for internal node

Attribute selection (also called feature selection) is an important research area of data mining and machine learning. It is a kind of data pre-processing strategy. Many classification algorithms such as decision tree and nearest neighbor can be benefited from an effective attribute selection process. We expect that by combining the attribute selection strategy and the lazy CSDT algorithm, we can utilize the limited resource budgets to build more efficient decision trees for each test example which minimize the objective cost.

There are many potential splitting strategies and they have different impacts on objective cost reduction with limited resources. Here we only introduce two simple and intuitive strategies (called the objective-first and the performance-first strategy) to select an attribute (feature) for each tree node. Both of them have resource budget constraint during the building phase.

The objective-first strategy is actually the same as the general lazy CSDT in Ling, et al. (2004) which only uses objective cost to select attributes to build the lazy general tree. It counts on resource consumption and stops at a node when any resources are exhausted; and makes a prediction according the cost matrix and classification distribution of the node.

The performance-first strategy is different which selects attribute for each node according to a heuristic measurement function on resource budgets. After making a node, it modifies the resource budgets to remaining resources and recursively builds the child node accordingly.

Assume the training set in a candidate node is *Y*, attribute *A* is a candidate attribute for splitting *Y*, and *A* is a discrete random variable with range $D = \{a_1, a_2, \ldots, a_n\}$, the test resource cost is t'_A. *A* resource budget *B* is specified prior to tree building, and *B'* is the remaining resource which is equal to *B* minuses the sum of test resource costs of all the parents nodes. *T* is the minimal objective cost calculated using Formula (5.2).

Object-first strategy

We call the first approach objective-first strategy. It assumes that the objective cost is the most important cost and totally ignores the resource issue. This approach attempts to minimize the total objective cost on misclassification. We could imagine this case as we have unlimited test resources or set all resource costs to 0. Objective-first strategy builds the same decision tree as that in Ling, et al. (2004) because only the objective cost is used in tree building phase. Before we decide if a candidate node is a leaf, we need to check all the candidate attributes for further splitting. The objective cost reduction for choosing attribute *A* to split is defined as below:

$$G_A = \mathrm{T} - \mathrm{T(A)} = T - \sum_{a \in D} p(x)(T(Y \mid A = a)) \qquad (5.5)$$

The node will be marked as a leaf if the cost reductions of all these potential splits are less than 0, otherwise the attribute with the maximum cost reduction will be selected to further splitting the node.

This approach totally ignores resource cost in tree building phase and only considers the resource budget at testing phase. Given a test example, we explore the tree and perform all needed tests alone the tree. Once any of the resource budgets are exhausted, we stop at the current node and give a result.

Performance-first strategy based on resource budget

This strategy considers the trade-off between objective cost and resource costs. It uses the "performance gain" to choose potential splitting attributes. Assuming the objective cost reduction is G_A = costRedu(*A*), the resource consumption of attribute *A* is shown in Formula (5.6).

$$R_A = \sum_{a \in D} p(x) \times t'_A \qquad (5.6)$$

To combine the resource budget *B* in the tree building phase, we define the ***remaining resource*** *B'* as resource budget minuses all the parent nodes' test resource costs. Attribute *A* will not be selected as splitting attributes if $B' - R_A < 0$, because the *remaining resource* cannot afford to obtain the value of attribute *A*. In case of $B' - R_A \geq 0$ and *B* is less than 100%, we define the ***performance gain*** of choosing *A* as the splitting attribute as following:

$$PerfG_A = \frac{G_A}{\left(\sum_i \frac{R_{Ai}}{B'_i} + 1\right)^w} \qquad (5.7)$$

, where, *i (1..n)* is the number *i* resource, B'_i is the remaining resource budget for the number *i* resource, *w* is a weight specified by domain expert. The attribute with the maximum performance gain will be selected as the splitting attribute. We can see that the ratio of objective cost gain (G_A) and resource consumption is used to select the best attribute. If B is equal or more than 100% we will use the objective-first attribute selection strategy specified in Formula (5.5) instead. On another word, we will focus on objective cost when we have enough resource.

Since performance-first strategy utilizes resource costs and resource budgets during the decision tree building phase, it is expected to outperform the objective-first strategy when the resource budgets are limited.

5.4 Experimental Evaluation

5.4.1 Experiment Setup

We conduct experiments on six UCI datasets (Blake and Merz 1998) and compare the performance of our proposed multiple-unit CSDT building and testing methods including attribute selection, performance-first strategy and LazyCSDT, to the original CSDT with or without test resource budget constraint. We compare their average objective cost and other key performance measurements such as AUC and test resource consumption across different cost ratios (FN/FP). All these algorithms are implemented in Weka. M-smoothing (specified in Chapter 2) is implemented for all the algorithms to reduce data over-fitting. These algorithms are listed in Table 5.4.

Table 5.4 List of Decision Tree Algorithms and Abbreviations

#	**Method**	**Abbreviation**
1	CSDT without test resource budget constraint	CSDT
2	CSDT with test resource budget constraint (Objective-first strategy)	CSDT-TF
3	Multiple-unit CSDT with test resource budget constraint (Performance-first strategy)	MSCSDT-PF
4	Attribute selection using Performance first strategy and test resource budgets, then use the selected attributes to build CSDT directly	MSCSDT-PFAS
5	Attribute selection using Performance first strategy and test resource budgets for each test example, then use the selected attributes to build CSDT directly to classify the example	MSCSDT-PFAS-Lazy

We then conduct experiment on six data sets chosen from UCI library. The details of these data sets are listed in Table 5.5. The imbalance level (the ratio of major class size to minor class size) in these data sets varies from 1.2 (Australia) to 11.9 (Car).

Table 5.5 Summary of the Data set Characteristics

Dataset	**No. of attributes**	**No. of examples**	**Class distribution (P/N)**
Statlog (heart)	14	270	120/150
Australia	15	653	296/357
Breast	9	683	239/444
Ecoli	6	332	102/230
Credit-a	15	690	307/383
Car	6	1728	134/1594

We conduct three experiments on the above datasets with different cost matrixes:

In the first experiment, we use the data set Statlog(heart). Statlog(heart) is one of a few data sets in UCI library with recommended cost matrix. The cost matrix is normalized and the cost ratio (FN/FP) is set to 5. The cost of TP and TN are both set to 0. This data set has been used extensively in cost-sensitive learning research previously. To better illustrate the effectiveness of the proposed multiple-unit CSDT algorithms, we only select the first 6 attributes from the Statlog (heart) data set to build decision trees. The test fee and test time are set as the Table 5.6 below. The budgets for both test resources are set to 50% of the total cost.

Table 5.6 Test resource costs and budgets

Attribute	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**	**Resource Budget**
Test time (Days)	3	1	1	5	2	2	7 (50%)
Test Fee (Dollar)	100	50	200	100	40	10	250(50%)

We then evaluate the classifier performance of CSDT, CSDT-TF, MSCSDT-PF and MSCSDT-PFAS by calculating their objective cost, AUC and test resource usage.

In the second experiment, the rest five UCI data sets are used to perform the test. The misclassification cost FP is always set to 1, and FN is set to an integer varying from

2 to 8 (2, 4, 8 respectively). We did not test with cost ratio more than 8 because when misclassification cost ratio is set too high, on many data sets, CSDT generates decision trees with only a single node. These decision trees are useless for evaluating our new multiple-unit CSDT algorithms. We assume that the misclassification of the minority class always incurs a higher cost. This is to simulate real-world scenarios in which the less frequent class is the more important class. The cost of *TP* and *TN* are both set to 0. All test fees are randomly set between 0 and 100, and all test time is randomly set between 0 and 10. We have two tests here. In the first test, the test budgets for both resources are set to 20% of the total test cost. In the second test, the test budgets are set to 60%. We then evaluate the classifier performance by calculating the average objective cost of the CSDT-TF, MSCSDT-PF and MSCSDT-PFAS algorithms.

In the last experiment, all the six UCI data sets are used to perform the test. Same as the second experiment, we always set the misclassification cost *FP* to 1, and *FN* to an integer varying from 2 to 8 (2, 4, 8 respectively). The cost of *TP* and *TN* are both set to 0. All test fees are randomly set between 0 and 100, and all test time is randomly set between 0 and 10. In this experiment, we focus on the example dependent resource budget, so the test budgets for both resources are randomly set between 20% and 100% for each example. We then evaluate the classifier performance by calculating the average objective cost of the CSDT-TF and MSCSDT-PFAS-Lazy algorithms.

All experiments are repeated for 10 times and ten-folder cross validation method is used in all tests to prevent over-fitting data.

5.4.2 Experiment Results and Discussion

In this section, we present an experimental comparison of the CSDT and the multiple-unit CSDT algorithms presented in the previous section.

The first experiment results are listed in Table 5.7. It lists the key performance measurements such as average objective cost, AUC and test resource usage for the Statlog(heart) data set. The aim of this experiment is to show that with imbalanced misclassification cost and limited test resources, compared to the naive CSDT-TF algorithm, our new methods can better utilize the limited test resources and build optimal decision trees which minimize the objective cost.

Table 5.7 Key performance measurements on Statlog(heart)

Table 5.7.1. Average Objective Cost

CSDT	CSDT-TF	MSCSDT-PF	MSCSDT-PFAS
0.555	0.889	0.646	0.592

Table 5.7.2. Area Under ROC (AUC)

CSDT	CSDT-TF	MSCSDT-PF	MSCSDT-PFAS
0.835	0.627	0.703	0.782

Table 5.7.3. Test Resource Usage (%)

Resource Usage	CSDT	CSDT-TF	MSCSDT-PF	MSCSDT-PFAS
Test fee	83%	50%	48%	48%
Test time	79%	30%	47%	47%

From the first experiment, we can draw some conclusions. First, in term of reducing objective cost, both of our new methods, MSCSDT-PF and MSCSDT-PFAS have achieved lower cost than CSDT-TF. MSCSDT-PFAS is the best among the three competing methods. Compared to CSDT-TF, it reduced the objective cost by more than 30%. Even when compared with the original CSDT method (without resource budget constraint), the average objective cost of MSCSDT-PFAS is very close, only increased by less than 10%. Second, by using AUC, a well-recognized measurement in cost-sensitive learning, we can see our two new methods always achieved higher AUC than CSDT-TF, and MSCSDT-PFAS is still the best. Third, in term of test resource usage, our two new methods can utilize the limited resource budgets on both test cost units. The test fee and test time resource budgets in this experiment are both set to 50%, MSCSDT-PF and MSCSDT-PFAS used most of test resources available (47% - 48%) to build more effective decision trees which minimizes the objective cost. However, the naive CSDT-TF used all 50% of test fee budget but only 30% of test time budget to build the decision tree. It cannot utilize all the test resources available, because it does not select tests based on test resource costs and budgets. Therefore, its objective cost is the highest among the four methods we tested. There is no surprise that the resource usage of the original CSDT is the highest (30% more than that of MSCSDT-PF and MSCSDT-PFAS) because it totally ignores the test resource costs and budgets during the decision tree building and test phase.

The test results of our second experiment are shown in Table 5.8 (with 20% test resource budget) and Table 5.9 (with 60% test resource budget).

Table 5.8 Average objective cost on selected UCI data sets (with 20% test resource budget)

Table 5.8.1. Cost Ratio (FP=1, FN=2)

Data Set	**CSDT-TF**	**MSCSDT-PF**	**MSCSDT-PFAS**
Australia	0.332	0.298	0.292
Breast	0.185	0.154	0.151
Ecoli	0.112	0.091	0.087
Credit_a	0.508	0.45	0.443
Car	0.48	0.432	0.421

Table 5.8.2. Cost Ratio (FP=1, FN=4)

Data Set	**CSDT-TF**	**MSCSDT-PF**	**MSCSDT-PFAS**
Australia	0.461	0.402	0.389
Breast	0.276	0.223	0.214
Ecoli	0.13	0.102	0.09
Credit_a	0.73	0.54	0.52
Car	0.594	0.503	0.499

Table 5.8.3. Cost Ratio (FP=1, FN=8)

Data Set	**CSDT-TF**	**MSCSDT-PF**	**MSCSDT-PFAS**
Australia	0.576	0.519	0.502
Breast	0.325	0.288	0.282
Ecoli	0.144	0.112	0.1
Credit_a	0.851	0.736	0.732
Car	0.69	0.561	0.513

Table 5.9 Average objective cost on selected UCI data sets (with 60% test resource budget)

Table 5.9.1. Cost Ratio (FP=1, FN=2)

Data Set	CSDT-TF	MSCSDT-PF	MSCSDT-PFAS
Australia	0.207	0.202	0.195
Breast	0.146	0.142	0.136
Ecoli	0.07	0.066	0.062
Credit_a	0.42	0.392	0.377
Car	0.306	0.288	0.286

Table 5.9.2. Cost Ratio (FP=1, FN=4)

Data Set	CSDT-TF	MSCSDT-PF	MSCSDT-PFAS
Australia	0.294	0.262	0.235
Breast	0.22	0.211	0.2
Ecoli	0.118	0.098	0.086
Credit_a	0.66	0.631	0.588
Car	0.489	0.435	0.384

Table 5.9.3. Cost Ratio (FP=1, FN=8)

Data Set	CSDT-TF	MSCSDT-PF	MSCSDT-PFAS
Australia	0.498	0.476	0.465
Breast	0.287	0.252	0.236
Ecoli	0.132	0.102	0.876
Credit_a	0.81	0.764	0.675
Car	0.654	0.543	0.452

From the second experiment we can draw the following conclusions: First our new method MSCSDT-PF and MSCSDT-PFAS consistently reduce the objective cost on all five selected data sets across different cost ratios. This approves that the "Performance First" strategy is better than the "Objective-First" strategy in multiple-unit decision tree learning. Second, our two new methods make more improvement in the test with only 20% resource budget, on all five data sets. The objective cost is reduced by more than

10%. When the test resource budget increases to 60%, the objective cost reduction that our new methods can make is much less, which means although our new methods can make improvement on tests with different resource budgets, they perform better when the test resource budget is low. Third, MSCSDT-PFAS always performs the best among the three competing methods. The improvement is made due to the introduction of the attribute selection strategy in the MSCSDT-PFAS algorithm.

The test results of our third experiment are shown in Table 5.10. It lists the average objective cost on the selected six UCI data sets when both test resource costs (test fee and test time) and budgets are randomly generated for each example. Among them, we deliberately set 20% of the test resource costs (both test fee and test time) to 0 which means these attribute values are known before the test.

Table 5.10 Average objective cost on selected UCI data sets

Table 5.10.1. Cost Ratio (FP=1, FN=2)

Data Set	**CSDT-TF**	**MSCSDT-PFAS-Lazy**
Australia	0.246	0.204
Breast	0.142	0.113
Ecoli	0.077	0.065
Credit_a	0.412	0.332
Car	0.321	0.195

Table 5.10.2. Cost Ratio (FP=1, FN=4)

Data Set	**CSDT-TF**	**MSCSDT-PFAS-Lazy**
Australia	0.395	0.356
Breast	0.223	0.195
Ecoli	0.114	0.095
Credit_a	0.677	0.52
Car	0.522	0.36

Table 5.10.3. Cost Ratio (FP=1, FN=8)

Data Set	**CSDT-TF**	**MSCSDT-PFAS-Lazy**
Australia	0.518	0.435
Breast	0.301	0.254
Ecoli	0.139	0.105
Credit_a	0.794	0.655
Car	0.66	0.372

The third experiment clearly shows that our new method, MSCSDT-PFAS-Lazy outperformed the CSDT-TF method on all six tested data sets across different cost ratios, often by a large margin. Compared to CSDT-TF, the objective cost reduction achieved by MSCSDT-PFAS-Lazy varies between 10% and 40%. Our new method works particularly well on the more imbalanced data set – Car. Also in this experiment, we can see that the higher the cost ratio, the more cost reduction achieved by the MSCSDT-PFAS-Lazy method. However, during our pre-test we noticed that when the cost ratio is set too high (more than 8), on many data sets, the original CSDT method only generates decision trees with a single node. In this case, our new method can not make any further improvement. We would like to extend our multiple-unit cost-sensitive learning framework to work with other classification algorithms in our future work to overcome this issue.

5.5 Conclusions

In this Chapter, we designed an efficient objective-resource framework to overcome the difficulty of handling multiple-cost-unit in cost-sensitive learning. We applied the framework with both a novel multiple-unit lazy CSDT algorithm and an attribute selection strategy to learn from real world medical data. With the two types of cost (objective and resource), this framework aims at minimizing the objective cost while controlling the resource costs within given resource budgets. Our experiments showed that the resource-budget-based performance-first strategy outperformed the naive objective-first strategy on all the UCI data sets we tested. The experiments also demonstrated that our proposed approach has outperformed a group of existing cost-sensitive learning algorithms in a cost/budget-changing environment, which mirrors the pressing demand of minimizing the objective cost while controlling the resource costs in the medical diagnosis domain.

Chapter 6 Cost-sensitive Classification with Inadequate Labelled Data

It is an actual and challenging issue to learn cost-sensitive models from datasets with few labelled data and plentiful unlabelled data, because some time labelled data are very difficult, time consuming and/or expensive to obtain. To solve this issue, in this Chapter we proposed two classification strategies to learn cost-sensitive classifier from training datasets with both labelled and unlabelled data, based on Expectation Maximization (EM). The first method, Direct-EM, uses EM to build a semi-supervised classifier, then directly compute the optimal class label for each test example using the class probability produced by the learning model. The second method, CS-EM, modifies EM by incorporating misclassification cost into the probability estimation process. We conducted extensive experiments to evaluate the efficiency, and results show that when using only a small number of labelled training examples, the CS-EM outperforms the other competing methods on majority of the selected UCI data sets across different cost ratios, especially when cost ratio is high.

6.1 Introduction

Cost-sensitive learning methods (Margineantu 2001) are developed based on the existing non-cost-sensitive learning methods. To make an error-based classifier cost-sensitive, a common method is to introduce biases into an error based classification system in three ways: (1) by changing the class distribution of the training data, (2) by modifying the learning algorithms, (3) and by taking the boosting approach (Li et al. 2005). An alternative method is called direct cost-sensitive learning which uses the conditional probability estimates provided by error based classifiers to directly compute the optimal class label for each test example using cost function (Zadrozny and Elkan 2001).

Existing cost-sensitive learning techniques work well when there are adequate labelled data in training datasets. It is undeveloped to learn cost-sensitive models from those datasets that are with few labelled data and plentiful unlabelled data. Many real world classification applications are facing the problem of inadequately labelled data, for example, spam email filtering and text classification applications. This is because labelled data are often very difficult, time consuming and/or expensive to obtain. In contrast with this, unlabelled data is often easy to be collected. Semi-supervised learning addresses this problem by using large amount of unlabelled data, together with the labelled data, to build classifiers with higher accuracy and less cost (Zhu 2005). It aims at training models from inadequate labelled data by utilizing large amount of unlabelled data, in order to reduce training costs and classification errors. Because of the potential to reduce need for expensive labelled data, semi-supervised learning has attracted an increasing amount of interest in recent years (Forman and Cohen 2004; Seeger, 2001). Many different methods (and algorithms) have been developed. Among them, Expectation Maximization (EM) (Dempster et al. 1977), self-training, co-training and transductive SVMs are the most commonly used methods. Applications such as text classification, web search and genetic research are examples where cheap unlabelled data can be used together with a group of labelled samples to build semi-supervised classifiers. They have illustrated the power of the semi-supervised classification as a new data mining tool aimed at improving the prediction accuracy and reducing costs.

For the example of spam email filtering, on one side, correctly identifying spam email requires human annotation, and collecting a large amount of spam email training data is time consuming and expensive. However, there are plenty unlabelled email data freely available. On the other side, the cost of misclassifying a spam email is very little. The email recipient can easily identify the spam email and delete it. However, the cost of misclassifying a non-spam email could be very expensive. For example, a company could lose a big business opportunity therefore a large amount of money because an important email is incorrectly filtered out. In addition, like computer virus, spam emails are also evolving over the time, spam email filters need to be very robust to handle any new types of spam emails. Utilizing the latest and unlabelled data can certainly reduce the on-going training cost and improve prediction accuracy, hence reduce the total cost.

Therefore, in this Chapter we study the issue of applying semi-supervised learning techniques to train cost-sensitive models from datasets with inadequate labelled data.

However, to the best of our knowledge, the semi-supervised classification has not been well studied to dealing with the inadequate labelled data in cost-sensitive learning. Some recent work includes Chen et al. (2010); Liu et al. (2008; 2009; 2010); and Qin et al. (2008). This research is among these first attempts to utilize unlabelled training data directly in cost-sensitive classification tasks. Two methods are presented, and both of them use EM as the base semi-supervised classification method. The first approach (we call it Direct-EM) is simple and straightforward, it uses EM (with both labelled and unlabelled training data) to build the classification model, then applies the cost function (Formula 2.1) directly to the model to classify the test examples. The second approach (we call it CS-EM, i.e. Cost-sensitive EM) involves modifying the EM algorithm so that it can use the labelled data and the cost matrix to assign "Cost-sensitive" probabilistically-weighted class labels to unlabelled training examples in the process of building the semi-supervised classification model. Extensive experiments are conducted for evaluating the efficiency, and show that when using only a small number of labelled training examples, the CS-EM outperforms the other competing methods on majority of the selected UCI data sets across different cost ratios, especially when cost ratio is high. The main contributions of this paper are:

- Addressing real world cost-sensitive classification issues in which there are few labelled data and plentiful unlabelled data
- Developing methods and algorithms for learning cost-sensitive classification model with small amount of labelled data and large amount of unlabelled data

6.2 Semi-supervised Learning

As per Seeger (2001), given an unknown probabilistic relationship $P(x; t)$ between input points x and class labels t *in* $T = \{1,\ldots,c\}$, the problem is to predict t from x, i.e. to find a predictor $\hat{t} = \hat{t}(x)$ such that the generalization error of $\hat{t}$,

$$P_{x,t}\{\hat{t}(x) \neq t\} \qquad (6.1)$$

- Where $\hat{t}$ is small, ideally close to the Bayes error, being the minimum of the generalization errors of all predictors. The aim of a semi-supervised learning algorithm is to compute ^*t* from: A labelled sample set $D_l = \{(x_i, t_i) \mid I = 1,\ldots,n\}$, where the (x_i, t_i) are drawn independently from $P(x, t)$.

- An unlabelled sample set $Du = \{x_i \mid I = n+1, \ldots, n+m\}$, where the x_i are drawn independently from the marginal input distribution $P(x) = \sum_{t=1}^{c} p(x,t)$. Du is sampled independently from D_l.
- Prior knowledge (or assumptions) about the unknown relationship.

6.3 Expectation Maximization (EM)

Given a set *Dl* (usually, small) of labelled data, and a set *Du* (usually, large) of unlabelled data, semi-supervised classification methods aim to design classifiers using both sets (Roli 2005). Some often-used methods include: EM, self-training and co-training. In this paper, we are interested in EM. The cost-sensitive method we developed either directly using EM as the base semi-supervised learner, or modifying it to make it sensitive to cost matrix.

EM is a class of iterative algorithms for maximum likelihood or maximum a posteriori estimation in problems with incomplete data (Dempster et al. 1977). In case of semi-supervised classification tasks, the unlabelled data are considered incomplete because they come without class labels. The **basic EM** approach first designs a probabilistic classifier (such as Naïve Bayes or Decision Tree in Bradford et al. 1998) with the labelled data set *Dl*. Then, use this classifier to assign probabilistically-weighted class labels to the unlabelled data. Then use both the original labelled data and the formerly unlabelled data to build a new classifier, and the process is iterated until the classifier does not change. This process is guaranteed to find model parameters that have equal or higher likelihood than at the previous iteration, as shown by Dempster et al. (1977). The main advantage of EM approach is that it allows exploiting labelled and unlabelled data in a theoretically well grounded way. Therefore, naturally it meets the main requirement of semi-supervised classification (Roli 2005).

Nigam et al. (2000) introduced a new parameter λ ($0 \leqslant \lambda \leqslant 1$) into the Basic EM. They term the resulting method EM-λ. Note that λ is the weight parameter of the unlabelled data. When λ is close to zero, the unlabelled data will have little influence on the shape of EM's hill-climbing surface. When $\lambda = 1$, each unlabelled data will be weighted the same as a labelled data, and the algorithm is the same as the basic EM. The details of the EM-λ algorithm are described below.

EM-λ algorithm:

1. Inputs: Collections *Dl* of labelled training data and *Du* of unlabelled training data.
2. Set the weight factor of the unlabelled data, λ, by cross-validation.
3. Build an initial probabilistic classifier, Θ, from the labelled training data only. Use maximum posteriori parameter estimation to find $\Theta = \arg\max_\theta P(D|\theta)P(\theta)$.
4. Loop while classifier parameters improve (the change in complete probability of the labelled and unlabelled data, and the prior):

   (E-step) Use the current classifier, Θ, to estimate the class membership of unlabelled data.

   (M-step) Re-estimate the classifier, Θ, given the estimated class membership of each unlabelled data. Use maximum a posteriori parameter estimation to find $\Theta = \arg\max_\theta P(D|\theta)P(\theta)$.

   When counting event for parameter estimation, weights from unlabelled training data are reduced by a factor λ.
5. Output: classifier, Θ, which takes an unlabelled test example and predicts its class label.

Results on different classification tasks showed that EM methods allow utilizing unlabelled data effectively. For instance, Nigam et al. (2000) showed that unlabelled data used with EM-λ method in a document categorization problem can improve classification accuracy by up to 30%.

6.4 Making Semi-supervised Classification Cost-sensitive

In this Chapter, we focus on binary classification problems. We propose two approaches to making semi-supervised learners sensitive to misclassification cost.

The first approach is quite simple. We use EM-λ algorithm (together with both labelled and unlabelled data) to train a semi-supervised classifier. Since the learned classifier can produce conditional probability estimates, using these probability estimates and formula (2.1) specified in Chapter 2, we can directly compute the optimal class label for each test example. We call this approach **Direct-EM**.

The second approach involves modifying the EM-λ algorithm. In a binary decision case, suppose that the probability of labelling an unlabelled example to P is P_p, and the probability of labelling this unlabelled example to N is P_n. If $P_p > P_n$, the unlabelled example is assigned to P, and the probability of this prediction is P_p. Otherwise, the unlabelled example is assigned to N, and the probability of this prediction is P_n. Note that here $P_p + P_n = 1$.

Suppose that in this situation, the cost of false positive (FP) and the cost of false negative (FN) are very different. To simplify the case, we assume that the cost of true positive (TP) and true negative (TN) are both 0. In cost-sensitive learning, the purpose is to minimize the misclassification cost instead of errors. Now we calculate the potential cost (C_p) of labeling this leaf to P is $FP * P_n$. And the potential cost (C_n) of labeling this leaf to N is $FN * P_p$.

If $C_p > C_n$, the unlabelled data is assigned to N, and the probability of this prediction is $C_p/(C_p + C_n)$. Otherwise, the unlabelled data is assigned to P, and the probability of this prediction is $C_n/(C_p + C_n)$.

We replace the probability estimate in the original EM-λ algorithm with this new probability estimate which is sensitive to the misclassification cost. The last step is using the learned classifier to predict class labels for all test examples. Since the cost has already been incorporated in the model building process, they will not be used again in the classification process. We call this approach **CS-EM**. Below are the modified steps (to the original EM-λ) in our new CS-EM algorithm.

CS-EM algorithm:

---

1. Build an initial cost-sensitive classifier, Θ, from the labelled training data only. Θ is a direct cost-sensitive classifier which can directly use any error based probabilistic classifiers and cost matrix to compute class memberships for unlabelled data.

2. Loop while classifier parameters improve (the change in complete probability of the labelled and unlabelled data, and the prior):

   E-step: Use the current classifier, Θ, to estimate the class membership of unlabelled data. Note the class membership is calculated based on our new approach which takes the misclassification cost into account.
   M-step: Re-estimate the classifier, Θ, given the estimated class membership of each unlabelled data. In this step, the probability estimates calculated are also based on our new approach which takes the misclassification cost into account.

We expect that the two new approaches can reduce the misclassification cost compared to the similar direct cost-sensitive learning method which only uses the labelled training data. We call this simple method **Direct-LBL**.

6.5 Experimental Evaluation

The main purpose of this experiment is to evaluate the performance of the two new cost-sensitive semi-supervised learning approaches (Direct-EM and CS-EM) we discussed in Section 6.4 by comparing their average misclassification cost to the Direct-LBL method. C4.5 decision tree and Naive Bayes are chosen as the base probabilistic classifier. Laplace correction is turned on for C4.5 to avoid data over-fitting. All these algorithms are implemented in Weka (a popular data mining software), and they are listed in Table 6.1.

Table 6.1 List of Algorithms and Abbreviations

#	**Method**	**Abbreviation**	**Base Classifier**
1	Cost-sensitive C4.5	Direct-LBL-C4.5	C4.5
2	Cost-sensitive Naive Bayes	Direct-LBL-NB	Naive Bayes
3	Direct EM with C4.5	Direct-EM-C4.5	C4.5
4	Cost-sensitive EM with C4.5	CS-EM-C4.5	C4.5
5	Direct EM with Naive Bayes	Direct-EM-NB	Naive Bayes
6	Cost-sensitive EM with Naive Bayes	CS-EM-NB	Naive Bayes

We then conduct experiments on 14 real world data sets chosen from UCI repository in Blake and Merz (1998). The details of these data sets are listed in Table 6.2. The imbalance level (the ratio of major class size to minor class size) in these data sets varies from 1.0 (Pendigits) to 15.3 (Sick).

Table 6.2 Summary of the Data set Characteristics

Dataset	**No. of attributes**	**No. of examples**	**Class distribution (P/N)**
car	7	1728	134/1594
creidt-g	21	1000	300/700
hypothyroid	30	3772	291/3481
kr-vs-kp	37	3196	1527/1699
mushroom	23	8124	3916/4208
nursery	9	12960	4320/8640
optdigits	65	1143	571/572
page-blocks	11	5473	560/4913
pendigits	17	2288	1144/1144
sick	30	3772	231/3541
spambase	58	4601	1813/2788
splice	62	2423	768/1655
vowel	14	990	90/900
waveform-5000	41	3347	1655/1692

We conduct experiments on all above datasets with different cost ratios. The misclassification cost FP is always set to 1, and FN is set to an integer varying from 2 to 20 (2, 5, 10, 20 respectively). We assume that the misclassification of the minority class always incurs a higher cost. This is to simulate real-world scenarios in which the less frequent class is the more important class. The cost of TP and TN are both set to 0.

All the selected data sets are randomly split into two sub sets: 60% for training and 40% for testing. Each test is repeated for 10 times, we then measure the performance of our cost-sensitive semi-supervised learning methods by calculating the average of the misclassification cost.

6.5.1 Evaluating the new cost-sensitive semi-supervised learning algorithms

In the first experiment, we choose 50 examples from the training set as the labelled data, and the rest of the examples in the training set as unlabelled data. The λ value is dynamically determined by cross validation. We then perform test on all the 14 data sets listed in Table 6.2 using all the algorithms listed in Table 6.1 with different cost ratios (2, 5, 10, 20 respectively). The aim of this experiment is to evaluate the performance of our new algorithms on different data sets with different cost ratios.

Table 6.3 lists the average misclassification cost on the selected UCI data sets when the number of labelled training examples is set to 50. Table 6.4 lists the corresponding results on the t-test. Each *w/t/l* in the table means our new algorithms at each row wins in *w* data sets, ties in *t* data sets and loses in *l* data sets.

From this kind of experimental result, we can draw several conclusions as follows.

- First, our new method, CS-EM works well as per our expectation. It achieved the lowest average misclassification cost on majority of the data sets, especially when cost ratio is high, and it does not increase a lot when the cost ratio increases. This is mainly because CS-EM takes the misclassification cost into account when assigning probabilistically-weighted class labels to the unlabelled data in the modified EM-λ algorithm. So the examples with lower potential misclassification cost are assigned higher weights in the final model. This method effectively reduces the misclassification cost, especially when cost ratio is high.
- Second, it is not a surprise to us that the Direct-LBL method, did not performs very well, because it only utilizes the labelled training data which contains only 50 examples. It is very hard to use such a small set of training data to build a classifier which can provide good class probability estimate. In cost-sensitive classification tasks, a classification model with good class probability estimate is critical to the success of reducing the misclassification cost.
- Third, the performance of another new method, Direct-EM is disappointing. Compared to CS-EM and Direct-LBL, its average misclassification cost is the highest on most of the data sets. The cause of this poor performance may be due to the inaccurate probability estimate generated by the final model. The probability estimate is hurt by using the unlabelled training data without considering the misclassification cost.

- Forth, the performance of our new algorithm, CS-EM, is not affected much by the base probabilistic classifier. For example, when the cost ratio is 20, both CS-EM-C4.5 and CS-EM-NB beat Direct-LBL-C4.5 and Direct-LBL-NB, the *w/t/l* values are 11/1/2 and 11/0/3 respectively. The similar results have been achieved on other cost ratios too.

Table 6.3 Average misclassification cost on selected UCI data sets

Table 6.3.1 C4.5 as the base classifier when Cost Ratio = 2

Data set	**Direct-LBL-C4.5**	**Direct-EM-C4.5**	**CS-EM-C4.5**
car	0.16	0.16	0.16
credit-g	0.56	0.55	0.55
hypothyroid	0.08	0.15	0.15
kr-vs-kp	0.16	0.16	0.14
mushroom	0.06	0.06	0.43
nursery	0	0.67	0.67
optdigits	0.14	0.15	0.11
page-blocks	0.15	0.2	0.2
pendigits	0.07	0.07	0.05
sick	0.1	0.1	0.09
spambase	0.3	0.6	0.27
splice	0.14	0.58	0.35
vowel	0.09	0.17	0.15
waveform-5000	0.34	0.31	0.31

Table 6.3.2 C4.5 as the base classifier when Cost Ratio = 5

Data set	Direct-LBL-C4.5	Direct-EM-C4.5	CS-EM-C4.5
car	0.39	0.39	0.39
credit-g	0.77	1.5	1.27
hypothyroid	0.15	0.39	0.38
kr-vs-kp	0.35	0.31	0.27
mushroom	0.19	0.15	0.52
nursery	0	1.67	0
optdigits	0.29	0.25	0.22
page-blocks	0.29	0.51	0.5
pendigits	0.16	0.16	0.15
sick	0.21	0.21	0.19
spambase	0.57	1.46	0.54
splice	0.22	1.45	0.44
vowel	0.21	0.41	0.15
waveform-5000	0.63	0.5	0.49

Table 6.3.3 C4.5 as the base classifier when Cost Ratio = 10

Data set	Direct-LBL-C4.5	Direct-EM-C4.5	CS-EM-C4.5
car	0.85	0.78	0.76
credit-g	0.7	3	2.25
hypothyroid	0.28	0.77	0.76
kr-vs-kp	0.5	0.68	0.5
mushroom	0.4	0.3	0.37
nursery	0	3.33	0
optdigits	0.52	0.5	0.47
page-blocks	0.49	1.02	1
pendigits	0.32	0.32	0.29
sick	0.43	0.61	0.4
spambase	0.84	2.91	0.61
splice	0.59	2.9	0.53
vowel	0.37	0.82	0.31
waveform-5000	0.87	0.71	0.51

Table 6.3.4 C4.5 as the base classifier when Cost Ratio = 20

Data set	Direct-LBL-C4.5	Direct-EM-C4.5	CS-EM-C4.5
car	1.11	1.55	0.93
credit-g	0.7	6	2.23
hypothyroid	0.56	1.54	0.51
kr-vs-kp	0.52	1.33	0.52
mushroom	0.6	0.6	0.52
nursery	0.57	6.67	0.27
optdigits	0.92	0.75	0.53
page-blocks	0.95	2.05	2
pendigits	0.63	0.63	0.5
sick	0.76	1.23	0.71
spambase	0.86	5.8	0.61
splice	0.68	5.8	0.63
vowel	0.71	1.63	0.62
waveform-5000	1.43	1.12	0.51

Table 6.3.5 Naïve Bayes as the base classifier when Cost Ratio = 2

Data set	Direct-LBL-NB	Direct-EM-NB	CS-EM-NB
car	0.14	0.16	0.13
credit-g	0.5	0.49	0.47
hypothyroid	0.11	0.15	0.15
kr-vs-kp	0.34	0.55	0.46
mushroom	0.15	0.2	0.2
nursery	0.02	0.67	0.64
optdigits	0.05	0.06	0.06
page-blocks	0.2	0.29	0.17
pendigits	0.01	0.01	0.01
sick	0.12	0.12	0.11
spambase	0.2	0.28	0.17
splice	0.18	0.47	0.23
vowel	0.12	0.12	0.07
waveform-5000	0.15	0.14	0.13

Table 6.3.6 Naïve Bayes as the base classifier when Cost Ratio = 5

Data set	**Direct-LBL-NB**	**Direct-EM-NB**	**CS-EM-NB**
car	0.28	0.39	0.37
credit-g	0.8	1.26	0.15
hypothyroid	0.24	0.36	0.36
kr-vs-kp	0.5	1.17	0.56
mushroom	0.31	0.5	0.43
nursery	0.09	1.67	0.05
optdigits	0.11	0.1	0.09
page-blocks	0.32	0.48	0.53
pendigits	0.01	0.01	0.01
sick	0.27	0.29	0.27
spambase	0.31	0.3	0.28
splice	0.32	1.16	0.29
vowel	0.22	0.2	0.18
waveform-5000	0.19	0.16	0.17

Table 6.3.7 Naïve Bayes as the base classifier when Cost Ratio = 10

Data set	Direct-LBL-NB	Direct-EM-NB	CS-EM-NB
car	0.44	0.78	0.71
credit-g	1.09	2.47	1.06
hypothyroid	0.47	0.72	0.72
kr-vs-kp	0.59	2.2	0.41
mushroom	0.53	0.99	0.47
nursery	0.19	3.33	0.1
optdigits	0.2	0.2	0.18
page-blocks	0.52	0.8	0.85
pendigits	0.01	0.01	0.01
sick	0.5	0.47	0.4
spambase	0.47	0.41	0.42
splice	0.47	2.32	0.46
vowel	0.36	0.54	0.65
waveform-5000	0.24	0.16	0.17

Table 6.3.8 Naïve Bayes as the base classifier when Cost Ratio = 20

Data set	Direct-LBL-NB	Direct-EM-NB	CS-EM-NB
car	0.66	1.55	1.41
credit-g	1.42	4.88	1.1
hypothyroid	0.9	1.44	1.45
kr-vs-kp	0.66	4.27	0.65
mushroom	0.9	1.97	0.72
nursery	0.32	6.66	0.18
optdigits	0.37	0.51	0.32
page-blocks	0.9	1.43	0.87
pendigits	0.02	0.02	0.01
sick	0.95	0.92	0.89
spambase	0.78	0.58	0.65
splice	0.71	4.64	0.65
vowel	0.64	1.07	1.34
waveform-5000	0.29	0.16	0.18

Table 6.4 Summary of t-test

Table 6.4.1 C4.5 as the base classifier

Cost Ratio (FN:FP)		**Direct-LBL-C4.5**
2	Direct-EM-C4.5	2/4/8
	CS-EM-C4.5	7/1/6
5	Direct-EM-C4.5	4/3/7
	CS-EM-C4.5	7/2/5
10	Direct-EM-C4.5	4/1/9
	CS-EM-C4.5	9/2/3
20	Direct-EM-C4.5	2/2/10
	CS-EM-C4.5	11/1/2

Table 6.4.2 Naïve Bayes as the base classifier

Cost Ratio (FN:FP)		Direct-LBL-NB
2	Direct-EM-NB	2/3/9
	CS-EM-NB	7/1/6
5	Direct-EM-NB	4/1/9
	CS-EM-NB	7/2/5
10	Direct-EM-NB	3/2/9
	CS-EM-NB	9/1/4
20	Direct-EM-NB	3/1/10
	CS-EM-NB	11/0/3

We also conducted the similar tests using different sizes of labelled training examples. The detailed analysis is in Section 6.5.2.

6.5.2 Evaluating the impact of the size of labelled examples

In the second experiment, we perform test on data set waveform-5000 using algorithms Direct-LBL-C4.5 and CS-EM-C4.5. The cost ratio is set to 10 and the λ value is still dynamically determined by cross validation. The aim of this experiment is to evaluate the impact of labelled training example size to CS-EM algorithm. So we conduct test with different number of labelled training examples: 5, 10, 20, 50, 100, 200, 400 respectively.

Figure 6.3 shows the average misclassification cost on the waveform-5000 data set for algorithms Direct-LBL-C4.5 and CS-EM-C4.5. From this figure, we can see that the performance of our new algorithm, CS-EM, is affected by the size of the labelled training examples. It performed well when the size of the labelled training examples is 10, 20, 50 and 100, but not 5, 200 and 400. The test results on other data sets vary but the trend is similar. This means that our new algorithm, CS-EM does not work well with too few or too many labelled training examples. The reasons are as follows. First, when the initial classifier is built only using a few labelled training examples, it can not generate good probability estimate for the unlabelled training examples at the first place, hence the final model suffers. Second, when there are many labelled training examples available, a good classification model can be generated from the base probabilistic classifier using only labelled data. The use of unlabelled training examples may actually reduce the classifier performance in cost-sensitive learning.

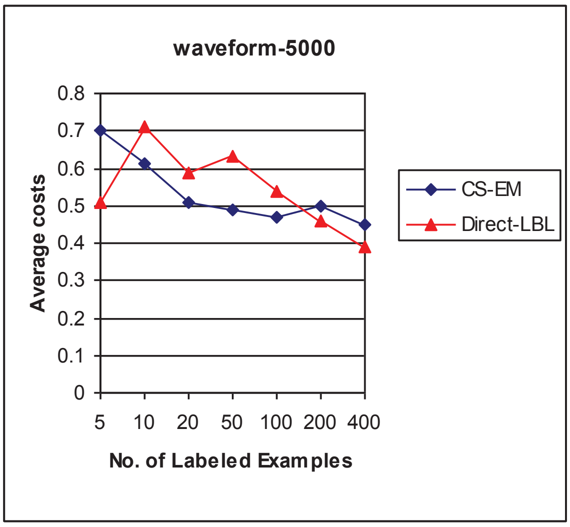

Figure 6.1 Comparing the average misclassification cost for different sizes of labelled training examples

6.5.3 Evaluating the impact of sample selection bias

In the third experiment, we extend the test in Section 6.5.2. We still use the same data sets, the same algorithms and the same experiment settings. The aim of this experiment is to evaluate the impact of sample selection bias to CS-EM algorithm. Therefore, instead of randomly selecting labelled examples (as we did in Section 6.5.2), this time we deliberately select our labelled examples with P/N ratio which is different from the class distribution. Two tests were performed, in the first test, the labelled example P/N ratio is set to 1/2, and in the second test, the ratio is set to 2/1. As you can see in Table 6.2, the actual P/N ratio of waveform-5000 data set is around 1.

Figures 6.4 and 6.5 show the average misclassification cost on the waveform-5000 data set for algorithms Direct-LBL-C4.5 and CS-EM-C4.5 with different P/N ratios. From these two figures you can see that with sample selection bias, the CS-EM-C4.5 outperforms Direct-LBL-C4.5 on all tests conducted. In both cases (P/N set to 1/2 or 2/1), CS-EM-C4.5 achieved significant lower cost than Direct-LBL-C4.5 with different sizes of labelled examples. Therefore, we can make a clear conclusion that if the sample selection bias does exist in the labelled examples, by using unlabelled data and applying semi-supervised learning techniques, we can build better cost-sensitive classification models with lower misclassification cost.

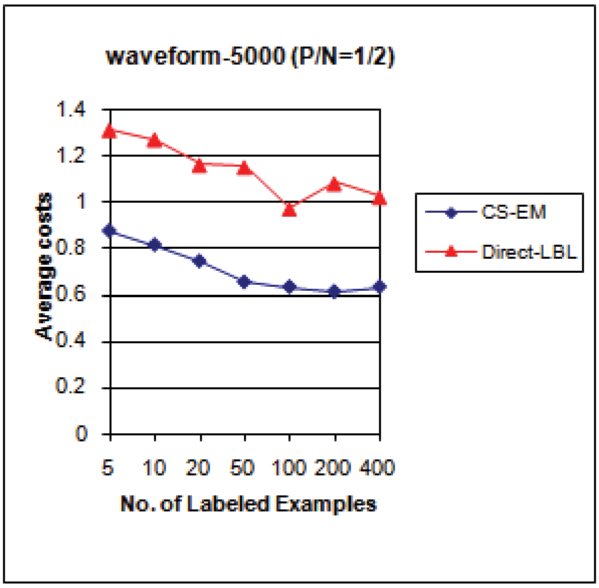

Figure 6.2 Comparing the average misclassification cost for different sizes of labelled training examples. The labelled example P/N ratio is 1/2

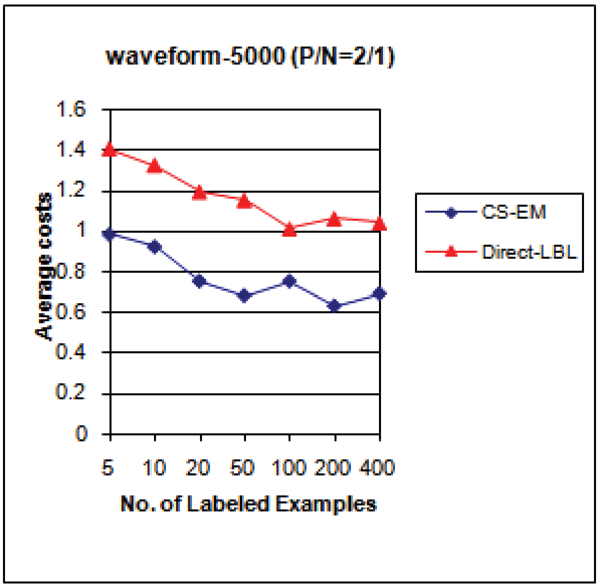

Figure 6.3 Comparing the average misclassification cost for different sizes of labelled training examples. The labelled example P/N ratio is 2/1

6.5.4 Evaluating the impact of feature selection strategy

In the third experiment, we perform test on data set waveform-5000 using only CS-EM-C4.5, with different feature selection strategies. In the first test, we randomly select labelled examples, and we call it CS-EM (Random). In the second test, we select labelled examples with the highest information gain, we all this strategy CS-EM (IG). The cost ratio is still set to 10 and the λ value is still dynamically determined by cross validation. The aim of this experiment is to evaluate the impact feature selection strategy to our CS-EM algorithm.

Figure 6.6 shows the average misclassification cost on the waveform-5000 data set for algorithm CS-EM-C4.5 with random and information gain feature selection strategy. From this figure, we can see that the performance of our new algorithm, CS-EM, is improved consistently by the introduction of the information gain feature selection strategy, especially when the size of labelled examples increases. Feature selection is an important component of any data mining and machine learning model, as it often accounts for big difference in performance. The test result in this experiment is exactly what we expected. By introducing a good feature selection strategy we increased the value of labelled examples and improved the performance of our new CS-EM algorithm. As a result, the misclassification cost was reduced significantly. In the future, we would like to apply other feature selection strategies to our cost-sensitive semi-supervised learning algorithms to maximize the benefit of this approach.

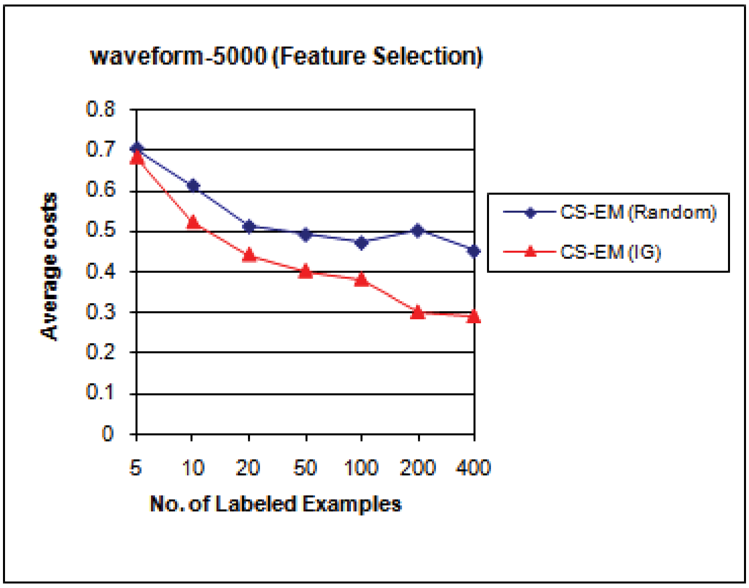

Figure 6.4 Comparing the average misclassification cost for different sizes of labelled training examples

6.5.5 Evaluating the impact of λ value

In the last experiment, we still perform test on data set waveform-5000 using algorithms Direct-LBL-C4.5 and CS-EM-C4.5. The size of labelled training examples is set to 50 and the cost ratio is set to 10. The aim of this experiment is to evaluate the impact of λ value. So we conduct test with different λ values: 0, 0.2, 0.4, 0.6, 0.8, 1.0 respectively.

Figure 6.7 shows the average misclassification cost on the waveform-5000 data set for algorithms Direct-LBL-C4.5 and CS-EM-C4.5. From this figure, we can see that the average misclassification cost is the highest when λ is set to 0 (no utilization of unlabelled training data). The average misclassification cost is also high when λ is set to too high, for example, 1.0. CS-EM works well when λ is set in the middle of 0 and 1.0. When λ is set to 0.2, it achieved the lowest average misclassification cost, 0.45, better than the average misclassification cost we achieved in Section 6.5.1 using cross validation.

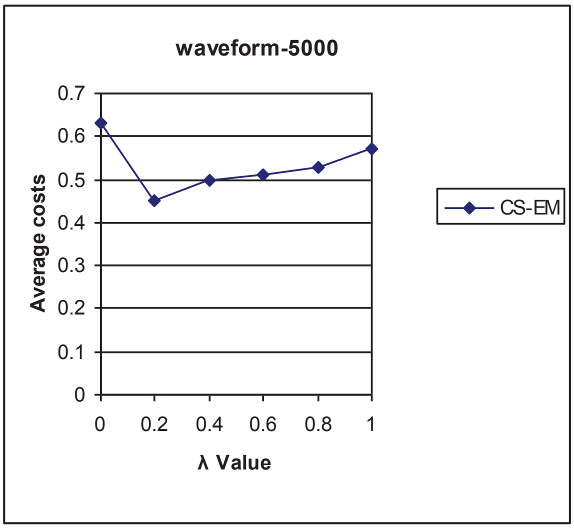

Figure 6.5 Comparing the average misclassification cost for different λ values

6.6 Conclusions

In this Chapter, we have studied the issue of learning cost-sensitive classifier from training datasets with few labelled data and plentiful unlabelled data. We designed two simple and effective methods, Direct-EM and CS-EM, for learning cost-sensitive classifiers using both the labelled and unlabelled training data. The CS-EM modifies the popular semi-supervised learning method, EM-based algorithm, by incorporating the misclassification cost into the class probability estimation process. The experiments have shown that when using only a small number of labelled training examples, the CS-EM outperforms the other competing methods on most of the selected UCI data sets across different cost ratios. We also analysed the impact of λ value and the size of labelled training examples to the CS-EM algorithm in the experiments. The results show that the CS-EM does not work well with λ value closing to either 0 or 1.0. And compared to the Direct-LBL method, the CS-EM cannot reduce much of misclassification cost when working with too few or too many labelled training examples.

Chapter 7 Conclusions and Future Work

7.1 Contributions

The main goal of this thesis was to develop a set of new and efficient techniques to improve the cost-sensitive decision making process and minimize the total cost involved in cost-sensitive classification. Based on the work specified in Chapter 3, 4, 5 and 6, we have achieved the following outcomes:

- We developed three simple but efficient methods - feature selection, smoothing and threshold pruning, on the TCSDT algorithm to reduce data over-fitting in cost-sensitive classification. Our new methods modify the TCSDT algorithm by introducing feature selection process before building decision tree, applying smoothing and pruning process before calculating the class probability estimate for each decision tree leaf. These methods removed noisy data and irrelevant/disturbing attributes from the training data, improved class membership probability estimation of the TCSDT algorithm, and eventually reduced the total cost of the classification.
- We modified the distance function of the popular KNN classifier to make it cost-sensitive. We also enhanced the modified cost-sensitive KNN classifier by applying the smoothing, minimum-cost *K* value selection, cost-sensitive feature selection and cost-sensitive stacking method to it. We demonstrated the effectiveness and performance our new cost-sensitive KNN algorithms.
- We designed an Objective-Resource cost-sensitive learning framework which addresses a real world issue where multiple cost units are considered. The Objective-Resource framework resolves this issue by defining two types cost, objective cost and resource cost, and building a lazy cost-sensitive decision tree which minimizes the objective cost subjecting to given budgets of other resource costs.
- We studied the issue of learning cost-sensitive classifier from training datasets with few labelled data and plentiful unlabelled data. We proposed a simple but effective

method (CS-EM) for learning cost-sensitive classifiers using a small number of labelled data and large amount unlabelled data. The proposed method modifies the popular EM algorithm, by incorporating the misclassification cost into the class probability estimation process. We also analysed the impact of the λ value, the size of labelled examples, the sample selection bias and the different feature selection strategies to the performance of our new semi-supervised cost-sensitive classification algorithm.

7.2 Directions for Future Research

The research and the outcomes described in this thesis can be extended in many different directions.

7.2.1 Cost-sensitive Classification on Multi-class Data Sets

In the future, we plan to test the new methods on more real world data sets which are relative to test cost-sensitive learning, and compare them to more cost-sensitive learning algorithms. We also would like to extend our new cost-sensitive classification algorithms to handle multi-class data sets and evaluate the effectiveness of our new methods on real world multi-class data sets.

7.2.2 Improving Objective-resource Cost-sensitive Learning Framework

Our new Objective-resource cost-sensitive learning framework (and algorithms), although is innovative and practical, it also has some shortcomings. For example, currently it only works with cost-sensitive decision tree, and does not handle cost matrixs with very high FN/FP cost ratio well. In the future, we would like to extend it to work with other classification algorithms to overcome those issues.

7.2.3 Semi-supervised Cost-sensitive Learning

In the future, we plan to test our new semi-supervised cost-sensitive classification methods (Direct-EM and CS-EM) on more real world data sets which are relative to both cost-sensitive and semi-supervised learning, and check if the result we achieved in this paper is consistent. It will be very beneficial to find out in which situation Direct-EM performs well (or worse) and why. Because if we had answered this question, we will be able to utilize all the existing semi-supervised learning methods (as long as they can provide conditional probability estimates) to build better cost-sensitive classification models with additional large amount of cheap unlabelled data. We also would like to investigate the possible improvements of the CS-EM including automatically determining the best λ value and the best size of labelled training examples, and applying new feature selection strategies to CS-EM.

7.2.4 Active Learning

Active learning is a data mining area in which training data is acquired incrementally. In many real world situations the costs of collecting and labelling training examples are very high. Both active learning and the semi-supervised learning can help in these situations. In the future, we plan to improve our new semi-supervised cost-sensitive classification algorithms by applying some active learning techniques, such as using Query by Committee method to select the most informative labelled training data before building the final classifiers.

Reference

1. Abe, N. and Mamitsuka, H. (1998). Query learning strategies using boosting and bagging. *Proceedings of the Fifteenth International Conference on Machine* Learning, pp 1-9.
2. Abe, N., Zadrozny, B. and Langford, J. (2004). An iterative method for multi-class cost-sensitive learning. *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 3-11.
3. Abrahams, A.S., Becker, A., Fleder, D. and MacMillan, I.C. (2005). Handling Generalized Cost Functions in the Partitioning Optimization Problem through Sequential Binary Programming. *Proceedings of the Fifth IEEE International Conference on Data Mining*, pp. 3-9.
4. Blake C.L., and Merz C.J. (1998). *UCI Repository of Machine Learning Databases*. Univ. of California at Irvine, Dept. of Information and Computer Science (http://www.ics.uci.edu/mlearn/MLRepository.html).
5. Bradford J.P., Kunz C., Kohavi R, Brunk C and Brodley C E. (1998). Pruning decision trees with misclassification costs. In: *Proceedings of 10th European Conference on Machine Learning*, pp. 131-136.
6. Bradley, A.P. and Lovell, B.C. (1995). Cost-Sensitive Decision Tree Pruning: Use of the ROC Curve, *Eighth Australian Joint Conference on Artificial Intelligence*, Canbarra, Australia, vol. 1, p. 1.
7. Brefeld, U., Geibel, P. and Wysotzki, F. (2003). Support vector machines with example dependent costs. *Proceedings of the European Conference on Machine Learning*, pp 23-34.
8. Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J. (1984) Classification and regression trees. Wadsworth, Belmont, CA, vol. 1, p. 358
9. Chai, X., Deng, L., Yang, Q. and Ling, C.X. (2004). Test-Cost Sensitive Naive Bayes Classification. *Proc. 4th IEEE International Conference on Data Mining* (*ICDM '04*), pp. 51-58.

10. Chawla, N.V. (2003). C4. 5 and imbalanced data sets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure. *Workshop on Learning from Imbalanced Data Sets II*, pp 8-13.
11. Chen, N., Ribeiro, B., Vieira, A., Duarte, J. and Neves, J. (2010). Weighted Learning Vector Quantization to Cost-Sensitive Learning. *Artificial Neural Networks*-CANN 2010, pp. 277-281.
12. Ciraco, M., Rogalewski, M. and Weiss, G. (2005). Improving classifier utility by altering the misclassification cost ratio. *Proceedings of the 1st international workshop on Utility-based data mining*, pp. 46-52.
13. Cohn, D., Atlas, L. and Ladner, R. (1994). Improving generalization with active learning', *Machine Learning*, Vol. 15(2), pp. 201-221.
14. Crone, S.F., Lessmann, S. and Stahlbock, R. (2005). Utility based data mining for time series analysis: cost-sensitive learning for neural network predictors. *Proceedings of the 1st international workshop on Utility-based data mining*, pp. 59-68.
15. Dasgupta, S., Hsu, D. and Monteleoni, C. (2007). *A general agnostic active learning algorithm*. Technical Report CS2007-0898, Department of Computer Science and Engineering, University of California, San Diego.
16. Davis, J.V., Ha, J., Rossbach, C.J., Ramadan, H.E. and Witchel, E. (2006). Cost-sensitive decision tree learning for forensic classification. *ECML-06*, pp 622-629.
17. Dempster, A.P., Laird, N.M. and Rubin, D.B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, Vol. 39, No. 1, pp. 1-38.
18. Domingos, P. (1999). MetaCost: a general method for making classifiers cost-sensitive. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 155-164.
19. Domingos, P. and Provost, F. (2000). Well-trained PETs: Improving probability estimation trees. CDER Working Paper #00-04-IS Stern School of Business.
20. Drummond, C. and Holte, R.C. (2000). Exploiting the cost (in) sensitivity of decision tree splitting criteria. *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 239-246.

21. Drummond, C. and Holte, R.C. (2003). C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. *Workshop on Learning from Imbalanced Data Sets II*, pp. 239-246.
22. Du, J., Cai, Z. and Ling, C.X. (2007). Cost-Sensitive Decision Trees with Pre-pruning. *Lecture Notes in Computer Science*, Vol. 4509, p. 171.
23. Elkan, C. (2001). The foundations of cost-sensitive learning. In: *Proceeding of the Seventeenth International Joint Conference of Artificial Intelligence*, ed. Bernhard Nebel, Morgan Kaufmann, 4-10 August 2001, Seattle, pp. 973-978.
24. Erray, W. and Hacid, H. (2006). ANew Cost Sensitive Decision Tree Method: Application for Mammograms Classification. *IJCSNS*, Vol. 6, No. 11, p. 130.
25. Esposito, F., Malerba, D. and Semeraro, G. (1997). A comparative analysis of methods for pruning decision trees. *IEEE transactions on pattern analysis and machine intelligence*, Vol. 19, pp. 476-491.
26. Fan, W., Stolfo, S.J., Zhang, J. and Chan, P.K. (1999). AdaCost: Misclassification cost-sensitive boosting. *Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 97-105.
27. Friedman, J.H., R. Kohavi, and Y. Yun (1996). Lazy decision trees. In: *Proceedings of the 13th National Conference Artificial Intelligence* (AAAI96), pp. 717-724.
28. Forman, G. and Cohen, I. (2004). Learning from Little: Comparison of Classifiers Given Little Training. *Knowledge Discovery in Databases: PKDD 2004: 8th European Conference on Principles and Practice of Knowledge Discovery in Databases,* Pisa, Italy, September 20-24, 2004.
29. Fumera, G. and Roli, F. (2002). Cost-sensitive learning in support vector machines. *In VIII Convegno Associazione Italiana per L'Intelligenza Artificiale*.
30. Gama, J. (2000). A cost-sensitive iterative Bayes. *Workshop on Cost-Sensitive Learning at the Seventeenth International Conference on Machine Learning (WCSL at ICML-2000)*.
31. Geibel, P. and Wysotzki, F. (2003). Perceptron based learning with example dependent and noisy costs. *Proceedings of the Twentieth International Conference on Machine Learning*., pp 583-590.
32. Greiner, R., Grove, A.J. and Roth, D. (2002). Learning cost-sensitive active classifiers. *Artificial Intelligence*, Vol. 139, no. 2, pp. 137-174.

33. Ikizler, N. (2002). *Benefit Maximizing Classification Using Feature Intervals*. The Department of Computer Engineering and the Institute of Engineering and Sciences, Bilkent University.
34. Japkowicz, N. (2000). The class imbalance problem: Significance and strategies. *Proceedings of the 2000 International Conference on Artificial Intelligence* (*ICAI'2000*), Vol(1), pp. 111-117.
35. Knoll, U., Nakhaeizadeh, G. and Tausend, B. (1994). Cost-sensitive pruning of decision trees. *Proceedings of the European conference on machine learning on Machine Learning table of contents*, pp. 383-386.
36. Kohavi, R. and John, G.H. (1997). Wrappers for feature subset selection. *Artificial intelligence*, vol. 97, no. 1-2, pp. 273-324.
37. Kotsiantis, S. and Pintelas, P. (2004), A cost sensitive technique for ordinal classification problems, *Methods and Applications of Artificial Intelligence*, pp. 220-229.
38. Kotsiantis, S., Kanellopoulos, D. and Pintelas, P. (2006), Handling imbalanced datasets: A review, *GESTS International Transactions on Computer Science and Engineering*, vol. 30, no. 1, pp. 25-36.
39. Kretowski, M. and Grzes, M. (2007). Evolutionary Induction of Decision Trees for Misclassification Cost Minimization. *Proceedings of the 8th international conference on Adaptive and Natural Computing Algorithms* (Part I), pp 1-10.
40. Kubat, M. and Matwin, S. (1997). Addressing the curse of imbalanced training sets: one-sided selection. *Proceedings of the Fourteenth International Conference on Machine Learning*, pp 179-186.
41. Kukar, M. and Kononenko, I. (1998). Cost-sensitive learning with neural networks. *Proceedings of the 13th European Conference on Artificial Intelligence* (ECAI-98), pp. 445-449.
42. Kwedlo, W. and Kretowski, M. (2001). An Evolutionary Algorithm for Cost-Sensitive Decision Rule Learning. *ECML-01*, pp. 288-299.
43. Langford, J. and Zadrozny, B. (2005). Estimating class membership probabilities using classier learners. *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*.
44. Li, J., Li, X. and Yao, X. (2005). Cost-Sensitive Classification with Genetic

Programming. *The 2005 IEEE Congress on Evolutionary Computation, 2005*, Vol. 3, pp 2114-2121.

45. Li, Y.F., Kwok, J.T. and Zhou, Z.H. (2010). Cost-Sensitive Semi-Supervised Support Vector Machine. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 500-505.
46. Liang, H., Zhang, H. and Yan, Y. (2006). Decision Trees for Probability Estimation: An Empirical Study. *Tools with Artificial Intelligence, 18th IEEE International Conference on*, pp. 756-764.
47. Lin, Y., Lee, Y. and Wahba, G. (2002). Support Vector Machines for Classification in Nonstandard Situations. *Machine Learning*, Vol. 46, No. 1, pp. 191-202.
48. Ling, C.X., Yang, Q., Wang, J., and Zhang, S. (2004). Decision trees with minimal costs. In: *Proceeding of the Twenty First International Conference on Machine Learning*, Vol. 69, ed. Carla E. Brodley, ACM Press, 4-8 July 2004, Banff, Alberta, pp. 69-76
49. Ling, C.X., Sheng, S., Bruckhaus, T. and Madhavji, N.H. (2005). Predicting Software Escalations with Maximum ROI. *Proceedings of the Fifth IEEE International Conference on Data Mining*, pp. 717-720.
50. Ling, C.X., Sheng, S. and Yang, Q. (2006). Test Strategies for Cost-Sensitive Decision Trees. *IEEE Transactions on Knowledge and Data Engineering*, 18(8): 1055-1067.
51. Liu, X.Y. (2006). Training Cost-Sensitive Neural Networks with Methods Addressing the Class Imbalance Problem. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 18, No. 1, pp. 63-77.
52. Liu, A., Jun, G. and Ghosh, J. (2009). Spatially Cost-Sensitive Active Learning. SDM 2009, pp. 14-825.
53. Liu, A., Jun, G. and Ghosh, J. (2009). A self-training approach to cost sensitive uncertainty sampling. *Machine learning,* 76**,** 257-270.
54. Lo, H. Y., Wang, J. C., Wang, H. M. and Lin, S. D. (2011) Cost-sensitive stacking for audio tag annotation and retrieval. ICASSP 2011, pp. 2308-2311
55. Masnadi-Shirazi, H. and Vasconcelos, N. (2010) Risk minimization, probability elicitation, and cost-sensitive SVMs. *Proceedings of the 27 th International Conference on Machine Learning.* Haifa, Israel. pp. 759-766.

56. Masnadi-Shirazi, H. and Vasconcelos, N. (2011) Cost-sensitive boosting. *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* 33**,** 294-309.

57. Maloof, M. (2003). Learning when data sets are imbalanced and when costs are unequal and unknown. *Proc of International Conference on Machine Learning*, pp 154-160.

58. Margineantu, D.D. (2001). *Methods for Cost-sensitive Learning*. Oregon State University, 2001.

59. Margineantu, D.D. (2002a). Class probability estimation and cost-sensitive classification decisions. *Proceedings of the 13th European Conference on Machine Learning*, pp. 270-281.

60. Margineantu, D.D. and Dietterich, T.G. (2002b). Improved class probability estimates from decision tree models. *Nonlinear Estimation and Classification: Lecture Notes in Statistics*, Vol. 171, pp. 169–184.

61. Margineantu, D.D. (2005). Active Cost-Sensitive Learning. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, IJCAI-05. pp. 1613-1622

62. Masnadi-Shirazi, H. and Vasconcelos, N. (2007). Asymmetric boosting. *Proceedings of the 24th international conference on Machine learning*, pp. 609-619.

63. Melville, P. and Mooney, R.J. (2004). Diverse ensembles for active learning. In *Proceedings of 21st International Conference on Machine Learning*, pp 584-591.

64. Merler, S., Furlanello, C., Larcher, B. and Sboner, A. (2003). Automatic model selection in cost-sensitive boosting. *Information Fusion*, Vol. 4, No. 1, pp. 3-10.

65. Mingers, J. (1987). Expert systems-rule induction with statistical data. *Journal of the operational research society*, pp. 39-47.

66. Niblett, T. and Bratko, I. (1986). Learning decision rules in noisy domains. *Developments in Expert Systems*, Cambridge University Press.

67. Niculescu-Mizil, A. and Caruana, R., (2005). Obtaining Calibrated Probabilities from Boosting. *Proc. 21st Conference on Uncertainty in Artificial Intelligence* (UAI ’05), pp 413-420.

68. Niculescu-Mizil, A. and Caruana, R. (2005). Predicting good probabilities with supervised learning. *Proc. Int. Conf. on Machine* Learning (ICML-05), pp 61-74.

69. Nigam, K., McCallum, A.K., Thrun, S. and Mitchell, T. (2000). Text Classification from Labelled and Unlabelled Documents using EM. *Machine Learning*, Vol. 39, No. 2, pp. 103-134.
70. Norton, S.W. (1989). Generating better decision trees. *Proceedings of the Eleventh International Conference on Artificial Intelligence*, pp. 800-805.
71. Nunez, M. (1991). The use of background knowledge in decision tree induction. *Machine Learning*, Vol. 6, No. 3, pp. 231-250.
72. Oza, N.C. (2000), Ensemble Data Mining Methods, *NASA Ame Research Center*.
73. Pagallo, G. and Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5: 71-99.
74. Pazzani, M., Merz, C., Murphy, P., Ali, K., Hume, T. and Brunk, C. (1994). Reducing misclassification costs. *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 217-225.
75. Platt, J.C. (1999). Probabilities for SV machines. *Advances in Neural Information Processing Systems*, pp. 61-74.
76. Provost, F. and Domingos, P. (2003). Tree Induction for Probability-Based Ranking. *Machine Learning*, Vol. 52, No. 3, pp. 199-215.
77. Qin, Z., Zhang, S. and Zhang, C. (2004). Cost-Sensitive Decision Trees with Multiple Cost Scales. *Proceedings of the 17th Australian Joint Conference on Artificial Intelligence* (*AI 2004*)*, Cairns, Queensland, Australia*, pp. 6-10.
78. Qin, Z., Zhang, C. and Zhang S. (2006). Missing or absent? A Question in Cost-sensitive Decision Tree. *Proceedings of the Fourth International Conference on Active Media Technology* (AMT006), pp 118-125.
79. Qin, Z., Zhang, C., Xie, X. and Zhang, S. (2005). Dynamic Test-Sensitive Decision Trees with Multiple Cost Scales. *Second International Conference on Fuzzy Systems and Knowledge Discovery*, FSKD 2005, p. 402-405.
80. Qin, Z., Zhang, S., Liu, L. and Wang, T. (2008). Cost-sensitive Semi-supervised Classification using CS-EM. *IEEE 8th International Conference on Computer and Information Technology*, 8 - 11 July, 2008: 131-136.
81. Qin, Z., Wang, T. and Zhang, S. (2010). Incorporating medical history to cost sensitive classification with lazy learning strategy. *Proceedings of the International Conference on Progress in Informatics and Computing conference.* IEEE. Vol. 1,

pp. 19-23.

82. Qin, Z., Zhang, C., Wang, T. and Zhang, S. (2011) Cost sensitive classification in data mining. *Advanced Data Mining and Applications*, pp. 1-11.
83. Quinlan, J.R. (1986). Induction of decision trees. *Machine learning*, vol. 1, no. 1, pp. 81-106.
84. Quinlan, J.R. (1987). Generating production rules from decision trees. vol. 30107.
85. Quinlan, J.R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo.
86. Rokach, L. and Maimon, OZ. (2008). Data mining with decision trees: theory and applications. *World Scientific Publishing Company* (March 2008).
87. Roli, F. (2005). Semi-supervised Multiple Classifier Systems: Background and Research Directions. *Proceedings of Multiple Classifier Systems: 6th International Workshop, MCS 2005*, Seaside, CA, USA, June 13-15, 2005.
88. Seeger, M. (2001). Learning with labelled and unlabelled data. *Inst. for Adaptive and Neural Computation, Technical Report*, Univ. of Edinburgh.
89. Sen, P. and Getoor, L. (2006). Cost-sensitive learning with conditional Markov networks. *Proceedings of the 23rd international conference on Machine learning*, pp. 801-808.
90. Seung, H.S., Opper, M. and Sompolinsky, H. (1992). Query by committee. *Proc. 5th Workshop on Computational Learning Theory*, pp. 287-294.
91. Sheng, S. and Ling, C.X. (2005). Hybrid Cost-Sensitive Decision Tree. *Knowledge Discovery in Databases: PKDD 2005*, pp. 274-284.
92. Sheng, S., Ling, C.X. and Yang, Q. (2005). Simple Test Strategies for Cost-Sensitive Decision Trees. *Machine Learning: ECML 2005*, pp. 365-376.
93. Sheng, V.S. and Ling, C.X. (2006). Feature value acquisition in testing: a sequential batch test algorithm. *Proceedings of the 23rd international conference on Machine learning*, pp. 809-816.
94. Sheng, V.S., Ling, C.X., Ni, A. and Zhang, S. (2006). Cost-Sensitive Test Strategies. *Proc. 21st Nat'l Conf. Artificial Intelligence*. pp. 482-487.
95. Sun, Q. and Pfahringer, B. (2011), Bagging Ensemble Selection, *AI 2011: Advances in Artificial Intelligence*, pp. 251-260.

96. Sun, Y., Wong, A.K.C. and Wang, Y. (2005). Parameter inference of cost-sensitive boosting algorithms. *Machine Learning and Data Mining in Pattern Recognition, 4th International Conference*. pp. 21-30.
97. Tan, M. (1993). Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning*, Vol. 13, No. 1, pp. 7-33.
98. Ting, K.M. (1998). Inducing Cost-sensitive trees via instance weighting. In: *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery*, pp. 23-26.
99. Ting, K.M. and Zheng, Z. (1998). Boosting Cost-Sensitive Trees. *Proceedings of the First International Conference on Discovery Science*, pp. 244-255.
100. Ting, K.M. and Zheng, Z. (1998). Boosting Trees for Cost-Sensitive Classifications. *Machine Learning: ECML-98: 10th European Conf on Machine Learning*, pp. 190-195.
101. Ting, K. M. (2002a). Cost-Sensitive Classification using Decision Trees, Boosting and MetaCost. *Heuristic and optimization for knowledge discovery. Hershey, PA: Idea Group Publishing*.
102. Ting, K.M. (2002b). An instance-weighting method to induce cost-sensitive trees. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 14, No. 3, pp. 659-665.
103. Turney, P. (2000). Types of cost in inductive concept learning. *Workshop on Cost-Sensitive Learning at the Seventeenth International Conference on Machine Learning*, pp. 1511.
104. Turney, P.D. (1995) Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, Vol. 2, No. 1995, pp. 369-409.
105. Vadera, S. (2005). Inducing cost-sensitive non-linear decision trees. *Technical report 03-05-2005*, School of Computing, Science and Engineering, University of Salford.
106. Venkatesan, A., Krishnan, N. C. and Panchanathan, S. (2010) Cost-sensitive boosting for concept drift. *In International Workshop on Handling Concept Drift in Adaptive Information Systems*. pp. 41-47.

107. Vapnik, V.N. (1982). *Estimation of Dependences Based on Empirical Data*. Springer-Verlag New York, Inc.
108. Viaene, S. and Dedene, G. (2005). Cost-sensitive learning and decision making revisited. *European journal of operational research*, Vol. 166, No. 1, pp. 212-220.
109. Viola, P., and Jones, M. (2002). Fast and robust classification using asymmetric adaboost and a detector cascade. *Advances in Neural Information Processing System* 14, pp 1311--1318.
110. Wang, T., Qin, Z., Jin, Z. and Zhang, S. (2009) Handling over-fitting in test cost-sensitive decision tree learning by feature selection, smoothing and pruning. *Journal of Systems and Software.* Vol. 83, pp. 1137-1147.
111. Wang, T., Qin, Z., Zhang, S. and Zhang, C. (2011) Cost-sensitive classification with inadequate labelled data. *Information Systems*. Vol. 37, pp. 508-516.
112. Webb, G.I. (1996). Cost-Sensitive Specialization. *Proceedings of the Fourteenth Pacific Rim International Conference on Artificial Intelligence*, pp. 23–34.
113. Wettschereck, D., Aha, D.W. and Mohri, T. (1997), A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms, *Artificial Intelligence Review*, vol. 11, no. 1, pp. 273-314.
114. Wilke, W. and Bergmann, R. (1996). Considering decision cost during learning of feature weights. *Proceedings of the Third European Workshop on Case-Based Reasoning*, pp. 460-472.
115. Witten, I. H., and Frank, E. (2000). *Data Mining: Practical Machine Learning Techniques with Java Implementations*. 2nd edition, Morgan Kaufmann Publishers.
116. Wolpert, D.H. (1992). *Stacked generalization*. Neural Networks, vol. 5, pp. 241-259.
117. Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B. and Yu, P.S. (2008), Top 10 algorithms in data mining, *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1-37.
118. Xu, J., Cao, Y., Li, H. and Huang, Y. (2006). Cost-sensitive Learning of SVM for Ranking. *Machine Learning: ECML 2006*, pp 833-840.
119. Zadrozny, B. and Elkan, C. (2001a). Learning and making decisions when costs and probabilities are both unknown. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 204-213.

120. Zadrozny, B. and Elkan, C. (2001b). Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 609–616.

121. Zadrozny, B. and Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. ACM New York, NY, USA, pp. 694-699.

122. Zadrozny, B., Langford, J. and Abe, N. (2003). Cost-sensitive learning by cost-proportionate example weighting. *Third IEEE International Conference on Data Mining,* ICDM-2003, pp. 435-442.

123. Zadrozny, B. (2004). Learning and evaluating classifiers under sample selection bias. *Proceedings of the twenty-first international conference on Machine learning*, pp. 114.

124. Zadrozny B. (2005). One-Benefit learning: cost-sensitive learning with restricted cost information. In: *Proceedings of the 1st international workshop on Utility-based data mining*, ACM Press, Chicago, Illinois, pp. 53-58.

125. Zhang, J. and Mani, I. (2009), kNN approach to unbalanced data distributions: a case study involving information extraction. *In Proceedings of Workshop on Learning from Imbalanced Datasets*.

126. Zhang, S. (2003), KNN-CF Approach: Incorporating Certainty Factor to kNN Classification, *IEEE Intelligent Informatics Bulletin Vol. 11 No*, vol. 1.

127. Zhang, S., Qin, Z., Ling, C.X. and Sheng, S. (2005). " Missing Is Useful': Missing Values in Cost-Sensitive Decision Trees. *IEEE Transactions on Knowledge and Data Engineering,* Vol. 17, No. 12, pp. 1689-1693.

128. Zhang, S. (2012). Decision Tree Classifiers Sensitive to Heterogeneous Costs. *Journal of Systems and Software*, 85(4): 771–779.

129. Zhang, S. (2010). Cost-Sensitive Classification With Respect To Waiting Cost. *Knowledge-Based Systems*, 23(5): 369–378.

130. Zhu, X. and Wu, X. (2005). Cost-Constrained Data Acquisition for Intelligent Data Preparation. *Knowledge and Data Engineering, IEEE Transactions on*, Vol. 17, No. 11, pp. 1542-1556.

131. Zhu, X. (2005). *Semi-Supervised Learning Literature Survey*. Computer Science TR 1530, University of Wisconsin-Madison.

132. Zubek, V.B. and Dietterich, T.G. (2002) Pruning Improves Heuristic Search for Cost-Sensitive Learning. In: *Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 27-34.