Cost-Sensitive Learning by Cost-Proportionate Example Weighting

Bianca Zadrozny, John Langford*, Naoki Abe
Mathematical Sciences Department
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598

Abstract

We propose and evaluate a family of methods for converting classifier learning algorithms and classification theory into cost-sensitive algorithms and theory. The proposed conversion is based on cost-proportionate weighting of the training examples, which can be realized either by feeding the weights to the classification algorithm (as often done in boosting), or by careful subsampling. We give some theoretical performance guarantees on the proposed methods, as well as empirical evidence that they are practical alternatives to existing approaches. In particular, we propose costing, *a method based on cost-proportionate rejection sampling and ensemble aggregation, which achieves excellent predictive performance on two publicly available datasets, while drastically reducing the computation required by other methods.*

1 Introduction

Highly non-uniform misclassification costs are very common in a variety of challenging real-world data mining problems, such as fraud detection, medical diagnosis and various problems in business decision-making. In many cases, one class is rare but the cost of not recognizing some of the examples belonging to this class is high. In these domains, classifier learning methods that do not take misclassification costs into account do not perform well. In extreme cases, ignoring costs may produce a model that is useless because it classifies every example as belonging to the most frequent class even though misclassifications of the least frequent class result in a very large cost.

Recently a body of work has attempted to address this issue, with techniques known as cost-sensitive learning in the machine learning and data mining communities. Current cost-sensitive learning research falls into three categories. The first is concerned with making particular classifier learners cost-sensitive [3, 7]. The second uses Bayes risk theory to assign each example to its lowest risk class [2, 19, 14]. This requires estimating class membership probabilities and, in the case where costs are nondeterministic, also requires estimating expected costs [19]. The third category concerns methods for converting arbitrary classification learning algorithms into cost-sensitive ones [2]. The work described here belongs to the last category.

In particular, the approach here is akin to the pioneering work of Domingos on MetaCost [2], which also is a general method for converting cost-sensitive learning problems to cost-insensitive learning problems. However, the method here is distinguished by the following properties: (1) it is even simpler; (2) it has some theoretical performance guarantees; and (3) it does not involve any probability density estimation in its process: MetaCost estimates conditional probability distributions via bagging with a classifier in its procedure, and as such it also belongs to the second category (Bayes risk minimization) mentioned above.

The family of proposed methods is motivated by a folk theorem that is formalized and proved in section 2.1. This theorem states that altering the original example distribution D to another $\hat{D}$, by multiplying it by a factor proportional to the relative cost of each example, makes any error-minimizing classifier learner accomplish expected cost minimization on the original distribution. Representing samples drawn from $\hat{D}$, however, is more challenging than it may seem. There are two basic methods for doing this: (i) Transparent Box: Supply the costs of the training data as example weights to the classifier learning algorithm. (ii) Black Box: resample according to these same weights.

While the transparent box approach cannot be applied to arbitrary classifier learners, it can be applied to many, including any classifier which only uses the data to calculate expectations. We show empirically that this method gives good results. The black box approach has the advantage that it can be applied to any classifier learner. It turns out, however, that straightforward sampling-with-replacement can result in severe overfitting related to duplicate examples.

We propose, instead, to employ *cost-proportionate rejection sampling* to realize the latter approach, which allows us to independently draw examples according to $\hat{D}$. This method comes with a theoretical guarantee: In the worst case it produces a classifier that achieves at least as good

*This author's present address: Toyota Technological Institute at Chicago, 427 East 60th Street, Second Floor - Press Building, Chicago, IL 60637.

approximate cost minimization as applying the base classifier learning algorithm on the entire sample. This is a remarkable property for a subsampling scheme: in general, we expect any technique using only a subset of the examples to compromise predictive performance.

The runtime savings made possible by this sampling technique enable us to run the classification algorithm on multiple draws of subsamples and average over the resulting classifiers. This last method is what we call *costing* (cost-proportionate rejection sampling with aggregation). Costing allows us to use an arbitrary cost-insensitive learning algorithm as a black box in order to accomplish cost-sensitive learning, achieves excellent predictive performance and can achieve drastic savings of computational resources.

2 Motivating Theory and Methods

2.1 A Folk Theorem

We assume that examples are drawn independently from a distribution D with domain $X \times Y \times C$ where X is the input space to a classifier, Y is a (binary) output space and $C \subset [0, \infty)$ is the importance (extra cost) associated with mislabeling that example. The goal is to learn a classifier $h : X \to Y$ which minimizes the expected cost,

$$E_{x,y,c\sim D}[cI(h(x) \neq y)]$$

given training data of the form: (x,y,c), where $I(\cdot)$ is the indicator function that has value 1 in case its argument is true and 0 otherwise. This model does not explicitly allow using cost information at prediction time although X might include a cost feature if that is available.

This formulation of cost-sensitive learning in terms of one number per example is more general than "cost matrix" formulations which are more typical in cost-sensitive learning [6, 2], when the output space is binary.[1] In the cost matrix formulation, costs are associated with false negative, false positive, true negative, and true positive predictions. Given the cost matrix and an example, only two entries (false positive, true negative) or (false negative, true positive) are relevant for that example. These two numbers can be further reduced to one: (false positive - true negative) or (false negative - true positive), because it is the difference in cost between classifying an example correctly or incorrectly which controls the importance of correct classification. This difference is the importance c we use here. This setting is more general in the sense that the importance may vary on a example-by-example basis.

A basic folk theorem [2] states that if we have examples drawn from the distribution:

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

[1]How to formulate the problem in this way when the output space is not binary is nontrivial and is beyond the scope of this paper.

[2]We say "folk theorem" here because the result appears to be known by some and it is straightforward to derive it from results in decision theory, although we have not found it published.

then optimal error rate classifiers for $\hat{D}$ are optimal cost minimizers for data drawn from D.

Theorem 2.1. *(Translation Theorem) For all distributions, D, there exists a constant $N = E_{x,y,c\sim D}[c]$ such that for all classifiers, h:*

$$E_{x,y,c\sim \hat{D}}[I(h(x) \neq y)] = \frac{1}{N} E_{x,y,c\sim D}[cI(h(x) \neq y)]$$

Proof.

$$\begin{aligned} E_{x,y,c\sim D}[cI(h(x) \neq y)] &= \sum_{x,y,c} D(x,y,c)cI(h(x) \neq y) \\ &= N \sum_{x,y,c} \hat{D}(x,y,c)I(h(x) \neq y) \\ &= NE_{x,y,c\sim \hat{D}}[I(h(x) \neq y)] \\ \text{where} \quad & \hat{D}(x,y,c) = \frac{c}{N} D(x,y,c). \end{aligned}$$

□

Despite its simplicity, this theorem is useful to us because the right-hand side expresses the expectation we want to control (via the choice of h) and the left-hand side is the probability that h errs under another distribution. Choosing h to minimize the rate of errors under $\hat{D}$ is equivalent to choosing h to minimize the expected cost under D. Similarly, ε-approximate error minimization under $\hat{D}$ is equivalent to $N\varepsilon$-approximate cost minimization under D.

The prescription for coping with cost-sensitive problems is straightforward: re-weight the distribution in your training set according to the importances so that the training set is effectively drawn from $\hat{D}$. Doing this in a correct and general manner is more challenging than it may seem and is the topic of the rest of the paper.

2.2 Transparent Box: Using Weights Directly

2.2.1 General conversion

Here we examine how importance weights can be used within different learning algorithms to accomplish cost-sensitive classification. We call this the transparent box approach because it requires knowledge of the particular learning algorithm (as opposed to the black box approach that we develop later).

The mechanisms for realizing the transparent box approach have been described elsewhere for a number of weak learners used in boosting, but we will describe them here for completeness. The classifier learning algorithm must use the weights so that it effectively learns from data drawn according to $\hat{D}$. This requirement is easy to apply for all learning algorithms which fit the statistical query model [13].

As shown in figure 1, many learning algorithms can be divided into two components: a portion which calculates the (approximate) expected value of some function (or query) f and a portion which forms these queries and uses their output to construct a classifier. For example, neural networks, decision trees, and Naive Bayes classifiers can be

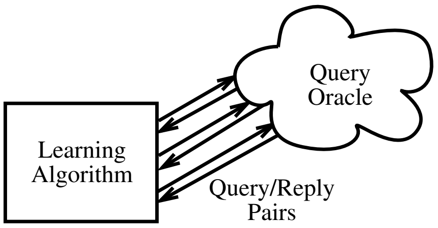

Figure 1. The statistical query model.

constructed in this manner. Support vector machines are not easily constructible in this way, because the individual classifier is explicitly dependent upon individual examples rather than on statistics derived from the entire sample.

With finite data we cannot precisely calculate the expectation $E_{x,y \sim D}[f(x,y)]$. With high probability, however, we can approximate the expectation given a set of examples drawn independently from the underlying distribution D.

Whenever we have a learning algorithm that can be decomposed as in figure 1, there is a simple recipe for using the weights directly. Instead of simulating the expectation with $\frac{1}{|S|}\sum_{(x,y)\in S} f(x,y)$, we use $\frac{1}{\sum_{(x,y,c)\in S} c}\sum_{(x,y,c)\in S} cf(x,y)$. This method is equivalent to importance sampling for $\hat{D}$ using the distribution D, and so the modified expectation is an unbiased Monte Carlo estimate of the expectation w.r.t. $\hat{D}$.

Even when a learning algorithm does not fit this model, it may be possible to incorporate importance weights directly. We now discuss how to incorporate importance weights into some specific learning algorithms.

2.2.2 Naive Bayes and boosting

Naive Bayes learns by calculating empirical probabilities for each output y using Bayes' rule and assuming that each feature is independent given the output:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{\prod_i P(x_i|y)P(y)}{\prod_i P(x_i)}$$

Each probability estimate in the above expression can be thought of as a function of empirical expectations according to D, and thus it can be formulated in the statistical query model. For example, $P(x_i|y)$ is just the expectation of $I(\mathbf{x_i} = x_i) \wedge I(\mathbf{y} = y)$ divided by the expectation of $I(\mathbf{y} = y)$. More specifically, to compute the empirical estimate of $P(x_i|y)$ with respect to D, we need to count the number of training examples that have y as output, and those having x_i as the i-th input dimension among those. When we compute these empirical estimates with respect to $\hat{D}$, we simply have to sum the weight of each example, instead of counting the examples. (This property is used in the implementation of boosted Naive Bayes [5].)

To incorporate importance weights into AdaBoost [8], we give the importance weights to the weak learner in the first iteration, thus effectively drawing examples from $\hat{D}$. In the subsequent iterations, we use the standard AdaBoost rule to update the weights. Therefore, the weights are adjusted according to the accuracy on $\hat{D}$, which corresponds to the expected cost on D.

2.2.3 C4.5

C4.5 [16] is a widely used decision tree learner. There is a standard way of incorporating example weights to it, which in the original algorithm was intended to handle missing attributes (examples with missing attributes were divided into fractional examples, each with a smaller weight, during the growth of the tree). This same facility was later used by Quinlan in the implementation of boosted C4.5 [15].

2.2.4 Support Vector Machine

The SVM algorithm [11] learns the parameters a and b describing a linear decision rule $h(x) = \text{sign}(a \cdot x + b)$, so that the smallest distance between each training example and the decision boundary (the margin) is maximized. It works by solving the following optimization problem:

$$\begin{array}{l} \text{minimize: } V(a,b,\xi) = \frac{1}{2} a \cdot a + C\sum_{i=1}^{n} \xi_i \\ \text{subject to: } \forall i: \;\; y_i[a \cdot x_i + b] \geq 1 - \xi_i,\; \xi_i > 0 \end{array}$$

The constraints require that all examples in the training set are classified correctly up to some slack ξ_i. If a training example lies on the wrong side of the decision boundary, the corresponding ξ_i is greater than 1. Therefore, $\sum_{i=1}^{n} \xi_i$ is an upper bound on the number of training errors. The factor C is a parameter that allows one to trade off training error and model complexity. The algorithm can be generalized to non-linear decision rules by replacing inner products with a kernel function in the formulas above.

The SVM algorithm does not fit the statistical query model. Despite this, it is possible to incorporate importance weights in a natural way. First, we note that $\sum_{i=1}^{n} c_i \xi_i$, where c_i is the importance of example i, is an upper bound on the total cost. Therefore, we can modify $V(a,b,\xi)$ to

$$V(a,b,\xi) = \frac{1}{2} a \cdot a + C\sum_{i=1}^{n} c_i \xi_i.$$

Now C controls model complexity versus total cost.

The SVMLight package [10] allows users to input weights c_i and works with the modified $V(a,b,\xi)$ as above, although this feature has not yet been documented.

2.3 Black Box: Sampling methods

Suppose we do not have transparent box access to the learner. In this case, sampling is the obvious method to convert from one distribution of examples to another to obtain a cost-sensitive learner using the translation theorem (Theorem 2.1). As it turns out, straightforward sampling does not work well in this case, motivating us to propose an alternative method based on rejection sampling.

2.3.1 Sampling-with-replacement

Sampling-with-replacement is a sampling scheme where each example (x,y,c) is drawn according to the distribution $p(x,y,c) = \frac{c}{\sum_{(x,y,c)\in S} c}$. Many examples are drawn to create a new dataset S'. This method, at first pass, appears useful because every example is effectively drawn from the distribution $\hat{D}$. In fact, very poor performance can result when using this technique, which is essentially due to overfitting because of the fact that the examples in S' are not drawn *independently* from $\hat{D}$, as we will elaborate in the section on experimental results (Section 3).

Sampling-without-replacement is also not a solution to this problem. In sampling-without-replacement, an example (x,y,c) is drawn from the distribution $p(x,y,c) = \frac{c}{\sum_{(x,y,c)\in S} c}$ and the next example is drawn from the set $S - \{x,y,c\}$. This process is repeated, drawing from a smaller and smaller set according to the weights of the examples remaining in the set.

To see how this method fails, note that sampling-without-replacement m times from a set of size m results in the original set, which (by assumption) is drawn from the distribution D, and not $\hat{D}$ as desired.

2.3.2 Cost-proportionate rejection sampling

There is another sampling scheme called rejection sampling [18] which allows us to draw examples independently from the distribution $\hat{D}$, given examples drawn independently from D. In rejection sampling, examples from $\hat{D}$ are obtained by first drawing examples from D, and then keeping (or accepting) the sample with probability proportional to $\hat{D}/D$. Here, we have $\hat{D}/D \propto c$, so we accept an example with probability c/Z, where Z is some constant chosen so that $\max_{(x,y,c)\in S} c \leq Z$,[3] leading to the name *cost-proportionate rejection sampling*. Rejection sampling results in a set S' which is generally smaller than S. Furthermore, because inclusion of an example in S' is independent of other examples, and the examples in S are drawn independently, we know that the examples in S' are distributed independently according to $\hat{D}$.

Using cost-proportionate rejection sampling to create a set S' and then using a learning algorithm $A(S')$ is guaranteed to produce an approximately cost-minimizing classifier, as long as the learning algorithm A achieves approximate minimization of classification error.

Theorem 2.2. *(Correctness) For all cost-sensitive sample sets S, if cost-proportionate rejection sampling produces a sample set S' and $A(S')$ achieves ε classification error:*

$$E_{x,y,c\sim\hat{D}}[I(h(x) \neq y)] \leq \varepsilon$$

then $h = A(S')$ approximately minimizes cost:

$$E_{x,y,c\sim D}[c\,I(h(x) \neq y)] \leq \varepsilon N$$

where $N = E_{x,y,c\sim D}[c]$.

Proof. Rejection sampling produces a sample set S' drawn independently from $\hat{D}$. By assumption $A(S')$ outputs a classifier h such that

$$E_{x,y,c\sim\hat{D}}[I(h(x) \neq y)] \leq \varepsilon$$

By the translation theorem (Theorem 2.1), we know that

$$E_{x,y,c\sim\hat{D}}[I(h(x) \neq y)] = \frac{1}{N} E_{x,y,c\sim D}[c\,I(h(x) \neq y)]$$

Thus,
$$E_{x,y,c\sim D}[c\,I(h(x) \neq y)] \leq \varepsilon N.$$

□

2.3.3 Sample complexity of cost-proportionate rejection sampling

The accuracy of a learned classifier generally improves monotonically with the number of examples in the training set. Since cost-proportionate rejection sampling produces a smaller training set (by a factor of about N/Z), one would expect worse performance than using the entire training set.

This turns out to not be the case, in the *agnostic PAC-learning model* [17, 12], which formalizes the notion of probably approximately optimal learning from arbitrary distributions D.

Definition 2.1. *A learning algorithm A is said to be an* agnostic PAC-learner *for hypothesis class H, with sample complexity $m(1/\varepsilon, 1/\delta)$ if for all $\varepsilon > 0$ and $\delta > 0$, $m = m(1/\varepsilon, 1/\delta)$ is the least sample size such that for all distributions D (over $X \times Y$), the classification error rate of its output h is at most ε more than the best achievable by any member of H with probability at least $1 - \delta$, whenever the sample size exceeds m.*

By analogy, we can formalize the notion of *cost-sensitive* agnostic PAC-learning.

Definition 2.2. *A learning algorithm A is said to be a* cost-sensitive agnostic PAC-learner *for hypothesis class H, with* cost-sensitive *sample complexity $m(1/\varepsilon, 1/\delta)$, if for all $\varepsilon > 0$ and $\delta > 0$, $m = m(1/\varepsilon, 1/\delta)$ is the least sample size such that for all distributions D (over $X \times Y \times C$), the expected cost of its output h is at most ε more than the best achievable by any member of H with probability at least $1 - \delta$, whenever the sample size exceeds m.*

We will now use this formalization to compare the cost-sensitive PAC-learning sample complexity of two methods: applying a given base classifier learning algorithm to a sample obtained through cost-proportionate rejection sampling, and applying the same algorithm on the original training set. We show that the cost-sensitive sample complexity of the latter method is lower-bounded by that of the former.

[3] In practice, we choose $Z = \max_{(x,y,w)\in S} c$ so as to maximize the size of the set S'. A data-dependent choice of Z is *not* formally allowed for rejection sampling. However, the introduced bias appears small when $|S| >> 1$. A precise measurement of "small" is an interesting theoretical problem.

Theorem 2.3. *(Sample Complexity Comparison) Fix an arbitrary base classifier learning algorithm A, and suppose that $m_{orig}(1/\varepsilon, 1/\delta)$ and $m_{rej}(1/\varepsilon, 1/\delta)$, respectively, are cost-sensitive sample complexity of applying A on the original training set, and that of applying A with cost-proportionate rejection sampling. Then, we have*

$$m_{orig}(1/\varepsilon, 1/\delta) = \Omega(m_{rej}(1/\varepsilon, 1/\delta)).$$

Proof. Let $m(1/\varepsilon, 1/\delta)$ be the (cost-insensitive) sample complexity of the base classifier learning algorithm A. (If no such function exists, then neither $m_{\text{orig}}(1/\varepsilon, 1/\delta)$ nor $m_{\text{rej}}(1/\varepsilon, 1/\delta)$ exists, and the theorem holds vacuously.) Since Z is an upper bound on the cost of misclassifying an example, we have that the cost-sensitive sample complexity of using the original training set satisfies

$$m_{\text{orig}}(1/\varepsilon, 1/\delta) = \Theta(m(Z/\varepsilon, 1/\delta))$$

This is because given a distribution that forces ε more classification error than optimal, another distribution can be constructed, that forces εZ more cost than optimal, by assigning cost Z to all examples on which A errs.

Now from Theorem 2.2 and noting that the central limit theorem implies that cost-proportionate rejection sampling reduces the sample size by a factor of $\Theta(N/Z)$, the cost-sensitive sample complexity for rejection sampling is:

$$m_{\text{rej}}(1/\varepsilon, 1/\delta) = \Theta\left(\frac{Z}{N} m(N/\varepsilon, 1/\delta)\right). \quad (1)$$

A fundamental theorem from PAC-learning theory states that $m(1/\varepsilon, 1/\delta) = \Omega((1/\varepsilon)\ln(1/\delta))$ [4]. When $m(1/\varepsilon, 1/\delta) = \Theta((1/\varepsilon)\ln(1/\delta))$, Equation (1) implies:

$$m_{\text{rej}}(1/\varepsilon, 1/\delta) = \Theta\left(\frac{Z}{N}\frac{N}{\varepsilon}\ln(1/\delta)\right) = \Theta\left(m_{\text{orig}}(1/\varepsilon, 1/\delta)\right)$$

Finally, note that when $m(1/\varepsilon, 1/\delta)$ grows faster than linear in $1/\varepsilon$, we have $m_{\text{rej}}(1/\varepsilon, 1/\delta) = o(m_{\text{orig}}(1/\varepsilon, 1/\delta))$, which finishes the proof. □

Note that the linear dependence of sample size on $1/\varepsilon$ is only achievable by an ideal learning algorithm, and in practice super-linear dependence is expected, especially in the presence of noise. Thus, the above theorem implies that *cost-proportionate rejection sampling* minimizes cost better than no sampling for worst case distributions.

This is a remarkable property about any sampling scheme, since one generally expects that predictive performance is compromised by using a smaller sample. Cost-proportionate rejection sampling seems to *distill* the original sample and obtains a sample of smaller size, which is at least as informative as the original.

2.3.4 Cost-proportionate rejection sampling with aggregation (costing)

From the same original training sample, different runs of cost-proportionate rejection sampling will produce different training samples. Furthermore, the fact that rejection sampling produces very small samples means that the time required for learning a classifier is generally much smaller.

We can take advantage of these properties to devise an ensemble learning algorithm based on repeatedly performing rejection sampling from S to produce multiple sample sets $S'_1, \ldots, S'_m$, and then learning a classifier for each set. The output classifier is the average over all learned classifiers. We call this technique *costing*:

Costing(Learner A, Sample Set S, count t)

1. **For** $i = 1$ **to** t **do**
   (a) S' = **rejection sample from** S **with acceptance probability** c/Z.
   (b) **Let** $h_i \equiv A(S')$
2. **Output** $h(x) = \textbf{sign}\left(\sum_{i=1}^{t} h_i(x)\right)$

The goal in averaging is to improve performance. There is both empirical and theoretical evidence suggesting that averaging can be useful. On the empirical side, many people have observed good performance from bagging despite throwing away a $1/e$ fraction of the samples. On the theoretical side, there has been considerable work which proves that the ability to overfit of an average of classifiers might be smaller than naively expected when a large margin exists. The preponderance of learning algorithms producing averaging classifiers provides significant evidence that averaging is useful.

Note that despite the extra computational cost of averaging, the overall computational time of costing is generally much smaller than that of a learning algorithm using sample set S (with or without weights). This is the case because most learning algorithms have running times that are superlinear in the number of examples.

3 Empirical evaluation

We show empirical results using two real-world datasets. We selected datasets that are publicly available and for which cost information is available on a per example basis. Both datasets are from the direct marketing domain. Although there are many other data mining domains that are cost-sensitive, such as credit card fraud detection and medical diagnosis, publicly available data are lacking.

3.1 The datasets used

3.1.1 KDD-98 dataset

This is the well-known and challenging dataset from the KDD-98 competition, now available at the UCI KDD repository [9]. The dataset contains information about persons who have made donations in the past to a particular charity. The decision-making task is to choose which donors to mail a request for a new donation. The measure of success is the total profit obtained in the mailing campaign.

The dataset is divided in a fixed way into a training set and a test set. Each set consists of approximately 96000 records for which it is known whether or not the person made a donation and how much the person donated, if a donation was made. The overall percentage of donors is about 5%. Mailing a solicitation to an individual costs the charity \$0.68. The donation amount for persons who respond varies from \$1 to \$200. The profit obtained by soliciting every individual in the test set is \$10560, while the profit attained by the winner of the KDD-98 competition was \$14712.

The importance of each example is the absolute difference in profit between mailing and not mailing an individual. Mailing results in the donation amount minus the cost of mailing. Not mailing results in zero profit. Thus, for positive examples (respondents), the importance varies from \$0.32 to \$199.32. For negative examples (non-respondents), it is fixed at \$0.68.

3.1.2 DMEF-2 dataset

This dataset can be obtained from the DMEF dataset library [1] for a nominal fee. It contains customer buying history for 96551 customers of a nationally known catalog. The decision-making task is to choose which customers should receive a new catalog so as to maximize the total profit on the catalog mailing campaign. Information on the cost of mailing a catalog is not available, so we fixed it at \$2.

The overall percentage of respondents is about 2.5%. The purchase amount for customers who respond varies from \$3 to \$6247. As is the case for the KDD-98 dataset, the importance of each example is the absolute difference in profit between mailing and not mailing a customer. Therefore, for positive examples (respondents), the importance varies from \$1 to \$6245. For negative examples (non-respondents), it is fixed at \$2.

We divided the dataset in half to create a training set and a test set. As a baseline for comparison, the profit obtained by mailing a catalog to every individual on the training set is \$26474 and on the test set is \$27584.

3.2 Experimental results

3.2.1 Transparent box results

Table 1 (top) shows the results for Naive Bayes, boosted Naive Bayes (100 iterations) C4.5 and SVMLight on the KDD-98 and DMEF-2 datasets, with and without the importance weights. Without the importance weights, the classifiers label very few of the examples positive, resulting in small (and even negative) profits. With the costs given as weights to the learners, the results improve significantly for all learners, except C4.5. Cost-sensitive boosted Naive Bayes gives results comparable to the best so far with this dataset [19] using more complicated methods.

We optimized the parameters of the SVM by cross-validation on the training set. Without weights, no setting of the parameters prevented the algorithm of labeling all examples as negatives. With weights, the best parameters were a polynomial kernel with degree 3 and $C = 5 \times 10^{-5}$ for KDD-98 and a linear kernel with $C = 0.0005$ for DMEF-2. However, even with this parameter setting, the results are not so impressive. This may be a hard problem for margin-based classifiers because the data is very noisy. Note also that running SVMLight on this dataset takes about 3 orders of magnitude longer than AdaBoost with 100 iterations.

KDD-98:

Method	Without Weights	With Weights
Naive Bayes	0.24	12367
Boosted NB	-1.36	14489
C4.5	0	118
SVMLight	0	13683

DMEF-2:

Method	Without Weights	With Weights
Naive Bayes	16462	32608
Boosted NB	121	36381
C4.5	0	478
SVMLight	0	36443

Table 1. Test set profits with transparent box.

The failure of C4.5 to achieve good profits with importance weights is probably related to the fact that the facility for incorporating weights provided in the algorithm is heuristic. So far, it has been used only in situations where the weights are fairly uniform (such as is the case for fractional instances due to missing data). These results indicate that it might not be suitable for situations with highly non-uniform costs. The fact that it is non-trivial to incorporate costs directly into existing learning algorithms is the motivation for the black box approaches that we present here.

3.2.2 Black box results

Table 2 shows the results of applying the same learning algorithms to the KDD-98 and DMEF-2 data using training sets of different sizes obtained by sampling-with-replacement. For each size, we repeat the experiments 10 times with different sampled sets to get mean and standard error (in parentheses). The training set profits are on the original training set from which we draw the sampled sets.

The results confirm that application of sampling-with-replacement to implement the black box approach can result in very poor performance due to overfitting. When there are large differences in the magnitude of importance weights, it is typical for an example to be picked twice (or more). In table 2, we see that as we increase the sampled training set size and, as a consequence, the number of duplicate examples in the training set, the training profit becomes larger while the test profit becomes smaller for C4.5.

Examples which appear multiple times in the training set of a learning algorithm can defeat complexity control mechanisms built into learning algorithms For example, suppose that we have a decision tree algorithm which divides the training data into a “growing set” (used to construct a tree)

KDD-98:

	1000		10000		100000	
	Training	Test	Training	Test	Training	Test
NB	11251 (330)	10850 (325)	12811 (155)	11993 (185)	12531 (242)	12026 (256)
BNB	11658 (311)	11276 (383)	13838 (65)	12886 (212)	14107 (152)	13135 (159)
C4.5	11124 (255)	9548 (331)	22083 (271)	7599 (310)	40704 (152)	2259 (107)
SVM	10320 (372)	10131 (281)	11228 (182)	11015 (161)	13565 (129)	12808 (220)

DMEF-2:

	1000		10000		100000	
	Training	Test	Training	Test	Training	Test
NB	33298 (495)	34264 (419)	32742 (793)	33956 (798)	33511 (475)	34506 (405)
BNB	33902 (558)	30304 (660)	34802 (806)	31342 (772)	34505 (822)	31889 (733)
C4.5	37905 (1467)	24011 (1931)	67960 (763)	9188 (458)	72574 (1205)	3149 (519)
SVM	28837 (1029)	30177 (1196)	31263 (1121)	32585 (891)	34309 (719)	33674 (600)

Table 2. Profits using sampling-with-replacement.

and a "pruning set" (used to prune the tree for complexity control purposes). If the pruning set contains examples which appear in the growing set, the complexity control mechanism is defeated.

Although not as markedly as for C4.5, we see the same phenomenon for the other learning algorithms. In general, as the size of the resampled size grows, the larger is the difference between training set profit and test set profit. And, even with 100000 examples, we do not obtain the same test set results as giving the weights directly to Boosted Naive Bayes and SVM.

The fundamental difficulty here is that the samples in S' are not drawn *independently* from $\hat{D}$. In particular, if $\hat{D}$ is a density, the probability of observing the same example twice given independent draws is 0, while the probability using sampling-with-replacement is greater than 0. Thus sampling-with-replacement fails because the sampled set S' is not constructed independently.

Figure 2 shows the results of costing on the KDD-98 and DMEF-2 datasets, with the base learners and $Z = 200$ or $Z = 6247$, respectively. We repeated the experiment 10 times for each t and calculated the mean and standard error of the profit. The results for $t = 1$, $t = 100$ and $t = 200$ are also given in table 3.

In the KDD-98 case, each resampled set has only about 600 examples, because the importance of the examples varies from 0.68 to 199.32 and there are few "important" examples. About 55% of the examples in each set are positive, even though on the original dataset the percentage of positives is only 5%. With $t = 200$, the C4.5 version yields profits around \$15000, which is exceptional performance for this dataset.

In the DMEF-2 case, each set has only about 35 examples, because the importances vary even more widely (from 2 to 6246) and there are even fewer examples with a large importance than in the KDD-98 case. The percentage of positive examples in each set is about 50%, even though on the original dataset it was only 2.5%.

For learning the SVMs, we used the same kernels as we did in section 2.2 and the default setting for C. In that

KDD-98:

	1	100	200
NB	11667 (192)	13111 (102)	13163 (68)
BNB	11377 (263)	14829 (92)	14714 (62)
C4.5	9628 (511)	14935 (102)	15016 (61)
SVM	10041 (393)	13075 (41)	13152 (56)

DMEF-2:

	1	100	200
NB	26287 (3444)	37627 (335)	37629 (139)
BNB	24402 (2839)	37376 (393)	37891 (364)
C4.5	27089 (3425)	36992 (374)	37500 (307)
SVM	21712 (3487)	33584 (1215)	35290 (849)

Table 3. Test set profits using costing.

section, we saw that by feeding the weights directly to the SVM, we obtain a profit of \$13683 on the KDD-98 dataset and of \$36443 on the DMEF-2 dataset. Here, we obtain profits around \$13100 and \$35000, respectively. However, this did not require parameter optimization and, even with $t = 200$, was much faster to train. The reason for the speed-up is that the time complexity of SVM learning is generally superlinear in the number of training examples.

4 Discussion

Costing is a technique which produces a cost-sensitive classification from a cost-insensitive classifier using only black box access. This simple method is fast, results in excellent performance and often achieves drastic savings in computational resources, particularly with respect to space requirements. This last property is especially desirable in applications of cost-sensitive learning to domains that involve massive amount of data, such as fraud detection, targeted marketing, and intrusion detection.

Another desirable property of any reduction is that it applies to the theory as well as to concrete algorithms. Thus, the reduction presented in the present paper allows us to automatically apply any future results in cost-insensitive classification to cost-sensitive classification. For example, a

KDD-98:

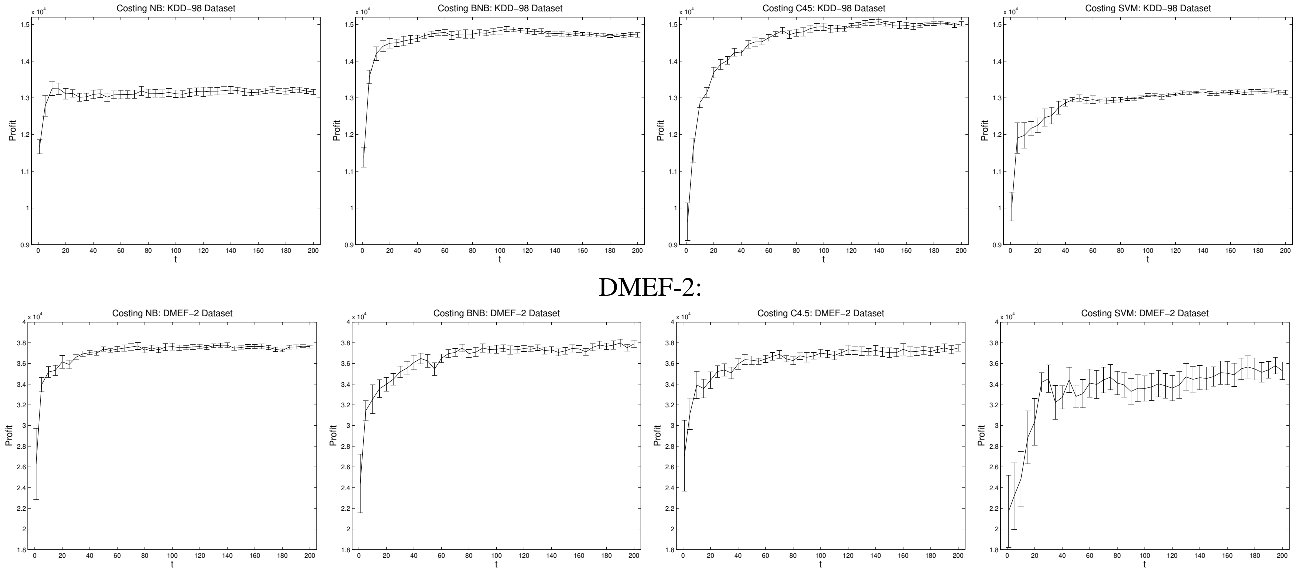

Figure 2. Costing: test set profit vs. number of sampled sets.

bound on the future error rate of $A(S')$ implies a bound on the expected cost with respect to the distribution D. This additional property of a reduction is especially important because cost-sensitive learning theory is still young and relatively unexplored.

One direction for future work is multiclass cost-sensitive learning. If there are K classes, the minimal representation of costs is $K-1$ weights. A reduction to cost-insensitive classification using these weights is an open problem.

References

[1] Anifantis, S. The DMEF Data Set Library. The Direct Marketing Association, New York, NY, 2002. [http://wwww.the-dma.org/dmef/dmefdset.shtml].

[2] Domingos, P. MetaCost: A general method for making classifiers cost sensitive. *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining*, 155-164, 1999.

[3] Drummond, C. & Holte, R. Exploiting the cost (in)sensitivity of decision tree splitting criteria. *Proceedings of the 17th International Conference on Machine Learning*, 239-246, 2000.

[4] Ehrenfeucht, A., Haussler, D., Kearns, M. & Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, *82:3*, 247-261, 1989.

[5] Elkan, C. *Boosting and naive bayesian learning* (Technical Report). University of California, San Diego, 1997.

[6] Elkan, C. The foundations of cost-sensitive learning. *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, 973-978, 2001.

[7] Fan, W., Stolfo, S., Zhang, J. & Chan, P. AdaCost: Misclassification cost-sensitive boosting. *Proceedings of the 16th International Conference on Machine Learning*, 97-105, 1999.

[8] Freund, Y. & Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55:1*, 119-139, 1997.

[9] Hettich, S. & Bay, S. D. The UCI KDD Archive. University of California, Irvine. [http://kdd.ics.uci.edu/].

[10] Joachims, T. Making large-scale SVM learning practical. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1999.

[11] Joachims, T. Estimating the generalization performance of a SVM efficiently. *Proceedings of the 17th International Conference on Machine Learning*, 431-438, 2000.

[12] Kearns, M., Schapire, R., & Sellie, L. Toward Efficient Agnostic Learning. *Machine Learning*, *17*, 115-141, 1998.

[13] Kearns, M. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, *45:6*, 983-1006, 1998.

[14] Margineantu, D. Class probability estimation and cost-sensitive classification decisions. *Proceedings of the 13th European Conference on Machine Learning*, 270-281, 2002.

[15] Quinlan, J. R. Boosting, Bagging, and C4.5. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 725-730, 1996.

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

[17] Valiant, L. A theory of the learnable. *Communications of the ACM*, *27:11*, 1134-1142, 1984.

[18] von Neumann, J. Various techniques used in connection with random digits, *National Bureau of Standards*, *Applied Mathematics Series*, *12*, 36-38, 1951.

[19] Zadrozny, B. and Elkan, C. Learning and making decisions when costs and probabilities are both unknown. *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining*, 203-213, 2001.