Cost-Sensitive Classification with k-Nearest Neighbors

Zhenxing Qin[1], Alan Tao Wang[1], Chengqi Zhang[1], and Shichao Zhang[1,2,*]

[1] The Centre for QCIS, Faculty of Engineering and Information Technology
University of Technology Sydney, Australia
[2] College of CS&IT, Guangxi Normal University, Guilin, 541004, China
alant.wang@gmail.com
{Zhenxing.Qin,Chengqi.Zhang,Shichao.Zhang}@uts.edu.au

Abstract. Cost-sensitive learning algorithms are typically motivated by imbalance data in clinical diagnosis that contains skewed class distribution. While other popular classification methods have been improved against imbalance data, it is only unsolved to extend k-Nearest Neighbors (kNN) classification, one of top-10 datamining algorithms, to make it cost-sensitive to imbalance data. To fill in this gap, in this paper we study two simple yet effective cost-sensitive kNN classification approaches, called Direct-CS-kNN and Distance-CS-kNN. In addition, we utilize several strategies (i.e., smoothing, minimum-cost k value selection, feature selection and ensemble selection) to improve the performance of Direct-CS-kNN and Distance-CS-kNN. We conduct several groups of experiments to evaluate the efficiency with UCI datasets, and demonstrate that the proposed cost-sensitive kNN classification algorithms can significantly reduce misclassification cost, often by a large margin, as well as consistently outperform CS-4.5 with/without additional enhancements.

1 Introduction

Classification aims at generating a classifier which minimizes classification errors, which is one of the main research topics in machine learning and data mining. Therefore, there are great many classification algorithms developed, typical examples include Decision Tree, Naïve Bayes, Instance based learning (e.g. kNN) and SVM. Twenty years ago, motivated by imbalance data in clinical diagnosis that contains skewed class distribution, these popular classification algorithms had been extended to deal with the classification tasks with non-uniform cost, called cost-sensitive classification, and attracted vast interest of data mining researchers [1, 2, 8, 16, 26]. These approaches incorporate misclassification cost and other costs in the classification technique, thus provide practical classification result in a multiple-cost environment. However, it is only unsolved to extend k-Nearest Neighbors (kNN) classification, one of top-10 datamining algorithms, to make it cost-sensitive to imbalance data.

* Corresponding author.

M. Wang (Ed.): KSEM 2013, LNAI 8041, pp. 112–131, 2013.

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. Despite of the popularity of KNN, very little work's been reported on KNN in the area cost-sensitive learning. Hence some challenges are must out there to stop us. However, regarding to the potential of KNN, it is still worth a good try to clarify the challenges and give some solutions.

To make KNN cost-sensitive, in this paper, two simple but effective approaches, Direct-CS-KNN and Distance-CS-KNN, are proposed to minimise the misclassification cost in cost sensitive learning. In order to clarify challenges and difficulties encountered in our new algorithms, completed performance studies are conducted and compared with the benchmark method cost-sensitive C4.5, by bunches of experiments with various cost settings and multiple typical datasets.

Furthermore, we also propose several additional enhancement methods - smoothing, minimum-cost K value selection, feature selection and ensemble selection, to further improve the performance of our new cost-sensitive KNN algorithms. The smoothing improvement is also applied to cost-sensitive C4.5 for performance study.

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 in case of with or without enhancements.

The rest of paper is organized as follows: Section 2 provides a brief review of the cost-sensitive learning, KNN classification Section 3 describes our two new cost-sensitive KNN algorithms and several improvements which can effectively reduce misclassification cost in KNN classification. Section 3 describes some additional enhancement methods: feature selection, direct cost-sensitive learning/probability calibration and ensemble selection methods. Experimental results for six UCI data sets are presented in Section 5. Finally, in Section 6 we conclude the work with a discussion of future improvements.

2 Related Work

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. Traditional data mining methods aimed at minimizing error rate will perform poorly in these areas, as they assume equal misclassification cost and relatively balanced class distributions. 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 [2].

Most of the cost-sensitive learning methods assume that for an M-class problem, an M by M cost matrix C is available at learning time, and does not change during the learning or decision making process. So it is static. 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.

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	$C(1, 0) = C_{10}$	$C(1, 1) = C_{11}$

In 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 labeling an example incorrectly should always be greater than the cost of labeling it correctly. Mathematically, it should always be the case that $C_{10} > C_{00}$ and $C_{01} > C_{11}$ [2].

As per Elkan, 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 $i \neq j$ the prediction is incorrect. The optimal prediction for an example x is the class i that minimizes:

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

The most popular base algorithms used in cost-sensitive learning include Decision Tree, Naïve Bayes and SVM.

2.2 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 [22].

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.

2.3 Direct Cost-Sensitive Classification

Any learned classifier that can provide conditional probability estimates for training examples can also provide conditional probability estimates for test examples. Using these probability estimates we can directly compute the optimal class label for each test example using the cost matrix. This cost-sensitive learning method is called direct cost-sensitive learning [26].

All direct cost-sensitive learning algorithms have one thing in common: They do not manipulate the internal behavior 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.

3 Making KNN Cost-Sensitive - The Proposed Approach

In this paper, 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.

3.1 Direct Cost-Sensitive KNN

Direct Cost-sensitive approach is quite simple and intuitive which has been studied with decision tree C4.5. In this paper, we use KNN algorithm to train a traditional non-cost-sensitive classifier as the baseline. After the K nearest neighbors is selected, we calculate class probability estimate using below formula:

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

Where Ki is the number of selected neighbors whose class label is i. Using the above probability estimate and Eq. 1, 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 paper, 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 generate very poor probability estimate

As we described in Section 2, several methods have been proposed for obtaining better probability estimate from traditional cost-blind classifiers in direct cost-sensitive learning. Zadrozny and Elkan [2] 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 paper, 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 section 2.2 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:

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

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 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 3.2), and compare this approach to the other proposed variations of Cost-sensitive KNN algorithms.

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 2.2:

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

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 unlabeled example to *P* is $P_p = Wpa/(Wpa+Wna)$.And the probability of labeling an unlabeled 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 unlabeled example is classified as N, and the probability of this prediction is $C_p/(C_p + C_n)$. Otherwise, the unlabeled 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**.

3.3 Potential and Challenges

We conduct performance for **DirectCS-KNN** and the **CS-C4.5** on different datasets. In most of datasets, the best result of **DirectCS-KNN** outperforms the **CS-C4.5** on both of the minimal cost and AUC measurements. A typical result is shown in Table 4 of section 5.2 (marked with gray background in the table). As we predicted before, KNN does has the great potential to cost-sensitive learning.

However, we do find some challenges when apply KNN into cost-sensitive learning environment. Firstly, the choice of K, which is also an open question in KNN study. Secondly, the large working loads to tuning the K while we change the cost ratio and datasets.

Good news is there are lot of work reported to tackle the challenges in general KNN. On the other hand, we introduce some additional enhancements on DirectCS-KNN which make us easier to search a good enough K and reduce the tuning work-load. The enhancements are described in next section.

4 Additional Enhancements

4.1 Calibration Methods for Improving Probability Estimates in Cost-Sensitive Learning

Any learned classifier that can provide conditional probability estimates for training examples can also provide conditional probability estimates for test examples. If the learned model does not explicitly compute these values, most classifiers can be modified to output some value that reflects the internal class probability estimate [9]. Using these probability estimates we can directly compute the optimal class label for each test example using the cost matrix.

However, it is well known that the probability estimates provided by classifiers from many error based learners are neither unbiased, nor well calibrated. Decision tree (C4.5) and instance based learners (KNN) are well known examples. Two smoothing methods, Laplace Correction and M-Smoothing, have been proposed for obtaining better probability estimates from decision tree classifiers [2, 13].

- **Laplace Correction.** Using C4.5, [13] 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.** [26] 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.

$$\mathrm{P_T}(i \mid x) = \frac{N_i + b*m}{N+m}$$

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×m = 10. The experiments conducted by Zadrozny and Elkan [24] 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 [1] 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 [12] and Isotonic Regression. These methods can also be used in directly cost-sensitive learning algorithms.

- **Platt Scaling.** 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 [10].
- **Isotonic Regression.** Compared to Platt Calibration, Isotonic Regression is a more powerful calibration method which can correct any monotonic distortion [10 24, 25] 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:

  $$m' = \mathrm{argmin}_z \sum (yi - z(fi))^2$$

4.2 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 neighbor 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 [4].

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" neighbors with the most relevant attributes which minimizes misclassification cost.

4.3 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:

1. Start with an empty ensemble;
2. 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;
3. Repeat Step 2 until all models have been examined;
4. 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.

4.4 KNN with Cost-Sensitive Feature Selection

Real world data sets usually 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 I mentioned in Section 4.2, 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 [4], 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 [4] 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 paper, 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:

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

4.5 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 paper, we propose an ensemble based method, more specific, **Cost-sensitive Stacking**, to handle the parameter *K*.

As we mentioned in section 2.5, 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 [21].

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 Equation 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 [1].

5 Experimental Evaluation

5.1 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 and its enhanced version with smoothing. All these algorithms are implemented in Weka (Witten and Frank 2000), and they are listed in Table 2 below.

Table 2. 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
8	C4.5 with Minimum Expected Cost and Smoothing	CS-C4.5-SM	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 3.

Table 3. 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-	41	3347	1655/1692

These data sets are chosen based on the following criteria:

- Data sets should be two-class because the cost-sensitive KNN classification algorithms we are evaluating 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.
- This experiment does 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 or replace them using Weka's "ReplacingMissingValues" filter.
- 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 1.02 (Waveform-5000) to 8.8 (Page-blocks).

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.

5.2 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. For easy reading and

discussing, the results without enhancements are marked with gray background on each table.

The first experiment aims to show the performance of enhancements. Results are listed in Table 4. It lists the key performance measurements such as average misclassification cost and AUC for the Statlog(heart) data set. The experiment shows 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. Key performance measurements on Statlog(heart)

Table 4.1. Average Misclassification Cost

Data Set	DirectCS-KNN	DirectCS-KNN-SM	DirectCS-KNN-CSK	CS-C4.5	CS-C4.5-SM
Statlog(heart)	0.3815	0.3605	0.3556	0.6704	0.4938

Table 4.2. Area Under ROC (AUC)

Data Set	DirectCS-KNN	DirectCS-KNN-SM	DirectCS-KNN-CSK	CS-C4.5	CS-C4.5-SM
Statlog(heart)	0.744	0.763	0.768	0.759	0.776

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 (with or without enhancements), 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 5 below.

Table 5. Key performance measurements on Statlog(heart)

Table 5.1. Average Misclassification Cost

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

Table 5.2. 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 pager. 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 6 and 7. Table 6 lists the average misclassification cost on selected five UCI data sets. Table 7 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 6. Average misclassification cost on selected UCI data sets

Table 6.1. Cost Ratio (FP=1, FN=2)

Data Set	DistanceCS-KNN	DistanceCS-KNN-FS	DistanceCS-KNN-STK	CS-C4.5	CS-C4.5-SM
Diabetes	0.3758	0.3596	0.3633	0.3828	0.3722
Credit-g	0.428	0.417	0.402	0.435	0.408
Page-blocks	0.0607	0.0592	0.0585	0.0422	0.0397
Spambase	0.1091	0.1044	0.0993	0.1052	0.0973
Waveform-5000	0.1341	0.1315	0.1298	0.1951	0.1933

Table 6.2. Cost Ratio (FP=1, FN=5)

Data Set	DistanceCS-KNN	DistanceCS-KNN-FS	DistanceCS-KNN-STK	CS-C4.5	CS-C4.5-SM
Diabetes	0.5573	0.536	0.5352	0.6003	0.5789
Credit-g	0.598	0.5815	0.582	0.77	0.681
Page-blocks	0.0965	0.0896	0.0838	0.0846	0.0767
Spambase	0.2006	0.1937	0.1864	0.2121	0.1905
Waveform-5000	0.1637	0.1596	0.1562	0.3756	0.3254

Table 6.3. Cost Ratio (FP=1, FN=10)

Data Set	DistanceCS-KNN	DistanceCS-KNN-FS	DistanceCS-KNN-STK	CS-C4.5	CS-C4.5-SM
Diabetes	0.5898	0.5832	0.5869	0.8268	0.7642
Credit-g	0.717	0.6756	0.62	1.043	0.832
Page-blocks	0.1214	0.1163	0.1031	0.1297	0.1108
Spambase	0.3019	0.2836	0.2712	0.3512	0.3122
Waveform-5000	0.1933	0.1896	0.1815	0.611	0.5752

Table 6.4. Cost Ratio (FP=1, FN=20)

Data Set	DistanceCS-KNN	DistanceCS-KNN-FS	DistanceCS-KNN-STK	CS-C4.5	CS-C4.5-SM
Diabetes	0.7591	0.725	0.717	0.9635	0.8281
Credit-g	0.939	0.822	0.8161	1.258	1.035
Page-blocks	0.1782	0.1665	0.1546	0.1838	0.1633
Spambase	0.4027	0.3817	0.3552	0.5781	0.4842
Waveform-5000	0.1963	0.1915	0.1848	1.0678	0.9912

Table 7. 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.

6 Conclusion and Future Work

In this paper, 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.

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

Other possible improvements include using calibration methods such as Platt Scaling or Isotonic Regression to get better class membership probability estimation, trying other search algorithms such as Genetic algorithm with cost-sensitive fitness function in our feature selection wrapper.

Acknowledgement. This work is supported in part by the Australian Research Council (ARC) under large grant DP0985456; the China "1000-Plan" National Distinguished Professorship; the China 863 Program under grant 2012AA011005; the Natural Science Foundation of China under grants 61170131 and 61263035; the China 973 Program under grant 2013CB329404; the Guangxi Natural Science Foundation under grant 2012GXNSFGA060004; the Guangxi "Bagui" Teams for Innovation and Research; the Guangxi Provincial Key Laboratory for Multi-sourced Data Mining and Safety; and the Jiangsu Provincial Key Laboratory of E-business at the Nanjing University of Finance and Economics.

References

1. Domingos, P.: MetaCost: a general method for making classifiers cost-sensitive. In: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 155–164 (1999)
2. Elkan, C.: The foundations of cost-sensitive learning. In: Nebel, B. (ed.) Proceeding of the Seventeenth International Joint Conference of Artificial Intelligence, Seattle, August 4-10, pp. 973–978. Morgan Kaufmann (2001)
3. Greiner, R., Grove, A.J., Roth, D.: Learning cost-sensitive active classifiers. Artificial Intelligence 139(2), 137–174 (2002)
4. Kohavi, R., John, G.H.: Wrappers for feature subset selection. Artificial intelligence 97(1-2), 273–324 (1997)
5. Kotsiantis, S., Pintelas, P.: A cost sensitive technique for ordinal classification problems. In: Vouros, G.A., Panayiotopoulos, T. (eds.) SETN 2004. LNCS (LNAI), vol. 3025, pp. 220–229. Springer, Heidelberg (2004)
6. Kotsiantis, S., Kanellopoulos, D., Pintelas, P.: Handling imbalanced datasets: A review. GESTS International Transactions on Computer Science and Engineering 30(1), 25–36 (2006)

7. Li, J., Li, X., Yao, X.: Cost-Sensitive Classification with Genetic Programming. In: The 2005 IEEE Congress on Evolutionary Computation, vol. 3 (2005)
8. Ling, C.X., Yang, Q., Wang, J., Zhang, S.: Decision trees with minimal costs. In: Brodley, C.E. (ed.) Proceeding of the Twenty First International Conference on Machine Learning, Banff, Alberta, July 4-8, vol. 69, pp. 69–76. ACM Press (2004)
9. Margineantu, D.D.: Methods for Cost-sensitive Learning. Oregon State University (2001)
10. Niculescu-Mizil, A., Caruana, R.: Predicting good probabilities with supervised learning. Association for Computing Machinery, Inc., New York (2005)
11. Oza, N.C.: Ensemble Data Mining Methods, NASA Ame Research Center (2000)
12. Platt, J.C.: Probabilities for SV machines. In: Advances in Neural Information Processing Systems, pp. 61–74 (1999)
13. Provost, F., Domingos, P.: Tree Induction for Probability-Based Ranking. Machine Learning 52, 199–215 (2003)
14. Quinlan, J.R.: C4.5: Programs for machine learning. Morgan Kaufmann, San Mateo (1993)
15. Sun, Q., Pfahringer, B.: Bagging Ensemble Selection. In: Wang, D., Reynolds, M. (eds.) AI 2011. LNCS, vol. 7106, pp. 251–260. Springer, Heidelberg (2011)
16. Turney, P.: Types of cost in inductive concept learning. In: Workshop on Cost-Sensitive Learning at the Seventeenth International Conference on Machine Learning, p. 1511 (2000)
17. Wang, T., Qin, Z., Jin, Z., Zhang, S.: Handling over-fitting in test cost-sensitive decision tree learning by feature selection, smoothing and pruning. Journal of Systems and Software (JSS) 83(7), 1137–1147 (2010)
18. Wang, T., Qin, Z., Zhang, S.: Cost-sensitive Learning - A Survey. Accepted by International Journal of Data Warehousing and Mining (2010)
19. Wettschereck, D., Aha, D.W., Mohri, T.: A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. Artificial Intelligence Review 11(1), 273–314 (1997)
20. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Techniques with Java Implementations, 2nd edn. Morgan Kaufmann Publishers (2000)
21. Wolpert, D.H.: Stacked generalization. Neural Networks 5, 241–259 (1992)
22. Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Yu, P.S.: Top 10 algorithms in data mining. Knowledge and Information Systems 14(1), 1–37 (2008)
23. Zadrozny, B., Elkan, C.: Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In: Proceedings of the 18th International Conference on Machine Learning, pp. 609–616 (2001)
24. Zadrozny, B., Elkan, C.: Learning and making decisions when costs and probabilities are both unknown. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 204–213. ACM Press, San Francisco (2001)
25. Zadrozny, B., Elkan, C.: Transforming classifier scores into accurate multiclass probability estimates, pp. 694–699. ACM, New York (2002)
26. Zadrozny, B.: One-Benefit learning: cost-sensitive learning with restricted cost information. In: Proceedings of the 1st International Workshop on Utility-Based Data Mining, pp. 53–58. ACM Press, Chicago (2005)
27. Zhang, J., Mani, I.: kNN approach to unbalanced data distributions: a case study involving information extraction (2009)
28. Zhang, S.: KNN-CF Approach: Incorporating Certainty Factor to kNN Classification. IEEE Intelligent Informatics Bulletin 11(1) (2003)