Technical Report TR-UNL-CSE-2006-0017, Department of Computer Science & Engineering, University of Nebraska

On Reoptimizing Multi-Class Classifiers[*]

Kun Deng[†] Chris Bourke[†] Stephen D. Scott[†] Robert E. Schapire[‡]
N. V. Vinodchandran[†]

December 5, 2006

Abstract

Significant changes in the instance distribution or associated cost function of a learning problem require one to reoptimize a previously learned classifier to work under new conditions. We study the problem of reoptimizing a multi-class classifier based on its ROC hypersurface and a matrix describing the costs of each type of prediction error. For a binary classifier, it is straightforward to find an optimal operating point based on its ROC curve and the relative cost of true positive to false positive error. However, the corresponding multi-class problem (finding an optimal operating point based on a ROC hypersurface and cost matrix) is more challenging and until now, it was unknown whether an efficient algorithm existed that found an optimal solution. We answer this question by first proving that the decision version of this problem is NP-complete. As a complementary positive result, we give an algorithm that finds an optimal solution in polynomial time if the number of classes n is a constant. We also present several heuristics for this problem, including linear, nonlinear, and quadratic programming formulations, genetic algorithms, and a customized algorithm. Empirical results suggest that under uniform costs several methods exhibit significant improvements while genetic algorithms and margin maximization quadratic programs fare the best under nonuniform cost models.

Keywords: Receiver Operator Characteristics, classifier reoptimization, multi-class classification.

1 Introduction

We study the problem of re-weighting classifiers to optimize them for new cost models. For example, given a classifier optimized to minimize classification error on its training set, one may attempt to tune it to improve performance in light of a new cost model. Equivalently, a change in the class distribution (the probability of seeing examples from each particular class) can be handled by modeling such a change as a change in cost model. More formally, we are concerned with finding a nonnegative weight vector $(w_1, \ldots, w_n)$ to minimize

$$\sum_{i=1}^{m} c\left(y_i, \underset{1\le j\le n}{\operatorname{argmax}}\{w_j\, f_j(x_i)\}\right) \quad , \tag{1}$$

given labeled examples $\{(x_1, y_1), \ldots, (x_m, y_m)\} \subset \mathcal{X} \times \{1, \ldots, n\}$ for instance space $\mathcal{X}$, a family of confidence functions $f_j : \mathcal{X} \to \mathbb{R}^+$ for $1 \le j \le n$, and a cost function $c : \{1, \ldots, n\}^2 \to \mathbb{R}^+$.

This models the problem of reoptimizing a multi-class classifier in machine learning. A machine learning algorithm takes a set $S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \subset \mathcal{X} \times \{1, \ldots, n\}$ of labeled training examples and selects a function $F : \mathcal{X} \to \{1, \ldots, n\}$ that minimizes misclassification cost on S, which is $\sum_{i=1}^{m} c(y_i, F(x_i))$, where cost $c(y_i, F(x_i))$ is a nonnegative function measuring the cost of predicting class $F(x_i)$ on example x_i whose

[*]Preliminary results appeared in Deng et al. (2006).

[†]Dept. of Computer Science, University of Nebraska, Lincoln, NE 68588-0115, USA, {kdeng, cbourke, sscott, vinod}@cse.unl.edu

[‡]Dept. of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08540, USA, schapire@cs.princeton.edu

true label is y_i. For convenience, we will assume that the classifier F is represented by a set of nonnegative functions $f_j : \mathcal{X} \rightarrow \mathbb{R}^+$ for $j \in \{1, \ldots, n\}$, where $f_j(x)$ is the classifier's confidence that x belongs in class j, and $F(x) = \mathrm{argmax}_{1 \le j \le n}\{f_j(x)\}$. Each confidence function f_j can be seen as a "base learner" (as in a one-versus-rest strategy).

An obvious solution would be to simply rerun the base learning algorithm to reoptimize each confidence function f_j for the new cost function[1]. However, this process may be very expensive or even impossible. Thus, the task before us is to reoptimize F *without* discarding the family of base learners. As an example, consider the machine learning application of predicting where a company should drill for oil. In this example the set of instances $\mathcal{X}$ consists of candidate drilling locations, each described by a set of attributes (e.g. fossil history of the site, geographic features that quantify how well oil is trapped in an area, etc.). The set of classes could be a discrete scale from 1 to n, where 1 indicates no oil would be found, and n indicates a highly abundant supply. To learn its classifier F, the learning algorithm was given a set $S \subset \mathcal{X} \times \{1, \ldots, n\}$ of instance-label pairs as well as a nonnegative, asymmetric cost function c, where $c(j, k)$ measures the cost of money and resources of thinking that an area of class j really was class k. (This function not only indicates the cost of committing excessive resources to an area with too little oil, but also of committing too few resources to an area with abundant oil.) Once the function F is learned and put into practice, it may become the case that the cost function changes from c to c', e.g. if new technologies in drilling and shipping of oil emerge. If this happens, then the function F is no longer appropriate to use. One option to remedy this is to discard F and train a new classifier F' on S under cost function c'. However this may be very resource-consuming since many learning algorithms take extensive time and effort to train. (It may also be impossible if the data S is unavailable, say due to proprietary restrictions.) In this case the best (or perhaps only) choice is to reoptimize F based on a new (possibly smaller) data set. Such problems have been studied extensively (Fieldsend & Everson, 2005; Ferri et al., 2003; Hand & Till, 2001; Lachiche & Flach, 2003; Mossman, 1999; O'Brien & Gray, 2005; Srinivasan, 1999).

For learning tasks with only $n = 2$ classes, this problem is equivalent to that of finding the *optimal operating point* of a classifier given a ratio of true positive cost to false positive cost and has a straightforward solution via Receiver Operating Characteristic (ROC) analysis (Lachiche & Flach, 2003). ROC analysis takes a classifier F that outputs confidences in its predictions (i.e. a ranking classifier), and precisely describes the tradeoffs between true positive and false positive errors. By ranking all examples $x \in S$ by their confidences $h(x)$ from largest to smallest (denoted $S = \{x_1, \ldots, x_m\}$), one achieves a set of $n+1$ binary classifiers by setting thresholds $\{\theta_i = (h(x_i) + h(x_{i+1}))/2, 1 \le i < m\} \cup \{h(x_1) - \epsilon, h(x_m) + \epsilon\}$ for some constant $\epsilon > 0$. Given a relative cost c of true positive error to false positive error and a validation set S of labeled examples, one can easily find the optimal threshold θ based on S and c (Lachiche & Flach, 2003). To do so, simply rank the examples in S, try every threshold θ_i as described above, and select the θ_i minimizing the total cost of all errors on S.

Though the binary case lends itself to straightforward optimization, working with multi-class problems makes things more difficult. A natural idea is to think of an n-class ROC space having dimension $n(n-1)$. A point in this space corresponds to a classifier, with each coordinate representing the misclassification rate of one class into some other class[2]. According to Srinivasan (1999), the optimal classifier lies on the convex hull of these points. Given this ROC polytope, a validation set, and an $n \times n$ cost matrix M with entries $c(y, \hat{y})$ (the cost associated with misclassifying a class y example as class $\hat{y}$), Lachiche and Flach (2003) define the optimization problem as finding a weight vector $\vec{w} \ge \vec{0}$ to minimize (1).

No efficient algorithm is known to optimally solve this problem for $n > 2$, and Lachiche and Flach (2003) speculate that the problem is computationally hard. We present a proof that the decision version of this problem is in fact NP-complete. As a complementary positive result, we give an algorithm that finds an optimal solution in polynomial time (w.r.t. the number of examples, m) when the number of classes n is constant. We also present several new heuristics for this problem, including an integer linear programming relaxation, a sum-of-linear fractional functions (SOLFF) formulation, and a quadratic programming formu-

[1]Equivalently, a change in the class distribution (the probability of seeing examples from each particular class) can be handled by modeling such a change as a change in cost function.

[2]Assuming that cost is zero if the classification is correct, we need only $n(n-1)$ instead of n^2 dimensions.

lation as well as a direct optimization of (1) with a genetic algorithm. Finally, we present a new custom algorithm based on partitioning the set of all classes into two *metaclasses*. This algorithm is similar to that of Lachiche and Flach (2003), but is more efficient. In our experiments, our algorithms yielded several significant improvements both in minimizing classification error and minimizing cost.

The rest of this paper is as follows. In Section 2 we discuss related work. In Section 3 we prove the decision version of this problem (which we call Reweight) is NP-complete and in Section 4 we present an algorithm for producing an optimal solution that is efficient for a constant number of classes. Next, in Section 5 we discuss our heuristic approaches to this problem. We then experimentally evaluate our algorithms in Section 6 and conclude in Section 7.

2 Related Work

The success of binary ROC analysis gives hope that it may be possible to adapt similar ideas to multi-class scenarios. However, research efforts (Srinivasan, 1999; Hand & Till, 2001; Ferri et al., 2003; Lachiche & Flach, 2003; Fieldsend & Everson, 2005) have shown that extending current techniques to multi-class problems is not a trivial task. One key aspect to binary ROC analysis is that it is highly efficient to represent trade-offs of misclassifying one class into the other via binary ROC curves. In addition, the "area under the curve" (AUC) nicely characterizes the classifier's ability to produce correct rankings without committing to any particular operating point. Decisions can be postponed until a desired trade-off is required (e.g. finding the lowest expected cost).

Now consider the problem of classification in an n-class scenario. A natural extension from the binary case is to consider a multi-class ROC space as having dimension $n(n-1)$. A point in this space corresponds to a classifier with each coordinate representing the misclassification rate of one class into some other class. Following from Srinivasan (1999), the optimal classifier lies on the convex hull of these points.

Previous investigations have all shared this basic framework (Mossman, 1999; Srinivasan, 1999; Hand & Till, 2001; Ferri et al., 2003; Lachiche & Flach, 2003; Fieldsend & Everson, 2005; O'Brien & Gray, 2005). They differ, however, in the metrics they manipulate and in the approach they use to solve multi-class optimization problems. Mossman (1999) addressed the special case of three-class problems, focusing on the statistical properties of the volume under the ROC surface. This motivated the later work of Ferri et al. (2003), Lachiche and Flach (2003), and O'Brien and Gray (2005). Hand and Till (2001) extended the definition of two-class AUC by averaging pairwise comparisons. They used this new metric in simple, artificial data sets and achieved some success. Ferri et al. (2003) took a different approach in which they strictly followed the definition of two-class AUC by using "Volume Under Surface" (VUS). They were able to compute the bounds of this measure in a three-class problem by using Monte Carlo methods. However, it is not known how well this measure performs in more complex problems.

Fieldsend and Everson (2005), Lachiche and Flach (2003) and O'Brien and Gray (2005) developed algorithms to minimize the overall multi-class prediction accuracy and cost given some knowledge of a multi-class classifier. In particular, Fieldsend and Everson approximate the ROC Convex Hull (ROCCH) using the idea of "Pareto front." Consider the following formulation: let $R_{j,k}(\theta)$ be the misclassification rate of predicting examples from class j as class k. This is a function of some generalized parameter θ that depends on the particular classifiers. For example, θ may be a combination of a weight vector $\vec{w}$ and hypothetical cost matrix M. The goal is to find θ that minimizes $R_{j,k}(\theta)$ for all j, k with $j \neq k$. Consider two classifiers θ and ϕ. Fieldsend and Everson say θ *strictly dominates* ϕ if all misclassification rates for θ are no worse than ϕ and at least one rate is strictly better. The set of all feasible classifiers such that no one is dominated by the other forms the *Pareto front.* Fieldsend and Everson present an evolutionary search algorithm to locate the Pareto front. This method is particularly useful when misclassification costs are not necessarily known.

More closely related to our work are the results of Lachiche and Flach (2003) and O'Brien and Gray (2005). Lachiche and Flach considered the case when the misclassification cost is known, and the goal is to find the optimal decision criterion that fits the training set. Recall that this can be solved optimally for the binary case. In particular, only one threshold θ is needed to make the decision for two-class problems. Since there are only $m + 1$ possible thresholds for m examples, it is efficient enough to simply test all possibilities and select the one that gives the minimum average error (or cost). However, the situation is more complicated for multi-class problems. The main obstacle in the multi-class case is that the number of possible classification assignments grows exponentially in the number of instances: $\Omega(n^m)$.

Lachiche and Flach (2003) formulated the multi-class problem as follows. Suppose the multi-class learning algorithm will output a positive, real-valued function $f : \{x_1, \ldots, x_m\} \times \{C_1, \ldots, C_n\} \to \mathbb{R}^+$. Here, $f_j(x_i)$ gives the confidence that example x_i belongs to class j. The decision criterion simply assigns example x_i to the class with maximum score. Reweighting the classes involves defining a nonnegative weight vector $\vec{w} = (w_1, w_2, \ldots, w_n)$, and predicting the class for an example x as

$$h(x) = \underset{1\le j\le n}{\operatorname{argmax}} \left\{ w_j f_j(x) \right\} \ .$$

Since $\vec{w}$ has only $n-1$ degrees of freedom, so we can fix $w_1 = 1$.

Lachiche and Flach (2003) used a hill climbing heuristic to find a good weight vector $\vec{w}$. In particular, they took advantage of the fact that the optimal threshold for the two-class problem can be found efficiently. For each coordinate in the weight vector, they mapped the problem to a binary problem. The algorithm starts by assigning $w_1 = 1$ and all other weights 0. It then tries to decide the weight for one class at a time as follows. Let S be the set of labeled examples and let j be the current class for which we want to assign a "good" weight w_j. Then the set of possible weights for w_j is

$$\left\{ \frac{\max_{i\in\{1,\ldots,j-1\}} f_i(x)}{f_j(x)} \,\middle|\, x \in S \right\} \ .$$

It is not difficult to see that at any stage there are at most $O(|S|)$ possible weights that can influence the prediction. Thus choosing the optimal weight in this setting can be easily achieved by checking all possibilities. Overall, their algorithm runs in time $\Theta(nm \log m)$. Though there is no guarantee that this approach can find an optimal solution, they gave empirical results that it works well for optimizing 1BC, a logic-based Bayes classifier (Lachiche & Flach, 1999).

Although only briefly mentioned by Lachiche and Flach (2003), this ROC thresholding technique is quite extensible to cost-sensitive scenarios. O'Brien and Gray (2005) investigated the role of a cost matrix in partitioning the estimated class probability space and as a replacement for the weights. Assuming that M is a misclassification cost matrix, an optimal decision criterion would be

$$h(x) = \underset{1\le j\le n}{\operatorname{argmax}} \left\{ \sum_{1\le k\le n} c(j,k)\, \hat{p}_k(x) \right\} \ .$$

If $\hat{p}_k(x)$ is a good probability estimate of example x belonging to class k, this prediction results in the lowest expected cost. However, if $\hat{p}_k(x)$ is not an accurate probability estimate, then to ensure optimality, the cost matrix M has to be altered accordingly. Thus the cost matrix M plays a similar role as the weight vector of Lachiche and Flach (2003) in defining the decision boundary in estimated probability space. O'Brien and Gray (2005) defined several standard operations to manipulate the cost matrix M and proposed the use of a greedy algorithm to find the altered cost matrix (called a *boundary matrix*).

While the multi-class problem has been studied via heuristics, no one has yet answered the question as to whether this problem is hard, and no one has found efficient algorithms to solve restricted cases of the multi-class problem. Below we provide answers to both of these open questions as well as extend the current literature of heuristics.

3 Hardness

We now prove our hardness result of this problem. For convenience, we use f_{ij} to denote $f_i(x_j)$. We will show hardness for the uniform cost case, i.e. $c(j,k) = 1$ when $j \neq k$ and 0 otherwise. This of course implies hardness for any nontrivial cost function.

Definition 1. Problem **Reweight**
Given: nonnegative real numbers f_{ij} ($i = 1, \dots, m$, $j = 1, \dots, n$), integers $y_i \in \{1, \dots, n\}$, and an integer K.
Question: does there exist a vector of nonnegative real numbers $(w_1, \dots, w_n)$ such that

$$\left| \left\{ i : \max_{j \neq y_i} \{w_j f_{ij}\} \geq w_{y_i} f_{iy_i} \right\} \right| \leq K \ ? \tag{2}$$

In other words, the problem is to find a vector $\vec{w} = (w_1, \dots, w_n)$ that maximizes how often $w_j f_{ij}$ is maximized (over j) by the correct label y_i.

To prove the hardness of Reweight, we will reduce from the minimum satisfiability problem MinSat, shown to be NP-complete by Kohli et al. (1994).

Definition 2. Problem **MinSat** (Kohli et al., 1994)
Given: a set of disjunctions of pairs of literals

$$\begin{array}{ccc} \ell_{11} & \vee & \ell_{12} \\ \ell_{21} & \vee & \ell_{22} \\ & \vdots & \\ \ell_{m1} & \vee & \ell_{m2} \ , \end{array}$$

where each ℓ_{ij} is a boolean variable x_i or its negation $\neg x_i$. We are also given an integer K.
Question: does there exist a setting of $x_1, \dots, x_n$ such that the number of satisfied clauses (disjuncts) is at most K?

Theorem 1. Reweight *is* NP*-complete.*

Proof. First, it is easy to see that Reweight is in NP. The certificate is simply the weight vector $\vec{w}$. This certificate is sized polynomially in the size of the input since its required precision is polynomially proportional to the precision of the input (the number of bits to represent each f_{ij}). We now reduce from MinSat. Note that a special case of the constraint

$$\max_{j \neq y_i} \{w_j f_{ij}\} \geq w_{y_i} f_{iy_i}$$

used in (2) is an inequality of the form

$$w_{j_0} f_{ij_0} \geq w_{y_i} f_{iy_i} \tag{3}$$

for one particular $j_0 \neq y_i$. This can be seen simply by setting all of the other f_{ij}'s to zero, which gives

$$\max\{0, w_{j_0} f_{ij_0}\} \geq w_{y_i} f_{iy_i} \ . \tag{4}$$

Since in our construction the w_j's and f_{ij}'s are nonnegative, these are equivalent. So, in what follows, we give constraints of the form of (3), but these really are of the form of (4). Thus while for the sake of clarity we map instances of MinSat to inequalities, it is straightforward to convert these to a collection of f_{ij} and y_i values in Reweight.

Given an instance of MinSat as above, we create an instance of Reweight. The new instance has $n' = 2n + 1$ weights: v_0; $w_1, \dots, w_n$; and $w'_1, \dots, w'_n$. The weight v_0 is forced to be strictly positive, and is used as a reference for all other weights. Roughly speaking, w_i will correspond to boolean variable x_i and w'_i will correspond to its negation. More specifically, we will force w_i to have a value close to $2v_0$ if x_i is true, and a value close to v_0 otherwise; w'_i will be forced to take just the opposite values (close to v_0 if x_i is true, close to $2v_0$ if x_i is false). We will also construct constraints corresponding to the MinSat clauses which are satisfied if and only if the MinSat clauses are satisfied.

To be more specific, we construct four classes of constraints. Each of these constraints actually gets repeated several times in the construction of the reduced instance, meaning that if the constraint holds, then it holds several times. In this way, the constraints can be assigned varying importance weights.

A. First, we force v_0 to be strictly positive. To do so, we include the constraint:

$$v_0 \leq 0 \ .$$

(Recall that the goal is to *minimize* how many of these constraints are satisfied, which effectively means that it will be forced to fail so that $v_0 > 0$.) This constraint gets repeated r_A times, as specified below.

B. Next, we force each w_i and w'_i to have a value roughly between v_0 and $2v_0$. To do so, we simply include constraints:

$$\begin{aligned} w_i &\leq 0.99v_0 \\ w_i &\geq 2.01v_0 \\ w'_i &\leq 0.99v_0 \\ w'_i &\geq 2.01v_0 \end{aligned}$$

for each i. Each of these is repeated r_B times.

C. Next, we add constraints that will effectively force (for each i) exactly one of w_i and w'_i to be close to v_0, and the other to be close to $2v_0$. These are the constraints:

$$\begin{aligned} w_i &\leq 1.99w'_i \\ w'_i &\leq 1.99w_i \ . \end{aligned}$$

In the optimal solution, we will see that exactly one of these two constraints will hold. These constraints each get repeated r_C times.

D. Finally, we encode the actual clauses of the MinSat instance. A MinSat clause of the form $x_i \vee x_j$ becomes the constraint

$$0.8w'_i \leq w_j \ .$$

A MinSat clause of the form $\neg x_i \vee x_j$ becomes the constraint

$$0.8w_i \leq w_j \ .$$

A MinSat clause of the form $x_i \vee \neg x_j$ becomes the constraint

$$0.8w'_i \leq w'_j \ .$$

And a MinSat clause of the form $\neg x_i \vee \neg x_j$ becomes the constraint

$$0.8w_i \leq w'_j \ .$$

Each of these is repeated only once.

The value K for the instance of Reweight that we constructed is denoted K' (reserving K for the corresponding value of the original MinSat instance). We let:

$$\begin{aligned} r_C &= K + 1 \\ K' &= K + n\,r_C \\ r_B &= K' + 1 \\ r_A &= K' + 1 \ . \end{aligned}$$

This completes the construction, which is clearly polynomial in all respects. We now need to argue that the MinSat instance is "yes" if and only if the reduced Reweight instance is also "yes".

Suppose then that $x_1, \ldots, x_n$ satisfies at most K of the MinSat clauses. In this case, we can easily construct settings of the weights so that at most K' of the constructed constraints are satisfied.

Let $v_0 = 1$ and let

$$w_i = \begin{cases} 1 & \text{if } x_i = 0 \\ 2 & \text{if } x_i = 1 \end{cases} ,$$

and let

$$w'_i = \begin{cases} 1 & \text{if } w_i = 2 \\ 2 & \text{if } w_i = 1 \end{cases} .$$

Then none of the constraint of types A and B is satisfied. Exactly half of the constraints of type C are satisfied, which means $n\, r_C$ constraints of type C are satisfied.

What about the constraints of type D? We claim that for each satisfied clause of the MinSat solution, the corresponding type-D constraint is satisfied. If a clause of the form $x_i \vee x_j$ is satisfied, then $x_i = 1$ or $x_j = 1$, which means that $w_i = 2$ or $w_j = 2$, which means $w'_i = 1$ or $w_j = 2$, which means that $0.8w'_i \leq w_j$. Conversely, if $x_i \vee x_j$ is not satisfied, then $x_i = 0$ and $x_j = 0$, which means that $w_i = 1$ and $w_j = 1$, which means $w'_i = 2$ and $w_j = 1$, which means that $0.8w'_i \not\leq w_j$. (The arguments are the same when some of the variables appear negated in the clause.)

Thus, because at most K of the clauses are satisfied, it follows that at most K of the type-D constraints are satisfied. Therefore, the total number of satisfied constraints is at most $K + n\, r_C = K'$.

We now argue the other direction. Suppose that $v_0, w_1, \ldots, w_n, w'_1, \ldots, w'_n$ satisfy at most K' of the constructed constraints.

First of all, this means that

$$v_0 > 0$$

since $r_A > K'$. Also, since $r_B > K'$, this means that

$$0.99v_0 < w_i < 2.01v_0 \tag{5}$$

and

$$0.99v_0 < w'_i < 2.01v_0 \ . \tag{6}$$

Next, we claim that either

$$w_i \leq 1.99w'_i \tag{7}$$

or

$$w'_i \leq 1.99w_i \ . \tag{8}$$

Otherwise, if neither of these were true, then we would have

$$w_i > 1.99w'_i > (1.99)^2 w_i \ ,$$

which implies that $w_i < 0$. However, we have already established that $w_i > 0.99v_0 > 0$.

Further, we claim that at most one of (7) or (8) can be satisfied. We already have established that at least one constraint of each pair must be satisfied. If, in addition, both held for some pair, then the number of satisfied type-C constraints would be at least

$$(n-1)r_C + 2r_C = n\, r_C + r_C > K' \ .$$

So exactly one of each pair of type-C constraints is satisfied.

We next claim that for each i, exactly one of w_i and w'_i is in the interval $(0.99v_0, 1.1v_0)$ and the other is in $(1.9v_0, 2.01v_0)$. We know that either (7) or (8) is satisfied. Suppose $w'_i > 1.99w_i$. If $w'_i \leq 1.9v_0$ then

$$w_i < \frac{w'_i}{1.99} \leq \frac{1.9v_0}{1.99} < 0.99v_0 \ ,$$

a contradiction since we have already shown that $w_i > 0.99v_0$. Also, if $w_i \geq 1.1v_0$, then

$$w'_i > 1.99w_i \geq 1.99 \cdot 1.1v_0 > 2.01v_0 \ ,$$

again a contradiction since $w_i' < 2.01v_0$. So in this case, $w_i < 1.1v_0$ and $w_i' > 1.9v_0$. Moreover, because (5) and (6) hold, we have in this case that $w_i \in (0.99v_0, 1.1v_0)$ and $w_i' \in (1.9v_0, 2.01v_0)$. In this case, we assign the boolean variable x_i the value 0. By a similar argument, if $w_i > 1.99w_i'$ then $w_i' \in (0.99v_0, 1.1v_0)$ and $w_i \in (1.9v_0, 2.01v_0)$. In this case, we assign the boolean variable x_i the value 1.

We have established that exactly $n\,r_C$ type-C constraints are satisfied, and none of the type-A and type-B constraints is satisfied. Since at most K' constraints are satisfied altogether, this means that at most K type-D constraints are satisfied. We complete the reduction by showing that a type-D constraint is satisfied if and only if the corresponding MinSat clause is satisfied (according to the assignment constructed above), which will mean that at most K of these are satisfied.

Suppose $x_i \vee x_j$ is satisfied. Then $x_i = 1$ or $x_j = 1$, which means either $w_i > 1.9v_0$ or $w_j > 1.9v_0$, which means either $w_i' < 1.1v_0$ or $w_j > 1.9v_0$. We claim, in either case, that the constraint $0.8w_i' \leq w_j$ is satisfied. For if $w_i' < 1.1v_0$, then because $w_j > 0.99v_0$, we have

$$0.8w_i' < 0.8 \cdot 1.1v_0 = 0.88v_0 < 0.99v_0 < w_j \ .$$

And if $w_j > 1.9v_0$ then because $w_i' < 2.01v_0$, we have

$$0.8w_i' < 0.8 \cdot 2.01v_0 = 1.608v_0 < 1.9v_0 < w_j \ .$$

Conversely, if $x_i \vee x_j$ is not satisfied then $x_i = 0$ and $x_j = 0$, which means that $w_i < 1.1v_0$ and $w_j < 1.1v_0$, which means that $w_i' > 1.9v_0$ and $w_j < 1.1v_0$, which means that

$$0.8w_i' > 0.8 \cdot 1.9v_0 = 1.52v_0 > 1.1v_0 > w_j \ ,$$

so the constraint $0.8w_i' \leq w_j$ is not satisfied. □

4 Constant-Class Algorithm

We now present an algorithm that finds an optimal solution for a nonuniform cost function in polynomial time when the number of classes n is constant. Our algorithm takes as input a set of nonnegative real numbers f_{ij} for $i = 1, \ldots, m$ and $j = 1, \ldots, n$, integers (labels) $y_i \in \{1, \ldots, n\}$, and a nonnegative cost function $c : \{1, \ldots, n\}^2 \to \mathbb{R}^+$. Assuming n is a constant, in time polynomial in m it will output a vector of weights $(w_1, \ldots, w_n)$ that minimizes (1).

Our algorithm is based on the observation that class j will be predicted for instance i if and only if $w_j/w_k > f_{ik}/f_{ij}$ for all $k \neq j$. Thus for a fixed (j, k) pair, there are only m values of f_{ik}/f_{ij} that can affect the value of (1). We will call these values *breakpoints*. For each (j, k) pair, one can easily compute all breakpoints, add in $-\infty$ and $+\infty$, sort them, and place them in an ordered set $B^{jk} = (-\infty, f_{1k}/f_{1j}, \ldots, f_{mk}/f_{mj}, +\infty)$. We use B_ℓ^{jk} to denote the ℓth element in B^{jk}. So a candidate range of values of w_j/w_k is $\left(B_\ell^{jk}, B_{\ell+1}^{jk}\right]$.

We now define a *configuration* C as a set of pairs of breakpoints across all (j, k) pairs:

$$C = \bigcup_{j \in \{1,\ldots,n\}, k \neq j} \left\{ B_{\ell_{jk}}^{jk} < w_j/w_k \leq B_{1+\ell_{jk}}^{jk} \right\} \ ,$$

for $\ell_{jk} \in \{1, \ldots, m+1\}$. We say that C is *realizable* if there exists a set of nonnegative weights that satisfies all of its constraints. If no such set of weights exists, we say C is *unrealizable*.

The idea of our algorithm is simple. It enumerates each configuration C and then uses linear programming to test if C is realizable[3]. If it is not, then we test the next configuration. If instead C is realizable, then the weight vector returned by the linear programming algorithm is one of our candidate solutions to minimize

[3]To handle the strict inequalities, we simply convert each constraint $a > b$ to $a \geq b + \epsilon$, constrain $1 \geq \epsilon \geq 0$, then maximize ϵ subject to the new constraints (note that we use the same ϵ for each strict constraint). If a solution is returned with $\epsilon > 0$ then we know the configuration is realizable. If no solution is found or if $\epsilon = 0$, then it is not realizable.

(1). Our algorithm stores this weight vector with its cost and then checks the remaining configurations. Once all configurations have been checked, the algorithm returns the one with minimum cost.

Consider an optimal solution $\vec{w}^*$. Let C^* be the configuration it satisfies. Any other positive weight vector $\vec{w}$ that also satisfies C^* must also be optimal since it induces the same collection of predictions and hence has the same value of (1). Since our algorithm tries all configurations, it also tries C^*. Since C^* is realizable, the algorithm finds a weight vector $\vec{w}$ satisfying it. Since $\vec{w}$ must be optimal, our algorithm returns an optimal weight vector.

Each (j,k) pair has at most $m+2$ breakpoints, which means that the number of $(B_\ell^{jk}, B_{\ell+1}^{jk})$ pairs for class pair (j,k) is at most $m+1$. So the number of configurations is at most

$$(m+1)^{2\binom{n}{2}} = (m+1)^{n^2-n} \ ,$$

which is polynomial for constant n. (We can also combine B^{jk} with B^{kj} before sorting, which would reduce the number of configurations to $(2m+1)^{\binom{n}{2}}$.) Further, it takes polynomial time to test each configuration for realizability via linear programming and it takes polynomial time to compute the cost of each candidate solution. Therefore this algorithm takes polynomial time.

Theorem 2. *There exists an algorithm to produce an optimal solution to* (1) *that runs in polynomial time when the number of classes n is constant.*

5 Heuristics

For even modest values of n the time complexity of the constant-class algorithm in the previous section is not practical. For this reason, we present several alternative heuristics to solve the reoptimization problem. Specifically, we present several new mathematical programming formulations. First, we reformulate the objective function (1) as a relaxed integer linear program. We also give formulations as a sum of linear fractional functions (SOLFF) as well as a quadratic program. Besides these formulations, we describe a tree-based heuristic algorithm approach, MetaClass. Finally, (in Section 6) we present experimental results from these formulations. We give evidence that, under nonuniform costs, the objective function landscape for this problem is highly discontinuous and thus more amenable to global optimization methods such as genetic algorithms and margin maximization methods.

5.1 Mathematical Programming Formulations

5.1.1 Relaxed Integer Linear Program

We start by reformulating (1) as follows:

$$\underset{\vec{w},\vec{I}}{\text{minimize}} \left\{ \sum_{j=1}^{m} \sum_{k=1}^{m} c(j,k) \sum_{x_i \in C_j} I_{i,k} \right\} \ , \tag{9}$$

where $C_j \subseteq S$ is the set of instances of class j, $c(j,k)$ is the cost of misclassifying an example from class j as k, and

$$I_{i,k} = \begin{cases} 1 & \text{if } w_k f_k(x_i) \geq w_\ell f_\ell(x_i), \ell \neq k \\ 0 & \text{otherwise} \end{cases} .$$

Recall that $f_k(x_i)$ is the base learner's confidence that example x_i belongs to class k. Also, we assume $c(j,j) = 0$ for all classes j. Formalizing this as a constrained optimization problem, we want to minimize (9) subject to

$$I_{i,j} w_j f_j(x_i) = I_{i,j} \max_{1 \leq k \leq m} \{w_k f_k(x_i)\} \tag{10}$$

$$\textstyle\sum_{j=1}^{m} I_{i,j} = 1 \tag{11}$$

$$I_{i,j} \in \{0,1\} \tag{12}$$

$$w_j \geq 0 \tag{13}$$

where each constraint holds for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. Equation (10) allows only the class that has the max value of $w_k f_k(x_i)$ to be indicated by $\vec{I}$ to be the predicted class of example x_i and (11) forces exactly one class to be predicted per example x_i.

We can change the optimization problem in two ways to get an equivalent problem. First, we change the "=" in (10) to "$\geq$". Second, we can relax (12) to be $I_{i,j} \in [0, 1]$. Note that (10) (even when amended with "$\geq$") will only be satisfied if $I_{i,j} = 0$ for all C_j that don't maximize the RHS of (10). Thus, so long as we never have $w_k f_k(x_i) = w_{k'} f_{k'}(x_i)$ for some $k \neq k'$, the relaxation is equivalent to the original problem. Further, even if there is such a tie for classes k and k', it will not be an issue if the corresponding entries in the cost matrix are different, since an optimal solution will set $I_{i,k} = 1$ and $I_{i,k'} = 0$ if $c(j, k) < c(j, k')$. The potential problem of both $w_k f_k(x_i) = w_{k'} f_{k'}(x_i)$ and $c(j, k) = c(j, k')$ is fixed by (after optimizing) checking for any $I_{i,k} \notin \{0, 1\}$ and arbitrarily choosing one to be 1 and the rest 0. Note that since there is a tie in this case, the prediction can go either way and the weight vector $\vec{w}$ returned is still valid.

Everything except (10) is linear. We now reformulate it. First, for each $i \in \{1, \ldots, n\}$, we substitute γ_i for $\max_{1 \leq k \leq m}\{w_k f_k(x_i)\}$:

$$I_{i,j} w_j f_j(x_i) \geq \gamma_i \, I_{i,j} \tag{14}$$

$$w_k f_k(x_i) \leq \gamma_i \ , \tag{15}$$

for all $i \in \{1, \ldots, n\}$ and $j, k \in \{1, \ldots, m\}$ where each γ_i is a new variable. Obviously (15) is a linear constraint, but (14) is not even quasiconvex (Boyd & Vandenberghe, 2004). The complexity of this optimization problem motivates us to reformulate it a bit further.

Let us assume that $f_k(x_i) \in (0, 1]$ (e.g. if $f_k(\cdot)$ are probability estimates from naïve Bayes or logistic regression). Now we can optimize (9) subject to:

$$\gamma_i - w_j f_j(x_i) + I_{i,j} \leq 1 \tag{16}$$

$$\gamma_i \geq w_j f_j(x_i) \tag{17}$$

$$\textstyle\sum_{j=1}^{m} I_{i,j} = 1 \tag{18}$$

$$I_{i,j} \in \{0, 1\} \tag{19}$$

$$w_j \geq 0 \tag{20}$$

for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$.

So long[4] as $w_j f_j(x_i) \in (0, 1]$ and $I_{i,j} \in \{0, 1\}$ for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, this is another equivalent optimization problem, this time a $\{0, 1\}$ integer linear program. Unfortunately, we cannot relax (19) to $I_{i,j} \in [0, 1]$ as we did before to get an equivalent problem. But we still use the relaxation as a linear programming heuristic. To help avoid overfitting, we also add a linear regularization term to (9):

$$\underset{\vec{w}, \vec{I}}{\text{minimize}} \left\{ \sum_{j=1}^{m} \sum_{k=1}^{m} c(j, k) \sum_{x_i \in C_j} I_{i,k} + \eta \|\vec{w} - \vec{1}\|_1 \right\} \tag{21}$$

where $\|\cdot\|_1$ is the 1-norm, $\vec{1}$ is the all-1s vector, and η is a parameter. This regularizer penalizes large deviations from the original classifier.

5.1.2 Sum of Linear Fractional Functions Formulation

Another formulation comes from changing how predictions are made from deterministic to probabilistic. In this prediction model, given a new example x to predict on, first compute $w_j f_j(x)$ for each $j \in \{1, \ldots, m\}$. Then predict class j for example x with probability

$$\frac{w_j f_j(x)}{\sum_{k=1}^{m} w_k f_k(x)} \ .$$

[4]We can ensure this happens by bounding each w_j appropriately.

Assuming a uniform distribution over the data set, the expected cost of this predictor is

$$\sum_{j=1}^{m}\sum_{k=1}^{m} c(j,k) \sum_{x_i \in C_j} \varphi(i,j) \ , \tag{22}$$

where

$$\varphi(i,j) = \frac{w_j f_j(x_i)}{\sum_{\ell=1}^{m} w_\ell f_\ell(x_i)}$$

subject to $w_j \geq 0$ for all $j \in \{1,\ldots,m\}$. We now have eliminated the variables $I_{i,j}$ and their integer constraints. However, we now have a nonlinear objective function in (22). Each individual term of the summation of (22) is a *linear fractional function*, which is quasiconvex and quasiconcave, and thus it is efficiently solvable optimally. However, the *sum of linear fractional functions* (SOLFF) problem is known to be hard (Matsui, 1996) and existing algorithms for this problem are inappropriate in solving (22) (they either restrict to few terms in the summation or to low-dimensional vectors). Instead, we apply a genetic algorithm to directly optimize (22).

5.1.3 Quadratic Programming Formulation

Convex programming is a special case of nonlinear programming in which the objective function and the inequality constraint functions are convex and the equality constraint functions are affine. The theory of convex programming is well-established (Rockafellar, 1970; Stoer & Witzgall, 1996). For a convex program, a local optimum is the global optimum and there are well-studied efficient algorithms to find such global optimum.

We tried several quadratic programming methods based on the idea of "margin maximization" in support vector machines. We found from our experiments that the ν-SVM-like formulation similar to that of Schölkopf and Smola (2002) gave the strongest result:

$$\underset{\vec{w},\vec{b},\vec{\zeta},\rho}{\text{minimize}} \quad \tfrac{1}{2}\|\vec{w}\|^2 + \tfrac{1}{m}\textstyle\sum_{i=1}^{m} \zeta_i - \nu\rho \tag{23}$$

$$\text{s.t.} \quad w_j f_j(x_i) + b_j \leq w_i f_{y_i}(x_i) + b_{y_i} + \zeta_i - \rho \quad \forall i, \forall j \neq y_i \tag{24}$$

$$\vec{\ell}_w \leq \vec{w} \leq \vec{u}_w \tag{25}$$

$$\vec{\ell}_b \leq \vec{b} \leq \vec{u}_b \tag{26}$$

$$\vec{0} \leq \vec{\zeta} \tag{27}$$

$$0 \leq \rho \tag{28}$$

where $\vec{w} = (w_1,\ldots,w_n)$ is our weight vector and $\vec{b} = (b_1,\ldots,b_n)$ is a reweighting *offset*. The vector $\vec{\zeta} = (\zeta_1,\ldots,\zeta_n)$ serves as set of slack variables and ρ as the margin with ν as a parameter. Furthermore, $\vec{l}_w, \vec{u}_w$ are the lower and upper bounds of $\vec{w}$ and $\vec{l}_b, \vec{u}_b$ are bounds for $\vec{b}$, all tunable parameters. In order to capture nonuniform costs we replace ζ_i with $c_i\zeta_i$ where $c_i = \max_{j=1,\ldots,m}\{c(y_i,j)\}$.

In our experiments, ν was fixed to be 0.1. The lower and upper bounds on w, $\vec{l}_w, \vec{u}_w$, were set to 0 and 1 respectively. We also tried several parameters for the offset, but little difference was observed. Thus, our experimental results use no offset (the lower and upper bounds on $\vec{b}$ were set to 0).

5.2 The MetaClass Heuristic Algorithm

We now present a new algorithm that we call MetaClass (Algorithm 1). This algorithm is similar to that of Lachiche and Flach (2003) in that we reduce the multi-class problem to a series of two-class problems. However, we take what can be considered a top-down approach while the algorithm of Lachiche and Flach (2003) can be considered bottom-up. Moreover, MetaClass has a faster time complexity. The output of the algorithm is a decision tree with each internal node labeled by two *metaclasses* and a threshold value.

Each leaf node is labeled by one of the classes in the original problem. At the root, the set of all classes is divided into two metaclasses. The criterion for this split may be based on any statistical measure, but for simplicity, experiments were performed by splitting classes so that each metaclass would have roughly the same number of examples. For each metaclass, our algorithm defines confidence functions $g_1(x_i)$ and $g_2(x_i)$ for each instance x_i, which are simply the sum of the confidences of the classes in $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. The ratio $G(x_i) = g_1(x_i)/g_2(x_i)$ is used to find a threshold θ. We find θ by sorting the instances according to $G(x_i)$ and choose a threshold that minimizes error. (This threshold will be the average of $G(x_i)$ and $G(x_{i+1})$ for some instance x_i.) We recursively perform this procedure on the two metaclasses until there is only a single class, at which point a leaf is formed.

The situation for nonuniform costs is slightly more complicated since it is not clear *which* class among those in metaclass an example is misclassified as. Recall that our cost function $c(y, \hat{y})$ represents the cost of misclassifying an instance x of class y as class $\hat{y}$. However, in this case we need a cost function to quantify the cost of misclassifying an example into a *set* of classes. Formally, we need a function $c' : \mathcal{C} \times 2^{\mathcal{C}} \to \mathbb{R}^+$. There are numerous natural extensions from c to c'. For our experiments, c' represents the average cost of misclassifying instances into metaclasses in $\mathcal{C}_1$ and $\mathcal{C}_2$. More formally, if $\mathcal{C}' \subseteq \mathcal{C}$, we define $c'(y, \mathcal{C}')$ to be 0 if $y \in \mathcal{C}'$ (that is, x's true label class is in the metaclass) and

$$\frac{1}{|\mathcal{C}'|} \sum_{j \in \mathcal{C}'} c(y, j)$$

otherwise. The MetaClass algorithm is presented as Algorithm 1.

```
   Input  : A set of instances, $S = \{x_1, \ldots, x_m\}$; a set of classes, $\mathcal{C} = \{1, \ldots, n\}$;
            a learned confidence function $f : S \times \mathcal{C} \to \mathbb{R}^+$ and a tree node $T$
   Output : A decision tree with associated weights.
 1 Split $\mathcal{C}$ into two metaclasses $\mathcal{C}_1, \mathcal{C}_2$ such that each metaclass has about an equal
   number of classes.
 2 foreach Instance $x_i \in S$ do
 3     $g_1(x_i) = \sum_{j \in \mathcal{C}_1} f_j(x_i)$
 4     $g_2(x_i) = \sum_{j \in \mathcal{C}_2} f_j(x_i)$
 5     $G(x_i) = g_1(x_i)/g_2(x_i)$
 6 end
 7 Sort instances according to $G$
 8 Select a threshold $\theta$ that minimizes
       $$\sum_{i=1}^{m} c'(y_i, M_\theta(x_i))$$
   where
       $$M_\theta(x_i) = \begin{cases} \mathcal{C}_1 & \text{if } \theta \geq G(x_i) \\ \mathcal{C}_2 & \text{otherwise} \end{cases}$$
 9 Label $T$ with $\theta, \mathcal{C}_1, \mathcal{C}_2$
10 Create two children of $T$: $T_{\text{left}}, T_{\text{right}}$
11 Split $S$ into two sets, $S_1, S_2$ according to $\mathcal{C}_1, \mathcal{C}_2$
12 Recursively perform this procedure on $S_1, \mathcal{C}_1, T_{\text{left}}$ and $S_2, \mathcal{C}_2, T_{\text{right}}$ until $|\mathcal{C}| = 1$
```

Algorithm 1: *MetaClass*

Figure 1 gives an example of a tree built by the MetaClass algorithm on the UCI (Blake & Merz, 2005) data set Nursery, a 5-class data set. At the root, the classes are divided into two metaclasses, each with about the same number of examples represented in their respective classes. In this case, the threshold $\theta = 0.8169$ favors the sum of confidences in metaclass $\mathcal{C}_1 = \{4, 3\}$ as an optimal weight.

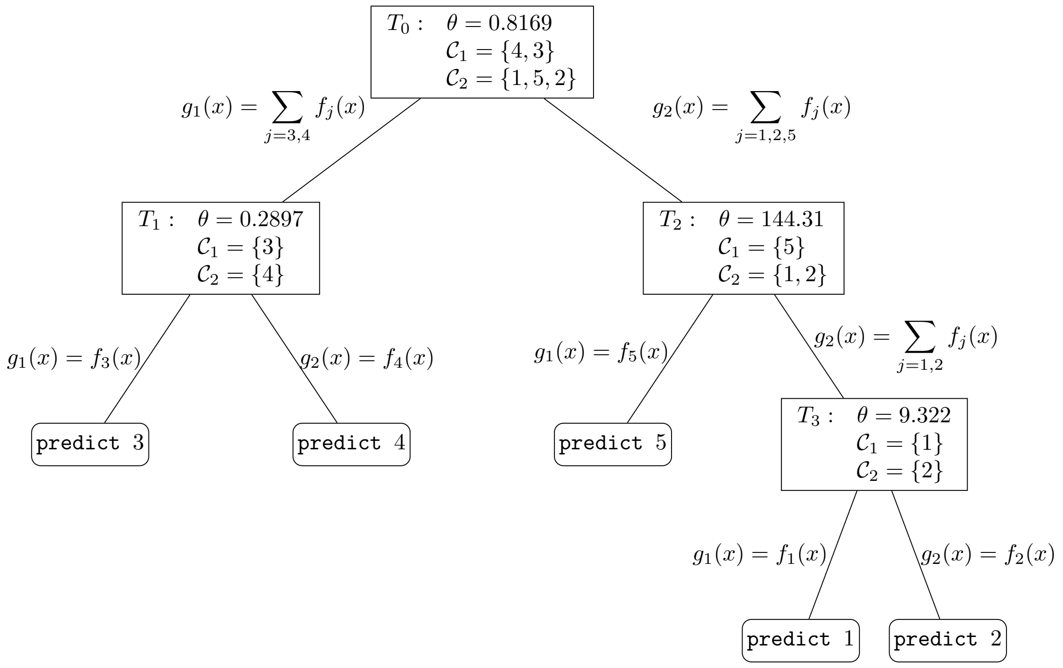
$T_0: \theta = 0.8169$, $\mathcal{C}_1 = \{4,3\}$, $\mathcal{C}_2 = \{1,5,2\}$
$g_1(x) = \sum_{j=3,4} f_j(x)$
$g_2(x) = \sum_{j=1,2,5} f_j(x)$
$T_1: \theta = 0.2897$, $\mathcal{C}_1 = \{3\}$, $\mathcal{C}_2 = \{4\}$
$T_2: \theta = 144.31$, $\mathcal{C}_1 = \{5\}$, $\mathcal{C}_2 = \{1,2\}$
$g_1(x) = f_3(x)$
$g_2(x) = f_4(x)$
$g_1(x) = f_5(x)$
$g_2(x) = \sum_{j=1,2} f_j(x)$
predict 3
predict 4
predict 5
$T_3: \theta = 9.322$, $\mathcal{C}_1 = \{1\}$, $\mathcal{C}_2 = \{2\}$
$g_1(x) = f_1(x)$
$g_2(x) = f_2(x)$
predict 1
predict 2

Figure 1: Example run of MetaClass on Nursery, a 5-class problem.

Predictions for a new example x are made as follows. Starting at the root node, we traverse the tree towards a leaf. At each node T we compute the sum of confidences of x with respect to each associated metaclass. We traverse left or right down the tree depending on whether $g_1(x)/g_2(x) \geq \theta$. When a leaf is reached, a final class prediction is made.

The number of nodes created by MetaClass is $\Theta(n)$, where n is the number of classes. Since the split into two metaclasses ensures each has an equal number of classes, MetaClass results in a $\log(n)$-depth tree. At each level, the most complex step is sorting at most m instances according to the confidence ratio. Thus, the overall time complexity is bounded by $\mathcal{O}(m \log m \log n)$. This represents a significant speedup to the algorithm of Lachiche and Flach (2003), which requires $\Theta(nm \log m)$ time. Classification is also efficient. At each node we compute a sum over an exponentially shrinking number of classes. The overall number of operations is thus

$$\sum_{i=0}^{\log(n)-1} \frac{n}{2^i} ,$$

which is linear in the number of classes: $\Theta(n)$. This matches the time complexity of Lachiche and Flach's with respect to classification.

6 Experimental Results

The following experiments were performed on 25 standard UCI data sets (Blake & Merz, 2005), using Weka's naïve Bayes (Witten et al., 2005) as the baseline classifier and Matlab's optimization functions for reoptimization. We ran experiments evaluating improvements both in classification accuracy and under nonuniform cost. We used 10-fold cross validation for error rate experiments (Table 1). For the cost experiments of Table 2, 10-fold cross validations were performed on 10 different cost matrices for each data set. Costs were integer values between 1 and 10 assigned uniformly at random. Costs on the diagonal were set to zero. The average cost per test instance was reported for each experiment. Table 2 gives the average cost over all 100 experiments per data set, per algorithm.

Table 1: Error Rates. Naïve Bayes is our baseline classifier. L&F is our implementation of Lachiche & Flach (2003). MC is MetaClass (Algorithm 1). LP, GA and QP are the Relaxed Integer Linear Programming, Genetic Algorithms and Quadratic Programming formulations optimizing their respective objective functions. **Bold** font denotes a significant difference to the baseline with at least 95% confidence according to the Student's t method. The overall best classifier among all algorithms is underlined.

Data Set	n	Naïve Bayes	L&F	MC	LP Eq. (21)	GA Eq. (22)	GA Eq. (1)	QP
Audiology	24	0.3095	0.2784	**0.4816**	0.2869	**0.2826**	0.2872	0.2337
Bridges 2 (material)	3	0.1582	**0.2427**	**0.2436**	**0.2709**	0.1764	**0.2136**	**0.2436**
Bridges 2 (rel-l)	3	0.3164	0.3527	0.3336	0.3081	0.3355	0.3455	0.3073
Bridges 2 (span)	3	0.2227	**0.3190**	0.2245	0.2427	**0.4018**	n/a	0.2518
Bridges 2 (type)	6	0.4564	0.5418	0.4564	0.4654	0.4664	0.4655	0.4936
Bridges 2 (t-or-d)	2	0.1755	0.2327	0.1955	0.1754	0.1936	**0.2500**	0.1654
Car	4	0.1464	**0.1197**	**0.1267**	**0.1336**	0.1724	**0.1209**	**0.1331**
Post-Op	3	0.4949	0.4738	0.4881	0.4948	0.4990	0.4908	0.4853
Horse-colic (code)	3	0.3173	0.3202	0.3179	0.3172	0.2931	0.3120	0.3174
Horse-colic (surgical)	2	0.2089	**0.1709**	**0.1710**	0.2089	**0.1791**	**0.1791**	0.1980
Horse-colic (site)	63	0.7634	0.7716	0.7770	0.7634	0.7444	0.7661	0.7661
Horse-colic (subtype)	2	0.0027	0.0081	0.0027	0.0027	0.0027	0.0027	0.0027
Horse-colic (type)	8	0.0409	0.0488	0.0272	0.0409	0.0327	0.0354	0.0326
Credit	2	0.2490	0.2560	0.2570	0.2490	**0.2720**	0.2580	0.2530
Dermatology	6	0.0273	0.0218	0.0192	0.0273	0.0273	0.0165	0.0219
Ecoli	8	0.1516	0.1972	0.1608	0.1545	0.1640	**0.1368**	0.1637
Flags	8	0.3761	0.4063	0.4213	0.3707	0.3705	0.3808	0.3860
Glass	7	0.5236	**0.3790**	0.5472	0.5235	**0.4260**	0.4959	0.5008
Mushroom	2	0.0420	**0.0179**	**0.0181**	**0.0374**	**0.0192**	**0.0178**	**0.0272**
Nursery	5	0.0968	**0.0851**	**0.0849**	n/a	n/a	**0.0847**	n/a
Image Segmentation	7	0.1974	0.1567	**0.1485**	0.1974	**0.1260**	**0.1727**	0.2000
Solar Flare (common)	8	0.2364	**0.1726**	**0.1745**	0.2364	**0.1708**	**0.1980**	**0.2176**
Solar Flare (moderate)	6	0.0732	**0.0337**	**0.0356**	0.0731	**0.0338**	**0.0507**	**0.0497**
Solar Flare (severe)	3	0.0282	**0.0056**	**0.0085**	**0.0234**	**0.0047**	**0.0207**	**0.0141**
Vote	2	0.0965	**0.1081**	0.1081	0.0964	0.0942	0.0965	**0.1056**
Win/Loss		–	11/14	11/12	6/5	13/11	15/9	11/11
Significant Win/Loss		–	8/3	8/2	3/1	7/3	8/3	5/2

In both tables, for each data set, n denotes the number of classes. The first column is the performance of our baseline classifier. For comparison, we have included results of our implementation for the algorithm of Lachiche and Flach (2003) on this baseline classifier. The results of the experiments on our heuristics can be found in the last five columns of each table. Here, MC is the MetaClass algorithm (Algorithm 1). LP is a linear programming algorithm (MOSEK ApS, 2005) on (21) with $\eta = 10^{-6}$. The first GA is the sum of linear fractional functions formulation (22) using a genetic algorithm. The second GA was a direct optimization performed on (1). Both GA implementations were from Abramson (2005). Parameters for both used the default Matlab settings with a population size of 20, a maximum of 200 generations and a crossover fraction of 0.8. The algorithm terminates if no change is observed in 100 continuous rounds. In addition, the mutation function of Abramson (2005) is guaranteed to only generate feasible solutions (in our case, all weights must be nonnegative). Upon termination, a direct pattern search is performed using the best solution from the GA. The final column is the quadratic program (QP) as in Section 5.1.3. Data for some entries were not available and are denoted "n/a" (data sets were too large for Matlab). For all columns, **bold** entries indicate a significant difference to the baseline with at least a 95% confidence according to a Student's t method. The overall best classifier for each data set is underlined.

Table 2: Nonuniform Costs. Naïve Bayes is our baseline classifier. L&F is our implementation of Lachiche & Flach (2003). MC is the MetaClass (Algorithm 1). LP, GA and QP are the Relaxed Integer Linear Programming, Genetic Algorithms and Quadratic Programming formulations optimizing their respective objective functions. **Bold** font denotes a significant difference to the baseline with at least 95% confidence according to the Student's t method. The overall best classifier among all algorithms is underlined.

Data Set	n	Naïve Bayes	L&F	MC	LP Eq. (21)	GA Eq. (22)	GA Eq. (1)	QP
Audiology	24	1.7720	**2.0928**	**2.7935**	1.6386	1.7245	**1.5194**	1.6877
Bridges 2 (material)	3	0.8611	0.9753	**1.3542**	**1.4725**	1.0059	**1.2181**	**1.3585**
Bridges 2 (rel-l)	3	1.9355	1.9165	1.9441	1.9003	**2.5553**	1.9875	1.9163
Bridges 2 (span)	3	1.1872	1.3348	**1.6153**	1.2930	**1.9934**	**1.4491**	1.3508
Bridges 2 (type)	6	2.4585	2.5110	2.3160	2.6020	**2.7605**	2.5220	2.7270
Bridges 2 (t-or-d)	2	0.9655	0.9685	0.8946	0.9568	1.0794	0.9705	**0.8096**
Car	4	0.8484	**1.0699**	0.8898	0.8074	**1.6665**	**0.6523**	**0.7558**
Post-Op	3	2.9242	**2.7495**	3.0450	**2.9993**	**3.6611**	**2.7057**	**2.8282**
Horse-colic (code)	3	1.7915	1.7218	1.7243	1.7863	1.7950	1.6989	1.7726
Horse-colic (surgical)	2	1.3788	**1.1150**	**1.0060**	1.3890	**1.6364**	**1.0629**	**1.1576**
Horse-colic (site)	63	4.0892	3.9775	4.2372	4.1084	**4.2647**	4.0809	4.1560
Horse-colic (subtype)	2	0.0114	0.0114	0.5741	0.0114	0.0113	0.0114	n/a
Horse-colic (type)	8	0.2225	0.1395	0.1447	0.2172	0.1704	0.1970	0.1894
Credit	2	1.3203	**1.1357**	**1.0444**	**1.3951**	**2.1906**	**1.0531**	**1.0805**
Dermatology	6	0.1744	**0.5532**	0.1105	0.1564	0.1775	0.1147	0.1166
Ecoli	8	0.8105	**1.3223**	0.8514	0.8766	**1.2116**	0.7678	0.9218
Flags	8	2.1590	2.2796	2.2803	2.1408	2.3013	**1.9888**	2.1644
Glass	7	3.1308	**2.1955**	2.9330	3.1301	**3.4910**	**2.6720**	2.9601
Mushroom	2	0.1930	**0.1017**	**0.0994**	**0.1784**	**0.1262**	**0.1031**	**0.1382**
Nursery	5	0.5565	n/a	0.6651	n/a	n/a	**0.4634**	n/a
Image Segmentation	7	1.0855	**1.4075**	**0.8416**	1.0855	1.0989	**0.8952**	n/a
Solar Flare (common)	8	1.3080	**0.9506**	**4.1595**	**1.3199**	**1.1622**	**0.9844**	1.2516
Solar Flare (moderate)	6	0.4644	**0.2137**	**5.6085**	0.4628	**0.2749**	**0.2652**	**0.3030**
Solar Flare (severe)	3	0.1682	**0.0280**	**6.0877**	0.1556	**0.0800**	**0.1070**	**0.0838**
Vote	2	0.4510	0.4096	**0.3770**	0.4510	**0.5346**	0.4577	0.4605
Win/Loss		–	12/10	11/14	12/9	6/17	19/6	15/8
Significant Win/Loss		–	8/5	5/6	1/4	4/11	13/2	7/2

We observe that no one technique consistently produced the best performance on the most data sets. For this reason, the most relevant measure among these heuristics is the ratio of significant wins to significant losses rather than merely total wins or losses.

As far as classification error is concerned, with the exception of the linear programming relaxation, all algorithms were competitive with no clear overall winner in terms of significant wins and significant losses. However, every algorithm successfully showed substantial improvement over the baseline classifier. This confirms that the techniques presented serve as good reclassification methods in practice.

However, when we consider nonuniform costs, Table 2 indicates that the quadratic programming formulation and the genetic algorithm optimization on (1) outperform all other methods. In some sense, this reflects the inherent combinatorial nature of the problem, giving evidence that the objective function surface is likely to be very rough with many local minimums (it is certainly discontinuous given the use of the argmax function). This also may explain why other methods did not perform as well. The GA is searching globally; in contrast all other methods (including Lachiche and Flach (2003)) search locally. Even the integer linear programming relaxation, which in general has a good track record, came up short. Intuitively, the quadratic programming formulation should also suffer for these reasons, but fares better due to the advantages characteristic of margin maximization techniques.

7 Conclusions and Future Work

Reoptimizing an already-learned classifier f is an important problem in machine learning, particularly in applications where the cost model or class distribution of a learning problem deviates from the conditions under which a classifier f was trained. We answered the open problem concerning the hardness of this reoptimization problem, and presented an algorithm that produces an optimal solution in polynomial time when the number of classes is constant. We also presented multiple algorithms for the multi-class version of this problem and empirically showed their competitiveness. Direct optimization by a genetic algorithm and quadratic programming were particularly effective under nonuniform cost models.

Acknowledgments

The authors would like to thank Nicolas Lachiche for helpful correspondence. We also thank anonymous reviewers for helpful suggestions and feedback. This work was supported in part by NSF grants CCR-0325463, CCR-0092761 and CCF-0430991.

References

Abramson, M. A. (2005). Genetic algorithm and direct search toolbox. `http://www.mathworks.com/`.

Blake, C., & Merz, C. (2005). UCI repository of machine learning databases. `http://www.ics.uci.edu/ mlearn/MLRepository.html`.

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.

Ferri, C., Hernández-Orallo, J., & Salido, M. (2003). Volume under the ROC surface for multi-class problems. *European Conference on Artificial Intelligence* (pp. 108–120).

Fieldsend, J., & Everson, R. (2005). Formulation and comparison of multi-class ROC surfaces. *Proceedings of the ICML Workshop on ROC Analysis in Machine Learning* (pp. 41–48).

Hand, D., & Till, R. (2001). A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning*, *45*, 171–186.

Kohli, R., Krishnamurti, R., & Mirchandani, P. (1994). The minimum satisfiability problem. *SIAM Journal of Discrete Mathematics*, *7*, 275–283.

Lachiche, N., & Flach, P. (1999). 1BC: A first-order bayesian classifier. *Proceedings of the 9th International Workshop on Inductive Logic Programming* (pp. 92–103).

Lachiche, N., & Flach, P. (2003). Improving accuracy and cost of two-class and multi-class probabilistic classifiers using ROC curves. *Proceedings of the 20th International Conference on Machine Learning* (pp. 416–423).

Matsui, T. (1996). NP-hardness of linear multiplicative programming and related problems. *Journal of Global Optimization*, *9*, 113–119.

MOSEK ApS (2005). The MOSEK optimization tools version 3.2. `http://www.mosek.com/`.

Mossman, D. (1999). Three-way ROCs. *Medical Decision Making*, 78–89.

O'Brien, D. B., & Gray, R. M. (2005). Improving classification performance by exploring the role of cost matrices in partitioning the estimated class probability space. *Proceedings of the ICML Workshop on ROC Analysis in Machine Learning* (pp. 79–86).

Rockafellar, R. (1970). *Convex analysis (2nd ed.).* Princeton University Press.

Schölkopf, B., & Smola, A. (2002). *Learning with kernels (2nd ed.).* MIT Press.

Srinivasan, A. (1999). *Note on the location of optimal classifiers in n-dimensional ROC space* (Technical Report PRG-TR-2-99). Oxford University Computing Laboratory, Oxford.

Stoer, I. J., & Witzgall, C. (1996). *Convexity and optimization in finite dimensions.* Springer-Verlag.

Witten, I. H., et al. (2005). Weka machine learning toolbox. `www.cs.waikato.ac.nz/ml/weka/`.