New Algorithms for Optimizing Multi-Class Classifiers via ROC Surfaces

Kun Deng KDENG@CSE.UNL.EDU
Chris Bourke CBOURKE@CSE.UNL.EDU
Stephen Scott SSCOTT@CSE.UNL.EDU
N. V. Vinodchandran VINOD@CSE.UNL.EDU

Department of Computer Science & Engineering, University of Nebraska–Lincoln, Lincoln, NE 68588-0115 USA

Abstract

We study the problem of optimizing 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. We present several heuristics for this problem, including linear and nonlinear programming formulations, genetic algorithms, and a customized algorithm. Empirical results suggest that genetic algorithms fare the best overall, improving performance most often.

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 (say, a change in the ratio of true positive to false positive error). 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.

For two-class problems, the problem of finding the *optimal operating point* of a classifier given a ratio of true positive cost to false positive cost is straightforward via Receiver Operating Characteristic (ROC) analysis. ROC analysis takes a classifier h 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 $x \in \mathcal{X}$ by their confidences $h(x)$ from largest to smallest (denoted $\mathcal{X} = \{x_1, \ldots, x_n\}$), one achieves a set of $n+1$ binary classifiers by setting thresholds $\{\theta_i = (h(x_i) + h(x_{i+1}))/2, 1 \leq i < n\} \cup \{h(x_1) - \epsilon, h(x_n) + \epsilon\}$ for some constant $\epsilon > 0$. Given a relative cost c of true positive error to false positive error and a validation set $\mathcal{X}$ of labeled examples, one can easily find the optimal threshold θ based on $\mathcal{X}$ and c (Lachiche & Flach, 2003). To do so, simply rank the examples in $\mathcal{X}$, try every threshold θ_i as described above, and select the θ_i minimizing the total cost of all errors on $\mathcal{X}$.

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 m-class ROC space having dimension $m(m-1)$. A point in this space corresponds to a classifier, with each coordinate representing the misclassification rate of one class into some other class[1]. According to Srinivasan (1999), the optimal classifier lies on the convex hull of these points. Given this ROC polytope, a validation[2] set $\mathcal{X}$, and an $m \times m$ cost matrix M with entries $c(C_j, C_k)$ (the cost associated with misclassifying a class C_j example as class C_k), Lachiche and Flach (2003) define the optimization problem as find-

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

[2] Lachiche and Flach ran their experiments with $\mathcal{X}$ as an independent validation set and with $\mathcal{X}$ as the original training set. They found little difference in their experimental results.

Appearing in *Proceedings of the ICML 2006 workshop on ROC Analysis in Machine Learning*, Pittsburgh, USA, 2006.

ing a weight vector $\vec{w} \geq \vec{0}$ to minimize[3]

$$\sum_{1 \leq j \leq m} \sum_{1 \leq k \leq m} p(C_j)\, r(C_j, C_k)\, c(C_j, C_k) \ , \qquad (1)$$

where $p(C_j)$ is the prior probability of class C_j, and $r(C_j, C_k)$ is the proportion of examples from $\mathcal{X}$ of actual class C_j that are predicted as class C_k. The predicted class of an example x is

$$\hat{y}_x = \underset{1 \leq i \leq m}{\operatorname{argmax}} \{w_i f(x, C_i)\} \ ,$$

where $f(x, C_i)$ is the classifier's confidence that example x is in class C_i.

No efficient algorithm is known to optimally solve (1), and Lachiche and Flach speculate that the problem is computationally hard. We present several new algorithms for this problem, including an integer linear programming relaxation, a sum-of-linear fractional functions (SOLFF) formulation, a direct optimization of (1) with a genetic algorithm and finally, a new custom algorithm based on partitioning $\mathcal{C}$ into *meta-classes*. 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 and in Section 3 we discuss our approaches to this problem. We then experimentally evaluate our algorithms in Section 4 and conclude in Section 5.

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 m-class scenario. A natural extension from the binary case is to consider a multi-class ROC space as having dimension $m(m-1)$. 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 (2005) 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. We 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 (2005) 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 (2003) considered the case

[3] Assuming $c(C_j, C_j) = 0 \ \forall j$.

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 $n+1$ possible thresholds for n 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: $\Omega(m^n)$.

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

$$h(x) = \operatorname*{argmax}_{1 \le j \le m} \left\{ w_j f(x, C_j) \right\} \ .$$

It should be apparent that $\vec{w}$ has only $m-1$ degrees of freedom, so we can fix $w_1 = 1$.

Lachiche and Flach (2003) used a hill climbing or sequential optimization 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 $\mathcal{X}$ be the set of training examples and let p be the current class for which we want to assign a "good" weight w_p. Then the set of possible weights for w_p is

$$\left\{ \left. \frac{\max_{j \in \{1,\ldots,p-1\}} f(x, C_j)}{f(x, C_p)} \right| x \in \mathcal{X} \right\}.$$

It is not difficult to see that at any stage there are at most $\mathcal{O}(|\mathcal{X}|)$ 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 $\Theta(mn \log n)$. 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 in 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

$$h(x) = \operatorname*{argmin}_{1 \le k \le m} \left\{ \sum_{1 \le j \le m} c(C_j, C_k)\, \hat{p}(x, C_j) \right\} \ .$$

If $\hat{p}(x, C_j)$ is a good probability estimate of example x belonging to class C_j, this prediction results in the lowest expected cost. However, if $\hat{p}(x, C_j)$ 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 Lachiche and Flach's weight vector 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*).

3. Our Contributions

In this section, we first present new mathematical programming formulations. In particular, we reformulate the objective function (1) given by Lachiche and Flach (2003) as a relaxed integer linear program as well as give a formulation that is a sum of linear fractional functions. We also describe a new heuristic algorithm approach, MetaClass. Finally (in Section 4) we present experimental results from these formulations. We give evidence that the objective function landscape for this problem is highly discontinuous and thus more amenable to global optimization methods such as genetic algorithms.

3.1. Mathematical Programming Formulations

3.1.1. Relaxed Integer Linear Program

We start by reformulating (1) as follows:

$$\underset{\vec{w}, \vec{I}}{\text{minimize}} \left\{ \sum_{1 \le j \le m} \frac{p(C_j)}{|C_j|} \sum_{1 \le k \le m} c(C_j, C_k) \sum_{x_i \in C_j} I_{i,k} \right\} , \tag{2}$$

where $C_j \subseteq \mathcal{X}$ is the set of instances of class j, $p(C_j)$ is the prior probability of class C_j, $c(C_j, C_k)$ is the cost of misclassifying an example from class C_j as C_k, and

$$I_{i,k} = \begin{cases} 1 & \text{if } w_k f(x_i, C_k) \ge w_l f(x_i, C_l), l \ne k \\ 0 & \text{otherwise.} \end{cases}$$

Here we assume $c(C_j, C_j) = 0$ for all C_j. Formalizing this as a constrained optimization problem, we want to minimize (2) subject to

$$I_{i,j} w_j f(x_i, C_j) = I_{i,j} \max_{1 \le k \le m} \{w_k f(x_i, C_k)\} \quad (3)$$
$$\textstyle\sum_{1 \le j \le m} I_{i,j} = 1 \quad (4)$$
$$I_{i,j} \in \{0, 1\} \quad (5)$$
$$w_j \ge 0 \quad (6)$$

where each constraint holds for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. Equation (3) allows only the class that has the max value of $w_k f(x_i, C_k)$ to be indicated by $\vec{I}$ to be the predicted class of example x_i and (4) 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 (3) to "$\ge$". Second, we can relax (5) to be $I_{i,j} \in [0, 1]$. Note that (3) (even when amended with "$\ge$") will only be satisfied if $I_{i,j} = 0$ for all C_j that don't maximize the RHS of (3). Thus, so long as we never have $w_k f(x_i, C_k) = w_{k'} f(x_i, C_{k'})$ for some $k \ne k'$, the relaxation is equivalent to the original problem. Further, even if there is such a tie for classes C_k and $C_{k'}$, 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 $c(C_j, C_k) < c(C_j, C_{k'})$. The potential problem of both $w_k f(x_i, C_k) = w_{k'} f(x_i, C_{k'})$ and $c(C_j, C_k) = c(C_j, C_{k'})$ 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 (3) is linear. We now reformulate it. First, for each $i \in \{1, \ldots, n\}$, we substitute γ_i for $\max_{1 \le k \le m} \{w_k f(x_i, C_k)\}$:

$$I_{i,j} w_j f(x_i, C_j) \ge \gamma_i I_{i,j} \quad (7)$$
$$w_k f(x_i, C_k) \le \gamma_i \ , \quad (8)$$

for all $i \in \{1, \ldots, n\}$ and $j, k \in \{1, \ldots, m\}$ where each γ_i is a new variable. Obviously (8) is a linear constraint, but (7) 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 $f(x_i, C_k) \in (0, 1]$ (e.g. if $f(\cdot, \cdot)$ are probability estimates from naïve Bayes or logistic regression). Now we can optimize (2) subject to:

$$\gamma_i - w_j f(x_i, C_j) + I_{i,j} \le 1 \quad (9)$$
$$\gamma_i \ge w_j f(x_i, C_j) \quad (10)$$
$$\textstyle\sum_{1 \le j \le m} I_{i,j} = 1 \quad (11)$$
$$I_{i,j} \in \{0, 1\} \quad (12)$$
$$w_j \ge 0 \quad (13)$$

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

So long as $w_j f(x_i, C_j) \in (0, 1]$ 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 (12) 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 (2):

$$\underset{\vec{w}, \vec{I}}{\text{minimize}} \left\{ \sum_{1 \le j \le m} \frac{p(C_j)}{|C_j|} \sum_{1 \le k \le m} c(C_j, C_k) \sum_{x_i \in C_j} I_{i,k} + \eta \|\vec{w} - \vec{1}\|_1 \right\} \quad (14)$$

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 $f(\cdot, \cdot)$.

3.1.2. Sum of Linear Fractional Function 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 $w_k f(x, C_k)$ for each $k \in \{1, \ldots, m\}$. Then predict class C_k for example x with probability

$$\frac{w_k f(x, C_k)}{\sum_{1 \le \ell \le m} w_\ell f(x, C_\ell)} \ .$$

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

$$\sum_{1 \le j \le m} \frac{p(C_j)}{|C_j|} \sum_{1 \le k \le m} c(C_j, C_k) \sum_{x_i \in C_j} \varphi(i, k) \ , \quad (15)$$

where

$$\varphi(i, k) = \frac{w_k f(x_i, C_k)}{\sum_{1 \le \ell \le m} w_\ell f(x_i, C_\ell)}$$

subject to $w_k \ge 0$ for all $k \in \{1, \ldots, m\}$. We now have eliminated the variables $I_{i,k}$ and their integer constraints. However, we now have a nonlinear objective function in (15). Each individual term of the summation of (15) 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 (they either restrict to few terms in the summation or to low-dimensional vectors). Instead, we apply a genetic algorithm to directly optimize (15).

3.2. The MetaClass Heuristic Algorithm

In addition to the linear programming formulations, we 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 f_1 and f_2 for each instance, which are simply the sum of the confidences of the classes in $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. The ratio $F = \frac{f_1}{f_2}$ is used to find a threshold θ. We find θ by sorting the instances according to F and choose a threshold that minimizes error. (This threshold will be the average of $F(x_i)$ and $F(x_{i+1})$ for some instance x_i.) The situation for cost is slightly more complicated since it is not clear *which* class in the metaclass an example is misclassified as. Instead, we use the *average* cost of misclassifying instances into metaclasses in $\mathcal{C}_1$ and $\mathcal{C}_2$. In this case, a threshold is chosen that minimizes the overall cost. We recursively perform this procedure on the two metaclasses until there is only a single class, at which point a leaf is formed. The MetaClass algorithm is presented as Algorithm 1.

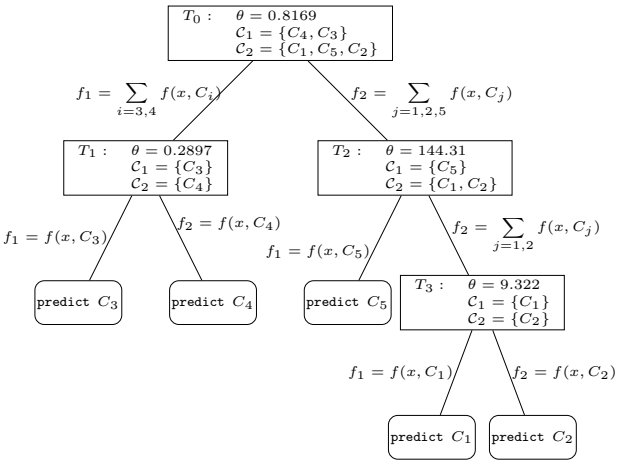

Figure 1. Example run of MetaClass on Nursery, a 5-class problem.

Input : A set of instances, $\mathcal{X} = \{x_1, \ldots, x_n\}$; a set of classes, $\mathcal{C} = \{C_1, \ldots, C_m\}$; a learned confidence function $f : \mathcal{X} \times \mathcal{C} \to \mathbb{R}^+$ and a tree node T

Output : A decision tree with associated weights.

Split $\mathcal{C}$ into two meta-classes $\mathcal{C}_1, \mathcal{C}_2$
foreach *Instance* $x_i \in \mathcal{X}$ **do**
  $f_1(x_i, \mathcal{C}_1) = \sum_{C_j \in \mathcal{C}_1} f(x_i, C_j)$
  $f_2(x_i, \mathcal{C}_2) = \sum_{C_j \in \mathcal{C}_2} f(x_i, C_j)$
  $F(x_i) = f_1(x_i)/f_2(x_i)$
end
Sort instances according to F
Compute a threshold θ that minimizes error/cost w.r.t. F
Label T with $\theta, \mathcal{C}_1, \mathcal{C}_2$
Create two children of T: $T_{\text{left}}, T_{\text{right}}$
Split $\mathcal{X}$ into two classes, $\mathcal{X}_1, \mathcal{X}_2$ according to $\mathcal{C}_1, \mathcal{C}_2$
Recursively perform this procedure on $\mathcal{X}_1, \mathcal{C}_1, T_{\text{left}}$ and $\mathcal{X}_2, \mathcal{C}_2, T_{\text{right}}$ 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 $\mathcal{C}_1 = \{C_4, C_3\}$ as an optimal weight.

Predictions for a new example y 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 y with respect to each associated metaclass. We traverse left or right down the tree depending on whether $f_1/f_2 \geq \theta$. When a leaf is reached, a final class prediction is made.

The number of nodes created by MetaClass is $\Theta(m)$, where m is the number of classes. At each node, the most complex step is sorting at most n instances according to the confidence ratio. Thus, the overall performance is bounded by $\Theta(n \log n \log m)$. Since for most applications, $n \gg m$, we may consider its actual running time to simply be $\mathcal{O}(n \log n)$. 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

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

which is linear in the number of classes: $\Theta(m)$.

4. 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. We ran experiments evaluating improvements both in classification accuracy and total 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.

In both tables, for each data set m denotes the number of classes and NB indicates our baseline classifier's performance. For comparison, we have included wins and losses (and significance) for the algorithms reported by Lachiche and Flach (2003) and O'Brien and Gray (2005). Raw numbers are omitted since these results are not directly comparable to ours: in addition to being based on different data partitions, the results from Lachiche and Flach (2003) were from an optimization run on the base classifier 1BC (Lachiche & Flach, 1999). Moreover, the results in O'Brien and Gray (2005) (here, we have used one of their best formulations, "column multiply") pruned classes that did not have "sufficient" representation. Furthermore, Lachiche and Flach (2003) did not consider cost-sensitive experiments. Thus, the results in Table 2 for Lachiche & Flach are taken from the implementation and results reported by O'Brien and Gray.

The results of our experiments can be found in the last four columns of each table. Here, MC is the MetaClass algorithm (Algorithm 1). LP is a linear programming algorithm (MOSEK ApS, 2005) on Equation (14) with $\eta = 10^{-6}$. The first GA is the Sum of Linear Fractional Functions formulation (Equation (15)) using a genetic algorithm. The final column is a genetic algorithm performed on Equation (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.

Data for some entries were not available and are denoted "n/a" (either the source did not report results or, in the case of our experiments, data sets were too large for Matlab). Therefore, for comparison it is important to note the *ratio* of significant wins to significant losses rather than merely total wins or losses. 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.

Regarding classification error, in every case each algorithm showed some significant performance improvements. With the exception of LPR, all algorithms were competitive with no clear overall winner. However, Table 3.2 does, in fact, show a clear winner when costs are non-uniform. The success when using a GA on Equation (1) gives 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.

5. Conclusion & Future Work

When the cost model or class distribution of a learning problem deviates from the conditions under which a classifier f was trained, one may wish to re-optimize f. For two-class problems, it is well-known how to do

this via ROC analysis, but the multi-class problem is more challenging. We presented multiple algorithms for the multi-class version of this problem and empirically showed their competitiveness. Direct optimization by a genetic algorithm was particularly effective.

Future work includes answering the question posed by Lachiche and Flach (2003): is this optimization problem computationally intractable? Assuming it is, then a more tractable and useful special case of this problem may be when the number of classes is restricted to a constant. In particular, can we find a provably optimal algorithm when the number of classes is 3?

Acknowledgments

The authors would like to thank Nicolas Lachiche for helpful correspondence. We also thank anonymous reviewers for helpful suggestions and feedback.

References

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

Bengio, S., Mariéthoz, J., & Keller, M. (2005). The expected performance curve. *International Conference on Machine Learning, ICML, Workshop on ROC Analysis in Machine Learning* (pp. 9–16).

Blake, C., & Merz, C. (2005). UCI repository of machine learning databases.

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. *ECAI 2003* (pp. 108–120).

Fieldsend, J., & Everson, R. (2005). Formulation and comparison of multi-class ROC surfaces. *International Conference on Machine Learning, 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.

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. See `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. *International Conference on Machine Learning, ICML, Workshop on ROC Analysis in Machine Learning* (pp. 79–86).

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.

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