

Efficient Multiclass Boosting Classification with Active Learning

Jian Huang¹, Seyda Ertekin², Yang Song², Hongyuan Zha³, C. Lee Giles^{1,2}

¹College of Information Sciences and Technology

²Department of Computer Science and Engineering

The Pennsylvania State University, University Park, PA 16802

³College of Computing, Georgia Institute of Technology, Atlanta, GA 30332

{jhuang, giles}@ist.psu.edu {sertekin, yasong}@cse.psu.edu {zha}@cc.gatech.edu

Abstract

We propose a novel multiclass classification algorithm Gentle Adaptive Multiclass Boosting Learning (GAMBLE). The algorithm naturally extends the two class Gentle AdaBoost algorithm to multiclass classification by using the multiclass exponential loss and the multiclass response encoding scheme. Unlike other multiclass algorithms which reduce the K-class classification task to K binary classifications, GAMBLE handles the task directly and symmetrically, with only one committee classifier. We formally derive the GAMBLE algorithm with the quasi-Newton method, and prove the structural equivalence of the two regression trees in each boosting step.

To scale up to large datasets, we utilize the generalized Query By Committee (QBC) active learning framework to focus learning on the most informative samples. Our empirical results show that with QBC-style active sample selection, we can achieve faster training time and potentially higher classification accuracy. GAMBLE's numerical superiority, structural elegance and low computation complexity make it highly competitive with state-of-the-art multiclass classification algorithms.

Keywords: Boosting, Multiclass Classification, Active Learning, Data Mining, Committee Machines

1 Introduction

Boosting algorithms [19, 16] have been very popular in the past decade. The general boosting framework only requires the underlying weak learners to perform better than random guessing. By combining these weak learners based on weighted majority vote, a committee classifier dramatically reduces the training and testing error rates. The representative AdaBoost algorithm [8, 11], in particular, adapts to the weighted error of the current weak learner, by strengthening the weights of the misclassified samples and dampening the weights of the correctly classified ones.

The most appealing feature of boosting is its empirical *resistance to overfitting* for various classification tasks. Researchers have gained insights into this seemingly mysterious phenomenon. Friedman et al. [11] provided an alternative viewpoint for boosting as the well-known forward stagewise additive modeling. By combining low-variance (high-bias) weak learners, boosting

acts as a bias reduction process. Based on the margin and VC dimension theory, Schapire et al. [8] proved that the generalization error is upper bounded, independent of the number of training iterations. Boosting is thus regarded as a margin increasing process, which connects it to Support Vector Machines [23].

Our work is motivated by two goals. First, boosting methods were initially designed to handle binary classification problems. Initial efforts typically decomposed the K-class classification problem to K binary classification problems by using the 'one-against-all' strategy, thus requiring K committees to be trained. A better approach should handle such problems directly and symmetrically. Since weak learners (e.g. classification trees) can naturally handle multiclass cases, a direct multiclass boosting method can unleash the innate power of multiclass weak learners that a 'one-against-all' or 'pairwise' method cannot. Generalizing to the multiclass cases, however, requires non-trivial extension of the loss function, encoding scheme and the algorithms per se.

Furthermore, multiclass labels are usually expensive to obtain (e.g. text classification). And even if such training data is abundant, multiclass classification is generally less efficient. The practicality of batch learning algorithms is limited in such learning settings. Boosting methods, specifically, require global weight updates in each boosting step, as such they are not tractable for large-scale multiclass learning. Active learning strategies enable large-scale learning for boosting algorithms by greatly improving data efficiency.

In this paper, we propose a novel multiclass boosting algorithm, Gentle Adaptive Multiclass Boosting Learning (GAMBLE), which addresses the aforementioned issues and has the following features:

- A K-class classification problem is treated simultaneously and symmetrically, without recasting it to multiple binary classification problems.
- Only one regression function per iteration is fitted,

yielding a highly efficient solution.

- The algorithm is less sensitive to outliers in training and robust to unseen data in testing, due to its numerical stability in each weak learner.
- With active sample selection, Active GAMBLE uses the informative portion of the training samples and is thus fast in training with early stopping. It is also efficient in prediction with a simpler model and potentially has greater generalization power.
- The algorithmic structure is succinct and the algorithm is easy to implement.

The paper is organized as follows. We review the related research on multiclass boosting algorithms and active learning in Section 2. In Section 3, we first derive the general GAMBLE algorithm using quasi-Newton methods and then propose an efficient implementation of GAMBLE using regression trees. We also comment on the advantages of GAMBLE over other multiclass boosting algorithms. In Section 4, we further investigate the active learning strategy for GAMBLE. We present experimental results on real world and synthetic data in Section 5 and conclude the paper in Section 6.

2 Related Work

2.1 Related research in multiclass boosting The first attempt to extend AdaBoost to multiclass classification is the AdaBoost.M1 algorithm [8], a direct generalization of AdaBoost by replacing the error with an indicator function $I[h_k(\mathbf{x}_i) \neq y_i]$, where h_k is the k -th dimension of the weak hypothesis. Since the same binary loss function is used as AdaBoost.Discrete in the binary case, AdaBoost.M1 fails when the underlying weak learners misclassify more than half of the samples. This is not appropriate for multiclass classification, when the underlying weak learners perform slightly better than random guessing ($err < \frac{K-1}{K}$).

More sophisticated approaches have been investigated for this case. AdaBoost.M2 [8, 18] (a special case of AdaBoost.MR [21]) solves the multiclass problem by using the ‘pairwise’ strategy. However, AdaBoost.M2 complicates the design of the weak learners since it requires the optimization of weak hypotheses for the pseudoloss measure, a weighted transform of the weak hypotheses. AdaBoost.MH [21] was proposed as an alternative to AdaBoost.M2. This method adopts the well-known ‘one-against-all’ [10] strategy to recast the K -class classification problem to K binary classification problems, which are in turn solved by the binary AdaBoost.Real [11] algorithm. Other multiclass variants exist such as AdaBoost.MO [21, 3] which uses the correcting output code.

These boosting algorithms do not handle the multiclass classification problems directly, but instead reduce them to multiple two-class classification problems. In practice, even when the classification boundaries for class pairs are simple, the pooling classes (as in the ‘against all’ classes) may form boundaries that are too complex to approximate and the methods above may fail [11, 10, 12]. Friedman et al. [11] proposed the multiclass LogitBoost algorithm by using the multiclass logistic model, which outperformed the asymmetric variants such as AdaBoost.MH. This gave insight into how to solve multiclass classification problems simultaneously. Recently, Zhu et al. [24] proposed the multiclass exponential loss function. By adopting the K -class response encoding scheme, they successfully generalized AdaBoost.Discrete and AdaBoost.Real to the multiclass version SAMME.Discrete and SAMME.Real. SAMME (Stagewise Additive Modeling using a Multi-class Exponential loss function) generates only one K -class classifier in each iteration, thus it is K times faster than AdaBoost.MH.

As noted by Schapire et al. [19] and convincingly shown by Dietterich [7], when there are a great number of outliers in the training instances, AdaBoost (SAMME.Discrete and SAMME.Real alike) may overemphasize them, which eventually leads to inferior hypotheses. Friedman et al. proposed the Gentle AdaBoost algorithm [11] to ameliorate such a problem, which is a gradient ascent approach in function space. Gentle AdaBoost also outperforms LogitBoost [11] for its numerical stability, since the updates are bounded. The Gentle AdaBoost.MH algorithm was also proposed in [11] but still using the ‘one-against-all’ strategy. This paper generalizes this successful boosting algorithm to the multiclass case in non-trivial ways, yielding a natural and efficient solution.

2.2 Related work in active learning Compared to the profusion of literature in boosting algorithms, there is relatively little work on the scalability issue of boosting methods. Active learning strategies can be used to scale up boosting algorithms by focusing training on the most informative samples. Query By Committee (QBC) [22] is a popular and theoretically well motivated active learning method. Abe et al. [1] proposed the QBoost algorithm which combines QBC and AdaBoost to efficiently learn the committee. This paper adopts the same active sample selection method as in QBoost and extends it to multiclass classification.

3 The GAMBLE Boosting Algorithm

We first formalize the K -class learning problem as:

Given N training samples $\{(\mathbf{x}_i, c_i)\}_{i=1}^N$ ($\mathbf{x}_i \in$

Gentle Adaptive Multiclass Boosting Learning (GAMBLE)

1. Initialization: set the observation weights $w_i \leftarrow \frac{1}{N}$, $i = 1, 2, \dots, N$, and set $\mathbf{F}(\mathbf{x}) \leftarrow 0$.
2. For $m \leftarrow 1$ to M :
 - (a) Using weight distribution $\{w_i\}$, fit two regression functions $\mathbf{g}^{(m)}(\mathbf{x})$ of \mathbf{y} and $\mathbf{h}^{(m)}(\mathbf{x})$ of \mathbf{z} to \mathbf{x} by weighted least-squares (WLS), where $z_j = y_j^2$, $j = 1, 2, \dots, K$.
 - (b) Set $r_j^{(m)}(\mathbf{x}) = K \cdot \frac{g_j^{(m)}(\mathbf{x})}{h_j^{(m)}(\mathbf{x})}$
 - (c) Obtain weak learner $f_j^{(m)}(\mathbf{x}) = r_j^{(m)}(\mathbf{x}) - \frac{1}{K} \sum_{k=1}^K r_k^{(m)}(\mathbf{x})$, $j = 1, 2, \dots, K$.
 - (d) Update committee $\mathbf{F}(\mathbf{x}) \leftarrow \mathbf{F}(\mathbf{x}) + \mathbf{f}^{(m)}(\mathbf{x})$.
 - (e) Update weights $w_i \leftarrow w_i \exp(-\frac{1}{K} \mathbf{y}_i^T \mathbf{f}^{(m)}(\mathbf{x}_i))$, $i = 1, 2, \dots, N$ and renormalize.
3. Output committee classifier: $C(\mathbf{x}) = \arg \max_k F_k(\mathbf{x})$.

Algorithm 1: The Gentle Adaptive Multiclass Boosting Learning (GAMBLE) Algorithm.

$\mathcal{R}^D, c_i \in \{1, 2, \dots, K\}$) drawn independently from the population, we need to obtain M weak hypotheses $\{\mathbf{f}^{(m)}(\mathbf{x}) | \mathbf{f}^{(m)} \in \mathcal{H}\}_{m=1}^M$, each performing better than random guessing ($\exists \gamma > 0$, $err(\mathbf{f}^{(m)}) \leq \frac{K-1}{K} - \gamma$). The combined committee $\mathbf{F}(\mathbf{x})$ constitutes a strong hypothesis, in terms of minimizing the empirical loss.

In what follows, we use the K -class response encoding scheme as in [15, 24] rather than a discrete number as a class label. A class label c is encoded by a K -dimensional response vector $\mathbf{y} = (y_1, \dots, y_K)^T$, where $y_k = 1$ if and only if $k = c$ and $y_k = -\frac{1}{K-1}$ otherwise. Hence, the response vector satisfies the sum-to-zero constraint, i.e. $\sum_{k=1}^K y_k = 0$. The advantage of this encoding scheme unfolds in the sequel.

3.1 The General GAMBLE Algorithm Given a classifier \mathbf{F} and a sample (\mathbf{x}, \mathbf{y}) , a loss function $J(\mathbf{F}(\mathbf{x}))$ measures the degree of disappointment in the difference between the true response and the learner's prediction. To generalize to the multiclass case, we adopt the multiclass AdaBoost exponential loss [24], $J(\mathbf{F}(\mathbf{x})) = \exp(-\frac{1}{K} \mathbf{y}^T \mathbf{F}(\mathbf{x}))$, a strictly convex function that has successfully generalized the original AdaBoost algorithm to the multiclass case. Our goal is to minimize the empirical loss. To this end, we use quasi-Newton steps to derive the multiclass extension of the Gentle AdaBoost algorithm, as shown in the following result.

RESULT 1. *The GAMBLE algorithm, in the population version, uses quasi-Newton steps for minimizing the multiclass exponential loss $E[\exp(-\frac{1}{K} \mathbf{y}^T \mathbf{F}(\mathbf{x})) | \mathbf{x}]$.*

Derivation. Using the **weighted conditional expectation** [11], $E_w[\mathbf{g}(\mathbf{x}, \mathbf{y})] \stackrel{\text{def}}{=} \frac{E[w(\mathbf{x}, \mathbf{y})\mathbf{g}(\mathbf{x}, \mathbf{y}) | \mathbf{x}]}{E[w(\mathbf{x}, \mathbf{y}) | \mathbf{x}]}$, where

weight $w = w(\mathbf{x}, \mathbf{y}) = \exp(-\frac{1}{K} \mathbf{y}^T \mathbf{F}(\mathbf{x}))$, we can derive the quasi-Newton update $\mathbf{f} = (f_1, \dots, f_K)$,

$$(3.1) \quad f_j(\mathbf{x}) \leftarrow K \cdot \frac{E_w[y_j | \mathbf{x}]}{E_w[y_j^2 | \mathbf{x}]}$$

See detailed derivation of this result in Appendix. \square

Result 1 translates directly into the Gentle Adaptive Multiclass Boosting Learning (GAMBLE) algorithm (see Algorithm 1). Note that in step (2c), the weak learner is 'centralized' in each dimension such that it satisfies the symmetric constraint $\sum_{k=1}^K f_k^{(m)} = 0$. This does not affect the weight update due to the encoding scheme of the response, because,

$$\begin{aligned} \mathbf{y}^T \mathbf{f}^{(m)}(\mathbf{x}) &= \sum_{k=1}^K y_k f_k^{(m)} = \sum_{k=1}^K y_k r_k^{(m)} \\ &- \frac{1}{K} \sum_{k=1}^K y_k \sum_{k=1}^K r_k^{(m)}(x) = \mathbf{y}^T \mathbf{r}^{(m)}(\mathbf{x}). \end{aligned}$$

Also note that our choice of multiclass exponential loss is valid, because in fact the population minimizer for the expected loss agrees with the Bayes optimal classification rule, as shown by the following theorem:

THEOREM 3.1. *With exponential loss $\exp(-\frac{1}{K} \mathbf{y}^T \mathbf{f}(x))$,*

$$\arg \max_k r_k(\mathbf{x}) = \arg \max_k f_k(\mathbf{x}) = \arg \max_k Pr(c = k | \mathbf{x}).$$

Proof. The constrained minimization problem for the empirical loss can be formulated as:

$$\begin{aligned} \arg \min_{\mathbf{f}(\mathbf{x})} \quad & \mathbf{E}_{\mathbf{Y} | \mathbf{x}} \exp(-\frac{1}{K} \mathbf{Y}^T \mathbf{f}(\mathbf{x})) \\ \text{subject to} \quad & f_1 + \dots + f_K = 0. \end{aligned}$$

The population minimizer can be found by the Lagrange multiplier (c.f. [24], details omitted in interest of space),

$$f_k = (K-1) \left(\log Pr(c = k|\mathbf{x}) - \frac{1}{K} \sum_{j=1}^K \log Pr(c = j|\mathbf{x}) \right)$$

Thus $\arg \max_k f_k(\mathbf{x}) = \arg \max_k Pr(c = k|\mathbf{x})$. Obviously, $\arg \max_k r_k(\mathbf{x}) = \arg \max_k f_k(\mathbf{x})$.

When implemented on data, the weighted expectation in (3.1) can be substituted by weighted regression methods. As an ensemble learning algorithm, the general GAMBLE algorithm may employ an arbitrary regressor as a weak learner. Since most boosting methods work well with ‘small’ classification trees as weak learners, we implement the general GAMBLE algorithm with weighted regression trees [5] in our case.

3.2 The GAMBLE Algorithm Using Weighted Regression Trees Boosting algorithms have been predominantly implemented using classification and regression trees. We first extend the regression tree method with the weighted least squares criterion. Then we explore the relationship of the two regression functions in the general GAMBLE algorithm, and propose an efficient implementation that requires only one regression.

3.2.1 Weighted Regression Trees Since weak learners work with weighted samples in boosting methods, we extend the classical regression tree method [13, 5] to the weighted version with the weighted least squares (WLS) criterion.

Suppose our model partitions the entire space (spanned by the variables X_1, X_2, \dots, X_D) into J regions R_1, R_2, \dots, R_J . In each region R_j , the model produces a constant response \mathbf{c}_{R_j} as the fitted value:

$$(3.2) \quad \mathbf{g}(\mathbf{x}) = \sum_{j=1}^J \mathbf{c}_{R_j} I[\mathbf{x} \in R_j]$$

\mathbf{c}_{R_j} is chosen by minimizing the WLS criterion, i.e.,

$$(3.3) \quad \mathbf{c}_{R_j} = \arg \min_{\mathbf{c}} \sum_{\mathbf{x}_l \in R_j} w_l (\mathbf{y}_l - \mathbf{c})^T (\mathbf{y}_l - \mathbf{c})$$

By taking the first order derivative and setting it to zero, \mathbf{c}_{R_j} is the weighted mean of the true responses,

$$(3.4) \quad c_{k,R_j} = \frac{\sum_{\mathbf{x}_l \in R_j} w_l y_{k,l}}{\sum_{\mathbf{x}_l \in R_j} w_l} \quad (k = 1, 2, \dots, K)$$

In terms of the WLS criterion, seeking an optimal binary regression tree is inhibitedly expensive. CART

[5], however, grows the tree in a top-down and greedy fashion and thus dramatically reduces the search space. Specifically, only one variable is taken into account in each splitting step, thus the splitting hyperplane is always orthogonal to the axis of the corresponding variable. More precisely, each splitting step is a decision making process: the region R is split into two subregions R_1 and R_2 , where the points satisfying the decision rule $X_j \leq h$ should be grouped into R_1 and otherwise R_2 . The parameters (j, h) in the decision rule are chosen such that they also minimize the WLS criterion:

$$(3.5) \quad (j, h) = \arg \min_{(j, h)} \left[\sum_{\mathbf{x}_l \in R_1(j, h)} w_l (\mathbf{y}_l - \mathbf{c}_{R_1(j, h)})^T (\mathbf{y}_l - \mathbf{c}_{R_1(j, h)}) + \sum_{\mathbf{x}_l \in R_2(j, h)} w_l (\mathbf{y}_l - \mathbf{c}_{R_2(j, h)})^T (\mathbf{y}_l - \mathbf{c}_{R_2(j, h)}) \right]$$

This decision rule serves as the splitting criterion for growing the weighted regression tree. We do not use more sophisticated strategies such as pruning by cross validation, since boosting strong hypotheses may lead to overfitting. Empirically, GAMBLE performs reasonably well using regression trees with about 15 leaves.

3.2.2 GAMBLE with Weighted Regression Trees In the general GAMBLE algorithm proposed earlier, two regression functions need to be fitted to the weighted samples in each iteration. We propose an implementation of the GAMBLE algorithm using weighted regression trees, which only requires one regression process. The key to this efficient solution lies in that \mathbf{y} only takes on categorical responses. Theorem 3.2 reveals the structural relationship of the two regression trees.

THEOREM 3.2. *The topological structure of the weighted regression trees with respect to \mathbf{y} and \mathbf{z} ($z_j = y_j^2, j = 1, \dots, K$) is identical.*

COROLLARY 3.1. *The regression functions \mathbf{h} and \mathbf{g} satisfy*

$$\mathbf{h}(\mathbf{x}) = \frac{K-2}{K-1} \mathbf{g}(\mathbf{x}) + \frac{1}{K-1}$$

Proof. See Appendix for the proofs.

Theorem 3.2 implies that we only need to grow a regression tree T_1 with respect to \mathbf{y} , and the regression tree T_2 with respect to \mathbf{z} has exactly the same structure as T_1 . Formally, $\mathbf{h}(\mathbf{x})$ satisfies the same form as $\mathbf{g}(\mathbf{x})$,

$$(3.6) \quad \mathbf{h}(\mathbf{x}) = \sum_{j=1}^J \mathbf{d}_{R_j} I[\mathbf{x} \in R_j]$$

Moreover, within the same region, Corollary 3.1 suggests that there is a simple way to transform the

Gentle Adaptive Multiclass Boosting Learning (GAMBLE)
Using Weighted Regression Trees

1. Initialization: set the observation weights $w_i \leftarrow \frac{1}{N}$, $i = 1, 2, \dots, N$, and set $\mathbf{F}(\mathbf{x}) \leftarrow 0$.
2. For $m \leftarrow 1$ to M :
 - (a) Using weight distribution $\{w_i\}$, fit a weighted regression tree $\mathbf{g}^{(m)}(\mathbf{x})$ of \mathbf{y} to \mathbf{x} .
 - (b) Set $\mathbf{r}^{(m)}(\mathbf{x}) = K \frac{(K-1)\mathbf{g}^{(m)}(\mathbf{x})}{(K-2)\mathbf{g}^{(m)}(\mathbf{x})+1}$.
 - (c) Obtain weak learner $f_j^{(m)}(\mathbf{x}) = r_j^{(m)}(\mathbf{x}) - \frac{1}{K} \sum_{k=1}^K r_k^{(m)}(\mathbf{x})$, $j = 1, 2, \dots, K$.
 - (d) Update committee $\mathbf{F}(\mathbf{x}) \leftarrow \mathbf{F}(\mathbf{x}) + \mathbf{f}^{(m)}(\mathbf{x})$.
 - (e) Update weights $w_i \leftarrow w_i \exp(-\frac{1}{K} \mathbf{y}_i^T \mathbf{f}^{(m)}(\mathbf{x}_i))$, $i = 1, 2, \dots, N$ and renormalize.
3. Output committee classifier: $C(\mathbf{x}) = \arg \max_k F_k(\mathbf{x})$.

Algorithm 2: The Gentle Adaptive Multiclass Boosting Learning (GAMBLE) algorithm using regression trees.

regression value from the regression function $g(\mathbf{x})$ to the (un-centralized) weak learner $r(\mathbf{x})$ (see Figure 1).

Theorem 3.2 and Corollary 1 clarify the relationship between the two regression functions based on the WLS criterion, thus yielding a more efficient solution (Algorithm 2). Obviously, Algorithm 2 works twice as fast as Algorithm 1 since it fits only one regression function in each step. In what follows, we refer to Algorithm 2 as GAMBLE unless stated otherwise.

3.3 Remarks on the GAMBLE Algorithm We characterize the relationship of the GAMBLE algorithm with other boosting variants, and remark on its superiority over other multiclass boosting alternatives.

First, GAMBLE is a natural extension of Gentle AdaBoost, and they are identical up to a factor of 2 in the binary case. The difference comes from the different coefficients in the loss functions. In this regard, Gentle AdaBoost is the binary version of GAMBLE. Also,

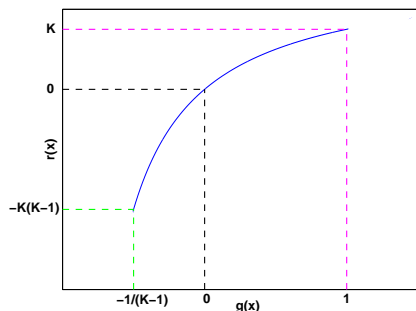


Figure 1: The concave transformation of the regression function $g(x)$ to the (un-centralized) weak learner $r(x)$.

GAMBLE retains the succinct form of Gentle AdaBoost when generalizing to the multiclass case.

Second, the same argument about numerical stability for Gentle AdaBoost and AdaBoost.Real [11] holds when we compare GAMBLE and SAMME.Real. Specifically, we reexamine how GAMBLE obtains a weak classifier from a regression function in step (2b) (Algorithm 2). Result 2 and Result 3 imply that the regression functions and the weak classifiers are both bounded.

RESULT 2. *The regression function $\mathbf{g}(\mathbf{x})$ satisfies $\sum_{k=1}^K g_k^{(m)}(\mathbf{x}) = 0$; also, $\dim(\{\mathbf{g}^{(m)}\}) = K - 1$.*

RESULT 3. *For any \mathbf{x} and dimension $k \in \{1, 2, \dots, K\}$, $g_k^{(m)}(\mathbf{x}) \in [-\frac{1}{K-1}, 1]$ and $r_k^{(m)}(\mathbf{x}) \in [-K(K-1), K]$; the weak learner $f_k^{(m)}(\mathbf{x}) \in [-(K^2 - K), K^2 - K]$.*

Proof. The derivation follows from the proof of Theorem 3.2. Details omitted due to space constraints.

SAMME.Real, on the other hand, relies on the logit-transform of the posterior probability, estimated by the training samples within a region. Due to the scarcity of data, it is highly likely that no training samples of a particular class appear in this region. Thus the posterior probability is estimated to be zero. As such, the logit will be $-\infty$, making the weak learner explode. Even when smoothing methods are used to account for unseen data, the logit is still large, and the weight updates are exponentially larger. In later iterations, the weights of training samples differ by orders of magnitude and result in numerical instability. The larger the K , the more severe the problem will be. Thus it explains the numerical stability of GAMBLE compared to SAMME.Real.

Besides the numerical stability issue, the key difference between GAMBLE and SAMME is how the multiclass loss is optimized. Although the same loss function is used, GAMBLE adopts adaptive quasi-Newton steps whereas SAMME.Real directly optimizes the Lagrange, the latter of which being more aggressive.

Finally, compared to Gentle AdaBoost.MH, GAMBLE is a natural extension for Gentle AdaBoost to the multiclass case. Moreover, GAMBLE is K times more efficient than Gentle AdaBoost.MH. Suppose a regression tree with depth d is used as the weak learner, the complexity of building it is $O(dDn \log(n))$, where D is the dimension of the input and n is the size of the training dataset. Note that GAMBLE builds only one regression tree per iteration, regardless of the number of fitted values. Thus its total cost is $O(MdDn \log(n))$, as compared to Gentle AdaBoost.MH's $O(KMdDn \log(n))$, where M is the number of iterations.

4 Query By Committee Style Active Learning

In practice, there are several learning settings where active learning is particularly valuable. First, labeled data is expensive to obtain due to the cost of human annotation, whereas unlabeled data is typically abundant. The cost of labeling can be minimized if the learner is able to learn from limited labeled data while actively querying only the most informative unlabeled instances. Second, if a tremendous amount of labeled data is available, boosting on the entire training dataset for even a small number of iterations can take large amount of time. Learning an optimal hypothesis by running a large number of iterations can be computationally infeasible. *Active learning* continuously augments the training dataset with the most informative instances at each step, thereby allowing the learner to focus on the ‘best’ portion of the training data. Thus it not only relieves the learner of redundant training samples, but it can also dispose of noisy samples.

In either case, Query By Committee (QBC) [22] is an effective active learning strategy. The general QBC method reduces the error of the learner by choosing to label the sample that splits the version space into two parts of comparable size. As such QBC theoretically has near-optimal data efficiency, which is logarithmic in the probability of error [9]. Such a result in the general QBC framework is however based on the assumption of using Gibbs algorithm as the component learner, generally intractable in practice.

In Active GAMBLE (algorithm 3), we adopt the same QBC sample selection method as in QBoost [1], and extend it to the multiclass case. In each trial, Active GAMBLE trains a committee using GAMBLE as the component learner, based on the current working

Active GAMBLE

1. Initialization: randomly choose a small set of samples from the query set $Q = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ to the working set S .
2. For $l \leftarrow 1$ to L :
 - (a) Train a committee $C_l = \text{GAMBLE}(S)$.
 - (b) Test C_l on data set $T = Q - S$.
 - (c) For each $\mathbf{x} \in T$, compute utility $u(\mathbf{x}) = [F_k(\mathbf{x}) - F_j(\mathbf{x})]^{-1}$, where $k = \arg \max_k F_k(\mathbf{x})$ and $j = \arg \max_{j \neq k} F_j(\mathbf{x})$.
 - (d) Let set D be the R samples $\mathbf{x} \in T$ with the highest utility, $Q \leftarrow Q - D$ and $S \leftarrow S \cup D$.
 - (e) If Q is empty, let $C_L \leftarrow C_l$ and exit loop.
3. Output the classifier: C_L .

Algorithm 3: The Active GAMBLE algorithm.

set rather than the entire training data. It then evaluates the potential utility of the unlabeled samples in the query set and selects the top R samples with the highest utility into the working set. *Utility* is here defined as the reciprocal of the difference between the committee’s output for the most and the second most popular class label. By Theorem 3.1, the committee’s output can be regarded as the surrogate of the posterior probability. As such utility measures the uncertainty in the committee about its predicted label, i.e. a sample has higher utility when the committee is less certain about its label. [20] has shown that the hypothesis having a larger margin generalizes better on unseen data. Since the definition of utility here is similar to the margin formulation in [20, 1] we expect that Active GAMBLE generalizes well by selecting samples with high utility or small margin, thereby increasing the margin in the hypothesis. Empirical results in Section 5.2 suggest that Active GAMBLE can potentially achieve higher classi-

Table 1: UCI benchmark datasets. K is the number of classes and D is the number of feature dimensions.

Dataset	K	D	#Train	#Test
Glass	6	9	113	101
Pendigits	10	16	7,494	3,498
Satimage	6	36	4,435	2,000
Segmentation	7	19	210	2,100
Soybean	19	35	307	376
Splice	3	60	2,860	316
Vehicle	4	18	422	422
Vowel	11	10	528	462

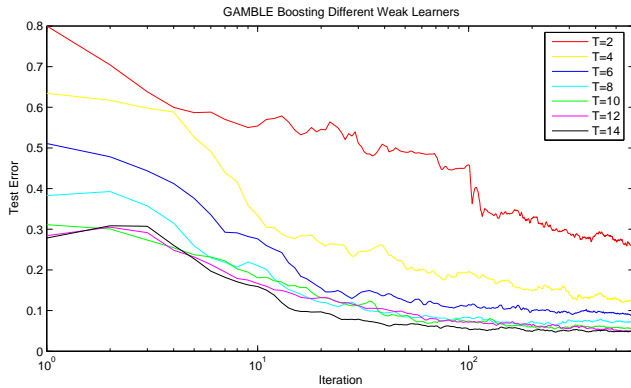


Figure 2: GAMBLE boosting regression trees of different sizes in the Pendigits dataset. T denotes the number of leaves in the regression tree. Note that the iteration number is shown on a log scale.

fication accuracies. The results also show that Active GAMBLE greatly improves data efficiency and creates simpler model, therefore reduces the training and testing time. For very large training datasets, one can further reduce the training time of Active GAMBLE by only evaluating the utility of a random portion of instances in the query set.

5 Experimental Results

We conduct a series of experiments on both simulated and real world data to showcase the performance of the algorithm in various multiclass classification problems. We choose the SAMME.Discrete algorithm¹ (SAMME for short) in our experiments for comparison for several reasons. First, it has been shown that SAMME outperforms the most popular multiclass boosting algorithm AdaBoost.MH, in terms of both accuracy and efficiency [24]. In addition, GAMBLE and SAMME can be treated as variants because the same multiclass loss function and K -class response encoding scheme are used in both algorithms. Note that unlike SAMME which uses the Lagrange multiplier for direct optimization, GAMBLE uses the quasi-Newton steps to minimize the exponential loss. Our goal in this paper is to provide a meaningful and efficient multiclass extension of Gentle AdaBoost, one of the best binary boosting algorithms.

5.1 Experiments on Real World Data We tested our algorithm on a collection of benchmark datasets (Table 1) available from the UCI Machine Learning Repository² [17]. We intentionally choose the same

¹SAMME.Discrete is numerically more stable than SAMME.Real, though empirical results have shown that they perform almost equally well [24].

²<http://www.ics.uci.edu/~mllearn/MLRepository.html>

multiclass datasets as in [21] so readers can compare our results with other multiclass boosting variants such as discrete and real AdaBoost.MH, AdaBoost.MR, etc. These real world datasets cover a wide variety of multiclass classification scenarios.

Figure 2 shows how GAMBLE performs when boosting regression trees of different sizes. Boosting stumps requires a lot more boosting steps than boosting sophisticated trees, in order to reach a desirable level of prediction error. Interestingly, the gap between test errors becomes more narrow when boosting relatively strong learners. For instance, the test errors of boosting trees with the number of leaves 10, 12 and 14 are almost the same after 600 iterations. Although boosting sophisticated learners may require fewer iterations for the convergence of the test error, each iteration takes longer training time and it may also risk overfitting the training data. Therefore we need to strike a balance in the size of the trees.

The test results of GAMBLE and SAMME in the UCI datasets are summarized in Figure 3. GAMBLE generally incurs less test error rates at 10, 100 and 1,000 iterations. The sign test reveals that the zero median hypothesis can be rejected ($p = 0.0066$) at the significance level 1%. In other words, GAMBLE performs statistically better than SAMME in these datasets. Figure 4 illustrates the test errors of GAMBLE and SAMME in the *Glass* and *Segmentation* datasets. Both datasets, compared to their dimensions, have small number of training samples for each class. GAMBLE however can handle these sparse datasets and generalize well.

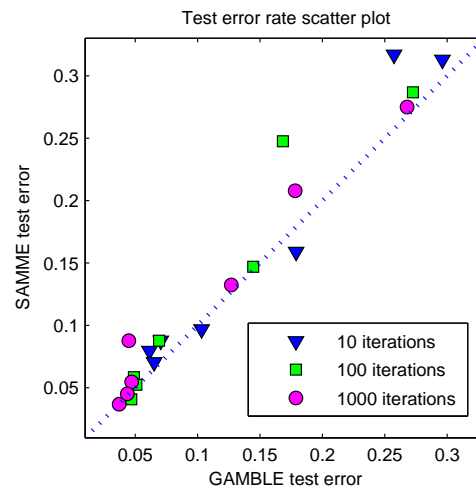


Figure 3: Test error scatter plot for GAMBLE and SAMME at 10, 100 and 1,000 iterations in 8 UCI multiclass datasets. Each point shows the error rate of GAMBLE and SAMME on a single benchmark dataset.

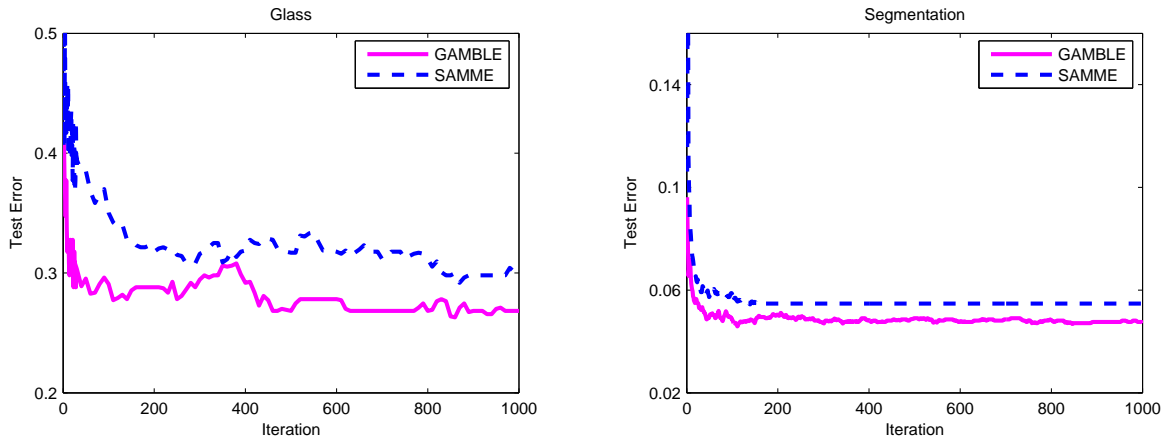


Figure 4: Test error (in the range of $[0, 1]$) for GAMBLE and SAMME.Discrete on two benchmark datasets.

5.2 Active Learning Performance We evaluate the effectiveness of the active learning strategy by test error, training time and data efficiency³. As a baseline algorithm, we implement the GAMBLE algorithm with random sampling, which we refer to as *Random GAMBLE* henceforth. Random GAMBLE selects training samples randomly from the training set regardless of their utility. The results of Random GAMBLE are averaged over 10 runs. The GAMBLE algorithm which uses the entire training dataset in each boosting step is referred to as *Batch GAMBLE*. For each dataset, the lowest error rate incurred by Batch GAMBLE throughout the first 1,000 iterations is set as the target error rate.

We present the results for two datasets which illustrate the characteristic learning curves of the active learning strategies. Figure 5 (left) shows that in the *Pendigits* dataset, Active GAMBLE quickly reaches the target error rate (4.7%) using about 300 samples and plateaus afterwards. Random GAMBLE converges to the same level of error rate using 1,000 samples. Figure 5 (right) shows a more interesting active learning curve. Active GAMBLE uses as few as 220 training samples (5% of all training data) to reach the target error rate (12.6%), while Random GAMBLE requires at least six times as many training samples. Interestingly, Active GAMBLE continues to reduce prediction error by about 2% after it reaches the target error rate. This implies that by using the 600 most informative samples (13.5% of all training data), Active GAMBLE is able to achieve higher classification accuracy. After this point, however, adding more training instances does not im-

³Data efficiency is defined as the fraction of training samples used to achieve the target error rate.

prove the accuracy and eventually the prediction error converges to that of Random GAMBLE.

Table 2 summarizes the results for the UCI datasets. By collecting the most informative instances, Active GAMBLE achieves far better **data efficiency** than Random GAMBLE, especially in the large training datasets, *Pendigits*, *Satimage* and *Splice*. This implies that those datasets contain redundant training samples. Batch GAMBLE requires global weight updates at each boosting step, whereas the component learner in Active GAMBLE only performs weight updates on a fraction of the training samples. Thus Active GAMBLE has faster **training time** than that of Batch GAMBLE, especially when high data efficiency is achieved. In small datasets such as *Glass*, *Segmentation* and *Soybean*, practically all training data is needed to converge to the target error rate. Batch GAMBLE works faster in these cases without having to search for the most informative samples. Table 2 also presents the lowest **test error rate** of Active GAMBLE, which in most cases is lower than the target error rate. In practice one can retain a portion of the training samples as a hold-out dataset to determine the early stopping point in training. This simpler model trained by less number of samples can then be used for prediction.

5.3 Simulation Study We tested GAMBLE with a popular synthetic example in boosting literature.

The Concentric Spheres Example The concentric spheres example has been used in [11] to showcase the performance of different boosting algorithms. Since geometrically complex classification boundaries are involved in this example, it is regarded to be more complicated than practical classification problems. In this

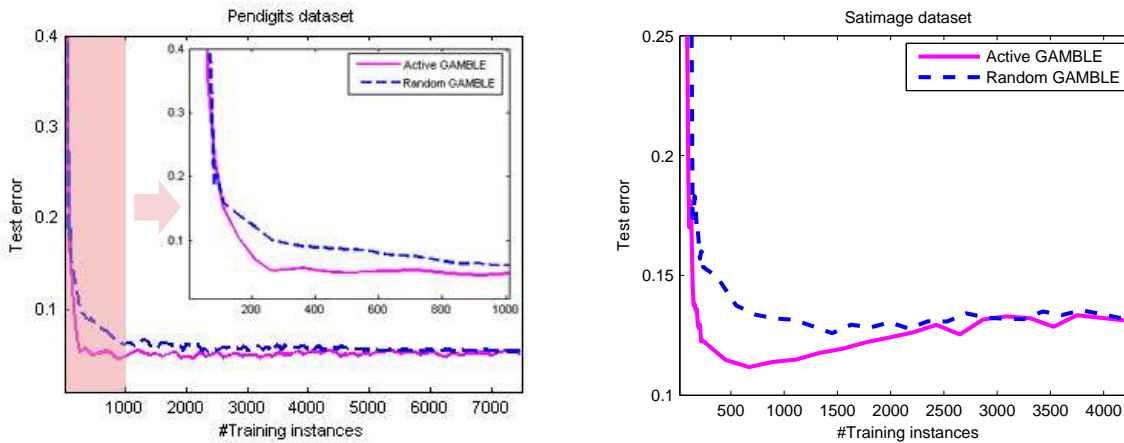


Figure 5: Test error of Active GAMBLE and Random GAMBLE in the *Pendigits* (left) and *Satimage* (right) datasets. Active GAMBLE incurs lower test error and uses fewer training samples than Random GAMBLE. Note that the test error curves for the first 1000 samples are zoomed in for the *Pendigits* dataset.

Table 2: Data efficiency and training time. *Lowest error rate* is the minimum error rate incurred by Active GAMBLE. *Target error rate* is the lowest error rate incurred by Batch GAMBLE in the first 1,000 iterations.

Dataset	Target Error Rate (%)	Lowest Error Rate (%)	# Training samples (pct.)			Training time (sec.)	
			Total Samples	Active GAMBLE	Random GAMBLE	Active GAMBLE	Batch GAMBLE
Glass	27.5%	27.5%	113	91 (80.5%)	102 (90.3%)	67	28
Pendigits	4.71%	3.92%	7,494	340 (4.39%)	2,510 (33.5%)	1,698	10,043
Satimage	12.6%	11.3%	4,435	220 (4.96%)	1,450 (32.7%)	362	6,066
Segmentation	4.62%	4.43%	210	135 (64.3%)	190 (90.5%)	158	72
Soybean	5.65%	5.25%	307	245 (79.8%)	300 (97.7%)	266	87
Splice	4.48%	3.42%	2,860	430 (15.0%)	970 (33.9%)	320	470
Vehicle	27.3%	25.1%	422	115 (27.3%)	225 (53.3%)	127	310
Vowel	47.0%	41.8%	528	155 (36.7%)	205 (48.6%)	136	367

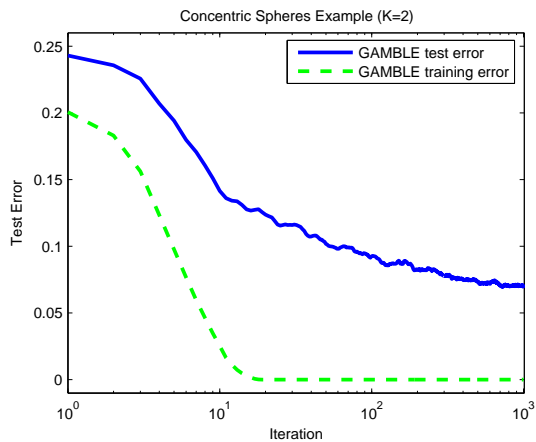
example, the samples are drawn from a ten-dimensional Gaussian distribution, i.e. $\mathbf{x} \sim N^{10}(\mathbf{0}, \mathbf{I})$. Each class is defined by thresholding the radius of the sphere $C_k = \{\mathbf{x}_i | t_{k-1} \leq r_i^2 \leq t_k\}$, where $r_i = \sqrt{\sum_{j=1}^{10} x_j^2}$ is the radius from the origin, $t_0 = 0$ and $t_k = \infty$. $\{t_k\}_{k=1}^K$ are chosen such that approximately equal number of instances are placed in each class. $K \cdot 1000$ instances are used as training samples, and an independently drawn set of 10,000 samples are used for testing.

Figure 6(a) represents the training and test error of GAMBLE when $K=2$, and it reveals an impressive aspect of GAMBLE. Even after the training error reaches zero at about 20 iterations, GAMBLE continues to fit base classifiers to the training samples and the test error keeps dropping thereafter without indications of overfitting. This can be interpreted as GAMBLE increases the margin of the training samples [20] even when they are all correctly classified. Eventually this

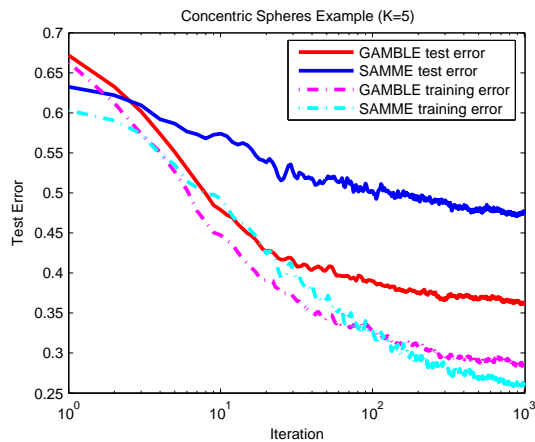
results in better generalization on the test data.

Figure 6(b) shows the training and test error curves of GAMBLE and SAMME. This figure demonstrates significant difference between the two algorithms. GAMBLE outperforms SAMME by incurring 10% less test error, which is consistent with the findings in [11]. Note that after 120 iterations, SAMME has lower training error whereas GAMBLE has substantially lower test error than SAMME. We can explain this by the inherent ‘gentle’ nature of the GAMBLE algorithm, which places less emphasis on samples that are hard to learn. This small sacrifice in training error actually translates to better generalization power in prediction. But in this complicated classification problem, SAMME overemphasizes the outliers of the Gaussian distribution and drives the weak learners to fit them. As such this effects prediction performance. This further confirms explanations in Section 3.3.

This example elicits two important aspects of GAM-



(a) Concentric spheres example (K=2).



(b) Concentric spheres example (K=5).

Figure 6: GAMBLE’s training and test errors in the concentric spheres example. Iteration number is shown on the log scale, highlighting the significant difference in training error iterations to those of testing.

BLE. GAMBLE continuously increases the margin regardless of the training error and is less sensitive to outliers. Both of these are crucial to prediction error reduction. In practice, classification problems are less noisy and less complex than this example and the difference in performance is not as much pronounced.

6 Conclusion

This work extends the successful Gentle AdaBoost algorithm to the multiclass case. The proposed GAMBLE algorithm follows closely the rationale of other boosting predecessors. It fits one weighted weak learner symmetrically for all classes in each boosting step and combines them into a powerful committee classifier. Compared to other ‘one-against-all’ variants, it reduces the time complexity by K . GAMBLE is numerically stable since all the weak learners are bounded, and is thus more robust to unseen data. Also, the only parameter required in GAMBLE is the complexity of the weak learner (e.g. the number of leaves in the regression tree). As such it demands minimum domain knowledge about the classification problem. Our experiments demonstrate that GAMBLE does not seem to overfit the training data after a large number of boosting steps, even after the training error reaches zero. To accelerate learning in large-scale datasets, we use active learning to select the most informative samples for training. Compared to Batch GAMBLE, Active GAMBLE is able to achieve even higher classification accuracy, using only a portion of the training samples and consuming far less training time. Our empirical results indicate that the proposed GAMBLE algorithm outperforms the state-of-the-art multiclass boosting methods.

Several issues emerge and are worthy of further investigation. It is of theoretical interest to show that the generalization error of GAMBLE is upper bounded and independent of the number of boosting steps, and Schapire’s result [8] may prove useful in this case. Additionally, the classes are more likely to be imbalanced in K -class classification in practice. We intend to improve our algorithm to handle rare classes [14] and the multiclass cost-sensitive learning problem [2]. It is also worthwhile to investigate the performance of boosting different types of regressors. Decision trees used in boosting methods are discrete and the changes in sample weights may dramatically alter the topology of the trees. Boosting other base classifiers, such as component-wise smoothing splines [6] and KNN [4] may yield even better empirical results.

Acknowledgments

The authors acknowledge partial support from the National Science Foundation (NSF).

References

- [1] N. Abe and H. Mamitsuka. Query learning strategies using boosting and bagging. In *Proc. of the 15th International Conference on Machine Learning*, 1998.
- [2] N. Abe, B. Zadrozny, and J. Langford. An iterative method for multi-class cost-sensitive learning. In *Proc. of the 10th ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 2004.
- [3] E. Allwein, R. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *J. Machine Learning Res.*, 1:113–141, 2000.

- [4] V. Athitsos and S. Sclaroff. Boosting nearest neighbor classifiers for multiclass recognition. In *IEEE Workshop on Learning in Comp. Vision & Pat. Recog.*, 2005.
- [5] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, 1984.
- [6] P. Buhmann and B. Yu. Boosting with the l2 loss: Regression and classification. *Journal of the American Statistical Association*, 98(462):324–339, 2003.
- [7] T. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Journal of Machine Learning Research*, 40(2):139–158, 2000.
- [8] 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, August 1997.
- [9] Y. Freund, H. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28:133–168, 1997.
- [10] J. Friedman. Another approach to polychotomous classification. Technical report, Stanford Univ., 1996.
- [11] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 28:337–407, 2000.
- [12] T. Hastie and R. Tibshirani. Classification by pairwise coupling. *The Annals of Statistics*, 26:451–471, 1998.
- [13] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, 2001.
- [14] M. Joshi, R. C. Agarwal, and V. Kumar. Predicting rare classes: Can boosting make any weak learner strong? In *Proc. of the 8th ACM KDD*, 2002.
- [15] Y. Lin. A note on margin-based loss functions in classification. *Stats. and Prob. Letters*, 68(1), 2004.
- [16] R. Meir and G. Ratsch. An introduction to boosting and leveraging. *Adv. Lect. on Machine Learning*, 2003.
- [17] C. Merz and P. Murphy. UCI repository of machine learning databases.
- [18] R. Schapire. Using output codes to boost multiclass learning problems. In *Proc. of the 14th International Conference on Machine Learning*, 1997.
- [19] R. Schapire. The boosting approach to machine learning: an overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.
- [20] R. Schapire, Y. Freund, P. Barlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *Proc. of the 14th Int'l Conf. on Machine Learning (ICML)*, 1997.
- [21] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated prediction. *Machine Learning*, 37(1):297–336, 1999.
- [22] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proc. of the 5th Annual Workshop on Computational Learning Theory*, 1992.
- [23] V. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, Inc., New York, 1998.
- [24] J. Zhu, S. Rosset, H. Zhou, and T. Hastie. Multiclass adaboost. Technical Report #430, Department of Statistics, University of Michigan, 2005.

Appendix

A. Derivation of Result 1

Derivation. Given the current committee $\mathbf{F}(\mathbf{x})$, we calculate the improved update $\mathbf{f}(\mathbf{x})$. Consider the expected multiclass exponential loss J ,

$$(A-7) \quad \begin{aligned} & E[J(\mathbf{F}(\mathbf{x}) + \mathbf{f}(\mathbf{x}))|\mathbf{x}] \\ &= E[\exp(-\frac{1}{K}\mathbf{y}^T(\mathbf{F}(\mathbf{x}) + \mathbf{f}(\mathbf{x})))|\mathbf{x}] \end{aligned}$$

- (i) Conditioning on input \mathbf{x} , we compute the first and the second order derivatives of the exponential loss at $\mathbf{f}(\mathbf{x}) = \mathbf{0} = (0, 0, \dots, 0)^T$. The first order derivative with respect to the j -th dimension is:

$$\begin{aligned} s_j(\mathbf{x}) &= \left. \frac{\partial(J(\mathbf{F}(\mathbf{x}) + \mathbf{f}(\mathbf{x})))}{\partial f_j(\mathbf{x})} \right|_{\mathbf{f}(\mathbf{x})=\mathbf{0}} \\ &= E\left[-\frac{1}{K}y_j \exp(-\frac{1}{K}\mathbf{y}^T(\mathbf{F}(\mathbf{x}) + \mathbf{f}(\mathbf{x})))|\mathbf{x}\right] \Big|_{\mathbf{f}(\mathbf{x})=\mathbf{0}} \\ &= E\left[-\frac{1}{K}y_j \exp(-\frac{1}{K}\mathbf{y}^T\mathbf{F}(\mathbf{x}))|\mathbf{x}\right] \end{aligned}$$

Likewise, the Hessian is:

$$\begin{aligned} H_{j,k}(\mathbf{x}) &= \left. \frac{\partial^2(J(\mathbf{F}(\mathbf{x}) + \mathbf{f}(\mathbf{x})))}{\partial f_j(\mathbf{x})\partial f_k(\mathbf{x})} \right|_{\mathbf{f}(\mathbf{x})=\mathbf{0}} \\ &= E\left[\frac{1}{K^2}y_j y_k \exp(-\frac{1}{K}\mathbf{y}^T\mathbf{F}(\mathbf{x}))|\mathbf{x}\right] \end{aligned}$$

- (ii) The quasi-Newton update is used to reduce the substantial computation overhead of Newton's method, the former of which uses the diagonal approximation of the Hessian, i.e.

$$(A-8) \quad f_j(\mathbf{x}) \leftarrow K \cdot \frac{E[y_j \exp(-\frac{1}{K}\mathbf{y}^T\mathbf{F}(\mathbf{x}))|\mathbf{x}]}{E[y_j^2 \exp(-\frac{1}{K}\mathbf{y}^T\mathbf{F}(\mathbf{x}))|\mathbf{x}]}$$

- (iii) Using the **weighted conditional expectation**,

$$(A-9) \quad E_w[\mathbf{g}(\mathbf{x}, \mathbf{y})] \stackrel{\text{def}}{=} \frac{E[w(\mathbf{x}, \mathbf{y})\mathbf{g}(\mathbf{x}, \mathbf{y})|\mathbf{x}]}{E[w(\mathbf{x}, \mathbf{y})|\mathbf{x}]}$$

where weight $w = w(\mathbf{x}, \mathbf{y}) = \exp(-\frac{1}{K}\mathbf{y}^T\mathbf{F}(\mathbf{x}))$,

(A-8) can be rewritten as

$$\begin{aligned} f_j(\mathbf{x}) &\leftarrow K \cdot \frac{E[y_j w(\mathbf{x}, \mathbf{y})|\mathbf{x}]}{E[y_j^2 w(\mathbf{x}, \mathbf{y})|\mathbf{x}]} \\ &= K \cdot \frac{E[y_j w(\mathbf{x}, \mathbf{y})|\mathbf{x}]/E[w(\mathbf{x}, \mathbf{y})|\mathbf{x}]}{E[y_j^2 w(\mathbf{x}, \mathbf{y})|\mathbf{x}]/E[w(\mathbf{x}, \mathbf{y})|\mathbf{x}]} \end{aligned}$$

Therefore, we express the update as

$$f_j(\mathbf{x}) \leftarrow K \cdot \frac{E_w[y_j|\mathbf{x}]}{E_w[y_j^2|\mathbf{x}]}$$

□

B. Proof of Theorem 3.2

Proof. For simplicity of notations, we provide the proof assuming that the regression function generates one response, and it is relatively straightforward to generalize to the K-response case. Suppose during the regression process with respect to \mathbf{y} , the region R in question is split into two subregions R_1 and R_2 ($R_1 \cup R_2 = R$), subject to the criterion (3.5). In other words, the following inequality holds for any two subregions S_1 and S_2 ($S_1 \cup S_2 = R$):

$$(B-10) \quad \begin{aligned} & \sum_{\mathbf{x}_l \in R_1} w_l (y_l - c_{R_1})^2 + \sum_{\mathbf{x}_l \in R_2} w_l (y_l - c_{R_2})^2 \\ & \leq \sum_{\mathbf{x}_l \in S_1} w_l (y_l - c_{S_1})^2 + \sum_{\mathbf{x}_l \in S_2} w_l (y_l - c_{S_2})^2 \end{aligned}$$

Substituting (3.4) into the weighted squares and canceling out the identical weighted square response $\sum_{\mathbf{x} \in R} w_l y_l^2$ on both sides, (B-10) becomes:

$$(B-11) \quad \begin{aligned} & \frac{\left(\sum_{\mathbf{x}_l \in R_1} w_l y_l \right)^2}{\sum_{\mathbf{x}_l \in R_1} w_l} + \frac{\left(\sum_{\mathbf{x}_l \in R_2} w_l y_l \right)^2}{\sum_{\mathbf{x}_l \in R_2} w_l} \\ & \geq \frac{\left(\sum_{\mathbf{x}_l \in S_1} w_l y_l \right)^2}{\sum_{\mathbf{x}_l \in S_1} w_l} + \frac{\left(\sum_{\mathbf{x}_l \in S_2} w_l y_l \right)^2}{\sum_{\mathbf{x}_l \in S_2} w_l} \end{aligned}$$

Since $y_l \in \{1, -\frac{1}{K-1}\}$, we further partition R_1 into R_1^+ when $y_l = 1$ and otherwise R_1^- ($R_1^+ \cup R_1^- = R_1$). Hence,

$$(B-12) \quad \begin{aligned} \sum_{\mathbf{x}_l \in R_1} w_l y_l &= \sum_{\mathbf{x}_l \in R_1^+} w_l \cdot 1 + \sum_{\mathbf{x}_l \in R_1^-} w_l \cdot \left(-\frac{1}{K-1}\right) \\ &= \sum_{\mathbf{x}_l \in R_1} w_l - \frac{K}{K-1} \sum_{\mathbf{x}_l \in R_1^-} w_l \end{aligned}$$

Applying the same trick to all the components and canceling out the common terms on both sides of (B-11), the following inequality holds:

$$(B-13) \quad \begin{aligned} & \frac{\left(\sum_{\mathbf{x}_l \in R_1^+} w_l \right)^2}{\sum_{\mathbf{x}_l \in R_1^+} w_l} + \frac{\left(\sum_{\mathbf{x}_l \in R_1^-} w_l \right)^2}{\sum_{\mathbf{x}_l \in R_1^-} w_l} \\ & \geq \frac{\left(\sum_{\mathbf{x}_l \in S_1^+} w_l \right)^2}{\sum_{\mathbf{x}_l \in S_1^+} w_l} + \frac{\left(\sum_{\mathbf{x}_l \in S_2^+} w_l \right)^2}{\sum_{\mathbf{x}_l \in S_2^+} w_l} \end{aligned}$$

Note that $z_l = y_l^2 \in \{1, \frac{1}{(K-1)^2}\}$, therefore,

$$(B-14) \quad \begin{aligned} \sum_{\mathbf{x}_l \in R_1} w_l z_l &= \sum_{\mathbf{x}_l \in R_1^+} w_l \cdot 1 + \sum_{\mathbf{x}_l \in R_1^-} w_l \cdot \frac{1}{(K-1)^2} \\ &= \sum_{\mathbf{x}_l \in R_1} w_l - \frac{K^2 - 2K}{(K-1)^2} \sum_{\mathbf{x}_l \in R_1^-} w_l \end{aligned}$$

Also note that

$$(B-15) \quad \begin{aligned} & \frac{\left(\sum_{\mathbf{x}_l \in R_1} w_l z_l \right)^2}{\sum_{\mathbf{x}_l \in R_1} w_l} + \frac{\left(\sum_{\mathbf{x}_l \in R_2} w_l z_l \right)^2}{\sum_{\mathbf{x}_l \in R_2} w_l} \\ & = \sum_{\mathbf{x}_l \in R_1} w_l + \sum_{\mathbf{x}_l \in R_2} w_l - 2 \frac{K^2 - 2K}{(K-1)^2} \left[\sum_{\mathbf{x}_l \in R_1^-} w_l + \sum_{\mathbf{x}_l \in R_2^-} w_l \right] \\ & + \frac{(K^2 - 2K)^2}{(K-1)^4} \left[\frac{\left(\sum_{\mathbf{x}_l \in R_1^-} w_l \right)^2}{\sum_{\mathbf{x}_l \in R_1^-} w_l} + \frac{\left(\sum_{\mathbf{x}_l \in R_2^-} w_l \right)^2}{\sum_{\mathbf{x}_l \in R_2^-} w_l} \right] \end{aligned}$$

Substituting (B-13), we derive this inequality:

$$(B-15) \geq \begin{aligned} & \sum_{\mathbf{x}_l \in R} w_l - 2 \frac{K^2 - 2K}{(K-1)^2} \sum_{\mathbf{x}_l \in R^-} w_l \\ & + \frac{(K^2 - 2K)^2}{(K-1)^4} \left[\frac{\left(\sum_{\mathbf{x}_l \in T_1^-} w_l \right)^2}{\sum_{\mathbf{x}_l \in T_1^-} w_l} + \frac{\left(\sum_{\mathbf{x}_l \in T_2^-} w_l \right)^2}{\sum_{\mathbf{x}_l \in T_2^-} w_l} \right] \\ & = \frac{\left(\sum_{\mathbf{x}_l \in T_1} w_l z_l \right)^2}{\sum_{\mathbf{x}_l \in T_1} w_l} + \frac{\left(\sum_{\mathbf{x}_l \in T_2} w_l z_l \right)^2}{\sum_{\mathbf{x}_l \in T_2} w_l} \end{aligned}$$

Thus we arrive at the following inequality,

$$(B-16) \quad \begin{aligned} & \sum_{\mathbf{x}_l \in R_1} w_l (z_l - d_{R_1})^2 + \sum_{\mathbf{x}_l \in R_2} w_l (z_l - d_{R_2})^2 \\ & \leq \sum_{\mathbf{x}_l \in S_1} w_l (z_l - d_{S_1})^2 + \sum_{\mathbf{x}_l \in S_2} w_l (z_l - d_{S_2})^2 \end{aligned}$$

where d is the weighted mean for the square response.

Hence the optimal splitting variable and value that minimize the weighted least squares with respect to \mathbf{y} also minimize that with respect to \mathbf{z} . Thus, we can reach the conclusion by induction. \square

C. Proof of Corollary 3.1

Proof. Substitute (B-12) into (3.4) and similarly (B-14) into the weighted mean of \mathbf{z} , we derive the following equation:

$$(C-17) \quad \mathbf{d}_R = \frac{1}{K-1} + \frac{K-2}{K-1} \mathbf{c}_R$$

By definition of $\mathbf{g}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$,

$$(C-17) \quad \begin{aligned} \mathbf{h}(\mathbf{x}) &= \frac{K-2}{K-1} \sum_{j=1}^J \mathbf{c}_{R_j} I[\mathbf{x} \in R_j] + \frac{1}{K-1} \sum_{j=1}^J I[\mathbf{x} \in R_j] \\ &= \frac{K-2}{K-1} \mathbf{g}(\mathbf{x}) + \frac{1}{K-1}. \end{aligned}$$

\square