A Multi-class Boosting Method with Direct Optimization

Shaodan Zhai, Tian Xia, and Shaojun Wang
Kno.e.sis Center
Department of Computer Science and Engineering
Wright State University
{zhai.6,xia.7,shaojun.wang}@wright.edu

ABSTRACT

We present a direct multi-class boosting (DMCBoost) method for classification with the following properties: (i) instead of reducing the multi-class classification task to a set of binary classification tasks, DMCBoost *directly* solves the multi-class classification problem, and only requires very weak base classifiers; (ii) DMCBoost builds an ensemble classifier by *directly* optimizing the non-convex performance measures, including the empirical classification error and margin functions, without resorting to any upper bounds or approximations. As a non-convex optimization method, DMCBoost shows competitive or better results than state-of-the-art convex relaxation boosting methods, and it performs especially well on the noisy cases.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Learning

Keywords

Classification; supervised learning; boosting; direct optimization; noise tolerance

1. INTRODUCTION

Boosting is a machine learning and data mining approach [34] that combines a set of weak classifiers to produce a single strong classifier. Most boosting methods were designed for binary classification tasks, while many real-world applications involve multiple classes, such as handwritten digit recognition, image segmentation, and automatic speech recognition. To effectively extend well-studied binary boosting algorithms to solve multi-class problems is still an ongoing research topic.

Multi-class boosting methods can be roughly divided into two categories. The first is to reduce the multi-class problem to multiple binary classification problems. Methods in this category include "one-vs-all", "all-vs-all", and other general output coding based approaches [2, 8, 10, 12, 13, 17, 18, 27]. Binary classification is well-studied, but there are some problems with the binary reduction, including (i) it may produce imbalanced data distributions, which are known to have a negative effect on the classifier performance [30, 16], (ii) a lack of guarantees of an optimal joint predictor [25], or (iii) using binary boosting scores that do not represent true class probabilities [21].

The second category is to build a multi-class classifier directly by using multi-class base classifiers, such as decision trees. Boosting methods of this category include AdaBoost.M1, SAMME, AdaBoost.MM, GD-MCBoost, et al. [10, 22, 25, 36, 37]. Usually, these methods require strong base classifiers which substantially increase complexity and have a high potential for overfitting [25]. Moreover, all of these methods formulate multi-class tasks as a convex optimization problem by using surrogates, and it has been shown that all boosting algorithms based on convex optimization are susceptible to random classification noise [19]. In addition, none are designed to directly maximize the multi-class margin, although some of them have the effects of margin enforcing.

In this paper, we introduce a new direct multi-class boosting algorithm named DMCBoost that extends the work of DirectBoost in [35] from binary classification to multi-class classification. DMCBoost uses multi-class decision trees as base classifiers to build an ensemble classifier by directly optimizing the performance measures, without reducing them to binary classification problems. The process of DMCBoost includes two phases: it first directly minimizes the empirical classification error by iteratively adding base classifiers to the ensemble classifier. Once the classification error reaches a coordinatewise local minimum[1], it continuously adds base classifiers by directly maximizing the average margin of a certain set of bottom samples.

Both DMCBoost and DirectBoost [35] can be viewed as a coordinate optimization in the hypothesis space, and in each iteration only one coordinate is chosen and the corresponding parameter is computed by line search approaches. However, DMCBoost is a non-trivial extension of DirectBoost in [35] to multi-class classification, where we have to mathematically re-formulate the multi-class classification problem and identify the scenarios that lead to efficient computation of the empirical error of a weak classifier in the first phase, and identify the scenarios that lead to efficient computation of the margin curve of a weak classifier in the second phase. Thus DMCBoost uses very different optimization

[1]See the definition of coordinatewise minimum/maximum on page 479 in [31].

KDD'14, August 24–27, 2014, New York, NY, USA.

http://dx.doi.org/10.1145/2623330.2623689.

techniques: since the objectives are more complex and more difficult to optimize, we propose new efficient line search algorithms that can find the parameter with the optimal objective value along one coordinate; moreover, constructing decision trees for DMCBoost is more challenging, while it is straightforward in [35].

DMCBoost is a non-convex optimization method. In general, non-convex problems are very difficult to solve. However, for the special 0-1 loss minimization and margin maximization problems, we propose efficient algorithms to find a local optima. In many real applications, it is possible that the local optima of non-convex optimization is less serious than the inconsistencies [20]. The computational cost of DMCBoost is K times larger than the computational cost of AdaBoost.M1 and SAMME, where K is the number of classes. Nevertheless, we will show that DMCBoost performs better than the convex relaxation algorithms such as AdaBoost.M1, SAMME, AdaBoost.MH, and GD-MCBoost in terms of accuracy under a bearable time limitation on a number of UCI datasets and it is more robust in noisy cases. Furthemore, DMCBoost only requires very weak base classifiers, and it is more efficient in driving down the empirical classification error than other multi-class boosting algorithms when the same depth trees are used as weak learners, as shown in our experimental results.

2. DMCBOOST ALGORITHM

In a multi-class classification, we want to predict the labels of examples lying in an instance space $\mathcal{X}$. Let $\mathcal{D}$ denote a distribution over $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{Y} = \{1, \cdots, K\}$ be the set of all the labels. We are provided a training set of labeled examples $\mathcal{S} = \{(x_1, y_1), \cdots, (x_n, y_n)\}$, where each example $x_i \in \mathcal{X}$ has a unique label y_i in the set $\mathcal{Y}$. Denoting $\mathcal{H} = \{h_1, ..., h_{|\mathcal{H}|}\}$ as the set of all possible weak classifiers that can be produced by the weak learning algorithm, where a weak classifier $h_j \in \mathcal{H}$ is a mapping from an instance space $\mathcal{X}$ to $\mathcal{Y}$.

Boosting combines weak classifiers to form a highly accurate ensemble classifier for multi-class classification by making a prediction according to the weighted plurality vote of the classifiers:

$$F(x) = \arg\max_{y \in \{1,\cdots,K\}} f(x, y), \quad (1)$$

where $f(x, y) = \sum_{h \in \mathcal{H}} \alpha_h 1(h(x) = y), \alpha_h \in \mathbb{R}$ is the inference function which is used to infer the predicted label, and $1(\cdot)$ is the indicator function. Our goal is to find an ensemble classifier F that generalizes well on any unseen dataset in $\mathcal{X}$. To this end, we design a boosting approach through the following two phases: we first directly minimize the 0-1 loss on training data, and then directly maximize the average margin of a certain set of bottom samples. Phase I, which is very efficient in minimizing the empirical 0-1 loss, serves as an initialization method of phase II. The motivation is that phase II often performs better when it starts with a low training error. In phase II, a margin objective is optimized, which leads to a further improvement of generalization.

2.1 Phase I: Minimizing Multi-class Classification Errors

In multi-class classification, the empirical error is given by

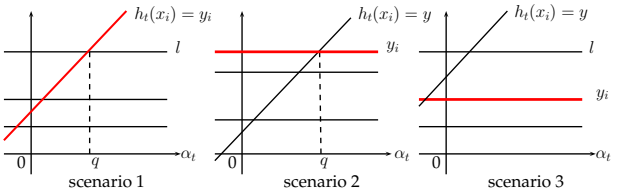

Figure 1: Three scenarios to compute the empirical error of a weak learner h_t over an example pair (x_i, y_i), where l denotes the incorrect label with the highest score, and q denotes the intersection point that results in an empirical error change. The red bold line for each scenario represents the inference function of example x_i and its true label y_i.

Algorithm 1 0-1 loss minimization algorithm.

1: **Initialize:** $t = 0$
2: **repeat**
3: $t \leftarrow t + 1$.
4: Select a weak classifier h_t by Algorithm 1.2.
5: Get the interval that has the minimum classification error by calling Algorithm 1.1 with h_t, and let α_t be the value within this interval.
6: Update: $f_t(x_i, y) = f_{t-1}(x_i, y) + \alpha_t 1(h_t(x_i) = y)$.
7: **until** the training error reaches the local coordinatewise minimum.
8: **Output:** $f_t(x_i, y)$.

$$\text{error}(F, \mathcal{S}) = \frac{1}{n} \sum_{i=1}^{n} 1(F(x_i) \neq y_i) \quad (2)$$

Due to the nonconvexity, nondifferentiability, and discontinuity of the classification error function (2), many previous multi-class boosting algorithms optimize the convex upper bounds of (2). While the convex surrogate losses are computationally efficient to globally optimize [3], they are sensitive to outliers [19, 23] and inconsistent under some conditions [20]. In contrast, our approach is to directly optimize 0-1 loss (2).

We use a greedy coordinate descent algorithm to directly minimize the empirical error (2) by constructing an ensemble classifier. Consider the tth iteration, the inference function is $f_t(x, y) = \sum_{k=1}^{t} \alpha_k 1(h_k(x) = y)$, $\forall y \in \mathcal{Y}$, where previous $t-1$ weak classifiers $h_k(x)$ and corresponding weights α_k, $k = 1, \cdots, t-1$ have been selected and determined, and our goal is to select a weak classifier h_t and its weight α_t such that (2) is minimized. Algorithm 1 outlines the greedy coordinate descent algorithm that sequentially minimizes 0-1 loss of (2), we will introduce the line search algorithm and the weak learning algorithm later. On each round, we first select a weak classifier h_t by Algorithm 1.2 (line 4), then the 0-1 loss (2) is a stepwise function w.r.t α_t. Next we can compute the interval that has the minimum classification error by using the line search algorithm (Algorithm 1.1) along the h_t coordinate. Since any value of α_t within this interval will lead to the largest error reduction, we can simply choose the middle point of the interval (line 5). In the end of each iteration, the inference function is updated (line 6). We repeat this procedure until the training error reaches a local coordinatewise minimum (line 7).

Algorithm 1.1 Line search algorithm to find the interval with the minimum 0-1 loss.

1: **Input:** a weak classifier $h_t \in \mathcal{H}$.
2: Let $\mathbf{e} : \mathbb{R} \to \mathbb{Q}$. ▷ $\mathbf{e}$ map intersection points to error updates.
3: **for** $i = 1, \cdots, n$ **do**
4: **if** $h_t(x_i) = y_i$ **then** ▷ scenario 1
5: $q = a(x_i, l) - a(x_i, y_i)$.
6: $error\_update = -1$.
7: **else if** $h_t(x_i) = y$, $y \neq y_i$, and $a(x_i, y_i) > a(x_i, y)$ **then** ▷ scenario 2
8: $q = a(x_i, y) - a(x_i, y_i)$.
9: $error\_update = 1$.
10: **end if**
11: $\mathbf{e}[q] = \mathbf{e}[q] + error\_update$. ▷ $\mathbf{e}[q] = 0$ if q is not in $\mathbf{e}$
12: **end for**
13: Sort $\mathbf{e}$ by the *keys* in an increasing order.
14: Incrementally calculate classification error on each interval.
15: **Output:** the interval with minimum 0-1 loss.

Algorithm 1.2 Constructing tree algorithm.

1: **Input:** a training set $\mathcal{S}$, current tree depth dep.
2: Let $dep \leftarrow dep + 1$.
3: **if** $dep \leq max\_dep$ **then**
4: **for** a binary split **do**
5: Split $\mathcal{S}$ into $\mathcal{S}_{left}$ and $\mathcal{S}_{right}$.
6: **if** $|\mathcal{S}_{left}| = 0$ or $|\mathcal{S}_{right}| = 0$ **then** continue.
7: $\ell_{left} = \arg\min_{\ell \in 1, \cdots, K}$ {0-1 loss that compute by Algorithm 1.1, while setting the input $h_t(x) = \ell$ if $x \in \mathcal{S}_{left}$}.
8: Let $h_t(x) = \ell_{left}$ if $x \in \mathcal{S}_{left}$.
9: $\ell_{right} = \arg\min_{\ell \in 1, \cdots, K}$ {0-1 loss that compute by Algorithm 1.1, while setting the input $h_t(x) = \ell$ if $x \in \mathcal{S}_{right}$}.
10: Let $h_t(x) = \ell_{right}$ if $x \in \mathcal{S}_{right}$.
11: **end for**
12: **end if**
13: Choose the optimal binary split which splits $\mathcal{S}$ into $\mathcal{S}^*_{left}$ and $\mathcal{S}^*_{right}$ with the corresponding ℓ^*_{left} and ℓ^*_{right}.
14: Let $h_t(x) = \ell^*_{left}$ if $x \in \mathcal{S}^*_{left}$, and $h_t(x) = \ell^*_{right}$ if $x \in \mathcal{S}^*_{right}$.
15: Call constructing tree algorithm with input $\mathcal{S}^*_{left}$ and $dep + 1$.
16: Call constructing tree algorithm with input $\mathcal{S}^*_{right}$ and $dep + 1$.
17: **Output:** a weak classifier $h_t \in \mathcal{H}$.

The line search algorithm describes how to find the optimal value of α for any given hypothesis $h \in \mathcal{H}$ such that (2) is minimized. The key idea is how to find the points that lead the 0-1 loss changes efficiently. Let $a(x_i, y) = \sum_{k=1}^{t-1} \alpha_k 1(h_k(x) = y)$, then let the inference functions for example x_i be

$$f_t(x_i, y) = a(x_i, y) + \alpha_t 1(h_t(x) = y), \tag{3}$$

which is a linear function of α_t with intercept $a(x_i, y)$ and slope $1(h_k(x) = y)$. Obviously, the inference function is either a line with slope 1 or a horizontal line. The inference functions are used to compute the empirical error (2). More specifically, given a weak learner $h_t \in \mathcal{H}$, for each example pair (x_i, y_i), there are 3 scenarios to compute the empirical error, see Figure 1. Scenario 1 is the case that $h_t(x_i) = y_i$. $f_t(x_i, y_i)$ is a line with slope 1, and assume that $l = \arg\max_{y \in \mathcal{Y}, y \neq y_i} a(x_i, y)$, then $f_t(x_i, l)$ is a line with slope 0. The intersection of $f_t(x_i, y_i)$ and $f_t(x_i, l)$ is at $\alpha_t = a(x_i, l) - a(x_i, y_i)$. Thus when α_t is set on the left side of the intersection point, there is an error for example x_i, otherwise there is no error. Scenario 2 is the case that $h_t(x_i) = y$, $y \neq y_i$, and $a(x_i, y_i) > a(x_i, y)$ $\forall y \in \mathcal{Y}$, $y \neq y_i$. Then $f_t(x_i, y)$ is a line with slope 1, and $f_t(x_i, y_i)$ is a line with slope 0. The intersection point of $f_t(x_i, y)$ and $f_t(x_i, y_i)$ is at $\alpha_t = a(x_i, y) - a(x_i, y_i)$. Thus when α_t is set on the right side of the intersection point, there is an error for example x_i, otherwise there is no error. Scenario 3 is the case that $h_t(x_i) = y$, and $y \neq y_i$, and $\exists l \in \mathcal{Y}$, $l \neq y_i$ such that $a(x_i, l) > a(x_i, y_i)$, in this case there is always an error no matter what value α_t has.

Formally, Algorithm 1.1 describes the line search procedure. We use $\mathbf{e}$ (bold letter denotes a vector valued function or variable) to record all the intersection points and their corresponding error updates on the right-hand side (line 2). More specifically, for each training example, we first categorize it into the three scenarios. For an example in Scenario 1, the intersection point is at $q = a(x_i, l) - a(x_i, y_i)$, and the error update on the right-hand side of q is -1 (line 4-6). In Scenario 2, the intersection point is at $q = a(x_i, y) - a(x_i, y_i)$ and the error update on the right-hand side of q is 1 (line 7-9). We add the intersection points as the *keys* and their corresponding error updates as the *values* into $\mathbf{e}$ (line 11). We only care about the examples in Scenario 1 and 2 since the examples in Scenario 3 do not lead to an error update no matter what value α_t has. Once all the intersection points are added into $\mathbf{e}$, we sort $\mathbf{e}$ by the *keys* in an increasing order (line 13). These intersections divide the coordinate to (at most) $|\mathbf{e}| + 1$ intervals, the classification error on each interval can be incrementally calculated by the *values* of $\mathbf{e}$ (line 14), and hence the interval which gives the minimum error is easy to obtain.

The weak learning algorithm is described in Algorithm 1.2. Here we only consider the decision trees algorithm with binary splits. The binary splits are preferred [15] since (i) multiway splits fragment the data too quickly, leaving insufficient data at the next level down; and (ii) multiway splits can be achieved by a series of binary splits. For a binary splitting node that splits the training examples into two subsets, we denote them as $\mathcal{S}_{left}$ and $\mathcal{S}_{right}$ (line 5). We first enumerate all the possible labels from 1 to K on $\mathcal{S}_{left}$, that produce K hypotheses that belong to $\mathcal{H}$. We choose the optimal label which leads to the minimum value of (2) by running the line search algorithm (Algorithm 1.1) with these K hypotheses (line 7). We then fix the selected label for $\mathcal{S}_{left}$ (line 8), and apply the same process on $\mathcal{S}_{right}$ (line 9). We simply choose the attribute to split by minimizing the 0-1 loss (line 13), and use a top-down, greedy search approach to build trees. If the problem involves real-valued variables, they are first binned into intervals, each interval being treated as an ordinal attribute. Note that since the historical information $f_{t-1}(x_i, y_i)$ is used in Algorithm 1.1 through building trees, Algorithm 1.2 will not end up with the same tree on each iteration t though DMCBoost does not maintain a distribution over training samples.

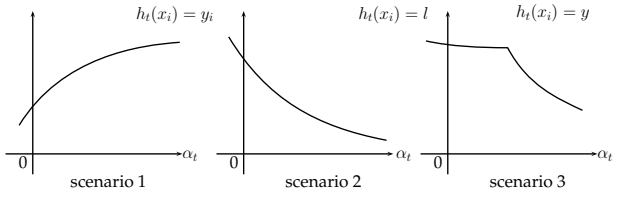

Figure 2: Three scenarios of margin curve of a weak learner h_t over an example pair (x_i, y_i), where l denotes the incorrect label with the highest score.

The computational cost of Algorithm 1 is $O(nMK)$ on each round when decision stumps[2] are used as weak learners, where M is the number of binary splits. It has the same computational cost as AdaBoost.MH, and is K times larger than the computational costs of AdaBoost.M1 and SAMME. Algorithm 1 may trap a coordinatewise local minimum of 0-1 loss. Nevertheless, we switch to the algorithm that directly maximizes various margins.

2.2 Phase II: Maximizing a Margin Objective

For boosting, the minimization of the training error is only one side of the story. To explain why boosting works, Schapire et al. [26] introduced the margin theory, which suggested that boosting is especially effective at maximizing the margins of training data. In multi-class classification, the most direct generalization of the margin is simply the difference between the score (weighted fraction of votes) obtained by the correct label and the score of the highest scoring incorrect label. We denote the (normalized) margin of an example (x_i, y_i) with respect to an inference function $f_t(x_i, y) = \sum_{k=1}^{t} \alpha_k 1(h_k(x_i) = y)$, $\forall y \in \mathcal{Y}$ by $\mathcal{M}_i$. Formally,

$$\mathcal{M}_i = \frac{f_t(x_i, y_i)}{\sum_{k=1}^{t} |\alpha_k|} - \max_{y \in \mathcal{Y}, y \neq y_i} \frac{f_t(x_i, y)}{\sum_{k=1}^{t} |\alpha_k|} \tag{4}$$

This definition of margin (4) is given in some earlier studies [2, 26]. A large margin implies the ensemble classifier confidently classifies the corresponding training sample.

While boosting's success can be ascribed to maximizing the margins, most boosting methods were not designed to specially optimize any margin functions [28]. Some exceptions, such as LPBoost [7], SoftBoost [33], and DirectBoost [35], explicitly maximize a relaxed minimum margin objective, but they are designed for binary classification problems. For the well-known multi-class boosting algorithms AdaBoost.M1 [10], SAMME [36], and AdaBoost.MM [22], none of them has been shown to maximize the multi-class margin [25]. The recently proposed algorithms CD-MCBoost and GD-MCBoost [25] optimize a margin enforcing loss function, but actually this objective is not related to the margin in the sense that one can minimize the loss function while simultaneously achieving a bad margin even for binary problems [24]. In this section, we introduce a coordinate ascent algorithm that directly maximizes the predefined margin objective functions for multi-class classification.

[2]Decision stumps are the special decision trees with a depth of 1. When more powerful trees are used, the complexity of Algorithm 1 has the same increasing rate as other boosting algorithms.

We first introduce the objective function that we are working on in this section. We can sort $\mathcal{M}_i$ in an increasing order, and consider n' worst training examples $n' \leq n$ that have smaller margins, then define the average margin over those n' labeled examples by $g_{avg\ n'}$. Formally,

$$g_{avg\ n'} = \frac{1}{n'} \sum_{i \in B_{n'}} \mathcal{M}_i \tag{5}$$

where $B_{n'}$ denotes the set of n' labeled examples having the smallest margins. The minimum margin (hard margin) $g_{min} = \min_{i \in \{1, \cdots, n\}} \mathcal{M}_i$ and average margin $g_{avg} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{M}_i$ are special cases for $n' = 1$ and $n' = n$ respectively. The parameter n' indicates how much we relax the hard margin on training examples, and we set n' based on knowledge of the number of noise examples in training data. The higher the noise rate, the larger the n' that should be used.

The following theorem shows the objective function (5) is equivalent to the soft margin, this conclusion is also given in [29] for binary classification, here we propose a general proof for multi-class classification and use different proof skills.

THEOREM 1. *Maximizing the average margin of the bottom n' examples (5) is equivalent to solving the soft margin maximization problem*

$$\begin{aligned} \max_{\boldsymbol{\alpha} \in \mathbb{R}^+, \boldsymbol{\xi} \in \mathbb{R}^+, \rho \in \mathbb{R}} & \rho - \tfrac{1}{n'} \textstyle\sum_{i=1}^{n} \xi_i \\ s.t. \quad & \mathcal{M}_i + \xi_i \geq \rho,\ i = 1, \cdots, n \end{aligned} \tag{6}$$

where $\boldsymbol{\xi}$ are slack variables.

Due to the max operator in the definition of margin (4), the soft margin optimization problem (6) is very difficult to solve, unlike its binary counterpart. To the best of our knowledge, there is no multi-class boosting designed to solve this soft margin optimization problem. Thus, one of the reasons for using (5) as objective is to solve the soft margin problem but from another point of view.

Another motivation of optimizing (5) is that the average of the bottom n' margins can be used to measure the generalization performance of a combined classifier, as shown in the following theorem.

THEOREM 2. *If the set of possible base classifiers $\mathcal{H}$ is finite, and $\theta = \frac{1}{n'} \sum_{i \in \mathcal{B}_{\{n'|f\}}} \mathcal{M}_i > \frac{8}{|\mathcal{H}|}$, then for any $\delta > 0$ and $n' \in \{1, \cdots, n\}$, with the probability at least $1 - \delta$ over the random choice of the training set $\mathcal{S}$ with size n, each inference function $f \in \mathcal{C}(\mathcal{H})$ satisfies the following bound:*

$$\Pr_{\mathcal{D}}[\mathcal{M}(f, x, y) < 0] \leq (K-1)\frac{\log |\mathcal{H}|}{2n} + \sqrt{\frac{\log |\mathcal{H}|}{2n^2}} + D^{-1}(\frac{n'-1}{n}, \frac{1}{n}u)$$

where $\Pr_{\mathcal{D}}[\mathcal{M}(f, x, y) < 0]$ denotes the generalization error of the ensemble classifier F, $\mathcal{M}(f, x, y)$ denotes the margin of an example (x, y) associated with f, $D^{-1}(v_1, v_2)$ denotes the inverse of K-L divergence between v_1 and v_2 when v_1 is fixed [32], and

$$u = \frac{8}{\theta^2} \log \frac{2n^2}{\log |\mathcal{H}|} \log 2|\mathcal{H}| + \log |\mathcal{H}| + \log \frac{1}{\delta}$$

The outline of the greedy coordinate ascent algorithm that sequentially maximizes the average margin of bottom n' examples is described in Algorithm 2. Similar to the 0-1 loss minimization algorithm, we intend to select a weak classifier

Algorithm 2 Margin maximization algorithm.

1: **Initialize:** t and $\boldsymbol{\alpha}$ from 0-1 loss minimization algorithm.
2: **repeat**
3: $t \leftarrow t+1$.
4: Select a weak learner h_t by weak learning algorithm.
5: Compute q^* by Algorithm 2.1 which maximizes (5) along the coordinate h_t. Set $\alpha_t = q^*$.
6: Update:$f_t(x_i, y) = f_{t-1}(x_i, y) + \alpha_t 1(h_t(x_i) = y)$
7: **until** the average margin of bottom n' examples reaches local coordinatewise maximum.
8: **Output:** $f_t(x_i, y)$.

h_t (line 4) and its weight α_t (line 5) on round t, but this time our target is maximizing (5). This procedure terminates if there is no increment in the average margin over the bottom n' examples over h_t (line 8). Its convergence can be proved in the same way as for the binary classification given in [35].

The key step is the line search algorithm which finds the value of α_t that maximizes (5) for a given weak classifier $h_t \in \mathcal{H}$. At tth iteration, let $c = \sum_{k=1}^{t-1} |\alpha_k|$, then the margin on the example (x_i, y_i) can be rewritten as,

$$\begin{aligned} \mathcal{M}_i &= \frac{a(x_i, y_i) + \alpha_t 1(h_t(x_i) = y_i)}{c + |\alpha_t|} \\ &\quad - \max_{y \in \mathcal{Y}, y \neq y_i} \frac{a(x_i, y) + \alpha_t 1(h_t(x_i) = y)}{c + |\alpha_t|} \end{aligned} \quad (7)$$

Consider the case that $\alpha_t \geq 0$. For each example pair (x_i, y_i), there are three scenarios of the margin (7) to consider, as shown in Figure 2. Scenario 1 is the case that $h_t(x_i) = y_i$, and assume that $l = \arg\max_{y \in \mathcal{Y}, y \neq y_i} a(x_i, y)$, then $\mathcal{M}_i = \frac{a(x_i,y_i) - a(x_i,l) + \alpha_t}{c + \alpha_t}$. This corresponds to the curve which is monotonically increasing in Figure 2. Scenario 2 is the case that $h_t(x_i) = l$, $y \neq y_i$, and $a(x_i, l) > a(x_i, y)$, $\forall y \in \mathcal{Y}$, $y \neq y_i$, then $\mathcal{M}_i = \frac{a(x_i,y_i) - a(x_i,l) - \alpha_t}{c + \alpha_t}$. This corresponds to the curve which is monotonically decreasing in Figure 2. Scenario 3 is the case that $h_t(x_i) = y$, and $y \neq y_i$, and $\exists l \in \mathcal{Y}$, $l \neq y_i$ such that $a(x_i, l) > a(x_i, y)$, in this case the margin curve of $\mathcal{M}_i$ has two pieces. When $\alpha_t < a(x_i, y) - a(x_i, l)$, $\mathcal{M}_i = \frac{a(x_i,y_i) - a(x_i,l)}{c + \alpha_t}$ and when $\alpha_t > a(x_i, y) - a(x_i, l)$, $\mathcal{M}_i = \frac{a(x_i,y_i) - a(x_i,y) - \alpha_t}{c + \alpha_t}$. The scenarios for the case that $\alpha_t < 0$ can be similarly identified.

Finding the exact solution of optimal α_t along the h_t coordinate is computationally difficult since the examples in Scenario 3 can either intersect with the examples in Scenario 1 or intersect with the examples in Scenario 2. Fortunately, we can prove that (5) is a quasi-concave function [4, 6], this property allows us to design an efficient line search algorithm.

THEOREM 3. *Denote the average margin of the bottom n' examples with respect to the set of weak classifiers $\mathcal{H}$ and their weights $\boldsymbol{\alpha}$ as*

$$\begin{aligned} g_{avg\ n'}(\boldsymbol{\alpha}) &= \frac{1}{n'} \sum_{i \in \{B_{n'} | \boldsymbol{\alpha}\}} \frac{\sum_{j=1}^{|\mathcal{H}|} \alpha_j 1(h_j(x_i) = y)}{\sum_{j=1}^{|\mathcal{H}|} |\alpha_j|} \\ &\quad - \max_{y \in \mathcal{Y}, y \neq y_i} \frac{\sum_{j=1}^{|\mathcal{H}|} \alpha_j 1(h_j(x_i) = y)}{\sum_{j=1}^{|\mathcal{H}|} |\alpha_j|} \end{aligned} \quad (8)$$

Algorithm 2.1 Line search algorithm to find the solution q^* that maximizes (5).

1: **Input:** a weak classifier $h_t \in \mathcal{H}$, an interval $[begin, end]$, and a small number th.
2: **repeat**
3: Set $q = \frac{begin+end}{2}$, calculate the value of $\frac{\partial g_{avg\ n'}}{\partial q}$ as equation (9).
4: **if** $\frac{\partial \mathcal{M}_{avg\ n'}}{\partial q} > 0$ **then**
5: $begin = q$.
6: **else**
7: $end = q$.
8: **end if**
9: **until** $end - begin < th$.
10: **Output:** $q^* = \frac{begin+end}{2}$.

where $\{B_{n'} | \boldsymbol{\alpha}\}$ denotes the set of n' examples whose margins are at the bottom for fixed $\boldsymbol{\alpha}$. Then $g_{avg\ n'}(\boldsymbol{\alpha})$ is a quasi-concave function for any $\boldsymbol{\alpha}$.

Therefore, we can design an algorithm that maximizes (5) efficiently by checking the derivative of (5) as

$$\frac{\partial g_{avg\ n'}}{\partial \alpha_t} = \frac{1}{n'} \sum_{i \in B_{n'}} \frac{\partial \mathcal{M}_i}{\partial \alpha_t}, \quad (9)$$

where $\frac{\partial \mathcal{M}_i}{\partial \alpha_t}$ denotes the derivative of $\mathcal{M}_i$ with respect to α_t, which is calculated as,

$$\frac{\partial \mathcal{M}_i}{\partial \alpha_t} = \begin{cases} \frac{c - (a(x_i,y_i) - a(x_i,l))}{(c+\alpha_t)^2} & \text{Scenario 1} \\ \frac{-c - (a(x_i,y_i) - a(x_i,l))}{(c+\alpha_t)^2} & \text{Scenario 2} \\ -\frac{a(x_i,y_i) - a(x_i,l)}{(c+\alpha_t)^2} & \text{Scenario 3} \end{cases} \quad (10)$$

Assume q^* is the optimal value of α_t that maximizes (5), then (5) is monotonically increasing at $\alpha_t < q^*$, otherwise it is monotonically decreasing. Thus, (9) is less than 0 at $\alpha_t < q^*$, and (9) is greater than 0 at $\alpha_t > q^*$. Formally, the line search algorithm to calculate the value of q^* with a small deviation threshold th is described in Algorithm 2.1.

To select the weak classifier, we use a similar procedure as in Algorithm 1.2 and replace the measure to the average margin of the bottom n' examples. Again, we only consider the decision trees algorithm with binary splits, and apply Algorithm 2.1 on the two subsets respectively. We choose the attributes to split by maximizing the average margin of the bottom n' examples, and use a top-down, greedy search approach to build trees. Same as Algorithm 1, the computational cost of Algorithm 2 is $O(nMK)$ on each round when decision stumps are used as weak learners, where M is the number of binary splits.

Since (5) is non-differentiable at turning points, the coordinate ascent algorithm may get stuck at a corner from which it is impossible to make progress along any coordinate direction. To overcome this difficulty, we use an ϵ-relaxation method [5], which allows a single coordinate to change even if this worsens the objective value. When a coordinate is changed, it is set to ϵ plus (or ϵ minus) the value that maximizes the objective function along that coordinate, where ϵ is a small positive number. If ϵ is small enough, the algorithm can eventually approach a small neighborhood of the optimal solution.

Table 1: Description of 13 UCI datasets

Data	# Examples	K	# Variables	Error Estimation
Abalone	4177	28	8	5-CV
Car	1728	4	6	5-CV
CNAE-9	1080	9	856	5-CV
Glass	214	6	10	5-CV
Krkopt	28056	18	6	5-CV
Letter	20000	26	16	5-CV
Nursery	12960	5	8	5-CV
Poker525k	525010/500000	10	11	test error
Segmentation	210/2100	7	19	test error
Vowel	990	11	10	5-CV
Waveform	5000	3	21	5-CV
Wine	178	3	13	5-CV
Yeast	1484	10	8	5-CV

3. EXPERIMENTS

To evaluate the performance of the DMCBoost algorithm, we first conduct experiments with 13 datasets from the UCI repository [9], then examine its noise robustness on two datasets with random label noise. For comparison, we also report the results of AdaBoost.M1 [11], AdaBoost.MH [27], SAMME [36], and GD-MCBoost [25]. All these algorithms use multi-class base classifiers except AdaBoost.MH, which essentially reduces the multi-class problem to a set of binary classification problems. The classification error is estimated either by a test error or five-fold cross-validation. The datasets which come with pre-specified training and testing sets are evaluated by the test error, where n' is set to $\frac{n}{4}$ for DMCBoost and the number of rounds is set to the maximum of 5000 for each method. For datasets which are evaluated by cross-validation, we partition them into five parts evenly for 5-fold. In each fold, we use three parts for training, one part for validation, and the remaining part for testing. We use the validation data to choose the optimal model for each algorithm. For AdaBoost.M1, AdaBoost.MH, SAMME, and GD-MCBoost the validation data is used to perform early stopping. We run these algorithms with a maximum of 5000 iterations, and then choose the ensemble classifier from the round with minimal error on the validation data. For DMCBoost, the parameter n' is chosen on the values $\{1, \frac{n}{10}, \frac{n}{5}, \frac{n}{4}, \frac{n}{3}, \frac{n}{2}, \frac{2n}{3}\}$ by the validation set. The stopping criterion of DMCBoost is defined as line 7 in Algorithm 2 where DMCBoost terminates at the margin maximization solution, thus we need not to apply early stopping. In all the experiments, the value of ϵ is set to be 0.01 and the value of th is set to be $1e$-5.

An overview of these 13 UCI datasets is shown in Table 1. The datasets have different numbers of input variables (6-856), classes (3-26), and instances (178-1,025,010), and represent a wide area of types of problems. In the # Examples column, the number of training/test examples are listed for datasets coming with pre-specified training and testing sets, and the entire number of examples is given for the rest datasets. The original Poker dataset has 25,010 training examples and 1,000,000 examples for testing. Since the test data is very large, same as the way Li did in [17, 18], we randomly divide it equally into two parts, and add them to training and testing sets respectively, thus its training size becomes 525,010 and the test size becomes 500,000. Therefore, the datasets we selected include fairly large datasets (Poker525k) as well as datasets of moderate sizes (Krkopt, Letter and Nursery).

Table 2: Test error (and standard deviation) of multi-class boosting methods AdaBoost.M1, SAMME, GD-MCBoost, and DMCBoost on 13 UCI datasets, using multi-class decision trees with a depth of 3.

Data	AdaBoost.M1	SAMME	GD-MCBoost	DMCBoost
Abalone	-	74.20(1.8)	74.62(1.5)	**74.03(2.0)**
Car	10.96(2.5)	4.75(1.0)	3.60(1.1)	**2.78(0.8)**
CNAE-9	-	14.91(2.4)	11.4(1.9)	**7.59(1.2)**
Glass	29.52(10.7)	31.9(8.0)	27.0(7.4)	**26.19(10.8)**
Krkopt	-	64.33(0.9)	26.55(0.4)	**22.76(0.7)**
Letter	-	24.94(0.9)	5.40(1.3)	**4.89(0.3)**
Nursery	9.70(1.5)	3.26(0.7)	0.2(0.0)	**0.02(0.0)**
Poker525k	49.16	69.09	-	**30.09**
Segmentation	8.29	6.43	6.0	**5.1**
Vowel	-	19.19(2.6)	9.2(2.6)	**5.66(1.9)**
Waveform	17.8(1.2)	16.96(1.2)	16.2(1.1)	**14.38(1.1)**
Wine	8.57(4.9)	7.43(4.8)	7.54(5.3)	**3.43(4.7)**
Yeast	43.65(2.6)	44.73(4.5)	43.6(3.5)	**42.43(2.8)**

3.1 Experimental Results on UCI Datasets

We compare all multi-class boosting algorithms on 13 UCI datasets. First, we restrict the base classifiers to smaller trees to test the performance of each algorithm when the base classifiers are very weak. We exclude the results of AdaBoost.MH as all the rest algorithms use multi-class base classifiers, and we want to compare the performance of each algorithm with the same hypothesis space $\mathcal{H}$. Table 2 shows the results of different methods when multi-class decision trees with a depth of 3 are used as weak learners[3]. With small trees, DMCBoost gives the best results on all datasets indicating that DMCBoost only requires very weak base classifiers even if there is no exact weak learner condition for DMCBoost. GD-MCBoost achieves the second best accuracy, this algorithm also requires weaker base classifiers since it is able to boost any type of weak learners with non-zero directional derivatives [25]. We do not report its results on the Poker525k dataset since its one iteration takes more than 12 hours to run by the authors' matlab code. For SAMME, the weak learner conditions can be satisfied easily, but it couldn't drive down the training error when the base classifier is very weak, and its performance is much worse. AdaBoost.M1 gives the worst results, and it is not able to boost the base classifiers for 5 of 13 datasets, as shown in Table 2.

With the same hypothesis space $\mathcal{H}$ (trees with a depth of 3), 0-1 loss minimization algorithm (Algorithm 1) usually achieves a lower training classification error rate. The left panel of Figure 3 shows a typical training error curve on the

[3]Whether a base classifier is weak or not often depends on the properties of the datasets, such as number of classes, examples and input variables. For the most multi-class datasets in Table 1, decision trees with a depth of 3 is weak enough. The previous multi-class boosting [10, 22, 36] studies often use much larger trees in the experiments.

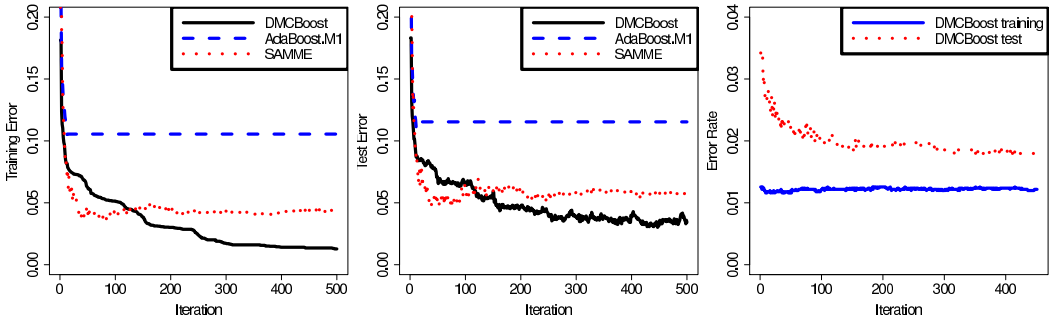

Figure 3: Left: Training errors of AdaBoost.M1, SAMME, and DMCBoost with 0-1 loss minimization algorithm on the Car dataset. Middle: Test errors of AdaBoost.M1, SAMME, and DMCBoost with 0-1 loss minimization algorithm on Car dataset. Right: Training and test error of DMCBoost when it switches to the margin maximization algorithm on the Car dataset.

Table 3: Test error (and standard deviation) of multi-class boosting methods AdaBoost.M1, AdaBoost.MH, SAMME, GD-MCBoost, and DMCBoost on 13 UCI datasets, using decision trees with a maximum depth of 12.

Data	AdaBoost.M1	AdaBoost.MH	SAMME	GD-MCBoost	DMCBoost
Abalone	76.41(1.4)	75.33(1.2)	73.70(1.7)	74.62(1.5)	**73.44(1.8)**
Car	3.36(0.8)	2.84(0.6)	3.65(0.9)	2.8(0.8)	**2.67(0.8)**
CNAE-9	20.9(2.7)	8.43(1.3)	13.43(1.3)	10.6(2.1)	**7.5(1.2)**
Glass	27.14(9.3)	29.52(9.2)	24.76(8.7)	**24.0(6.8)**	24.76(9.9)
Krkopt	14.3(0.3)	11.68(0.3)	12.71(0.2)	12.20(0.3)	**11.04(0.2)**
Letter	3.48(0.3)	**3.1(0.1)**	4.88(0.3)	3.37(0.2)	**3.1(0.2)**
Nursery	0.12(0.1)	0.03(0.0)	0.16(0.1)	**0.0(0.0)**	**0.0(0.0)**
Poker525k	30.19	**2.01**	18.74	-	2.77
Segmentation	4.86	6.14	5.1	6.0	**4.52**
Vowel	5.96(2.9)	7.68(1.8)	6.25(2.3)	**5.6(3.0)**	5.66(1.9)
Waveform	15.2(1.4)	14.56(1.4)	15.08(1.0)	15.2(0.8)	**14.26(1.1)**
Wine	8.57(4.9)	9.16(5.3)	7.43(4.8)	7.54(5.3)	**3.43(4.7)**
Yeast	41.69(1.8)	41.82(2.1)	41.22(3.1)	43.2(3.6)	**40.23(2.5)**

Car dataset, and the middle panel shows the corresponding test error curve. Once the 0-1 loss minimization algorithm terminates at a coordinatewise local minimum, DMCBoost switches to the margin maximization algorithm (Algorithm 2), and it can still drive down the test error even when the training error does not decrease, as shown in the right panel of Figure 3.

We now analyze the running time of AdaBoost.M1, SAMME, and DMCBoost on the Poker525k dataset, which has 525,000 training examples. We implemented each algorithm by C++, and test them on a PC with Core2 Duo 2.6GHz. We left GD-MCBoost out of the comparison since it is unfair to compare a matlab implementation with C++ implementations, and GD-MCBoost runs too slow to record the running time. AdaBoost.M1 and SAMME are very efficient in terms of running time, they take about 10s on each round. For DMCBoost, it takes about 90s on each round, which is slower than AdaBoost.M1 and SAMME but it is bearable on such a scale as the dataset.

We next investigate how these algorithms perform with more powerful base classifiers. We tried all tree depths in the candidate set {3,5,8,12} for each dataset. This time we compare the algorithms not restricted in the same hypothesis space $\mathcal{H}$, so we also add AdaBoost.MH in the comparison. As shown in Table 3, among all the methods, DMCBoost gives the most accurate results in 10 of the 13 datasets, and its results are close to the best results produced by the other methods for the remaining 3 datasets.

3.2 Evaluation of Noise Robustness

In many real-world applications, training samples are obtained through manual labeling and there will be unavoidable human errors that provoke wrong labels. Hence, it is desirable that the designed classification algorithm is robust to noise. In the experiments conducted below, we check the noise robustness of each boosting algorithm on Car and Nursery datasets with additional label noise. We randomly change the labels on training and validation data at the rates of 5% and 20% respectively, and keep the the test data clean. Again, the tree depths are chosen from the candidate set {3,5,8,12}. The results via 5-fold cross-validation (as described in the begining of section 3) are reported in Table 4. The affection of label noise to the performance of DMCBoost is very limited, especially for the Nursery dataset, the test error only increases from 0 to 1.6% when 20% of the training examples have wrong labels. Similar to AdaBoost in binary cases, AdaBoost.M1 and SAMME are quite sen-

Table 4: Test error (and standard deviation) of multi-class boosting methods AdaBoost.M1, AdaBoost.MH, SAMME, GD-MCBoost, and DMCBoost on the two UCI datasets with random noise, using decision trees with a maximum depth of 12.

Data	Noise rate	AdaBoost.M1	AdaBoost.MH	SAMME	GD-MCBoost	DMCBoost
Car	0	3.36(0.8)	2.84(0.6)	3.65(0.9)	3.6(1.1)	**2.67(0.8)**
	0.05	9.22(1.5)	6.09(1.0)	7.94(2.0)	5.8(1.1)	**3.54(1.2)**
	0.2	14.55(2.4)	9.1(1.0)	14.2(1.9)	9.6(1.1)	**6.55(1.6)**
Nursery	0	0.12(0.1)	0.03(0.0)	0.16(0.1)	**0.0(0.0)**	**0.0(0.0)**
	0.05	3.93(1.3)	1.82(0.5)	2.29(0.2)	2.4(0.5)	**0.37(0.2)**
	0.2	6.61(0.9)	3.47(0.6)	6.61(0.9)	4.4(0.9)	**1.61(0.5)**

sitive to noise, their performance is hurt badly even with a 5% noise rate.

DMCBoost achieves good performance by varying the parameter n', the higher the noise rate, the larger n' should be used. Consider the Car dataset as an example, for the case that the training set is clean (noise rate is 0), the optimal n' via cross-validation is $\frac{n}{10}$ and the training error is 1.5%, thus the algorithm focus on the difficult examples. For the case that the noise rate is 5%, the optimal n' is $\frac{n}{4}$ and the training error is 7.1%, indicating that DMCBoost allows some misclassification to achieve a better performance. For a noise rate of 20%, DMCBoost considers more examples in the bottom set, the optimal n' via cross-validation is $\frac{n}{2}$. In this case, DMCBoost further allows more misclassification (the training error is 22.2%), but its corresponding test error is only 6.55%.

4. CONCLUSION AND FUTURE WORKS

In this paper we have proposed a multi-class boosting approach that directly optimizes the 0-1 loss and the targeted margins. Experiments show that our method gives better results in the case of restricting the weak learning algorithm to small decision trees, and performs highly competitively with the existing boosting algorithms in the case of deeper decision trees. More importantly, our method is more robust on the noisy data.

In this study we restrict the weak learners to multi-class decision trees as the combination of boosting with a decision trees is the state-of-the-art classification method [1]. For future works, we will consider more weak learning algorithms and the case that $|\mathcal{H}|$ is infinite, such as k-Nearest Neighbors, Naive Bayes, and Neural Networks.

5. ACKNOWLEDGMENTS

This research is supported in part by AFOSR under grant FA9550-10-1-0335, NSF under grant IIS:RI-small 1218863, DoD under grant FA2386-13-1-3023, and a Google research award.

6. REFERENCES

[1] R. Appel, T. Fuchs, T. Dollar and P. Perona. Quickly boosting decision trees - Pruning underachieving features early. *International Conference on Machine Learning* (ICML), 2013.

[2] E. Allwein, R. Schapire and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113-141, 2000.

[3] P. Bartlett, M. Jordan and J. McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(March):138-156, 2006.

[4] M. Bazaraa, H. Sherali and C. Shetty. *Nonlinear Programming: Theory and Algorithms*, 3rd Edition. Wiley-Interscience, 2006.

[5] D. Bertsekas. *Network Optimization: Continuous and Discrete Models*. Athena Scientific, 1998.

[6] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University, 2004.

[7] A. Demiriz, K. Bennett and J. Shawe-Taylor. Linear programming boosting via column generation, *Machine Learning*, 46:225–254, 2002.

[8] T. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[9] A. Frank and A. Asuncion. UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science, 2010.

[10] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. *The 13th International Conference in Machine Learning*, 148-156, 1996.

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

[12] J. Friedman, T. Hastie and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–374, 2000.

[13] J. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 28(2):1189–1232, 2001.

[14] W. Gao, Z. Zhou. On the doubt about margin explanation of boosting. *Artificial Intelligence*, 203:1–18, 2013.

[15] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning*, 2nd Edition. Springer, 2009.

[16] H. He, E.A. Garcia. *Learning from imbalanced data*. IEEE Trans. Knowl. Data Eng, 21(9):1263–1284, 2009.

[17] P. Li. ABC-Boost: Adaptive base class boost for multi-class classification. *International Conference on Machine Learning* (ICML), 2009.

[18] P. Li. Robust LogitBoost and adaptive base class (ABC) LogitBoost. *Uncertainty on Artificial Intelligence* (UAI), 2010.

[19] P. Long and R. Servedio. Random classification noise defeats all convex potential boosters. *Machine Learning*, 78:287-304, 2010.

[20] D. McAllester, T. Hazan and J. Keshet. Direct loss minimization for structured prediction. *Advances in Neural Information Processing Systems* (NIPS), 23:1594–1602, 2010.

[21] D. Mease and A. Wyner. Evidence contrary to the statistical view of boosting. *Journal of Machine Learning Research*, 9:131–156, 2008.

[22] I. Mukherjee and R. Schapire. A theory of multiclass boosting. *Journal of Machine Learning Research*, 14:437–497, 2013.

[23] T. Nguyen and S. Sanner. Algorithms for direct 0-1 loss optimization in binary classification. *International Conference on Machine Learning* (ICML), 2013.

[24] C. Rudin, I. Daubechies and R. Schapire. The dynamics of AdaBoost: Cyclic behavior and convergence of margins. *Journal of Machine Learning Research*, 5:1557-1595, 2004.

[25] M. Saberian and N. Vasconcelos. Multiclass boosting: Theory and algorithms. In Proc. *Neural Information Processing Systems* (NIPS), 2011.

[26] R. Schapire, Y. Freund, P. Bartlett and W. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

[27] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[28] R. Schapire and Y. Freund. *Boosting: Foundations and Algorithms*. MIT Press, 2012.

[29] S. Shalev-Shwartz and Y. Singer. On the equivalence of weak learnability and linear separability: new relaxations and efficient boosting algorithms. *Machine Learning*, 80(2-3): 141-163, 2010.

[30] Y. Sun, M.S. kamel, A.K.C. Wong, Y. Wang. Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognition*, 40(12): 3358-3378, 2007.

[31] P. Tseng. Convergence of block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications*, 475–494, 2001.

[32] L. Wang, M. Sugiyama, Z. Jing, C. Yang, Z. Zhou and J. Feng. A refined margin analysis for boosting algorithms via equilibrium margin. *The Journal of Machine Learning Research*, 12:1835-1863, 2011.

[33] M. Warmuth, K. Glocer and G. Ratsch. Boosting algorithms for maximizing the soft margin. *Advances in Neural Information Processing Systems* (NIPS), 21, 2007.

[34] X. Wu, V. Kumar, J. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. McLachlan, A. Ng, B. Liu, P. Yu, Z. Zhou, M. Steinbach, D. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information System*, 14(1):1-37, 2008.

[35] S. Zhai, T. Xia, M. Tan and S. Wang. Direct 0-1 loss minimization and margin maximization with boosting. *Advances in Neural Information Processing Systems* (NIPS), 2013.

[36] J. Zhu, H. Zou, S. Rosset and T. Hastie. Multi-class AdaBoost. *Statistics and Its Interface*, 2:349–366, 2009.

[37] H. Zou, J. Zhu and T. Hastie. New multi-category boosting algorithms based on multicategory Fisher-consistent losses. *Annals of Applied Statistics*, 2(4): 1290-1306, 2008.

APPENDIX

A. PROOF OF THEOREM 1

PROOF. : (i) We first show that when the optimal solution of soft margin optimization is achieved, $\mathcal{M}_i + \xi_i = \rho$ always holds for those $\xi_i > 0$. Suppose when the optimal solution is achieved, there exists an example such that $\mathcal{M}_i + \xi_i > \rho$ and $\xi_i > 0$. The slack variable $\xi_i > 0$ indicates $\mathcal{M}_i < \rho$, then we can always find a $\xi_i' < \xi$ such that $\mathcal{M}_i + \xi_i' = \rho$, and therefore replace ξ to ξ' in objective function that produce a greater objective value, a contradiction.

(ii) We next prove under the optimal solution, there are at most n' examples that are allowed to lie below (or equal) ρ, that is, $|\{x_i|\mathcal{M}_i \leq \rho\}| \leq n'$. Suppose we arrived a optimal solution and $n' < |\{x_i|\mathcal{M}_i \leq \rho\}| \leq m'$, with the corresponding slack variable $\boldsymbol{\xi}$. Then we can always find a $\rho' < \rho$ such that $|\{x_i|\mathcal{M}_i \leq \rho'\}| \leq n'$, with the slack variable $\boldsymbol{\xi}'$. We denote $\delta = \rho - \rho'$, then by (i) $\xi_i' = \xi_i - \delta$ for the examples with $\xi_i' > 0$. We use $B_{m'}$ to denote the set of m' examples having the smallest margins, then $B_{n'} \subset B_{m'}$ and $B_{n'}^c \cap B_{m'} \neq \emptyset$. We have

$$\begin{aligned}\rho - \frac{1}{n'}\sum_{i=1}^{n}\xi_i &= \rho' - \frac{1}{n'}(\sum_{i\in B_{m'}}\xi_i - n'\delta)\\ &= \rho' - \frac{1}{n'}\sum_{i\in B_{n'}}\xi_i' - \frac{1}{n'}\sum_{i\in B_{n'}^c\cap B_{m'}}\xi_i\\ &\leq \rho' - \frac{1}{n'}\sum_{i\in B_{n'}}\xi_i'\end{aligned}$$

which is a contradiction.

(iii) We further show that $|\{x_i|\mathcal{M}_i \leq \rho\}| \geq n'$ under the optimal solution. Suppose there is an optimal solution with $|\{x_i|\mathcal{M}_i \leq \rho\}| \leq m' < n'$ and slack variable $\boldsymbol{\xi}$, then there must exist a $\rho' > \rho$ such that $m' < |\{x_i|\mathcal{M}_i \leq \rho\}| \leq n'$ and corresponding $\boldsymbol{\xi}'$. We denote $\delta = \rho' - \rho$ and we have

$$\begin{aligned}\rho - \frac{1}{n'}\sum_{i=1}^{n}\xi_i &= \rho' - \frac{1}{n'}(\sum_{i\in B_{m'}}\xi_i + n'\delta)\\ &\leq \rho' - \frac{1}{n'}(\sum_{i\in B_{m'}}\xi_i' + \sum_{i\in B_{m'}^c\cap B_{n'}}\xi_i')\\ &= \rho' - \frac{1}{n'}\sum_{i\in B_{n'}}\xi_i'\end{aligned}$$

where the inequation holds since $\xi_i' \leq \delta$ if $x_i \in B_{m'}^c \cap B_{n'}$. A contradiction.

(iv) Combine the results (ii) and (iii), we have $|\{x_i|\mathcal{M}_i \leq \rho\}| = n'$, and therefore $\sum_{i\in B_{n'}}(\mathcal{M}_i + \xi_i) = n'\rho$, it follows that $\sum_{i\in B_{n'}}\mathcal{M}_i = n'\rho - \sum_{i\in B_{n'}}\xi_i = n'\rho - \sum_{i=1}^{n}\xi_i$. Dividing by n' on both sides of the equation, we get $\frac{1}{n'}\sum_{i\in B_{n'}}\mathcal{M}_i = \rho - \frac{1}{n'}\sum_{i=1}^{n}\xi_i$. □

B. PROOF OF THEOREM 2

PROOF. : Our proof is inspired by the works in [14, 26, 32]. Let $\mathcal{C}(\mathcal{H})$ denote the convex hull of $\mathcal{H}$, and let $\mathcal{C}_N(\mathcal{H})$ denote the set of unweighted averages over N elements from $\mathcal{H}$. Formally,

$$\mathcal{C}(\mathcal{H}) = \{f : f = \sum_k \alpha_k 1(h_k(x) = y), \sum_k \alpha_k = 1, \alpha_k \geq 0, h_k \in \mathcal{H}\}$$

$$\mathcal{C}_N(\mathcal{H}) = \{g : g = \frac{1}{N}\sum_{j=1}^{N} 1(h_j(x) = y), h_j \in \mathcal{H}\}$$

We denote the distribution over $\mathcal{H}$ by the coefficients $\{\alpha_k\}$ to be $\mathcal{Q}(f)$. Let $B_{\{n'|f\}}$ be the set of n' examples having the smallest margins associate with the inference function f. By [26] we know that for any fixed $\beta > 0$

$$\Pr_{\mathcal{D}}[\mathcal{M}(f,x,y) < 0] \leq \Pr_{D,g\sim\mathcal{Q}(f)}[\mathcal{M}(g,x,y) < \beta] + \exp(\frac{-N\beta^2}{2}) \tag{11}$$

Let $\mathcal{M}(g, x_{(k)}, y_{(k)})$ denote the k-th smallest margin with respect to g. For any $n' \in \{1, \cdots, n\}$ and $\epsilon_N > 0$, we consider the following probability:

$$\begin{aligned}&\Pr_S[\Pr_{\mathcal{D}}[\mathcal{M}(g,x,y) < \beta] > \mathbf{1}(\mathcal{M}(g, x_{(n')}, y_{(n')}) \leq \beta) + \epsilon_N]\\ &\leq \sum_{i=0}^{n'-1}\binom{n}{i}\epsilon_N^i(1-\epsilon_N)^{n-i} \leq \exp(-nD(\frac{n'}{n}|\epsilon))\end{aligned}$$

by using the relative entropy Chernoff bound.

Let $\mathcal{Z} = \{i/|\mathcal{H}| : i = 1, \cdots, |\mathcal{H}|\}$, we only consider β at the values in $\mathcal{Z}$. By the fact that $|\mathcal{C}_N(\mathcal{H})| \leq |\mathcal{H}|^N$ and applying the union bound, for any $n' \in \{1, \cdots, n\}$,

$$\begin{aligned}
&\Pr_{S, g\sim\mathcal{Q}(f)}\big[\exists g \in \mathcal{C}_N(\mathcal{H}), \exists \beta \in \mathcal{Z}, \Pr_{\mathcal{D}}[\mathcal{M}(g,x,y) < \beta] \\
&\qquad > \mathbf{1}[\mathcal{M}(g, x_{(n')}, y_{(n')}) \leq \beta] + \epsilon_N\big] \\
\leq & |\mathcal{H}|^{N+1} \exp(-nD(\frac{n'}{n}|\epsilon))
\end{aligned}$$

Let $\delta_N = |\mathcal{H}|^{N+1} \exp(-nD(\frac{n'}{n}|\epsilon))$, then

$$\epsilon_N = D^{-1}(\frac{n'-1}{n}, \frac{1}{n}\log\frac{|\mathcal{H}|^{N+1}}{\delta_N})$$

Thus, with a probability at least $1-\delta_N$ over the training sample S, for all $f \in \mathcal{C}(\mathcal{H})$, all $\beta \in \mathcal{Z}$, and all fixed n', we have

$$\begin{aligned}
&\Pr_{\mathcal{D}}[\mathcal{M}(g,x,y) < \beta] \leq \mathbf{1}[\mathcal{M}(g, x_{(n')}, y_{(n')}) \leq \beta] \\
&+ D^{-1}(\frac{n'-1}{n}, \frac{1}{n}\log\frac{|\mathcal{H}|^{N+1}}{\delta_N})
\end{aligned}$$

Since

$$\begin{aligned}
&\Pr_{D, g\sim\mathcal{Q}(f)}[\mathcal{M}(g,x,y) \leq \beta] = E_{g\sim\mathcal{Q}(f)}[\Pr_{\mathcal{D}}[\mathcal{M}(g,x,y) < \beta]] \\
&\leq \Pr_{g\sim\mathcal{Q}(f)}[\mathcal{M}(g, x_{(n')}, y_{(n')}) \leq \beta] + D^{-1}(\frac{n'-1}{n}, \frac{1}{n}\log\frac{|\mathcal{H}|^{N+1}}{\delta_N})
\end{aligned}$$

And for any $\theta > \beta$,

$$\begin{aligned}
&\Pr_{g\sim\mathcal{Q}(f)}[\mathcal{M}(g, x_{(n')}, y_{(n')}) \leq \beta] \\
\leq & \mathbf{1}[\frac{1}{n'}\sum_{i\in B_{\{n'|f\}}} \mathcal{M}(f, x_i, y_i) < \theta] \\
&+ \Pr_{g\sim\mathcal{Q}(f)}[\mathcal{M}(f, x_{(n')}, y_{(n')}) \geq \theta, \mathcal{M}(g, x_{(n')}, y_{(n')}) \leq \beta]
\end{aligned}$$

Thus, we have

$$\begin{aligned}
&\Pr_{D, g\sim\mathcal{Q}(f)}[\mathcal{M}(g,x,y) \leq \beta] \\
\leq & \mathbf{1}[\frac{1}{n'}\sum_{i\in B_{\{n'|f\}}} \mathcal{M}(f, x_i, y_i) < \theta] + D^{-1}(\frac{n'-1}{n}, \frac{1}{n}\log\frac{|\mathcal{H}|^{N+1}}{\delta_N}) \\
&+ \Pr_{g\sim\mathcal{Q}(f)}[\mathcal{M}(f, x_{(n')}, y_{(n')}) \geq \theta, \mathcal{M}(g, x_{(n')}, y_{(n')}) \leq \beta] \qquad (12)
\end{aligned}$$

We now prove if $\mathcal{M}(f, x_{(n')}, y_{(n')}) \geq \theta$ and $\mathcal{M}(g, x_{(n')}, y_{(n')}) \leq \beta$ with $\theta > \beta$, there always exists an example (x_i, y_i) such that $\mathcal{M}(f, x_i, y_i) \geq \theta$ and $\mathcal{M}(g, x_i, y_i) \leq \beta$. Since there exists a bijection between $\{\mathcal{M}(f, x_{(1)}, y_{(1)}), \cdots, \mathcal{M}(f, x_{(n)}, y_{(n)})\}$ and $\{\mathcal{M}(g, x_{(1)}, y_{(1)}), \cdots, \mathcal{M}(g, x_{(n)}, y_{(n)})\}$, we can assume $\mathcal{M}(f, x_{(n')}, y_{(n')})$ corresponding to $\mathcal{M}(g, x_{(\tilde{n})}, y_{(\tilde{n})})$ for some $\tilde{n} \leq n$. If $\tilde{n} \leq n'$, then $(x_{(n')}, y_{(n')})$ of $\mathcal{M}(f, x_{(n')}, y_{(n')})$ is desired. On the other hand, if $\tilde{n} > n'$, then there are at least $n - n'$ examples greater than or equal to θ in $\{\mathcal{M}(f, x_i, y_i) : i \neq n'\}$ but at most $n - n' - 1$ examples greater than β in $\{\mathcal{M}(g, x_i, y_i) : i \neq \tilde{n}\}$. Thus,

$$\begin{aligned}
&\Pr_{g\sim\mathcal{Q}(f)}[\mathcal{M}(f, x_{(n')}, y_{(n')}) \geq \theta, \mathcal{M}(g, x_{(n')}, y_{(n')}) \leq \beta] \\
\leq & \Pr_{g\sim\mathcal{Q}(f)}[\exists (x_i, y_i) : \mathcal{M}(f, x_i, y_i) \geq \theta, \mathcal{M}(g, x_i, y_i) \leq \beta] \\
\leq & \Pr_{g\sim\mathcal{Q}(f)}[\exists (x_i, y_i) : \forall \widetilde{y} : f(x_i, y_i) - f(x_i, \widetilde{y}) \geq \theta, \exists \widetilde{y} : g(x_i, y_i) - g(x_i, \widetilde{y}) \leq \beta] \\
\leq & n(K-1)\exp(\frac{-N(\theta-\beta)^2}{2})
\end{aligned}$$

By combining (11) and (12), we obtain that with probability at least $1-\delta_N$ over the training sample S, for all $f \in \mathcal{C}(\mathcal{H})$, all $\beta \in \mathcal{Z}$ and all $\theta > \beta$ and all $n' = \{1, \cdots, n\}$, but fixed N,

$$\begin{aligned}
&\Pr_{\mathcal{D}}[\mathcal{M}(f,x,y) < 0] \\
\leq & \mathbf{1}[\frac{1}{n'}\sum_{i\in B_{\{n'|f\}}} \mathcal{M}(f, x_i, y_i) < \theta] + \exp(\frac{-N\beta^2}{2}) \\
&+ n(K-1)\exp(\frac{-N(\theta-\beta)^2}{2}) + D^{-1}(\frac{n'-1}{n}, \frac{1}{n}\log\frac{|\mathcal{H}|^{N+1}}{\delta_N})
\end{aligned}$$

To let β takes values only in $\mathcal{Z}$, we set $\beta = \frac{\theta}{2} - \frac{\eta}{|\mathcal{H}|}$ with $0 \leq \eta < 1$ and $N = \left\lceil \frac{8}{\theta^2}\log\frac{2n^2}{\log|\mathcal{H}|} \right\rceil$, the following inequality holds for $\theta \geq \frac{8}{|\mathcal{H}|}$

$$\begin{aligned}
&\exp(\frac{-N\beta^2}{2}) + n(K-1)\exp(\frac{-N(\theta-\beta)^2}{2}) \\
\leq & (K-1)\frac{\log|\mathcal{H}|}{2n} + \sqrt{\frac{\log|\mathcal{H}|}{2n^2}}
\end{aligned}$$

Setting $\delta = 2^N\delta_N$. Thus, with probability at least $1-\delta$ over the random choice of the training data S of n examples, for all $f \in \mathcal{C}(\mathcal{H})$ and all $n' = \{1, \cdots, n\}$, we obtain

$$\begin{aligned}
\Pr_{\mathcal{D}}[\mathcal{M}(f,x,y) < 0] \leq & \mathbf{1}[\frac{1}{n'}\sum_{i\in B_{\{n'|f\}}} \mathcal{M}(f, x_i, y_i) < \theta] \\
&+ (K-1)\frac{\log|\mathcal{H}|}{2n} + \sqrt{\frac{\log|\mathcal{H}|}{2n^2}} + D^{-1}(\frac{n'-1}{n}, \frac{1}{n}u)
\end{aligned}$$

where

$$u = \frac{8}{\theta^2}\log\frac{2n^2}{\log|\mathcal{H}|}\log 2|\mathcal{H}| + \log|\mathcal{H}| + \log\frac{1}{\delta}$$

□

C. PROOF OF THEOREM 3

PROOF. : By definition of quasiconcave, $g_{avg\ n'}(\boldsymbol{\alpha})$ is quasiconcave if and only if its upper contour sets are convex sets. The γ-upper-contour set $\mathcal{S}_\gamma$ of $g_{avg\ n'}(\boldsymbol{\alpha})$ is denoted as

$$\begin{aligned}
\mathcal{S}_\gamma = \Big\{ \boldsymbol{\alpha} : & \sum_{i\in\{B_{n'|\boldsymbol{\alpha}}\}} \Big(\sum_{j=1}^{|\mathcal{H}|} \alpha_j 1(h_j(x_i) = y_i) \\
& - \max_{y\in\mathcal{Y}, y\neq y_i} \sum_{j=1}^{|\mathcal{H}|} \alpha_j 1(h_j(x_i) = y) \Big) \geq \gamma \sum_{j=1}^{|\mathcal{H}|} |\alpha_j| \Big\}
\end{aligned}$$

We now prove that $\mathcal{S}_\gamma$ is a convex set. For $\forall \boldsymbol{\alpha}^{(1)}$, $\boldsymbol{\alpha}^{(2)} \in \mathcal{S}_\gamma$, $\forall \lambda \in [0,1]$, we have

$$\begin{aligned}
&\sum_{i\in\{B_{n'|(1-\lambda)\boldsymbol{\alpha}^{(1)}+\lambda\boldsymbol{\alpha}^{(2)}\}}} (\sum_{j=1}^{|\mathcal{H}|}((1-\lambda)\alpha_j^{(1)} + \lambda\alpha_j^{(2)})1(h_j(x_i) = y_i) \\
&- \max_{y\in\mathcal{Y}, y\neq y_i} \sum_{j=1}^{|\mathcal{H}|}((1-\lambda)\alpha_j^{(1)} + \lambda\alpha_j^{(2)})1(h_j(x_i) = y)) \\
\geq & (1-\lambda) \sum_{i\in\{B_{n'|\boldsymbol{\alpha}^{(1)}\}}} (\sum_{j=1}^{|\mathcal{H}|} \alpha_j^{(1)} 1(h_j(x_i) = y_i) - \max_{y\in\mathcal{Y}, y\neq y_i} \sum_{j=1}^{|\mathcal{H}|} \alpha_j^{(1)} 1(h_j(x_i) = y)) \\
&+ \lambda \sum_{i\in\{B_{n'|\boldsymbol{\alpha}^{(2)}\}}} (\sum_{j=1}^{|\mathcal{H}|} \alpha_j^{(2)} 1(h_j(x_i) = y_i) - \max_{y\in\mathcal{Y}, y\neq y_i} \sum_{j=1}^{|\mathcal{H}|} \alpha_j^{(2)} 1(h_j(x_i) = y)) \\
\geq & (1-\lambda)\gamma \sum_{j=1}^{|\mathcal{H}|} |\alpha_j^{(1)}| + \lambda\gamma \sum_{j=1}^{|\mathcal{H}|} |\alpha_j^{(2)}| \geq \gamma \sum_{j=1}^{|\mathcal{H}|} |(1-\lambda)\alpha_j^{(1)} + \lambda\alpha_j^{(2)}|
\end{aligned}$$

Therefore, $(1-\lambda)\boldsymbol{\alpha}^{(1)} + \lambda\boldsymbol{\alpha}^{(2)} \in \mathcal{S}_\gamma$. $g_{avg\ n'}(\boldsymbol{\alpha})$ is quasiconcave. □