Roulette Sampling for Cost-Sensitive Learning

Victor S. Sheng and Charles X. Ling

Department of Computer Science, University of Western Ontario,
London, Ontario, Canada N6A 5B7
{ssheng,cling}@csd.uwo.ca

Abstract. In this paper, we propose a new and general preprocessor algorithm, called *CSRoulette*, which converts any cost-insensitive classification algorithms into cost-sensitive ones. *CSRoulette* is based on cost proportional roulette sampling technique (called *CPRS* in short). *CSRoulette* is closely related to Costing, another cost-sensitive meta-learning algorithm, which is based on rejection sampling. Unlike rejection sampling which produces smaller samples, *CPRS* can generate different size samples. To further improve its performance, we apply ensemble (bagging) on *CPRS*; the resulting algorithm is called *CSRoulette*. Our experiments show that *CSRoulette* outperforms Costing and other meta-learning methods in most datasets tested. In addition, we investigate the effect of various sample sizes and conclude that reduced sample sizes (as in rejection sampling) cannot be compensated by increasing the number of bagging iterations.

Keywords: meta-learning, cost-sensitive learning, decision trees, classification, data mining, machine learning.

1 Introduction

Classification is a primary task of inductive learning in machine learning. However, most original classification algorithms ignore different misclassification errors; or they implicitly assume that all misclassification errors cost equally. In many real-world applications, this assumption is not true. For example, in medical diagnosis, misdiagnosing a cancer patient as non-cancer is very serious than the other way around; the patient could die because of the delay in treatment.

Cost-sensitive classification [11, 5, 8, 10, 12] has received much attention in recent years. Many works have been done; and they can be categorized into two groups. One is to design cost-sensitive learning algorithms directly [11 4, 7]. Another is to make wrappers that convert existing cost-insensitive inductive learning algorithms into cost-sensitive ones. This method is called cost-sensitive meta-learning, such as MetaCost [3], Costing [14], and CostSensitiveClassifier [13]. Section 2 provides a more complete review of cost-sensitive meta-learning approaches.

These cost-sensitive meta-learning techniques are useful because they let us reuse existing inductive learning algorithms and their related improvements. *CSRoulette* (see Section 3 and 4) is another effective cost-sensitive meta-learning algorithm. *CSRoulette* is very similar to Costing, as they both are based on sampling. Costing is

J. N. Kok et al. (Eds.): ECML 2007, LNAI 4701, pp. 724–731, 2007.

based on rejection sampling [14]. The weakness of rejection sampling is that the size of resampled training set is reduced dramatically, and much useful information is thrown away in the preprocessing procedure.

To overcome this weakness, *CSRoulette* is based on a cost-sensitive sampling technique – cost proportionate roulette sampling (CPRS in short). CPRS is an improvement of the advanced sampling. It can generate samples of any sizes from the original dataset. By default it generates samples with the same size as the original dataset (details in Section 3).

We compare *CSRoulette* with Costing and other cost-sensitive meta-learning methods. The experimental results show that *CSRoulette* outperforms others (see Section 5.1). Furthermore, we investigate the effect of various sample sizes, and conclude that reduced sample sizes cannot be compensated by increasing the number of iterations in ensemble learning (see Section 5.2).

2 Related Work

Cost-sensitive meta-learning converts existing cost-insensitive learning algorithms into cost-sensitive ones without modifying them. Thus, all the cost-sensitive meta-learning techniques become middleware components that pre-process the training data for a cost-insensitive learning algorithm or post-process the output of a cost-insensitive learning algorithm.

Currently, cost-sensitive meta-learning techniques fall into three categories. The first is *relabeling* the classes of instances, by applying the minimum expected cost criterion [3]. This approach can be further divided into two branches: relabeling training instances and relabeling test instances. MetaCost [3] belongs to the former branch. CostSensitiveClassifier (called CSC in short) [13] belongs to the latter branch. The second category is *Weighting* [10]. It assigns a certain weight to each instance in terms of its class, according to the misclassification costs, such that the learning algorithm is in favor of the class with high weight/cost. The Third is *sampling*, which changes the distribution of the training data according to their costs. Both Costing [14] and *CSRoulette* belong to this category. Thus, we provide a detailed comparison between them in Section 5.

Costing [14] uses the advanced sampling - rejection sampling - to change the distribution of the training set according to the misclassification costs shown in a cost matrix $C(i,j)$. More specifically, each example in the original training set is drawn once, and accepted into the sample with the accepting probability $C(i)/Z$, where $C(i)$ is the misclassification cost of class i, and Z is an arbitrary constant chosen with the condition $maxC(i) \leq Z$. However, we notice that rejection sampling has a shortcoming. The sample S' produced by rejection sampling is much smaller than the original training set S (i.e. $|S'| << |S|$); even the constant Z is set as $Z=maxC(i)$ to maximize the size of the sample $|S'|$, if the misclassification costs are not equal or the dataset is imbalanced. The learning model built on the reduced samples is not stable (see Section 5.1). Apparently to reduce the instability, Costing [14] applies bagging [2] to the rejection sampling.

However, cost proportionate roulette sampling can generate samples with any size. If it generates samples with the same size as the original training set, it uses more

available information in the training set. Thus, it can be expected that cost proportionate roulette sampling outperform rejection sampling.

3 Cost Proportionate Roulette Sampling (CPRS)

Roulette sampling is a stochastic sampling with replacement [6]. The training examples are mapped into segments of a line. The size of each segment is equal to the weight of the corresponding examples. The weight of each example is assigned according to the misclassification costs. To simplify this issue, we discuss binary classification here.

For binary classes, we assume we have the misclassification costs false positive (*FP*, the cost of misclassifying a negative instance into positive) and false negative (*FN*, the cost of misclassifying a positive instance into negative), and the cost of correct classification is zero. We simply assign *FP* as the weight to each negative instance, and assign *FN* as the weight to each positive instance. That is, the weight ratio of a positive instance to a negative instance is proportional to *FN/FP* [5].

In the cost proportional roulette sampling, we normalize their weights such that the sum of them equals to the number of instances. If we have a training set with *P* positive instances and *N* negative instances, then the weight of each negative instance is $\frac{FP\times(P+N)}{P\times FN+N\times FP}$, and the weight of each positive instance is $\frac{FN\times(P+N)}{P\times FN+N\times FP}$. Thus, the sum of the weights is *P+N*. That is, the length of the mapped line is *P+N*.

One of the important steps of roulette sampling is to select an example. Each time, roulette sampling generates a random number within the range of the length of the line. Then the example whose segment spans the random number is selected. The selection process does not stop until the total number of examples is reached. That is, users can indicate the size of samples generated by roulette sampling.

Like rejection sampling, roulette sampling makes use of the misclassification cost information. However, unlike rejection sampling, which draws examples independently from the distribution of the original training set, roulette sampling takes the distribution of original training set into consideration. In sum, roulette sampling integrates the effect of misclassification cost, the distribution of original training set, and the size of samples.

4 CPRS with Aggregation (CSRoulette)

Many researchers have shown that bagging (bootstrap aggregating) [2] can reliably improve base classifiers. Bagging is a voting process, which counts the votes from the base classifiers trained on different bootstrap samples. The bootstrap samples are generated from the original training set by uniformly sampling with replacement. In order to improve the performance of rejection sampling, Zadrozny et al. [14] take advantage of a voting algorithm, i.e., Costing, a procedure that is similar to bagging. Instead of using uniformly sampling with replacement, Costing uses rejection sampling to generate multiple samples and build a classifier on each sample.

Algorithm: *CSRoulette(T, B, C, I, x)*
Input: cost proportional roulette sampling (*CPRS*), training set *T*, based-classifier *B*, cost matrix *C*, a testing example *x*, and integer *I* (number of iterations of bootstrap).
For $r = 1$ to I do //build I classifiers
S'=*CPRS* (T, C)
Let $h_r = B(S')$
Output $h(x) = sign\left(\sum_{r=1}^{I} h_r(x)\right)$

Fig. 1. Pseudo-code of *CSRoulette*

In the same way that Zadrozny et al. apply bagging to rejection sampling to become Costing, we also apply bagging on cost proportional roulette sampling. That is, we repeatedly perform cost proportional roulette sampling to generate multiple set $S'_1, \ldots, S'_i$ from the original training set T, and learn a classifier from each sampled set. The resulting method is called *CSRoulette*. The procedure of *CSRoulette* is shown in Figure 1.

5 Empirical Comparisons

To compare *CSRoulette* with Costing and other existing cost-sensitive meta-learning algorithms, we choose seventeen datasets (Breast-cancer, Breast-w, Car, Credit-g, Cylinder-bands, Diabetes, Heart-c, Heart-statlog, Hepatitis, Ionosphere, Labor, Molecular, Sick, Sonar, Spect, Spectf, and Tic-tac-toe) from the UCI Machine Learning Repository [1]. Since misclassification costs are not available for the datasets in the UCI Machine Learning Repository, we reasonably assign their values that are inversely proportional to the ratio of the number of class instances in the experiments. For each dataset in experiments, we set the misclassification cost ratio is inversely proportional to the number of positive/negative instances.

We choose C4.5 [9] as the base learning algorithm with Laplace correction. We first conduct experiments to compare *CSRoulette* with MetaCost, CSC, Weighting, and Costing. In order to maximize the size of the sample produced by rejection sampling in Costing, we set the constant $Z = maxC(i)$ in all following experiments. Furthermore, we also conduct experiments to investigate the effect of various sample sizes in *CPRS*.

5.1 Average Cost

The average cost is an ultimate measure for the performance of a cost-sensitive learning algorithm. It is the average misclassification cost of testing examples. In this section, we conduct experiments to compare *CSRoulette* (under default sample sizes) with Costing, MetaCost, CSC, and Weighting measured by the average cost. To be comparable with *CSRoulette* and Costing, we also apply bagging on MetaCost, CSC, and Weighting. The results are presented in terms of the average of the misclassification cost per test example via 10 runs over ten-fold cross-validation shown in Figure 2. The vertical axis represents the average cost, and the horizontal axis represents the number of iterations. The legends of the first dataset are shared by others.

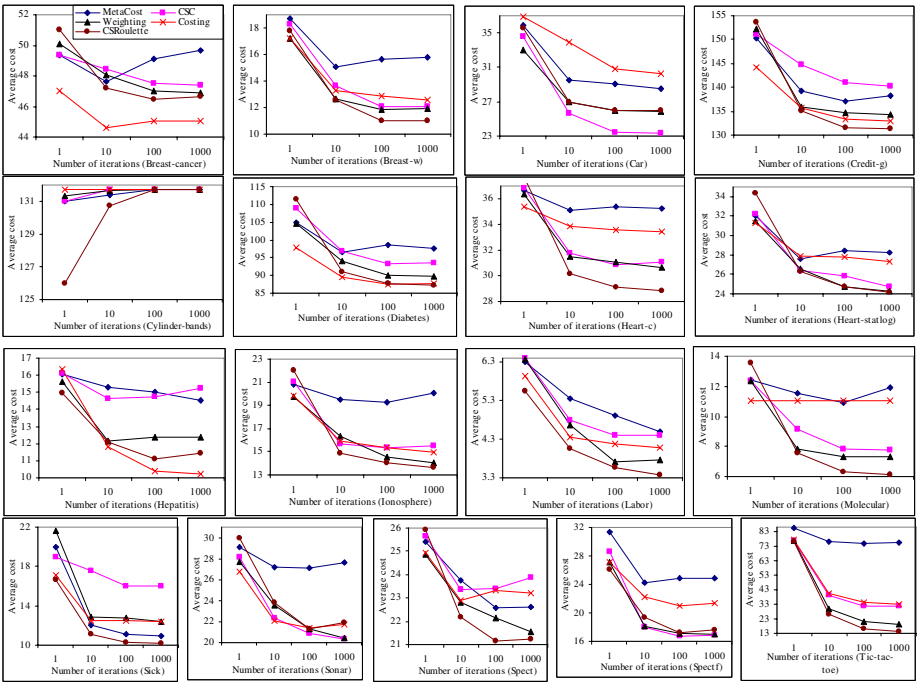

Fig. 2. Average cost of each dataset (the lower, the better)

We can draw the following interesting conclusions from the results shown in Figure 2. First of all, MetaCost is almost always the worst (twelve out of seventeen datasets). Bagging does improve its performance. However, the improvement is not as significant as bagging applied on *CSRoulette*, Costing, CSC, and Weighting. Second, CSC performs better than Costing on nine datasets. In other datasets, it is similar or worse. Third, only in three datasets (Car, Sick and Sonar) is CSC better than Weighting. In all other datasets, it is similar or worse. Fourth, overall, *CSRoulette* and Weighting perform better than MetaCost, Costing, and CSC. However, *CSRoulette* performs better than Weighting in thirteen out of seventeen datasets. In others, it is similar to Weighting in three datasets (Car, Heart-statlog and Spectf). It per-forms worse only in the dataset Sonar.

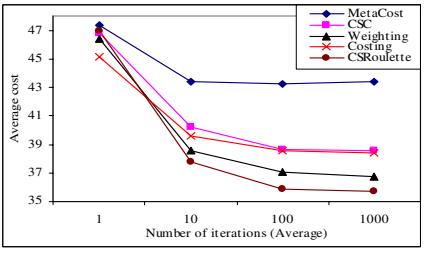

Fig. 3. The average cost over the seventeen datasets of meta-learning algorithms (the lower, the better)

From above individual analysis for the seventeen datasets in our experiments, we can conclude that *CSRoulette* performs better than Weighting, Weighting does better than CSC and Costing, and CSC and Costing do better than MetaCost.

We further summarize the experimental results in average over all the datasets represented in Figure 3. The results in Figure 3 agree with our conclusion made from the analysis for all seventeen datasets in previous paragraphy and show quantitively that *CSRoulette* is the best, followed by Weighing and followed by CSC and Costing. MetaCost is the worst.

5.2 Sample Size vs. Number of Bagging

We further investigate the effect of the size of the sample on the cost-sensitive sampling techniques between *CSRoulette* and Costing. *CPRS* (cost proportional roulette sampling) can generate any sample sizes according to the size indicated by users. Thus, *CSRoulette* can use the various sizes samples to build learning models. In Costing, we use maximum sample size generated by rejection sampling. The maximum sample size for each dataset is listed in the following table (Table 1), where $k = |S'|/|S|$, where $|S'|$ is the maximum sample size, and $|S|$ is the size of the original data. For *CSRoulette*, we generate samples under four different sizes indicated ($0.2\times|S|$, $0.5\times|S|$, $1\times|S|$, $2\times|S|$, and $5\times|S|$). Note that when $k=0.5$, the size of the sample generated by *CPRS* is similar to the maximum size of most samples generated by rejection sampling (see Table 1).

Table 1. The ratio of the maximum sample size of the *i*th dataset shown in Figure 2 over its original size with the rejection sampling. The 12th dataset (Molecular) is absolute balanced.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0.59	0.69	0.60	0.60	0.84	0.70	0.91	0.89	0.41	0.72	0.70	1.00	0.12	0.93	0.41	0.54	0.69

From the above table, we can see that the maximum sample size generated by rejection sampling of each dataset is less than the original training set, particularly in the dataset Sick. This is one of reasons why *CSRoulette* performs much better than Costing in Section 5.1. That is, the sample size affects the quality of the cost-sensitive classifier, as *CPRS* generates samples with the same size of the dataset. Does the original training set size is optimum for the sample generated? Or does each dataset have its own optimal sample size for cost sensitive sampling techniques (such as cost proportional roulette sampling used in *CSRoulette*)? Can we compensate the effect of reduced sample sizes by increasing the number of iterations in bagging? To investigate this question we conduct experiments controlling the number of iterations for different sample sizes to make the total training size equal. The total training size is defined as the sample size multiples the number of iterations. To be specific, if we have i iterations for the sample size $|S|$, the total training size is $i\times|S|$. Thus, for the sample size $0.5\times|S|$, we have to iterate $2i$ times to make its total training size as $i\times|S|$. We compare Costing (with its maximum sample size) and *CSRoulette* (with five different sample sizes). The detail experimental results could not be shown here as the limited space.

Generally, our experimental results show that bagging could not compensate the effect of small sample sizes. In eleven out of seventeen datasets (Breast-w, Car, Cylinder-bands, Hepatitis, Ionosphere, Labor, Sick, Sonar, Spect, Spectf, and Tic-tac-toe), bagging could not compensate the effect of the small sample sizes. For these

datasets, larger sample sizes are preferred. Although in six out of seventeen datasets (such as Breast-cancer, Credit-g, Diabetes, Heart-c, Heart-statlog, and Molecular), *CSRoulette* prefers small sample sizes with more iterations of bagging.

CSRoulette outperforms Costing in most cases. *CSRoulette* can generate samples with different sizes. For most sample sizes, *CSRoulette* have a lower average cost compared to Costing. Even when the sample size of *CSRoulette* is similar (when k=0.5) to the sample size of Costing, *CSRoulette* still performs better than Costing.

We further summarize the experimental results in average in Figure 4. Its vertical axis represents the average cost (the lower, the better), and its horizontal axis represents the total training size. The results are represented in terms of the average of misclassification cost per test example of 10 runs over ten-fold cross validation. There are five curves in each figure, one for Costing, and four for *CSRoulette* (with size $k \times |S|$, where *k=0.2, 0.5, 1.0*, *2.0,* and *5.0* respectively).

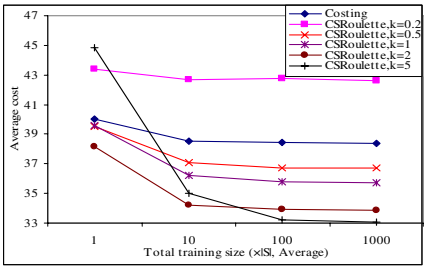

Fig. 4. The average cost over all seventeen datasets of Costing and *CSRoulette* under different cases (the lower, the better)

From Figure 4, we can see that *CSRoulette* and Costing perform better when the total training size increases. Under the same total training size, *CSRoulette* performs better when the sample size increases. However, when the sample size increases from twice to five times of the sizes of original training sets, *CSRoulette* performs similarly. We can also see that *CSRoulette* performs badly without bagging (i.e., bagging iteration is 1) when the sample size is five times (i.e., the curve of *CSRoulette, k=5*) of that of original training set. When we further increase the iterations, the average costs of all curves do not further go down. Thus, we can conclude that bagging is useful, but it could not compensate the effect of small sample sizes.

6 Conclusions and Future Work

In this paper, we propose a new cost-sensitive meta-learning algorithm *CSRoulette*. It is based on the cost proportional roulette sampling technique, called *CPRS*. It overcomes the shortcomings of rejection sampling. Rejection sampling reduces significantly the sample size of an original training set. *CPRS* can generate different size samples according to users' indication. If it generates samples with the same size as the original training set by default, it makes better use of the available information in the training set.

Similar to Costing, *CSRoulette* takes advantage of ensemble learning to further improve the performance of cost proportional roulette sampling. The experimental results show that *CSRoulette* outperforms Costing in most cases. Comparing with other cost-sensitive meta-learning algorithms (MetaCost, CSC, and Weighting), *CSRoulette* also performs better. In general, we can conclude that *CSRoulette* is the best, followed by Weighting, followed by CSC and Costing. MetaCost is the worst

Our experiment results indicate that the best sample size is a characteristic of a dataset. The effect of small sample size could not be compensated by increasing the iterations of bagging, although bagging is useful to improve the performance of all the meta-learning cost-sensitive algorithms.

In our future work, we will further investigate the performance of *CSRoulette* by searching its best sample size for each dataset and compare it with other cost-sensitive learning algorithms under different cost ratios.

References

1. Blake, C.L., Merz, C.J.: UCI Repository of machine learning databases (website). University of California, Department of Information and Computer Science, Irvine, CA (1998)
2. Breiman, L.: Bagging predictors. Machine Learning 24, 123–140 (1996)
3. Domingos, P.: MetaCost: A general method for making classifiers cost-sensitive. In: Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining, pp. 155–164. ACM Press, New York (1999)
4. Drummond, C., Holte, R.: Exploiting the cost (in)sensitivity of decision tree splitting criteria. In: Proceedings of the 17th International Conference on Machine Learning, pp. 239–246 (2000)
5. Elkan, C.: The foundations of cost-sensitive learning. In: Proceedings of the Seventeenth International Joint Conference of Artificial Intelligence, pp. 973–978. Morgan Kaufmann, Seattle, Washington (2001)
6. Goldberg, D.E.: Genetic Algorithm in Search, Optimization and Machine Learning. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts (1989)
7. Ling, C.X., Yang, Q., Wang, J., Zhang, S.: Decision trees with minimal costs. In: Proceedings of the Twenty-First International Conference on Machine Learning, Morgan Kaufmann, Banff, Alberta (2004)
8. Lizotte, D., Madani, O., Greiner, R.: Budgeted learning of naïve-Bayes classifiers. In: Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, Acapulco, Mexico (2003)
9. Quinlan, J.R. (ed.): C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco (1993)
10. Ting, K.M.: 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. Springer, Heidelberg (1998)
11. Turney, P.D.: Cost-sensitive classification: empirical evaluation of a hybrid genetic decision tree induction algorithm. Journal of Artificial Intelligence Research 2, 369–409 (1995)
12. Weiss, G., Provost, F.: Learning when training data are costly: the effect of class distribution on tree induction. Journal of Artificial Intelligence Research 19, 315–354 (2003)
13. Witten, I.H., Frank, E.: Data Mining – Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann Publishers, San Francisco (2005)
14. Zadrozny, B., Langford, J., Abe, N.: Cost-sensitive learning by cost-proportionate instance weighting. In: Proceedings of the 3th International Conference on Data Mining (2003)