An Empirical Evaluation of Four Algorithms for Multi-Class Classification: Mart, ABC-Mart, Robust LogitBoost, and ABC-LogitBoost

Ping Li
Department of Statistical Science
Faculty of Computing and Information Science
Cornell University
Ithaca, NY 14853
pingli@cornell.edu

Abstract
This empirical study is mainly devoted to comparing four tree-based boosting algorithms: mart, abc-mart, robust logitboost, and abc-logitboost, for multi-class classification on a variety of publicly available datasets. Some of those datasets have been thoroughly tested in prior studies using a broad range of classification algorithms including SVM, neural nets, and deep learning.

In terms of the empirical classification errors, our experiment results demonstrate:
1. Abc-mart considerably improves mart.
2. Abc-logitboost considerably improves (robust) logitboost.
3. (Robust) logitboost considerably improves mart on most datasets.
4. Abc-logitboost considerably improves abc-mart on most datasets.
5. These four boosting algorithms (especially abc-logitboost) outperform SVM on many datasets.
6. Compared to the best deep learning methods, these four boosting algorithms (especially abc-logitboost) are competitive.

1 Introduction
Boosting algorithms [16, 4, 5, 2, 17, 7, 15, 6] have become very successful in machine learning. In this paper, we provide an empirical evaluation of four tree-based boosting algorithms for multi-class classification: mart[6], abc-mart[11], robust logitboost[13], and abc-logitboost[12], on a wide range of datasets.

Abc-boost[11], where “abc” stands for adaptive base class, is a recent new idea for improving multi-class classification. Both abc-mart[11] and abc-logitboost[12] are specific implementations of abc-boost. Although the experiments in [11, 12] were reasonable, we consider a more thorough study is necessary. Most datasets used in [11, 12] are (very) small. While those datasets (e.g., pendigits, zipcode) are still popular in machine learning research papers, they may be too small to be practically very meaningful. Nowadays, applications with millions of training samples are not uncommon, for example, in search engines[14].

It would be also interesting to compare these four tree-based boosting algorithms with other popular learning methods such as support vector machines (SVM) and deep learning. A recent study[9] conducted a thorough empirical comparison of many learning algorithms including SVM, neural nets, and

\[\text{http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DeepVsShallowComparisonICML2007}\]
deep learning. The authors of [9] maintain a nice Web site from which one can download the datasets and compares the test mis-classification errors.

In this paper, we provide extensive experiment results using mart, abc-mart, robust logitboost, and abc-logitboost on the datasets used in [9], plus other publicly available datasets. One interesting dataset is the UCI Poker. By private communications with C.J. Lin (the author of LibSVM), we learn that SVM achieved a classification accuracy of $\leq 60\%$ on this dataset. Interestingly, all four boosting algorithms can easily achieve $> 90\%$ accuracies.

We try to make this paper self-contained by providing a detailed introduction to abc-mart, robust logitboost, and abc-logitboost in the next section.

2 LogitBoost, Mart, Abc-mart, Robust LogitBoost, and Abc-LogitBoost

We denote a training dataset by $\{y_i, x_i\}_{i=1}^N$, where $N$ is the number of feature vectors (samples), $x_i$ is the $i$th feature vector, and $y_i \in \{0, 1, 2, \ldots, K-1\}$ is the $i$th class label, where $K \geq 3$ in multi-class classification.

Both logitboost[7] and mart (multiple additive regression trees)[6] algorithms can be viewed as generalizations to logistic regression, which assumes class probabilities $p_{i,k}$ as

$$
p_{i,k} = \Pr(y_i = k | x_i) = \frac{e^{F_{i,k}(x_i)}}{\sum_{s=0}^{K-1} e^{F_{i,s}(x_i)}}. \tag{1}
$$

While traditional logistic regression assumes $F_{i,k}(x_i) = \beta^T x_i$, logitboost and mart adopt the flexible “additive model,” which is a function of $M$ terms:

$$
F^{(M)}(x) = \sum_{m=1}^{M} \rho_m h(x; a_m), \tag{2}
$$

where $h(x; a_m)$, the base learner, is typically a regression tree. The parameters, $\rho_m$ and $a_m$, are learned from the data, by maximum likelihood, which is equivalent to minimizing the negative log-likelihood loss

$$
L = \sum_{i=1}^{N} L_i, \quad L_i = -\sum_{k=0}^{K-1} r_{i,k} \log p_{i,k} \tag{3}
$$

where $r_{i,k} = 1$ if $y_i = k$ and $r_{i,k} = 0$ otherwise.

For identifiability, $\sum_{k=0}^{K-1} F_{i,k} = 0$, i.e., the sum-to-zero constraint, is routinely adopted [7, 6, 19, 10, 18, 21, 20].

2.1 Logitboost

As described in Alg. 1 [7] builds the additive model (2) by a greedy stage-wise procedure, using a second-order (diagonal) approximation, which requires knowing the first two derivatives of the loss function (3) with respective to the function values $F_{i,k}$. [7] obtained:

$$
\frac{\partial L_i}{\partial F_{i,k}} = - (r_{i,k} - p_{i,k}), \quad \frac{\partial^2 L_i}{\partial F_{i,k}^2} = p_{i,k} (1 - p_{i,k}). \tag{4}
$$
Those derivatives can be derived by assuming no relations among $F_{i,k}$, $k = 0$ to $K - 1$. However, [7] used the “sum-to-zero” constraint $\sum_{k=0}^{K-1} F_{i,k} = 0$ throughout the paper and they provided an alternative explanation. [7] showed (4) by conditioning on a “base class” and noticed the resultant derivatives are independent of the choice of the base.

Algorithm 1 LogitBoost[7, Alg. 6]. $\nu$ is the shrinkage.

0: $r_{i,k} = 1$, if $y_i = k$, $r_{i,k} = 0$ otherwise.
1: $F_{i,k} = 0$, $p_{i,k} = \frac{1}{K}$, $k = 0$ to $K - 1$, $i = 1$ to $N$
2: For $m = 1$ to $M$ Do
3: For $k = 0$ to $K - 1$, Do
4: Compute $w_{i,k} = p_{i,k}(1 - p_{i,k})$.
5: Compute $z_{i,k} = r_{i,k} - p_{i,k} p_{i,k}(1 - p_{i,k})$.
6: Fit the function $f_{i,k}$ by a weighted least-square of $z_{i,k}$ to $x_i$ with weights $w_{i,k}$.
7: $F_{i,k} = F_{i,k} + \nu \frac{K-1}{K} \left( f_{i,k} - \frac{1}{K} \sum_{k=0}^{K-1} f_{i,k} \right)$
8: End
9: $p_{i,k} = \frac{\exp(F_{i,k})}{\sum_{s=0}^{K-1} \exp(F_{i,s})}$
10: End

At each stage, logitboost fits an individual regression function separately for each class. This is analogous to the popular individualized regression approach in multinomial logistic regression, which is known [3, 1] to result in loss of statistical efficiency, compared to the full (conditional) maximum likelihood approach.

On the other hand, in order to use trees as base learner, the diagonal approximation appears to be a must, at least from the practical perspective.

2.2 Adaptive Base Class Boost (ABC-Boost)

[11] derived the derivatives of the loss function (3) under the sum-to-zero constraint. Without loss of generality, we can assume that class 0 is the base class. For any $k \neq 0$,

$$\frac{\partial L_i}{\partial F_{i,k}} = (r_{i,0} - p_{i,0}) - (r_{i,k} - p_{i,k}), \quad \frac{\partial^2 L_i}{\partial F_{i,k}^2} = p_{i,0}(1 - p_{i,0}) + p_{i,k}(1 - p_{i,k}) + 2p_{i,0}p_{i,k}. \quad (5)$$

The base class must be identified at each boosting iteration during training. [11] suggested an exhaustive procedure to adaptively find the best base class to minimize the training loss (3) at each iteration.

[11] combined the idea of abc-boost with mart. The algorithm, named abc-mart, achieved good performance in multi-class classification on the datasets used in [11].

2.3 Robust LogitBoost

The mart paper[6] and a recent (2008) discussion paper [8] commented that logitboost (Alg. 1) can be numerically unstable. In fact, the logitboost paper[7] suggested some “crucial implementation protections” on page 17 of [7]:

- In Line 5 of Alg. 1 compute the response $z_{i,k}$ by $\frac{1}{p_{i,k}}$ (if $r_{i,k} = 1$) or $\frac{1}{1 - p_{i,k}}$ (if $r_{i,k} = 0$).
- Bound the response $|z_{i,k}|$ by $z_{max} \in [2, 4]$. The value of $z_{max}$ is not sensitive as long as in [2, 4].
Note that the above operations were applied to each individual sample. The goal was to ensure that the response \(|z_{i,k}|\) should not be too large. On the other hand, we should hope to use larger \(|z_{i,k}|\) to better capture the data variation. Therefore, this thresholding operation occurs very frequently and it is expected that part of the useful information is lost.

The next subsection explains that, if implemented carefully, logitboost is almost identical to mart. The only difference is the tree-splitting criterion.

### 2.4 Tree-Splitting Criterion Using Second-Order Information

Consider \(N\) weights \(w_i\), and \(N\) response values \(z_i, i = 1\) to \(N\), which are assumed to be ordered according to the sorted order of the corresponding feature values. The tree-splitting procedure is to find the index \(s, 1 \leq s < N\), such that the weighted mean square error (MSE) is reduced the most if split at \(s\). That is, we seek the \(s\) to maximize

\[
Gain(s) = MSE_T - (MSE_L + MSE_R)
\]

where \(\bar{z} = \frac{\sum_{i=1}^{N} z_i w_i}{\sum_{i=1}^{N} w_i}, \bar{z}_L = \frac{\sum_{i=1}^{s} z_i w_i}{\sum_{i=1}^{s} w_i}, \bar{z}_R = \frac{\sum_{i=s+1}^{N} z_i w_i}{\sum_{i=s+1}^{N} w_i}\). After simplification, one can obtain

\[
Gain(s) = \left[\frac{\sum_{i=1}^{s} z_i w_i}{\sum_{i=1}^{s} w_i}\right]^2 + \left[\frac{\sum_{i=s+1}^{N} z_i w_i}{\sum_{i=s+1}^{N} w_i}\right]^2 - \left[\frac{\sum_{i=1}^{N} z_i w_i}{\sum_{i=1}^{N} w_i}\right]^2
\]

Plugging in \(w_i = p_{i,k}(1 - p_{i,k}), z_i = \frac{r_{i,k} - p_{i,k}}{p_{i,k}(1 - p_{i,k})}\) yields,

\[
Gain(s) = \frac{\sum_{i=1}^{s} (r_{i,k} - p_{i,k})^2}{\sum_{i=1}^{s} p_{i,k}(1 - p_{i,k})} + \frac{\sum_{i=s+1}^{N} (r_{i,k} - p_{i,k})^2}{\sum_{i=s+1}^{N} p_{i,k}(1 - p_{i,k})} - \frac{\sum_{i=1}^{N} (r_{i,k} - p_{i,k})^2}{\sum_{i=1}^{N} p_{i,k}(1 - p_{i,k})}
\]

Because the computations involve \(\sum p_{i,k}(1 - p_{i,k})\) as a group, this procedure is actually numerically stable.

In comparison, mart only used the first order information to construct the trees, i.e.,

\[
MartGain(s) = \left[\sum_{i=1}^{s} (r_{i,k} - p_{i,k})\right]^2 + \left[\sum_{i=s+1}^{N} (r_{i,k} - p_{i,k})\right]^2 - \left[\sum_{i=1}^{N} (r_{i,k} - p_{i,k})\right]^2
\]

Alg. describes robust logitboost using the tree-splitting criterion in Sec. 2.4. Note that after trees are constructed, the values of the terminal nodes are computed by

\[
\frac{\sum_{\text{node}} z_{i,k} w_{i,k}}{\sum_{\text{node}} w_{i,k}} = \frac{\sum_{\text{node}} (r_{i,k} - p_{i,k})}{\sum_{\text{node}} p_{i,k}(1 - p_{i,k})},
\]

which explains Line 5 of Alg.
For \( k = 0 \) to \( K - 1 \), \( i = 1 \) to \( N \)

2. At each boosting iteration, adaptively select the base class according to the training loss. [11] suggested an exhaustive search strategy.

[11] combined \( \text{abc-boost} \) with \( \text{mart} \) to develop \( \text{abc-mart} \). More recently, [12] developed \( \text{abc-logitboost} \), the combination of \( \text{abc-boost} \) with (robust) logitboost.

Algorithm 3 \textit{Abc-logitboost} using the exhaustive search strategy for the base class, as suggested in [11].

The vector \( B \) stores the base class numbers.

1. \( F_{i,k} = 0 \), \( p_{i,k} = \frac{1}{K} \), \( k = 0 \) to \( K - 1 \), \( i = 1 \) to \( N \)
2. For \( m = 1 \) to \( M \) Do
3. For \( b = 0 \) to \( K - 1 \), Do
4. For \( k = 0 \) to \( K - 1 \), \( k \neq b \), Do
5. \( \{R_{j,k,m}\}_{j=1}^{J} = J\text{-terminal node regression tree from } \{r_{i,k} - p_{i,k} \text{, } x_i\}_{i=1}^{N}, \)
6. \( \beta_{j,k,m} = \frac{1}{K} \sum_{x_i \in R_{j,k,m}} r_{i,k} - p_{i,k} \)
7. \( F_{i,k} = F_{i,k} + \nu \sum_{j=1}^{J} \beta_{j,k,m} 1_{x_i \in R_{j,k,m}} \)
8. End
9. \( G_{i,b,b} = -\sum_{k \neq b} G_{i,k,b} \)
10. \( q_{i,k} = \exp(G_{i,b,b}) / \sum_{b=0}^{K-1} \exp(G_{i,b,b}) \)
11. \( L(b) = -\sum_{i=1}^{N} \sum_{k=0}^{K-1} r_{i,k} \log(q_{i,k}) \)
12. End
13. \( B(m) = \arg\min_{b} L(b) \)
14. \( F_{i,k} = G_{i,k,B(m)} \)
15. \( p_{i,k} = \exp(F_{i,k}) / \sum_{s=0}^{K-1} \exp(F_{i,s}) \)
16. End
2.6 Main Parameters

Alg. 2 and Alg. 3 have three parameters \((J, \nu \text{ and } M)\), to which the performance is in general not very sensitive, as long as they fall in some reasonable range. This is a significant advantage in practice.

The number of terminal nodes, \(J\), determines the capacity of the base learner. \[6\] suggested \(J = 6\). [7][21] commented that \(J > 10\) is unlikely. In our experience, for large datasets (or moderate datasets in high-dimensions), \(J = 20\) is often a reasonable choice; also see [14] for more examples.

The shrinkage, \(\nu\), should be large enough to make sufficient progress at each step and small enough to avoid over-fitting. \[6\] suggested \(\nu \leq 0.1\). Normally, \(\nu = 0.1\) is used.

The number of boosting iterations, \(M\), is largely determined by the affordable computing time. A commonly-regarded merit of boosting is that, on many datasets, over-fitting can be largely avoided for reasonable \(J\), and \(\nu\).

3 Datasets

Table 1 lists the datasets used in our study. [11][12] provided experiments on several other (small) datasets.

| dataset            | \(K\) | \# training | \# test | \# features |
|--------------------|-------|-------------|--------|-------------|
| Covertype290k      | 7     | 290506      | 290506 | 54          |
| Covertype145k      | 7     | 145253      | 290506 | 54          |
| Poker525k          | 10    | 525010      | 500000 | 25          |
| Poker275k          | 10    | 275010      | 500000 | 25          |
| Poker150k          | 10    | 150010      | 500000 | 25          |
| Poker100k          | 10    | 100010      | 500000 | 25          |
| Poker25kT1         | 10    | 25010       | 500000 | 25          |
| Poker25kT2         | 10    | 25010       | 500000 | 25          |
| Mnist10k           | 10    | 10000       | 60000  | 784         |
| M-Basic            | 10    | 12000       | 50000  | 784         |
| M-Rotate           | 10    | 12000       | 50000  | 784         |
| M-Image            | 10    | 12000       | 50000  | 784         |
| M-Rand             | 10    | 12000       | 50000  | 784         |
| M-RotImg           | 10    | 12000       | 50000  | 784         |
| M-Noise1           | 10    | 10000       | 2000   | 784         |
| M-Noise2           | 10    | 10000       | 2000   | 784         |
| M-Noise3           | 10    | 10000       | 2000   | 784         |
| M-Noise4           | 10    | 10000       | 2000   | 784         |
| M-Noise5           | 10    | 10000       | 2000   | 784         |
| M-Noise6           | 10    | 10000       | 2000   | 784         |
| Letter15k          | 26    | 15000       | 5000   | 16          |
| Letter4k           | 26    | 4000        | 16000  | 16          |
| Letter2k           | 26    | 2000        | 18000  | 16          |

3.1 Covertype

The original UCI Covertype dataset is fairly large, with 581012 samples. To generate Covertype290k, we randomly split the original data into halves, one half for training and another half for testing. For
Covertype145k, we randomly select one half from the training set of Covertype290k and still keep the test set.

3.2 Poker

The UCI Poker dataset originally used only 25010 samples for training and 1000000 samples for testing. Since the test set is very large, we randomly divide it equally into two parts (I and II). Poker25kT1 uses the original training set for training and Part I of the original test set for testing. Poker25kT2 uses the original training set for training and Part II of the original test set for testing. This way, Poker25kT1 can use the test set of Poker25kT2 for validation, and Poker25kT2 can use the test set of Poker25kT1 for validation. As the two test sets are still very large, this treatment will provide reliable results.

Since the original training set (about 25k) is too small compared to the size of the test set, we enlarge the training set to form Poker525k, Poker275k, Poker150k, and Poker100k. All four enlarged training datasets use the same test set as Pokere25kT2 (i.e., Part II of the original test set). The training set of Poker525k contains the original (25010) training set plus Part I of the original test set. Similarly, the training set of Poker275k / Poker150k / Poker100k contains the original training set plus 250k/125k/75k samples from Part I of the original test set.

The original Poker dataset provides 10 features, 5 “suit” features and 5 “rank” features. While the “ranks” are naturally ordinal, it appears reasonable to treat “suits” as nominal features. By private communications, R. Cattral, the donor of the Poker data, suggested us to treat the “suits” as nominal. C.J. Lin also kindly told us that the performance of SVM was not affected whether “suits” are treated nominal or ordinal. In our experiments, we choose to use “suits” as nominal feature; and hence the total number of features becomes 25 after expanding each “suite” feature with 4 binary features.

3.3 Mnist

While the original Mnist dataset is extremely popular, this dataset is known to be too easy[9]. Originally, Mnist used 60000 samples for training and 10000 samples for testing.

Mnist10k uses the original (10000) test set for training and the original (60000) training set for testing. This creates a more challenging task.

3.4 Mnist with Many Variations

[9] (www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DeepVsShallowComparisonICML2007) created a variety of much more difficult datasets by adding various background (correlated) noise, background images, rotations, etc, to the original Mnist dataset. We shortened the notations of the generated datasets to be M-Basic, M-Rotate, M-Image, M-Rand, M-RotImg, and M-Noise1, M-Noise2 to M-Noise6.

By private communications with D. Erhan, one of the authors of [9], we learn that the sizes of the training sets actually vary depending on the learning algorithms. For some methods such as SVM, they retrained the algorithms using all 120000 training samples after choosing the best parameters; and for other methods, they used 10000 samples for training. In our experiments, we use 12000 training samples for M-Basic, M-Rotate, M-Image, M-Rand and M-RotImg; and we use 10000 training samples for M-Noise1 to M-Noise6.

Note that the datasets M-Noise1 to M-Noise6 have merely 2000 test samples each. By private communications with D. Erhan, we understand this was because [9] did not mean to compare the statistical significance of the test errors for those six datasets.
3.5 Letter

The UCI Letter dataset has in total 20000 samples. In our experiments, Letter4k (Letter2k) use the last 4000 (2000) samples for training and the rest for testing. The purpose is to demonstrate the performance of the algorithms using only small training sets.

We also include Letter15k, which is one of the standard partitions of the Letter dataset, by using 15000 samples for training and 5000 samples for testing.

4 Summary of Experiment Results

We simply use logitboost (or even logit in the plots) to denote robust logitboost.

Table 2 summarizes the test mis-classification errors. For all datasets except Poker25kT1 and Poker25kT2, we report the test errors with the tree size $J=20$ and shrinkage $\nu = 0.1$. For Poker25kT1 and Poker25kT2, we use $J = 6$ and $\nu = 0.1$. We report more detailed experiment results in Sec. 5.

For C Uncovertype290k, Poker525k, Poker275k, Poker150k, and Poker100k, as they are fairly large, we only train $M = 5000$ boosting iterations. For all other datasets, we always train $M = 10000$ iterations or terminate when the training loss (3) is close to the machine accuracy. Since we do not notice obvious over-fitting on those datasets, we simply report the test errors at the last iterations.

Table 2: Summary of test mis-classification errors.

| Dataset            | mart  | abc-mart | logitboost | abc-logitboost | # test |
|--------------------|-------|----------|------------|----------------|--------|
| Covertype290k      | 11350 | 10454    | 10765      | 9727           | 290506 |
| Covertype145k      | 15767 | 14665    | 14928      | 13986          | 290506 |
| Poker525k          | 7061  | 2424     | 2704       | 1736           | 500000 |
| Poker275k          | 15404 | 3679     | 6533       | 2727           | 500000 |
| Poker150k          | 22289 | 12340    | 16163      | 5104           | 500000 |
| Poker100k          | 27871 | 21293    | 25715      | 13707          | 500000 |
| Poker25kT1         | 43575 | 34879    | 46789      | 37345          | 500000 |
| Poker25kT2         | 42935 | 34326    | 46600      | 36731          | 500000 |
| Mnist10k           | 2815  | 2440     | 2381       | 2102           | 60000  |
| M-Basic            | 2058  | 1843     | 1723       | 1602           | 50000  |
| M-Rotate           | 7674  | 6634     | 6813       | 5959           | 50000  |
| M-Image            | 5821  | 4727     | 4703       | 4268           | 50000  |
| M-Rand             | 6577  | 5300     | 5020       | 4725           | 50000  |
| M-RotImg           | 24912 | 23072    | 22962      | 22343          | 50000  |
| M-Noise1           | 305   | 245      | 267        | 234            | 2000   |
| M-Noise2           | 325   | 262      | 270        | 237            | 2000   |
| M-Noise3           | 310   | 264      | 277        | 238            | 2000   |
| M-Noise4           | 308   | 243      | 256        | 238            | 2000   |
| M-Noise5           | 294   | 244      | 242        | 227            | 2000   |
| M-Noise6           | 279   | 224      | 226        | 201            | 2000   |
| Letter15k          | 155   | 125      | 139        | 109            | 5000   |
| Letter4k           | 1370  | 1149     | 1252       | 1055           | 16000  |
| Letter2k           | 2482  | 2220     | 2309       | 2034           | 18000  |
4.1 \(P\)-Values

Table 3 summarizes the following four types of \(P\)-values:

- \(P_1\): for testing if \(abc\)-mart has significantly lower error rates than \(mart\).
- \(P_2\): for testing if (robust) logitboost has significantly lower error rates than \(mart\).
- \(P_3\): for testing if \(abc\)-logitboost has significantly lower error rates than \(abc\)-mart.
- \(P_4\): for testing if \(abc\)-logitboost has significantly lower error rates than (robust) logitboost.

The \(P\)-values are computed using binomial distributions and normal approximations. Recall, if a random variable \(z \sim \text{Binomial}(n, p)\), then the probability parameter \(p\) can be estimated by \(\hat{p} = \frac{z}{n}\), and the variance of \(\hat{p}\) can be estimated by \(\hat{p}(1 - \hat{p})/n\). The \(P\)-values can then be computed using normal approximation of binomial distributions.

Note that the test sets for \(M\)-Noise1 to \(M\)-Noise6 are very small because [9] originally did not intend to compare the statistical significance on those six datasets. We compute their \(P\)-values anyway.

Table 3: Summary of test \(P\)-Values.

| Dataset         | \(P_1\)    | \(P_2\)    | \(P_3\)    | \(P_4\)    |
|-----------------|------------|------------|------------|------------|
| Covertype290k   | \(3 \times 10^{-10}\) | \(3 \times 10^{-6}\) | \(9 \times 10^{-8}\) | \(8 \times 10^{-14}\) |
| Covertype145k   | \(4 \times 10^{-11}\) | \(4 \times 10^{-7}\) | \(2 \times 10^{-5}\) | \(7 \times 10^{-9}\) |
| Poker525k       | 0          | 0          | 0          | 0          |
| Poker275k       | 0          | 0          | 0          | 0          |
| Poker150k       | 0          | 0          | 0          | 0          |
| Poker100k       | 0          | 0          | 0          | 0          |
| Poker25kT1      | 0          | ---        | ---        | 0          |
| Poker25kT2      | 0          | ---        | ---        | 0          |
| Mnist10k        | \(5 \times 10^{-8}\) | \(3 \times 10^{-10}\) | \(1 \times 10^{-7}\) | \(1 \times 10^{-5}\) |
| M-Basic         | \(2 \times 10^{-4}\) | \(1 \times 10^{-8}\) | \(1 \times 10^{-5}\) | 0.0164     |
| M-Rotate        | 0          | \(5 \times 10^{-15}\) | \(6 \times 10^{-11}\) | \(3 \times 10^{-16}\) |
| M-Image         | 0          | 0          | \(2 \times 10^{-7}\) | \(7 \times 10^{-7}\) |
| M-Rand          | 0          | 0          | \(7 \times 10^{-10}\) | \(8 \times 10^{-4}\) |
| M-RotImg        | 0          | 0          | \(2 \times 10^{-6}\) | \(4 \times 10^{-5}\) |
| M-Noise1        | 0.0029     | 0.0430     | 0.2961     | 0.0574     |
| M-Noise2        | 0.0024     | 0.0072     | 0.1158     | 0.0583     |
| M-Noise3        | 0.0190     | 0.0701     | 0.1073     | 0.0327     |
| M-Noise4        | 0.0014     | 0.0090     | 0.4040     | 0.1935     |
| M-Noise5        | 0.0102     | 0.0079     | 0.2021     | 0.2305     |
| M-Noise6        | 0.0043     | 0.0058     | 0.1189     | 0.1002     |
| Letter15k       | 0.0345     | 0.1718     | 0.1449     | 0.0268     |
| Letter4k        | \(2 \times 10^{-6}\) | 0.008       | 0.019      | \(1 \times 10^{-5}\) |
| Letter2k        | \(2 \times 10^{-5}\) | 0.003       | 0.001      | \(4 \times 10^{-6}\) |

The results demonstrate that \(abc\)-logitboost and \(abc\)-mart considerably outperform logitboost and \(mart\), respectively. In addition, except for Poker25kT1 and Poker25kT2, we observe that \(abc\)-logitboost outperforms \(abc\)-mart, and logitboost outperforms \(mart\).
4.2 Comparisons with SVM and Deep Learning

For UCI Poker, we know that SVM could only achieve an error rate of about 40% (by private communications with C.J. Lin). In comparison, all four algorithms, mart, abc-mart, (robust) logitboost, and abc-logitboost, could achieve much smaller error rates (i.e., < 10%) on Poker25kT1 and Poker25kT2.

Figure 1 provides the comparisons on the six (correlated) noise datasets: M-Noise1 to M-Noise6. Table 4 compares the error rates on M-Basic, M-Rotate, M-Image, M-Rand, and M-RotImg.

Figure 1: Six datasets: M-Noise1 to M-Noise6. Left panel: Error rates of SVM and deep learning [9]. Middle and right panels: Errors rates of four boosting algorithms. X-axis: degree of correlation from high to low; the values 1 to 6 correspond to the datasets M-Noise1 to M-Noise6.

Table 4: Summary of error rates of various algorithms on the modified Mnist dataset[9].

| Algorithm   | M-Basic | M-Rotate | M-Image | M-Rand | M-RotImg |
|-------------|---------|----------|---------|--------|----------|
| SVM-RBF     | 3.05%   | 11.11%   | 22.61%  | 14.58% | 55.18%   |
| SVM-POLY    | 3.69%   | 15.42%   | 24.01%  | 16.62% | 56.41%   |
| NNET        | 4.69%   | 18.11%   | 27.41%  | 20.04% | 62.16%   |
| DBN-3       | 3.11%   | 10.30%   | 16.31%  | 6.73%  | 47.39%   |
| SAA-3       | 3.46%   | 10.30%   | 23.00%  | 11.28% | 51.93%   |
| DBN-1       | 3.94%   | 14.69%   | 16.15%  | 9.80%  | 52.21%   |
| mart        | 4.12%   | 15.35%   | 11.64%  | 13.15% | 49.82%   |
| abc-mart    | 3.69%   | 13.27%   | 9.45%   | 10.60% | 46.14%   |
| logitboost  | 3.45%   | 13.63%   | 9.41%   | 10.04% | 45.92%   |
| abc-logitboost | 3.20% | 11.92% | 8.54% | 9.45% | 44.69% |
4.3 Performance vs. Boosting Iterations

Figure 2 presents the training loss, i.e., Eq. (3), on \textit{Covertype290k} and \textit{Poker525k}, for all boosting iterations. Figures 3 and 4 provide the test mis-classification errors on \textit{Covertype}, \textit{Poker}, \textit{Mnist10k}, and \textit{Letter}.

![Figure 2: Training loss, Eq. (3), on \textit{Covertype290k} and \textit{Poker525k}.](image1)

![Figure 3: Test mis-classification errors on \textit{Mnist10k}, \textit{Letter15k}, \textit{Letter4k}, and \textit{Letter2k}.](image2)
Figure 4: Test mis-classification errors on Covertype and Poker.
5 More Detailed Experiment Results

Ideally, we would like to demonstrate that, with any reasonable choice of parameters $J$ and $\nu$, abc-mart and abc-logitboost will always improve mart and logitboost, respectively. This is actually indeed the case on the datasets we have experimented. In this section, we provide the detailed experiment results on Mnist10k, Poker25kT1, Poker25kT2, Letter4k, and Letter2k.

5.1 Detailed Experiment Results on Mnist10k

For this dataset, we experiment with every combination of $J \in \{4, 6, 8, 10, 12, 14, 16, 18, 20, 24, 30, 40, 50\}$ and $\nu \in \{0.04, 0.06, 0.08, 0.1\}$. We train the four boosting algorithms till the training loss (3) is close to the machine accuracy, to exhaust the capacity of the learner so that we could provide a reliable comparison, up to $M = 10000$ iterations.

Table 5 presents the test mis-classification errors and Table 6 presents the $P$-values. Figures 5, 6, and 7 provide the test mis-classification errors for all boosting iterations.

Table 5: Mnist10k. Upper table: The test mis-classification errors of mart and abc-mart (bold numbers). Bottom table: The test mis-classification errors of logitboost and abc-logitboost (bold numbers)

|          | mart   | abc-mart |          | mart   | abc-mart |          | mart   | abc-mart |          | mart   | abc-mart |
|----------|--------|----------|----------|--------|----------|----------|--------|----------|----------|--------|----------|
|          | $\nu = 0.04$ | $\nu = 0.06$ | $\nu = 0.08$ | $\nu = 0.1$ | $\nu = 0.04$ | $\nu = 0.06$ | $\nu = 0.08$ | $\nu = 0.1$ | $\nu = 0.04$ | $\nu = 0.06$ | $\nu = 0.08$ | $\nu = 0.1$ |
| $J = 4$  | 3356  3060 | 3329  3019 | 3318  2855 | 3326  2794 | 3287  2720 | 3308  2812 | 3330  2874 | 3339  2896 | 3329  2812 | 3318  2855 | 3308  2720 |
| $J = 6$  | 3815  3086 | 3093  2626 | 3129  2656 | 3217  2590 | 3287  2720 | 3318  2855 | 3372  2900 | 3412  2940 | 3372  2720 | 3318  2855 | 3372  2900 |
| $J = 8$  | 3049  2558 | 3054  2555 | 3054  2534 | 3035  2577 | 3287  2720 | 3318  2855 | 3372  2900 | 3412  2940 | 3372  2720 | 3318  2855 | 3372  2900 |
| $J = 10$ | 3020  2547 | 2973  2521 | 2990  2520 | 2978  2506 | 3287  2720 | 3318  2855 | 3372  2900 | 3412  2940 | 3372  2720 | 3318  2855 | 3372  2900 |
| $J = 12$ | 2927  2499 | 2917  2457 | 2945  2480 | 2907  2490 | 3287  2720 | 3318  2855 | 3372  2900 | 3412  2940 | 3372  2720 | 3318  2855 | 3372  2900 |
| $J = 14$ | 2925  2487 | 2901  2471 | 2877  2470 | 2884  2454 | 3287  2720 | 3318  2855 | 3372  2900 | 3412  2940 | 3372  2720 | 3318  2855 | 3372  2900 |
| $J = 16$ | 2899  2478 | 2893  2452 | 2873  2465 | 2860  2451 | 3287  2720 | 3318  2855 | 3372  2900 | 3412  2940 | 3372  2720 | 3318  2855 | 3372  2900 |
| $J = 18$ | 2857  2469 | 2880  2460 | 2870  2437 | 2855  2454 | 3287  2720 | 3318  2855 | 3372  2900 | 3412  2940 | 3372  2720 | 3318  2855 | 3372  2900 |
| $J = 20$ | 2833  2441 | 2834  2448 | 2834  2444 | 2815  2440 | 3287  2720 | 3318  2855 | 3372  2900 | 3412  2940 | 3372  2720 | 3318  2855 | 3372  2900 |
| $J = 22$ | 2840  2447 | 2827  2431 | 2801  2427 | 2784  2455 | 3287  2720 | 3318  2855 | 3372  2900 | 3412  2940 | 3372  2720 | 3318  2855 | 3372  2900 |
| $J = 30$ | 2826  2457 | 2822  2443 | 2828  2470 | 2807  2450 | 3287  2720 | 3318  2855 | 3372  2900 | 3412  2940 | 3372  2720 | 3318  2855 | 3372  2900 |
| $J = 40$ | 2837  2482 | 2809  2440 | 2836  2447 | 2782  2506 | 3287  2720 | 3318  2855 | 3372  2900 | 3412  2940 | 3372  2720 | 3318  2855 | 3372  2900 |
| $J = 50$ | 2813  2502 | 2826  2459 | 2824  2469 | 2786  2499 | 3287  2720 | 3318  2855 | 3372  2900 | 3412  2940 | 3372  2720 | 3318  2855 | 3372  2900 |
Table 6: *Mnist10k*; *P*-values. See Sec. 4.1 for the definitions of P1, P2, P3, and P4.

| J  | P1          | P2          | P3          | P4          |
|----|-------------|-------------|-------------|-------------|
|    | $\nu = 0.04$ | $\nu = 0.06$ | $\nu = 0.08$ | $\nu = 0.1$ |
| 4  | $7 \times 10^{-3}$ | $3 \times 10^{-3}$ | $7 \times 10^{-10}$ | $1 \times 10^{-12}$ |
| 6  | $8 \times 10^{-9}$ | $1 \times 10^{-10}$ | $9 \times 10^{-11}$ | $0$ |
| 8  | $9 \times 10^{-12}$ | $4 \times 10^{-12}$ | $5 \times 10^{-13}$ | $2 \times 10^{-10}$ |
| 10 | $4 \times 10^{-11}$ | $2 \times 10^{-10}$ | $4 \times 10^{-11}$ | $3 \times 10^{-11}$ |
| 12 | $1 \times 10^{-9}$ | $7 \times 10^{-11}$ | $1 \times 10^{-10}$ | $3 \times 10^{-9}$ |
| 14 | $6 \times 10^{-10}$ | $1 \times 10^{-9}$ | $6 \times 10^{-9}$ | $9 \times 10^{-10}$ |
| 16 | $2 \times 10^{-9}$ | $3 \times 10^{-10}$ | $6 \times 10^{-9}$ | $5 \times 10^{-9}$ |
| 18 | $3 \times 10^{-8}$ | $2 \times 10^{-9}$ | $6 \times 10^{-10}$ | $9 \times 10^{-9}$ |
| 20 | $2 \times 10^{-8}$ | $3 \times 10^{-8}$ | $2 \times 10^{-8}$ | $6 \times 10^{-8}$ |
| 24 | $2 \times 10^{-8}$ | $1 \times 10^{-8}$ | $6 \times 10^{-8}$ | $2 \times 10^{-6}$ |
| 30 | $1 \times 10^{-7}$ | $5 \times 10^{-8}$ | $2 \times 10^{-7}$ | $2 \times 10^{-7}$ |
| 40 | $3 \times 10^{-7}$ | $1 \times 10^{-7}$ | $2 \times 10^{-8}$ | $5 \times 10^{-5}$ |
| 50 | $6 \times 10^{-6}$ | $1 \times 10^{-7}$ | $3 \times 10^{-7}$ | $3 \times 10^{-5}$ |
|    | $\nu = 0.04$ | $\nu = 0.06$ | $\nu = 0.08$ | $\nu = 0.1$ |
| 4  | $2 \times 10^{-8}$ | $2 \times 10^{-6}$ | $6 \times 10^{-6}$ | $3 \times 10^{-6}$ |
| 6  | $1 \times 10^{-10}$ | $4 \times 10^{-8}$ | $9 \times 10^{-9}$ | $8 \times 10^{-12}$ |
| 8  | $4 \times 10^{-10}$ | $2 \times 10^{-9}$ | $1 \times 10^{-10}$ | $1 \times 10^{-9}$ |
| 10 | $7 \times 10^{-11}$ | $4 \times 10^{-10}$ | $3 \times 10^{-11}$ | $2 \times 10^{-11}$ |
| 12 | $1 \times 10^{-10}$ | $2 \times 10^{-10}$ | $2 \times 10^{-11}$ | $3 \times 10^{-10}$ |
| 14 | $2 \times 10^{-11}$ | $8 \times 10^{-12}$ | $2 \times 10^{-10}$ | $3 \times 10^{-11}$ |
| 16 | $1 \times 10^{-11}$ | $8 \times 10^{-11}$ | $7 \times 10^{-12}$ | $3 \times 10^{-11}$ |
| 18 | $5 \times 10^{-11}$ | $9 \times 10^{-12}$ | $9 \times 10^{-12}$ | $9 \times 10^{-12}$ |
| 20 | $2 \times 10^{-10}$ | $2 \times 10^{-9}$ | $1 \times 10^{-9}$ | $4 \times 10^{-10}$ |
| 24 | $1 \times 10^{-8}$ | $3 \times 10^{-9}$ | $3 \times 10^{-8}$ | $1 \times 10^{-7}$ |
| 30 | $2 \times 10^{-7}$ | $2 \times 10^{-8}$ | $5 \times 10^{-9}$ | $2 \times 10^{-7}$ |
| 40 | $3 \times 10^{-5}$ | $1 \times 10^{-5}$ | $4 \times 10^{-6}$ | $2 \times 10^{-4}$ |
| 50 | $0.0026$ | $0.0023$ | $3 \times 10^{-4}$ | $0.0013$ |
|    | $\nu = 0.04$ | $\nu = 0.06$ | $\nu = 0.08$ | $\nu = 0.1$ |
| 4  | $3 \times 10^{-9}$ | $5 \times 10^{-9}$ | $4 \times 10^{-6}$ | $7 \times 10^{-6}$ |
| 6  | $4 \times 10^{-13}$ | $2 \times 10^{-8}$ | $2 \times 10^{-10}$ | $3 \times 10^{-8}$ |
| 8  | $2 \times 10^{-9}$ | $3 \times 10^{-10}$ | $3 \times 10^{-10}$ | $6 \times 10^{-11}$ |
| 10 | $1 \times 10^{-10}$ | $8 \times 10^{-10}$ | $6 \times 10^{-11}$ | $4 \times 10^{-10}$ |
| 12 | $2 \times 10^{-10}$ | $2 \times 10^{-8}$ | $1 \times 10^{-9}$ | $1 \times 10^{-9}$ |
| 14 | $5 \times 10^{-10}$ | $6 \times 10^{-9}$ | $4 \times 10^{-10}$ | $4 \times 10^{-10}$ |
| 16 | $2 \times 10^{-8}$ | $2 \times 10^{-7}$ | $1 \times 10^{-8}$ | $1 \times 10^{-8}$ |
| 18 | $4 \times 10^{-9}$ | $8 \times 10^{-9}$ | $6 \times 10^{-8}$ | $3 \times 10^{-8}$ |
| 20 | $1 \times 10^{-6}$ | $2 \times 10^{-7}$ | $6 \times 10^{-8}$ | $2 \times 10^{-7}$ |
| 24 | $2 \times 10^{-5}$ | $9 \times 10^{-6}$ | $3 \times 10^{-6}$ | $9 \times 10^{-7}$ |
| 30 | $5 \times 10^{-4}$ | $0.0011$ | $1 \times 10^{-4}$ | $2 \times 10^{-5}$ |
| 40 | $0.0056$ | $0.0103$ | $0.0024$ | $1 \times 10^{-4}$ |
| 50 | $0.0145$ | $0.0707$ | $0.0218$ | $0.0102$ |
|    | $\nu = 0.04$ | $\nu = 0.06$ | $\nu = 0.08$ | $\nu = 0.1$ |
| 4  | $1 \times 10^{-3}$ | $2 \times 10^{-7}$ | $4 \times 10^{-10}$ | $5 \times 10^{-12}$ |
| 6  | $5 \times 10^{-11}$ | $7 \times 10^{-11}$ | $1 \times 10^{-12}$ | $6 \times 10^{-13}$ |
| 8  | $4 \times 10^{-11}$ | $5 \times 10^{-13}$ | $2 \times 10^{-12}$ | $8 \times 10^{-12}$ |
| 10 | $6 \times 10^{-11}$ | $5 \times 10^{-10}$ | $8 \times 10^{-11}$ | $7 \times 10^{-10}$ |
| 12 | $2 \times 10^{-9}$ | $6 \times 10^{-9}$ | $6 \times 10^{-9}$ | $1 \times 10^{-8}$ |
| 14 | $1 \times 10^{-8}$ | $4 \times 10^{-7}$ | $1 \times 10^{-8}$ | $9 \times 10^{-9}$ |
| 16 | $1 \times 10^{-6}$ | $5 \times 10^{-7}$ | $3 \times 10^{-6}$ | $9 \times 10^{-7}$ |
| 18 | $1 \times 10^{-6}$ | $8 \times 10^{-7}$ | $2 \times 10^{-6}$ | $8 \times 10^{-6}$ |
| 20 | $4 \times 10^{-5}$ | $2 \times 10^{-6}$ | $8 \times 10^{-7}$ | $1 \times 10^{-5}$ |
| 24 | $3 \times 10^{-5}$ | $3 \times 10^{-5}$ | $7 \times 10^{-6}$ | $1 \times 10^{-5}$ |
| 30 | $3 \times 10^{-4}$ | $0.0016$ | $0.0012$ | $2 \times 10^{-5}$ |
| 40 | $2 \times 10^{-4}$ | $5 \times 10^{-4}$ | $6 \times 10^{-5}$ | $3 \times 10^{-5}$ |
| 50 | $9 \times 10^{-5}$ | $7 \times 10^{-5}$ | $2 \times 10^{-4}$ | $4 \times 10^{-4}$ |
Figure 5: Mnist10k. Test mis-classification errors of four algorithms. \( J = 4, 6, 8, 10. \)
Figure 6: Mnist10k. Test mis-classification errors of four algorithms. $J = 12, 14, 16, 18$. 
Figure 7: **Mnist10k**. Test mis-classification errors of four algorithms. $J = 20, 24, 30, 40, 50$. 
The experiment results illustrate that the performances of all four algorithms are stable on a wide-range of base class tree sizes $J$, e.g., $J \in [6, 30]$. The shrinkage parameter $\nu$ does not affect much the test performance, although smaller $\nu$ values result in more boosting iterations (before the training losses reach the machine accuracy).

We further randomly divide the test set of $Mnist10k$ (60000 test samples) equally into two parts (I and II). We then test algorithms on Part I (using the same training results). We name this “new” dataset $Mnist10kT1$. The purpose of this experiment is to further demonstrate the stability of the algorithms.

Table 7 presents the test mis-classification errors of $Mnist10kT1$. Compared to Table 5, the mis-classification errors of $Mnist10kT1$ are roughly 50% of the mis-classification errors of $Mnist10k$ for all $J$ and $\nu$. This helps establish that our experiment results on $Mnist10k$ provide a very reliable comparison.

### Table 7: $Mnist10kT1$

Upper table: The test mis-classification errors of $mart$ and $abc-mart$ (bold numbers). Bottom table: The test mis-classification errors of $logitboost$ and $abc-logitboost$ (bold numbers). $Mnist10kT1$ only uses a half of the test data of $Mnist10k$.

| $J$ | $mart$ | $abc-mart$ | $logitboost$ | $abc-logitboost$ |
|-----|--------|------------|--------------|------------------|
|     | $\nu = 0.04$ | $\nu = 0.06$ | $\nu = 0.08$ | $\nu = 0.1$ |
| 4   | 1682   | 1514       | 1668         | 1505            | 1666         | 1416       | 1663         | 1380 |
| 6   | 1573   | 1382       | 1523         | 1320            | 1533         | 1329       | 1582         | 1288 |
| 8   | 1501   | 1263       | 1515         | 1257            | 1523         | 1250       | 1491         | 1279 |
| 10  | 1492   | 1270       | 1457         | 1248            | 1470         | 1239       | 1459         | 1236 |
| 12  | 1432   | 1244       | 1427         | 1234            | 1444         | 1228       | 1436         | 1227 |
| 14  | 1424   | 1237       | 1420         | 1231            | 1407         | 1223       | 1419         | 1212 |
| 16  | 1430   | 1226       | 1426         | 1224            | 1411         | 1223       | 1418         | 1204 |
| 18  | 1400   | 1222       | 1413         | 1218            | 1390         | 1210       | 1404         | 1211 |
| 20  | 1398   | 1213       | 1381         | 1205            | 1388         | 1213       | 1382         | 1198 |
| 24  | 1402   | 1221       | 1366         | 1201            | 1372         | 1199       | 1346         | 1205 |
| 30  | 1384   | 1211       | 1374         | 1208            | 1368         | 1224       | 1366         | 1205 |
| 40  | 1397   | 1244       | 1375         | 1220            | 1397         | 1222       | 1365         | 1246 |
| 50  | 1371   | 1239       | 1380         | 1221            | 1382         | 1223       | 1362         | 1242 |

| $J$ | $\nu = 0.04$ | $\nu = 0.06$ | $\nu = 0.08$ | $\nu = 0.1$ |
|-----|--------------|--------------|--------------|--------------|
| 4   | 1419         | 1299         | 1449         | 1281         | 1446         | 1251         | 1460         | 1244 |
| 6   | 1313         | 1111         | 1313         | 1114         | 1326         | 1101         | 1317         | 1097 |
| 8   | 1278         | 1058         | 1287         | 1050         | 1270         | 1036         | 1262         | 1058 |
| 10  | 1252         | 1061         | 1244         | 1057         | 1237         | 1040         | 1229         | 1041 |
| 12  | 1224         | 1020         | 1219         | 1049         | 1217         | 1053         | 1224         | 1047 |
| 14  | 1213         | 1038         | 1207         | 1050         | 1201         | 1039         | 1198         | 1026 |
| 16  | 1185         | 1050         | 1205         | 1058         | 1189         | 1044         | 1178         | 1041 |
| 18  | 1186         | 1048         | 1184         | 1038         | 1184         | 1046         | 1167         | 1056 |
| 20  | 1185         | 1077         | 1199         | 1063         | 1183         | 1042         | 1184         | 1045 |
| 24  | 1208         | 1095         | 1196         | 1083         | 1191         | 1064         | 1194         | 1068 |
| 30  | 1225         | 1113         | 1201         | 1117         | 1190         | 1113         | 1211         | 1087 |
| 40  | 1254         | 1159         | 1247         | 1145         | 1248         | 1127         | 1249         | 1127 |
| 50  | 1292         | 1177         | 1284         | 1174         | 1275         | 1161         | 1276         | 1176 |
5.2 Detailed Experiment Results on Poker25kT1 and Poker25kT2

Recall the original UCI Poker dataset used 25010 samples for training and 1000000 samples for testing. To provide a reliable comparison (and validation), we form two datasets Poker25kT1 and Poker25kT2 by equally dividing the original test set into two parts (I and II). Both use the same training set. Poker25kT1 uses Part I of the original test set for testing and Poker25kT2 uses Part II for testing.

Table 8 and Table 9 present the test mis-classification errors, for $J \in \{4, 6, 8, 10, 12, 14, 16, 18, 20\}$ and $\nu \in \{0.04, 0.06, 0.08, 0.1\}$. Comparing these two tables, we can see the corresponding entries are very close to each other, which again verifies that the four boosting algorithms provide reliable results on this dataset.

For most $J$ and $\nu$, all four algorithms achieve error rates $< 10\%$. For both Poker25kT1 and Poker25kT2, the lowest test errors are attained at $\nu = 0.1$ and $J = 6$. Unlike Mnist10k, the test errors, especially using mart and logitboost, are slightly sensitive to the parameters.

Note that when $J = 4$ (and $\nu$ is small), only training $M = 10000$ steps would not be sufficient in this case.

Table 8: Poker25kT1. Upper table: The test mis-classification errors of mart and abc-mart (bold numbers). Bottom table: The test mis-classification errors of logitboost and abc-logitboost (bold numbers)

|       | mart     | abc-mart |       | logitboost | abc-logit |
|-------|----------|----------|-------|------------|-----------|
|       | $\nu = 0.04$ | $\nu = 0.06$ | $\nu = 0.08$ | $\nu = 0.1$ |
| $J = 4$ | 145880.90323 | 132526.67417 | 124283.49403 | 113985.49403 |
| $J = 6$ | 71628.38017 | 59046.36839 | 48064.35467 | 43573.34879 |
| $J = 8$ | 64090.39220 | 53400.37112 | 47360.36407 | 44131.35777 |
| $J = 10$ | 60456.39661 | 52464.38547 | 47203.36990 | 46351.36647 |
| $J = 12$ | 61452.41362 | 52697.39221 | 46822.37723 | 46965.37345 |
| $J = 14$ | 58348.42764 | 56047.40993 | 50476.40155 | 47935.37780 |
| $J = 16$ | 63518.44386 | 55418.43360 | 50612.41952 | 49179.40050 |
| $J = 18$ | 64426.46463 | 55708.45607 | 54033.45838 | 52113.43040 |
| $J = 20$ | 65528.49577 | 59236.47901 | 56384.45725 | 53506.44295 |

|       | logitboost | abc-logit |
|-------|------------|-----------|
|       | $\nu = 0.04$ | $\nu = 0.06$ | $\nu = 0.08$ | $\nu = 0.1$ |
| $J = 4$ | 147064.102905 | 140068.71450 | 128161.51226 | 117085.42140 |
| $J = 6$ | 81566.43156 | 59324.39164 | 51526.37954 | 48516.37546 |
| $J = 8$ | 68278.46076 | 56922.40162 | 52532.38422 | 46789.37345 |
| $J = 10$ | 63796.44830 | 55834.40754 | 53262.40486 | 47118.38141 |
| $J = 12$ | 66732.48412 | 56867.44886 | 51248.42100 | 47485.39798 |
| $J = 14$ | 64263.52479 | 55614.48093 | 51735.44688 | 47806.43048 |
| $J = 16$ | 67092.53363 | 58019.51308 | 53746.47831 | 51267.46968 |
| $J = 18$ | 69104.57147 | 56514.55468 | 55290.50292 | 51871.47986 |
| $J = 20$ | 68899.62345 | 61314.57677 | 56648.53696 | 51608.49864 |
Table 9: Poker25kT2. Upper table: The test mis-classification errors of \textit{mart} and \textit{abc-mart} (bold numbers). Bottom table: The test mis-classification errors of \textit{logitboost} and \textit{abc-logitboost} (bold numbers).

\begin{tabular}{cccccc}
\hline
 & \textit{mart} & \textit{abc-mart} \\
\hline
 & $\nu = 0.04$ & $\nu = 0.06$ & $\nu = 0.08$ & $\nu = 0.1$ \\
\hline
$J = 4$ & 144020 & 89608 & 131243 & 67071 & 123031 & 48855 & 113232 & 41688 \\
$J = 6$ & 71004 & 37567 & 58487 & 36345 & 47564 & 34920 & 42935 & 34326 \\
$J = 8$ & 63452 & 38703 & 52990 & 36586 & 46914 & 35836 & 43647 & 35129 \\
$J = 10$ & 60061 & 39078 & 52125 & 38025 & 46912 & 36455 & 45863 & 36076 \\
$J = 12$ & 61098 & 40834 & 52296 & 38657 & 46458 & 37203 & 46698 & 36781 \\
$J = 14$ & 57924 & 42348 & 55622 & 40363 & 50243 & 39613 & 47619 & 37243 \\
$J = 16$ & 63213 & 44067 & 55461 & 45133 & 53652 & 45308 & 51870 & 42485 \\
$J = 18$ & 64056 & 46050 & 55461 & 45133 & 53652 & 45308 & 51870 & 42485 \\
$J = 20$ & 65215 & 49046 & 58911 & 47430 & 56099 & 45390 & 53213 & 43888 \\
\hline
\end{tabular}

\begin{tabular}{cccccc}
\hline
 & \textit{logitboost} & \textit{abc-logit} \\
\hline
 & $\nu = 0.04$ & $\nu = 0.06$ & $\nu = 0.08$ & $\nu = 0.1$ \\
\hline
$J = 4$ & 145368 & 102014 & 138734 & 70886 & 126980 & 50783 & 116346 & 41551 \\
$J = 6$ & 80782 & 42699 & 58769 & 38592 & 51202 & 37397 & 48199 & 36914 \\
$J = 8$ & 68065 & 45737 & 56678 & 39648 & 52504 & 37935 & 46600 & 36731 \\
$J = 10$ & 63153 & 44517 & 55419 & 40286 & 52835 & 40044 & 46913 & 37504 \\
$J = 12$ & 66240 & 47948 & 56619 & 44602 & 50918 & 41582 & 47128 & 39378 \\
$J = 14$ & 63763 & 52063 & 55238 & 47642 & 51526 & 44296 & 47545 & 42720 \\
$J = 16$ & 66543 & 52937 & 57473 & 50842 & 53287 & 47578 & 51106 & 46635 \\
$J = 18$ & 68477 & 56803 & 57070 & 55166 & 54954 & 49956 & 51603 & 47707 \\
$J = 20$ & 68311 & 61980 & 61047 & 57383 & 56474 & 53364 & 51242 & 49506 \\
\hline
\end{tabular}
5.3 Detailed Experiment Results on *Letter4k* and *Letter2k*

Table 10: *Letter4k*. Upper table: The test mis-classification errors of *mart* and *abc-mart* (bold numbers). Bottom table: The test mis-classification errors of *logitboost* and *abc-logitboost* (bold numbers)

|       | *mart*       |       | *abc-mart*     |       |       |       |       | *logitboost* |       | *abc-logit*   |       |
|-------|--------------|-------|----------------|-------|-------|-------|-------|--------------|-------|---------------|-------|
|       | *ν* = 0.04   | *ν* = 0.06 | *ν* = 0.08 | *ν* = 0.1 | *ν* = 0.04   | *ν* = 0.06 | *ν* = 0.08 | *ν* = 0.1 | *ν* = 0.04   | *ν* = 0.06 | *ν* = 0.08 | *ν* = 0.1 |
| *J* = 4 | 1681 | 1415 | 1660 | 1380 | 1671 | 1368 | 1655 | 1323 | 1460 | 1296 | 1471 | 1241 | 1452 | 1202 | 1446 | 1208 |
| *J* = 6 | 1618 | 1320 | 1584 | 1288 | 1588 | 1266 | 1577 | 1240 | 1390 | 1143 | 1394 | 1117 | 1382 | 1090 | 1374 | 1074 |
| *J* = 8 | 1531 | 1266 | 1522 | 1246 | 1516 | 1192 | 1521 | 1184 | 1383 | 1174 | 1406 | 1174 | 1401 | 1177 | 1404 | 1209 |
| *J* = 10 | 1499 | 1228 | 1463 | 1208 | 1479 | 1186 | 1470 | 1185 | 1458 | 1211 | 1455 | 1224 | 1441 | 1233 | 1454 | 1215 |
| *J* = 12 | 1420 | 1213 | 1434 | 1186 | 1409 | 1170 | 1437 | 1162 | 1426 | 1204 | 1415 | 1234 | 1295 | 1202 | 1220 | 1057 |
| *J* = 14 | 1410 | 1190 | 1388 | 1156 | 1377 | 1151 | 1396 | 1160 | 1426 | 1204 | 1415 | 1234 | 1295 | 1202 | 1220 | 1057 |
| *J* = 16 | 1395 | 1167 | 1402 | 1156 | 1396 | 1157 | 1387 | 1146 | 1458 | 1211 | 1455 | 1224 | 1444 | 1233 | 1454 | 1215 |

21
Table 11: Letter2k. Upper table: The test mis-classification errors of *mart* and *abc-mart* (bold numbers). Bottom table: The test mis-classification errors of *logitboost* and *abc-logitboost* (bold numbers).

|      | mart  | abc-mart |      | logitboost | abc-logit |
|------|-------|----------|------|------------|-----------|
|      | $\nu = 0.04$ | $\nu = 0.06$ | $\nu = 0.08$ | $\nu = 0.1$ | $\nu = 0.04$ | $\nu = 0.06$ | $\nu = 0.08$ | $\nu = 0.1$ |
| $J = 4$ | 2694 2512 | 2598 2470 | 2684 2419 | 2689 2435 | 2629 2300 | 2598 2470 | 2684 2419 | 2689 2435 |
| $J = 6$ | 2683 2360 | 2664 2321 | 2640 2313 | 2629 2321 | 2598 2470 | 2684 2419 | 2689 2435 | 2689 2435 |
| $J = 8$ | 2569 2279 | 2603 2289 | 2563 2259 | 2571 2251 | 2598 2470 | 2684 2419 | 2689 2435 | 2689 2435 |
| $J = 10$ | 2534 2242 | 2516 2215 | 2504 2210 | 2491 2185 | 2598 2470 | 2684 2419 | 2689 2435 | 2689 2435 |
| $J = 12$ | 2503 2202 | 2516 2215 | 2473 2198 | 2492 2201 | 2598 2470 | 2684 2419 | 2689 2435 | 2689 2435 |
| $J = 14$ | 2488 2203 | 2467 2231 | 2460 2204 | 2460 2183 | 2598 2470 | 2684 2419 | 2689 2435 | 2689 2435 |
| $J = 16$ | 2503 2219 | 2501 2219 | 2496 2235 | 2500 2205 | 2598 2470 | 2684 2419 | 2689 2435 | 2689 2435 |
| $J = 18$ | 2494 2225 | 2497 2212 | 2472 2205 | 2439 2213 | 2598 2470 | 2684 2419 | 2689 2435 | 2689 2435 |
| $J = 20$ | 2499 2199 | 2512 2198 | 2504 2188 | 2482 2220 | 2598 2470 | 2684 2419 | 2689 2435 | 2689 2435 |
| $J = 22$ | 2549 2200 | 2549 2191 | 2526 2218 | 2538 2248 | 2598 2470 | 2684 2419 | 2689 2435 | 2689 2435 |
| $J = 24$ | 2579 2237 | 2566 2232 | 2574 2244 | 2574 2285 | 2598 2470 | 2684 2419 | 2689 2435 | 2689 2435 |
| $J = 26$ | 2641 2303 | 2632 2304 | 2606 2271 | 2667 2351 | 2598 2470 | 2684 2419 | 2689 2435 | 2689 2435 |

Note: The values in bold indicate mis-classification errors.
6 Conclusion

Classification is a fundamental task in machine learning. This paper presents extensive experiment results of four tree-based boosting algorithms: \textit{mart}, \textit{abc-mart}, (robust) \textit{logitboost}, and \textit{abc-logitboost}, for multi-class classification, on a variety of publicly available datasets. From the experiment results, we can conclude the following:

1. \textit{Abc-mart} considerably improves \textit{mart}.
2. \textit{Abc-logitboost} considerably improves (robust) \textit{logitboost}.
3. (Robust) \textit{logitboost} considerably improves \textit{mart} on most datasets.
4. \textit{Abc-logitboost} considerably improves \textit{abc-mart} on most datasets.
5. These four boosting algorithms (especially \textit{abc-logitboost}) outperform SVM on many datasets.
6. Compared to the best deep learning methods, these four boosting algorithms (especially \textit{abc-logitboost}) are competitive.

References

[1] Alan Agresti. \textit{Categorical Data Analysis}. John Wiley & Sons, Inc., Hoboken, NJ, second edition, 2002.

[2] Peter Bartlett, Yoav Freund, Wee Sun Lee, and Robert E. Schapire. Boosting the margin: a new explanation for the effectiveness of voting methods. \textit{The Annals of Statistics}, 26(5):1651–1686, 1998.

[3] Colin B. Begg and Robert Gray. Calculation of polychotomous logistic regression parameters using individualized regressions. \textit{Biometrika}, 71(1):11–18, 1984.

[4] Yoav Freund. Boosting a weak learning algorithm by majority. \textit{Inf. Comput.}, 121(2):256–285, 1995.

[5] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. \textit{J. Comput. Syst. Sci.}, 55(1):119–139, 1997.

[6] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. \textit{The Annals of Statistics}, 29(5):1189–1232, 2001.

[7] Jerome H. Friedman, Trevor J. Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. \textit{The Annals of Statistics}, 28(2):337–407, 2000.

[8] Jerome H. Friedman, Trevor J. Hastie, and Robert Tibshirani. Response to evidence contrary to the statistical view of boosting. \textit{Journal of Machine Learning Research}, 9:175–180, 2008.

[9] Hugo Larochelle, Dumitru Erhan, Aaron C. Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In \textit{ICML}, pages 473–480, Corvalis, Oregon, 2007.

[10] Yoonkyung Lee, Yi Lin, and Grace Wahba. Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data. \textit{Journal of the American Statistical Association}, 99(465):67–81, 2004.
[11] Ping Li. Abc-boost: Adaptive base class boost for multi-class classification. In ICML, Montreal, Canada, 2009.

[12] Ping Li. Abc-logitboost for multi-class classification. Technical report, Department of Statistical Science, Cornell University, 2009.

[13] Ping Li. Robust logitboost. Technical report, Department of Statistical Science, Cornell University, 2009.

[14] Ping Li, Christopher J.C. Burges, and Qiang Wu. Mcrank: Learning to rank using classification and gradient boosting. In NIPS, Vancouver, BC, Canada, 2008.

[15] Liew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent. In NIPS, 2000.

[16] Robert Schapire. The strength of weak learnability. Machine Learning, 5(2):197–227, 1990.

[17] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. Machine Learning, 37(3):297–336, 1999.

[18] Ambuj Tewari and Peter L. Bartlett. On the consistency of multiclass classification methods. Journal of Machine Learning Research, 8:1007–1025, 2007.

[19] Tong Zhang. Statistical analysis of some multi-category large margin classification methods. Journal of Machine Learning Research, 5:1225–1251, 2004.

[20] Ji Zhu, Hui Zou, Sharon Rosset, and Trevor Hastie. Multi-class adaboost. Statistics and Its Interface, 2(3):349–360, 2009.

[21] Hui Zou, Ji Zhu, and Trevor Hastie. New multiclass boosting algorithms based on multiclass fisher-consistent losses. The Annals of Applied Statistics, 2(4):1290–1306, 2008.