ABC-LogitBoost for Multi-Class Classification

Ping Li

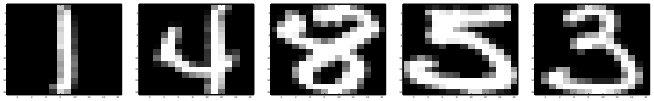
Department of Statistical Science

Cornell University

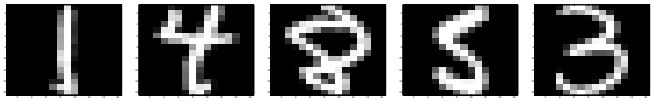
What is Classification?

An Example: USPS Handwritten Zipcode Recognition

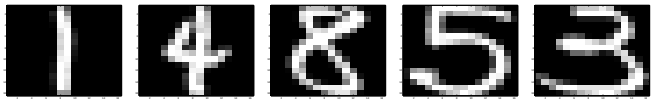
Person 1:



Person 2:



Person 3:



The task: Teach the machine to automatically recognize the 10 digits.

Multi-Class Classification

Given a training data set

$$\{y_i,\ X_i\}_{i=1}^N,\qquad X_i \in \mathbb{R}^p,\qquad y_i \in \{0, 1, 2, ..., K-1\}$$

the task is to learn a function to predict the class label y_i from X_i.

- $K = 2$: binary classification
- $K > 2$: multi-class classification

Many important practical problems can be cast as (multi-class) classification.

For example, Li, Burges, and Wu, NIPS 2007

Learning to Ranking Using Multiple Classification and Gradient Boosting.

Logistic Regression for Classification

First learn the class probabilities

$$\hat{p}_k = \mathbf{Pr}\left\{y = k|X\right\}, \qquad k = 0, 1, ..., K-1,$$

$$\sum_{k=0}^{K-1} \hat{p}_k = 1, \qquad \left(\text{only } K-1 \text{ degrees of freedom}\right).$$

Then assign the class label according to

$$\hat{y}|X = \underset{k}{\text{argmax}}\ \hat{p}_k$$

Multinomial Logit Probability Model

$$p_k = \frac{e^{F_k}}{\sum_{s=0}^{K-1} e^{F_s}}$$

where $F_k = F_k(\mathbf{x})$ is the function to be learned from the data.

Classical logistic regression:

$$F(\mathbf{x}) = \beta^{\mathsf{T}}\mathbf{x}$$

The task is to learn the coefficients β.

Flexible additive modeling:

$$F(\mathbf{x}) = F^{(M)}(\mathbf{x}) = \sum_{m=1}^{M} \rho_m h(\mathbf{x}; \mathbf{a}_m),$$

$h(\mathbf{x}; \mathbf{a})$ is a pre-specified function (e.g., trees).

The task is to learn the parameters ρ_m and $\mathbf{a}_m$.

————————-

Both **LogitBoost** (Friedman et. al, 2000) and **MART** (Multiple Additive Regression Trees, Friedman 2001) adopted this model.

Learning Logistic Regression by Maximum Likelihood

Seek $F_{i,k}$ to maximize the mutlinomial likelihood: Suppose $y_i = k$,

$$Lik \propto p_{i,0}^0 \times ... \times {p_{i,k}}^1 \times ... \times p_{i,K-1}^0 = p_{i,k}$$

or equivalently, maximizing the log likelihood:

$$\log Lik \propto \log p_{i,k}$$

Or equivalently, minimizing the negative log likelihood loss

$$L_i = -\log p_{i,k}, \qquad (y_i = k)$$

The Negative Log-Likelihood Loss

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

$$r_{i,k} = \begin{cases} 1 & \text{if } y_i = k \\ 0 & \text{otherwise} \end{cases}$$

Two Basic Optimization Methods for Maximum Likelihood

1. **Newton's Method**

   Uses the first and second derivatives of the loss function.

   The method in LogitBoost.

2. **Gradient Descent**

   Only uses the first order derivative of the loss function.

   ——————————-

   MART used a creative combination of gradient descent and Newton's method.

Derivatives Used in LogitBoost and MART

The loss function:

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

The first derivative:

$$\frac{\partial L_i}{\partial F_{i,k}} = - \left(r_{i,k} - p_{i,k} \right)$$

The second derivative:

$$\frac{\partial^2 L_i}{\partial F_{i,k}^2} = p_{i,k} \left(1 - p_{i,k} \right).$$

The Original LogitBoost Algorithm

1: $F_{i,k} = 0,\ p_{i,k} = \frac{1}{K}, \quad k = 0$ to $K-1,\ i = 1$ to N

2: For $m = 1$ to M Do

3: For $k = 0$ to $K-1$, Do

4: $w_{i,k} = p_{i,k}\left(1 - p_{i,k}\right), \qquad z_{i,k} = \frac{r_{i,k} - p_{i,k}}{p_{i,k}\left(1 - p_{i,k}\right)}.$

5: Fit the function $f_{i,k}$ by a weighted least-square of $z_{i,k}$ to $\mathbf{x}_i$ with weights $w_{i,k}$.

6: $F_{i,k} = F_{i,k} + \nu \frac{K-1}{K}\left(f_{i,k} - \frac{1}{K}\sum_{k=0}^{K-1} f_{i,k}\right)$

7: End

8: $p_{i,k} = \exp(F_{i,k}) / \sum_{s=0}^{K-1} \exp(F_{i,s}), \quad k = 0$ to $K-1,\ i = 1$ to N

9: End

The Original MART Algorithm

1: $F_{i,k} = 0,\ p_{i,k} = \frac{1}{K}, \quad k = 0$ to $K-1,\ i = 1$ to N

2: For $m = 1$ to M Do

3: For $k = 0$ to $K-1$ Do

4: $\{R_{j,k,m}\}_{j=1}^{J} = J$-terminal node regression tree from $\{r_{i,k} - p_{i,k},\ \mathbf{x}_i\}_{i=1}^{N}$

5: $\beta_{j,k,m} = \frac{K-1}{K} \frac{\sum_{\mathbf{x}_i \in R_{j,k,m}} r_{i,k} - p_{i,k}}{\sum_{\mathbf{x}_i \in R_{j,k,m}} \left(1-p_{i,k}\right) p_{i,k}}$

6: $F_{i,k} = F_{i,k} + \nu \sum_{j=1}^{J} \beta_{j,k,m} 1_{\mathbf{x}_i \in R_{j,k,m}}$

7: End

8: $p_{i,k} = \exp(F_{i,k}) / \sum_{s=0}^{K-1} \exp(F_{i,s}), \quad k = 0$ to $K-1,\ i = 1$ to N

9: End

Comparing LogitBoost with MART

- LogitBoost used first and second derivatives to construct the trees.
- MART only used the first order information to construct the trees.
- Both used second-order information to update values of the terminal nodes.
- LogitBoost was believed to have numerical instability problems.

The Numerical Issue in LoigtBoost

4: $w_{i,k} = p_{i,k}\left(1 - p_{i,k}\right), \qquad z_{i,k} = \frac{r_{i,k} - p_{i,k}}{p_{i,k}\left(1 - p_{i,k}\right)}.$

5: Fit the function $f_{i,k}$ by a weighted least-square of $z_{i,k}$ to $\mathbf{x}_i$ with weights $w_{i,k}$.

6: $F_{i,k} = F_{i,k} + \nu \frac{K-1}{K}\left(f_{i,k} - \frac{1}{K}\sum_{k=0}^{K-1} f_{i,k}\right)$

The "instability issue":

When $p_{i,k}$ is close to 0 or 1, $z_{i,k} = z_{i,k} = \frac{r_{i,k} - p_{i,k}}{p_{i,k}(1 - p_{i,k})}$ may approach infinity.

The Numerical Issue in LoigtBoost

(Friedman et al 2000) used regression trees and suggested some "crucial implementation protections":

- In Line 4, compute $z_{i,k}$ by $\frac{1}{p_{i,k}}$ (if $r_{i,k}=1$) or $\frac{-1}{1-p_{i,k}}$ (if $r_{i,k}=0$).
- Bound $\left|z_{i,k}\right|$ by $z_{max} \in [2, 4]$.

Robust LogitBoost avoids this **pointwise thresholding** and is essentially free of numerical problems.

It turns out, the numerical issue does not really exist, especially when the trees are not too large.

Tree-Splitting Using the Second-Order Information

Feature values: x_i, $i = 1$ to N. Assume $x_1 \leq x_2 \leq ... \leq x_N$.

Weight values: w_i, $i = 1$ to N. **Response values**: z_i, $i = 1$ to N.

We seek the index s, $1 \leq s < N$, to maximize the gain of weighted SE:

$$\begin{aligned} Gain(s) =& SE_T - (SE_L + SE_R) \\ =& \sum_{i=1}^{N}(z_i - \bar{z})^2 w_i - \left[\sum_{i=1}^{s}(z_i - \bar{z}_L)^2 w_i + \sum_{i=s+1}^{N}(z_i - \bar{z}_R)^2 w_i\right] \end{aligned}$$

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, we obtain

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

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

Recall $w_i = p_{i,k}(1-p_{i,k}), \quad z_i = \frac{r_{i,k}-p_{i,k}}{p_{i,k}(1-p_{i,k})}$.

This procedure is numerically stable.

MART only used the first order information to construct the trees:

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

Which can also be derived by letting weights $w_{i,k} = 1$ and response $z_{i,k} = r_{i,k} - p_{i,k}$.

LogitBoost used more information and could be more accurate in many datasets.

Update terminal node values

In LogitBoost:

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

which is the same as in MART.

Robust LogitBoost

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: $\{R_{j,k,m}\}_{j=1}^{J} = J$-terminal node regression tree from $\{r_{i,k} - p_{i,k},\ \mathbf{x}_i\}_{i=1}^{N}$,

: with weights $p_{i,k}(1 - p_{i,k})$.

5: $\beta_{j,k,m} = \frac{K-1}{K} \frac{\sum_{\mathbf{x}_i \in R_{j,k,m}} r_{i,k} - p_{i,k}}{\sum_{\mathbf{x}_i \in R_{j,k,m}} (1-p_{i,k}) p_{i,k}}$

6: $F_{i,k} = F_{i,k} + \nu \sum_{j=1}^{J} \beta_{j,k,m} 1_{\mathbf{x}_i \in R_{j,k,m}}$

7: End

8: $p_{i,k} = \exp(F_{i,k}) / \sum_{s=0}^{K-1} \exp(F_{i,s}), \quad k = 0$ to $K-1,\ i = 1$ to N

9: End

Experiments on Binary Classification

(Multi-class classification is even more interesting!)

Data

IJCNN1: 49990 training samples, 91701 test samples

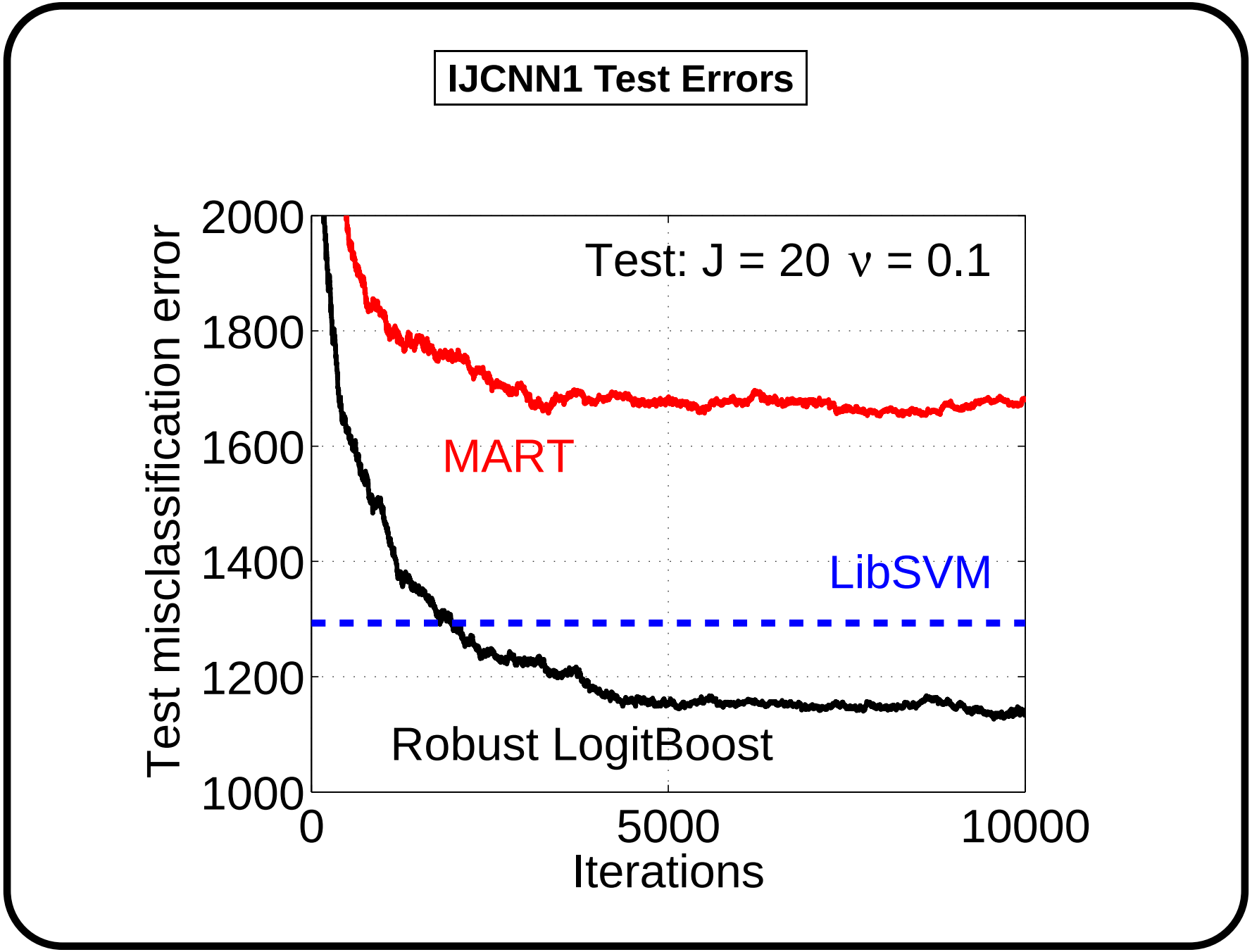
This dataset was used in a competition. LibSVM was the winner.

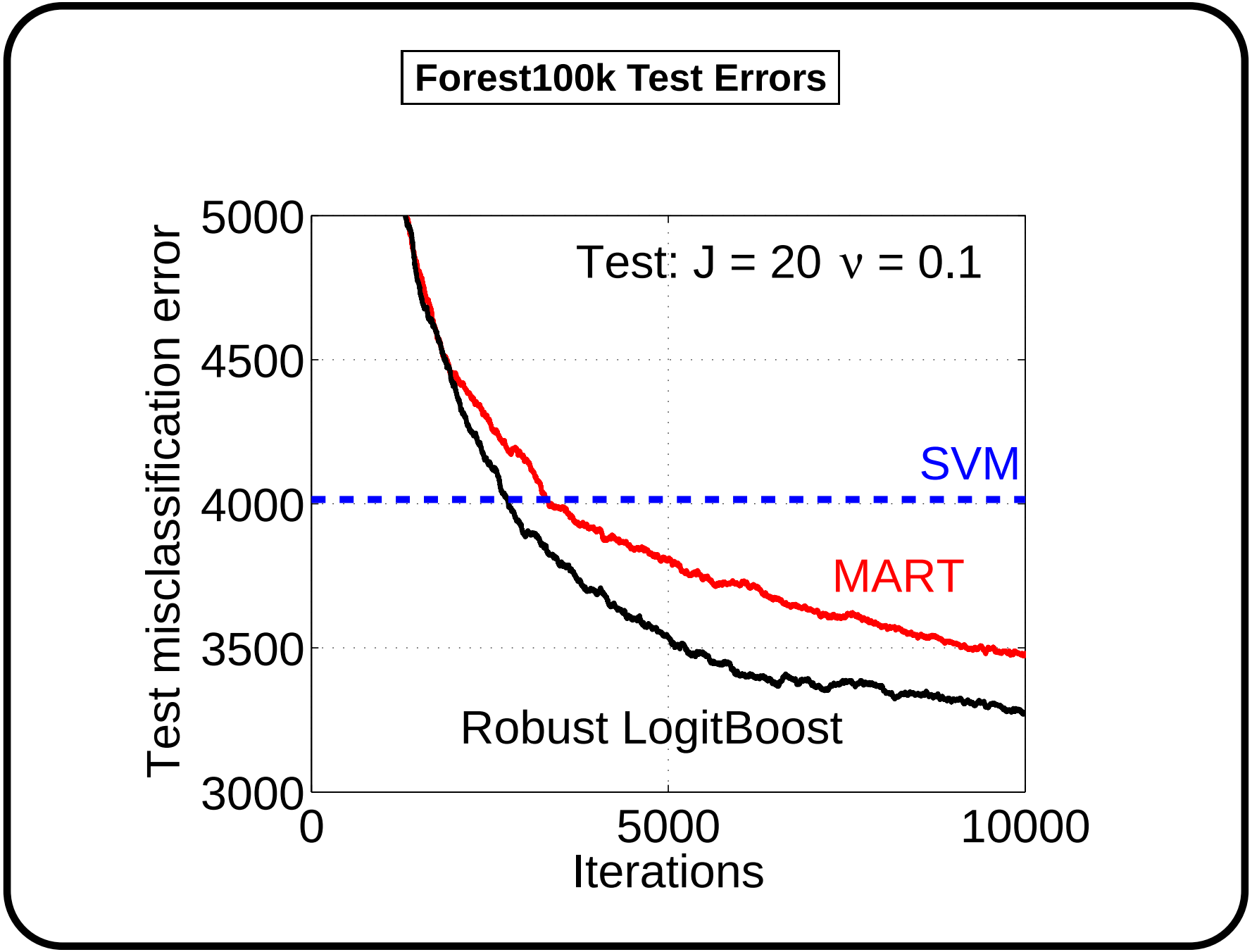
Forest100k: 100000 training samples, 50000 test samples

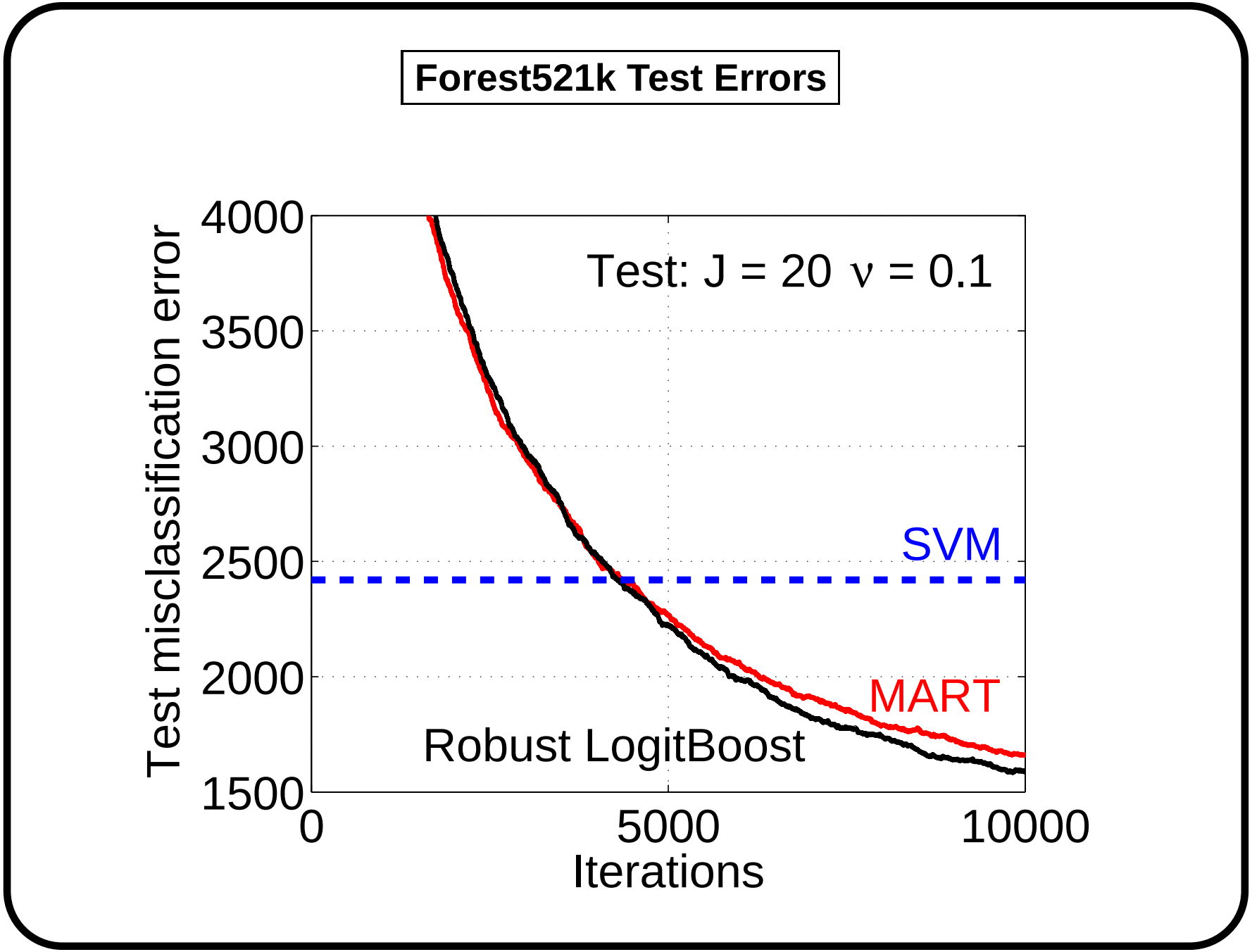
Forest521k: 521012 training samples, 50000 test samples

The two largest datasets from Bordes et al. JMLR 2005,

Fast Kernel Classifiers with Online and Active Learning


IJCNN1 Test Errors
Test misclassification error
2000
1800
1600
1400
1200
1000
0
5000
10000
Iterations
Test: J = 20 ν = 0.1
MART
LibSVM
Robust LogitBoost


Forest100k Test Errors
Test misclassification error
5000
4500
4000
3500
3000
0
5000
10000
Iterations
Test: J = 20 ν = 0.1
SVM
MART
Robust LogitBoost


Forest521k Test Errors
Test misclassification error
4000
3500
3000
2500
2000
1500
0
5000
10000
Iterations
Test: J = 20 ν = 0.1
SVM
MART
Robust LogitBoost

ABC-Boost for Multi-Class Classification

ABC = Adaptive Base Class

ABC-MART = ABC-Boost + MART

ABC-LogitBoost = ABC-Boost + (Robust) LogitBoost

The key to the success of ABC-Boost is the use of “better” derivatives.

Review Components of Logistic Regression

The multinomial logit probability model:

$$p_k = \frac{e^{F_k}}{\sum_{s=0}^{K-1} e^{F_s}}, \qquad \sum_{k=0}^{K-1} p_k = 1$$

where $F_k = F_k(\mathbf{x})$ is the function to be learned from the data.

The sum-to-zero constraint:

$$\sum_{k=0}^{K-1} F_k(\mathbf{x}) = 0$$

is commonly used to obtain a unique solution (only $K - 1$ degrees of freedom).

Why the sum-to-zero constraint?

$$\frac{e^{F_{i,k}+C}}{\sum_{s=0}^{K-1} e^{F_{i,s}+C}} = \frac{e^C e^{F_{i,k}}}{e^C \sum_{s=0}^{K-1} e^{F_{i,s}}} = \frac{e^{F_{i,k}}}{\sum_{s=0}^{K-1} e^{F_{i,s}}} = p_{i,k}.$$

For identifiability, one should impose a constraint.

One popular choice is to assume $\sum_{k=0}^{K-1} F_{i,k} = const$, equivalent to

$$\sum_{k=0}^{K-1} F_{i,k} = 0.$$

This is the assumption used in many papers including LogitBoost and MART.

The negative log-Likelihood loss

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

$$r_{i,k} = \begin{cases} 1 & \text{if } y_i = k \\ 0 & \text{otherwise} \end{cases} \qquad \sum_{k=0}^{K-1} r_{i,k} = 1$$

Derivatives used in LogitBoost and MART:

$$\frac{\partial L_i}{\partial F_{i,k}} = -\left(r_{i,k} - p_{i,k}\right)$$

$$\frac{\partial^2 L_i}{\partial F_{i,k}^2} = p_{i,k}\left(1 - p_{i,k}\right),$$

which could be derived without imposing any constraints on F_k.

Singularity of Hessian without Sum-to-zero Constraint

Without sum-to-zero constraint $\sum_{k=0}^{K-1} F_{i,k} = 0$:

$$\frac{\partial L_i}{\partial F_{i,k}} = -\left(r_{i,k} - p_{i,k}\right), \qquad \frac{\partial^2 L_i}{\partial F_{i,k}^2} = p_{i,k}\left(1 - p_{i,k}\right).$$

For example, when $K = 3$.

$$\begin{vmatrix} \frac{\partial^2 L_i}{\partial F_{i,0}^2} & \frac{\partial^2 L_i}{\partial F_{i,0} F_{i,1}} & \frac{\partial^2 L_i}{\partial F_{i,0} F_{i,2}} \\ \frac{\partial^2 L_i}{\partial F_{i,1} F_{i,0}} & \frac{\partial^2 L_i}{\partial F_{i,1}^2} & \frac{\partial^2 L_i}{\partial F_{i,1} F_{i,2}} \\ \frac{\partial^2 L_i}{\partial F_{i,2} F_{i,0}} & \frac{\partial^2 L_i}{\partial F_{i,2} F_{i,1}} & \frac{\partial^2 L_i}{\partial F_{i,2}^2} \end{vmatrix} = \begin{vmatrix} p_0(1-p_0) & -p_0 p_1 & -p_0 p_2 \\ -p_1 p_0 & p_1(1-p_1) & -p_1 p_2 \\ -p_2 p_0 & -p_2 p_1 & p_2(1-p_2) \end{vmatrix}$$

$$= 0$$

Diagonal Approximation

(Friedman et al 2000 and Friedman 2001) used diagonal approximation of Hessian:

$$\frac{K-1}{K}\begin{bmatrix} \frac{\partial^2 L_i}{\partial F_{i,0}^2} & & \\ & \frac{\partial^2 L_i}{\partial F_{i,1}^2} & \\ & & \frac{\partial^2 L_i}{\partial F_{i,2}^2} \end{bmatrix}$$

$$=\frac{K-1}{K}\begin{bmatrix} p_0(1-p_0) & & \\ & p_1(1-p_1) & \\ & & p_2(1-p_2) \end{bmatrix}$$

Derivatives Under Sum-to-zero Constraint

The loss function:

$$L_i = -\sum_{k=0}^{K-1} r_{i,k} \log p_{i,k}$$

The probability model and sum-to-zero constraint:

$$p_{i,k} = \frac{e^{F_{i,k}}}{\sum_{s=0}^{K-1} e^{F_{i,s}}}, \qquad \sum_{k=0}^{K-1} F_{i,k} = 0$$

Without loss of generality, we assume $k = 0$ is the base class

$$\boxed{F_{i,0} = -\sum_{i=1}^{K-1} F_{i,k}}$$

New derivatives:

$$\frac{\partial L_i}{\partial F_{i,k}} = (r_{i,0} - p_{i,0}) - (r_{i,k} - p_{i,k}),$$

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

MART and LogitBoost used:

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

Assume class 0 is the base class. The determinant of the Hessian is

$$\begin{vmatrix} \frac{\partial^2 L_i}{\partial F_{i,1}^2} & \frac{\partial^2 L_i}{\partial F_{i,1} F_{i,2}} \\ \frac{\partial^2 L_i}{\partial F_{i,2} F_{i,1}} & \frac{\partial^2 L_i}{\partial F_{i,2}^2} \end{vmatrix} =$$

$$\begin{vmatrix} p_0(1-p_0) + p_1(1-p_1) + 2p_0p_1 & p_0 - p_0^2 + p_0p_1 + p_0p_2 - p_1p_2 \\ p_0 - p_0^2 + p_0p_1 + p_0p_2 - p_1p_2 & p_0(1-p_0) + p_2(1-p_2) + 2p_0p_2 \end{vmatrix}$$

$$= p_0p_1 + p_0p_2 + p_1p_2 - p_0p_1^2 - p_0p_2^2 - p_1p_2^2 - p_2p_1^2 - p_1p_0^2 - p_2p_0^2 + 6p_0p_1p_2,$$

independent of the choice of the base class.

However, diagonal approximation appears to be a must when using trees. Thus **the choice of base class matters**.

Adaptive Base Class Boost (ABC-Boost)

Two Key Ideas of ABC-Boost:

1. Formulate the multi-class boosting algorithm by considering a base class.

   Do not have to train for the base class, which is inferred from the sum-zero-constraint $\sum_{k=0}^{K-1} F_{i,k} = 0$.

2. At each boosting step, adaptively choose the base class.

How to choose the base class?

- We should choose the base class based on performance (training loss). How?

- (One Idea) Exhaustively search for all K base classes and choose the base class that leads to the best performance (smallest training loss).

  - Computationally expensive (but not too bad, unless K is really large)

  - Good performance can be achieved.

- Many other ideas.

ABC-MART = ABC-Boost + MART.

ABC-LogitBoost = ABC-Boost + Robust LogitBoost.

The Original MART Algorithm

1: $F_{i,k} = 0,\ p_{i,k} = \frac{1}{K}, \quad k = 0 \text{ to } K-1,\ i = 1 \text{ to } N$

2: For $m = 1$ to M Do

3: For $k = 0$ to $K-1$ Do

4: $\{R_{j,k,m}\}_{j=1}^{J} = J$-terminal node regression tree from $\{r_{i,k} - p_{i,k},\ \mathbf{x}_i\}_{i=1}^{N}$

5: $\beta_{j,k,m} = \frac{K-1}{K} \frac{\sum_{\mathbf{x}_i \in R_{j,k,m}} r_{i,k} - p_{i,k}}{\sum_{\mathbf{x}_i \in R_{j,k,m}} (1-p_{i,k}) p_{i,k}}$

6: $F_{i,k} = F_{i,k} + \nu \sum_{j=1}^{J} \beta_{j,k,m} 1_{\mathbf{x}_i \in R_{j,k,m}}$

7: End

8: $p_{i,k} = \exp(F_{i,k}) / \sum_{s=0}^{K-1} \exp(F_{i,s}), \quad k = 0 \text{ to } K-1,\ i = 1 \text{ to } N$

9: End

ABC-MART

1: $F_{i,k} = 0$, $k = 0$ to $K-1$, $i = 1$ to N

2: For $m = 1$ to M Do

: For $b = 0$ to $K-1$ DO

3: For $k = 0$ to $K-1$ (and $k \neq b$) Do

4: $\{R_{j,k,m}\}_{j=1}^{J} = J$-terminal node tree from $\{r_{i,k} - p_{i,k} - (r_{i,b} - p_{i,b}),\ \mathbf{x}_i\}_{i=1}^{N}$

5: $$\beta_{j,k,m} = \frac{\sum_{\mathbf{x}_i \in R_{j,k,m}} (r_{i,k} - p_{i,k}) - (r_{i,b} - p_{i,b})}{\sum_{\mathbf{x}_i \in R_{j,k,m}} (1-p_{i,k}) p_{i,k} + (1-p_{i,b}) p_{i,b} + 2 p_{i,k} p_{i,b}}$$

6: $$F_{i,k} = F_{i,k} + \nu \sum_{j=1}^{J} \beta_{j,k,m} 1_{\mathbf{x}_i \in R_{j,k,m}}$$

7: End

8: $p_{i,k} = \exp(F_{i,k}) / \sum_{s=0}^{K-1} \exp(F_{i,s})$

9: End

Datasets

- **UCI-Covertype** Total 581012 samples.

  Two datasets were generated: **Covertype290k**, **Covertype145k**

- **UCI-Poker** Original 25010 training samples and 1 million test samples.

  Poker25kT1, **Poker25kT2**, **Poker525k**, **Poker275k**, **Poker150k**, **Poker100k**.

- **MNIST** Originally 60000 training samples and 10000 test samples.

  MNIST10k swapped the training with test samples.

- **Many variations of MNIST** Original MNIST is a well-known easy problem. (`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.

- **UCI-Letter** Total 20000 samples.

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

Summary of test mis-classification errors

Dataset	**mart**	**abc-mart**	**logitboost**	**abc-logitboost**	logistic regression	# test
Covertype290k	11350	10454	10765	9727	80233	290506
Covertype145k	15767	14665	14928	13986	80314	290506
Poker525k	7061	2424	2704	1736	248892	500000
Poker275k	15404	3679	6533	2727	248892	500000
Poker150k	22289	12340	16163	5104	248892	500000
Poker100k	27871	21293	25715	13707	248892	500000
Poker25kT1	43575	34879	46789	37345	250110	500000
Poker25kT2	42935	34326	46600	36731	249056	500000
Mnist10k	2815	2440	2381	2102	13950	60000
M-Basic	2058	1843	1723	1602	10993	50000
M-Rotate	7674	6634	6813	5959	26584	50000
M-Image	5821	4727	4703	4268	19353	50000
M-Rand	6577	5300	5020	4725	18189	50000
M-RotImg	24912	23072	22962	22343	33216	50000
M-Noise1	305	245	267	234	935	2000
M-Noise2	325	262	270	237	940	2000
M-Noise3	310	264	277	238	954	2000
M-Noise4	308	243	256	238	933	2000
M-Noise5	294	244	242	227	867	2000
M-Noise6	279	224	226	201	788	2000
Letter15k	155	125	139	109	1130	5000
Letter4k	1370	1149	1252	1055	3712	16000
Letter2k	2482	2220	2309	2034	4381	18000

P-Values

- $P1$: for testing if abc-mart has significantly lower **error rates** than mart.
- $P2$: for testing if (robust) logitboost has significantly lower error rates than mart.
- $P3$: for testing if abc-logitboost has significantly lower error rates than abc-mart.
- $P4$: for testing if abc-logitboost has significantly lower error rates than (robust) logitboost.

P-values are computed using binomial distributions and normal approximations.

Summary of test P-Values

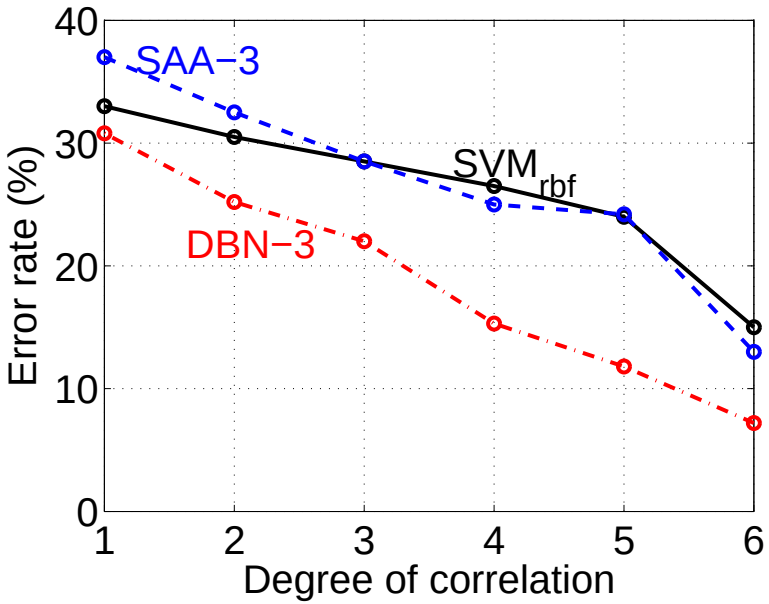
Dataset	$P1$	$P2$	$P3$	$P4$
Covertype290k	3×10^{-10}	3×10^{-5}	9×10^{-8}	8×10^{-14}
Covertype145k	4×10^{-11}	4×10^{-7}	2×10^{-5}	7×10^{-9}
Poker525k	0	0	0	0
Poker275k	0	0	0	0
Poker150k	0	0	0	0
Poker100k	0	0	0	0
Poker25kT1	0	—	—	0
Poker25kT2	0	—	—	0
Mnist10k	5×10^{-8}	3×10^{-10}	1×10^{-7}	1×10^{-5}
M-Basic	2×10^{-4}	1×10^{-8}	1×10^{-5}	0.0164
M-Rotate	0	5×10^{-15}	6×10^{-11}	3×10^{-16}
M-Image	0	0	2×10^{-7}	7×10^{-7}
M-Rand	0	0	7×10^{-10}	8×10^{-4}
M-RotImg	0	0	2×10^{-6}	4×10^{-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	2×10^{-6}	0.008	0.019	1×10^{-5}
Letter2k	2×10^{-5}	0.003	0.001	4×10^{-6}

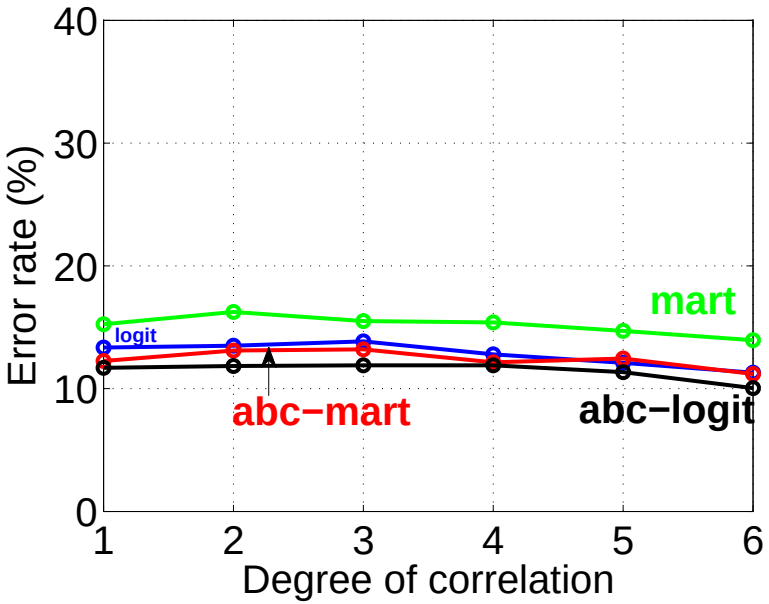
Comparisons with SVM and Deep Learning

Datasets: M-Noise1 to M-Noise6

Results on SVM, Neural Nets, and Deep Learning are from

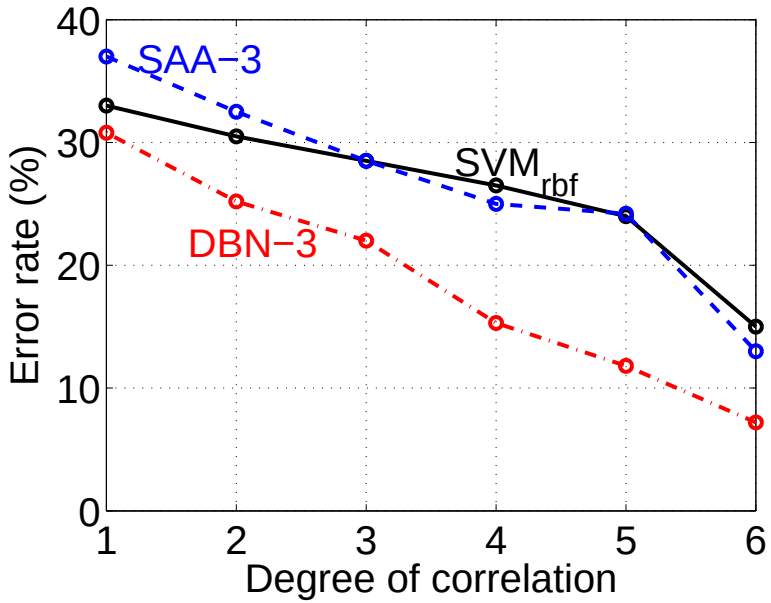
`www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DeepVsShallowComparisonICML2007`

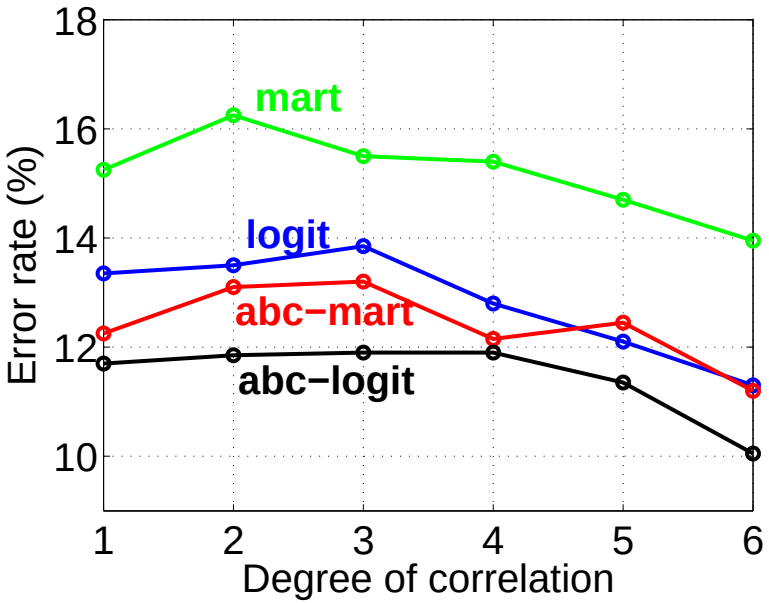




Comparisons with SVM and Deep Learning

Datasets: M-Noise1 to M-Noise6

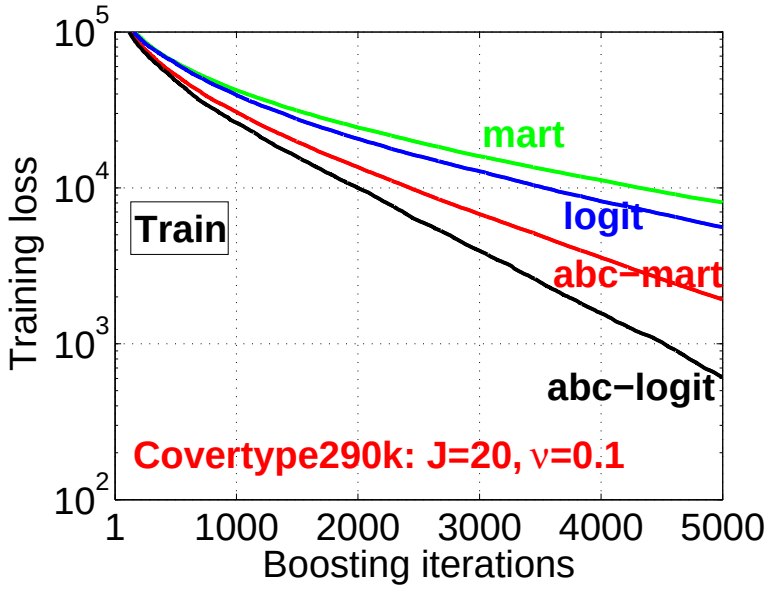


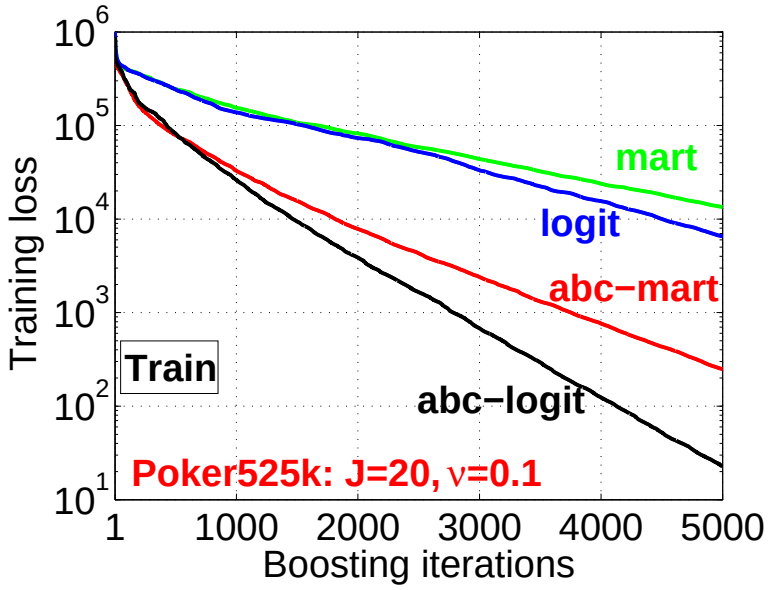


More Comparisons with SVM and Deep Learning

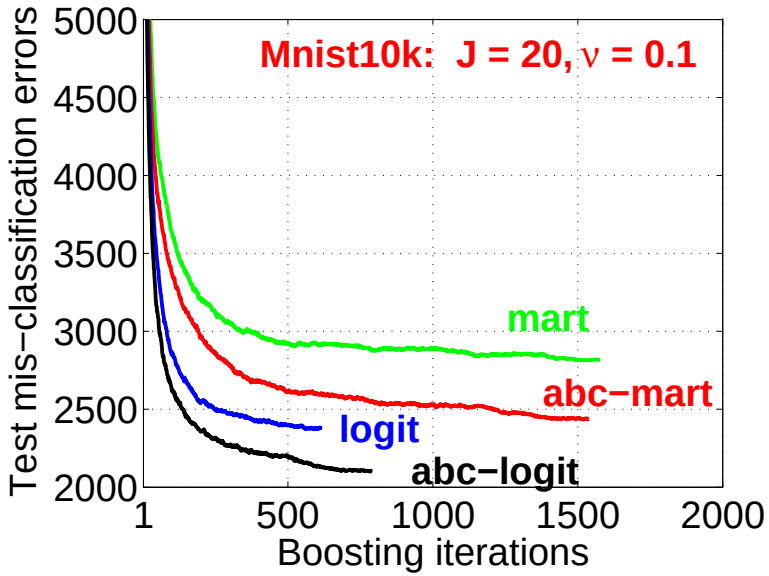
	M-Basic	M-Rotate	M-Image	M-Rand	M-RotImg
SVM-RBF	**3.05%**	11.11%	22.61%	14.58%	55.18%
SVM-POLY	3.69%	15.42%	24.01%	16.62%	56.41%
NNET	4.69%	18.11%	27.41%	20.04%	62.16%
DBN-3	3.11%	**10.30%**	16.31%	**6.73%**	47.39%
SAA-3	3.46%	**10.30%**	23.00%	11.28%	51.93%
DBN-1	3.94%	14.69%	16.15%	9.80%	52.21%
mart	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%**

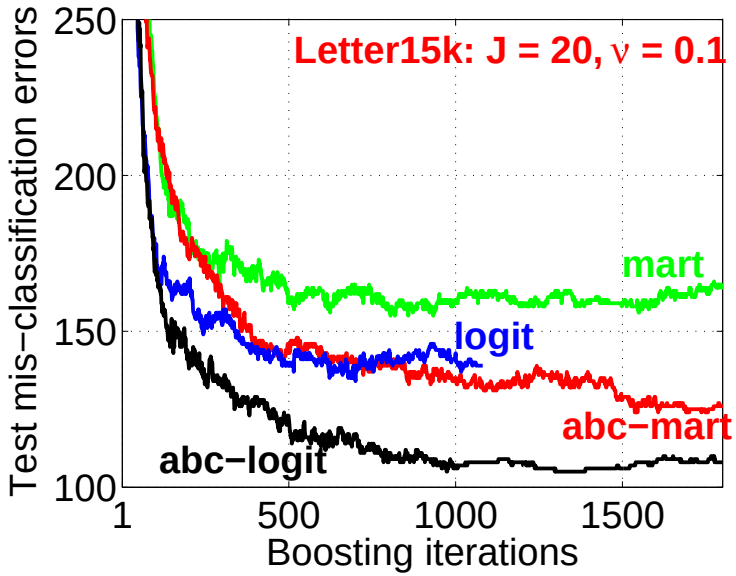
Training Loss Vs Boosting Iterations

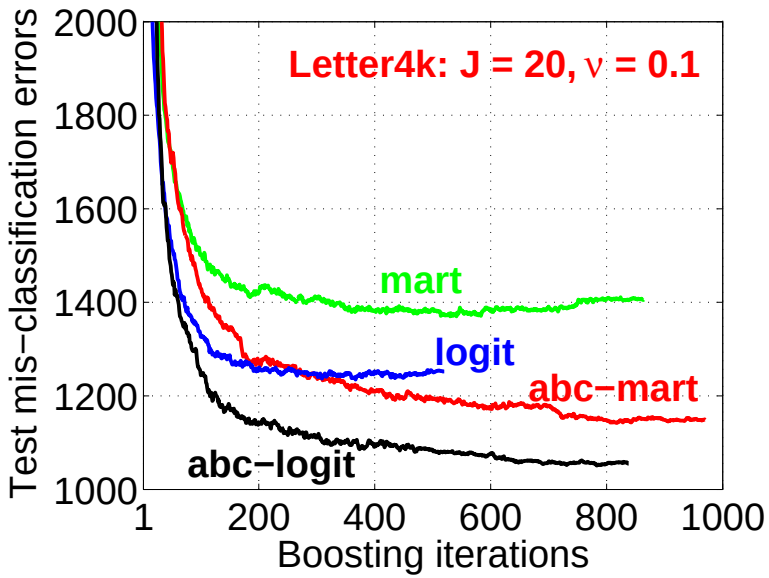


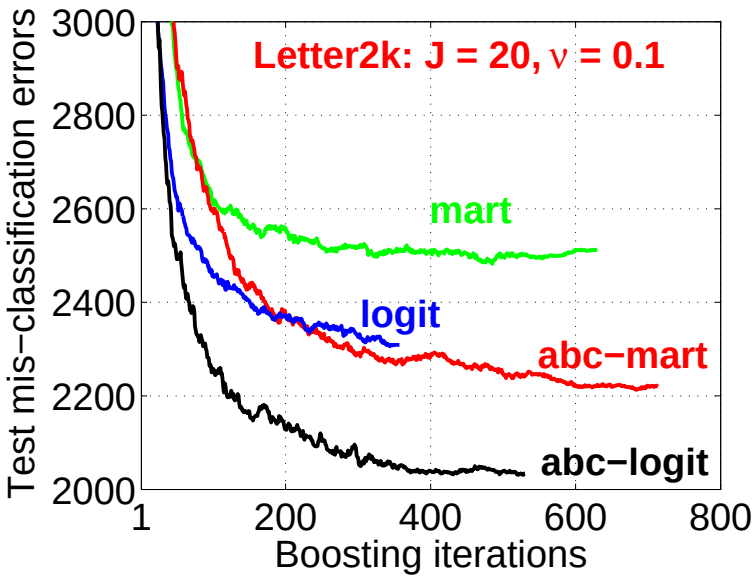


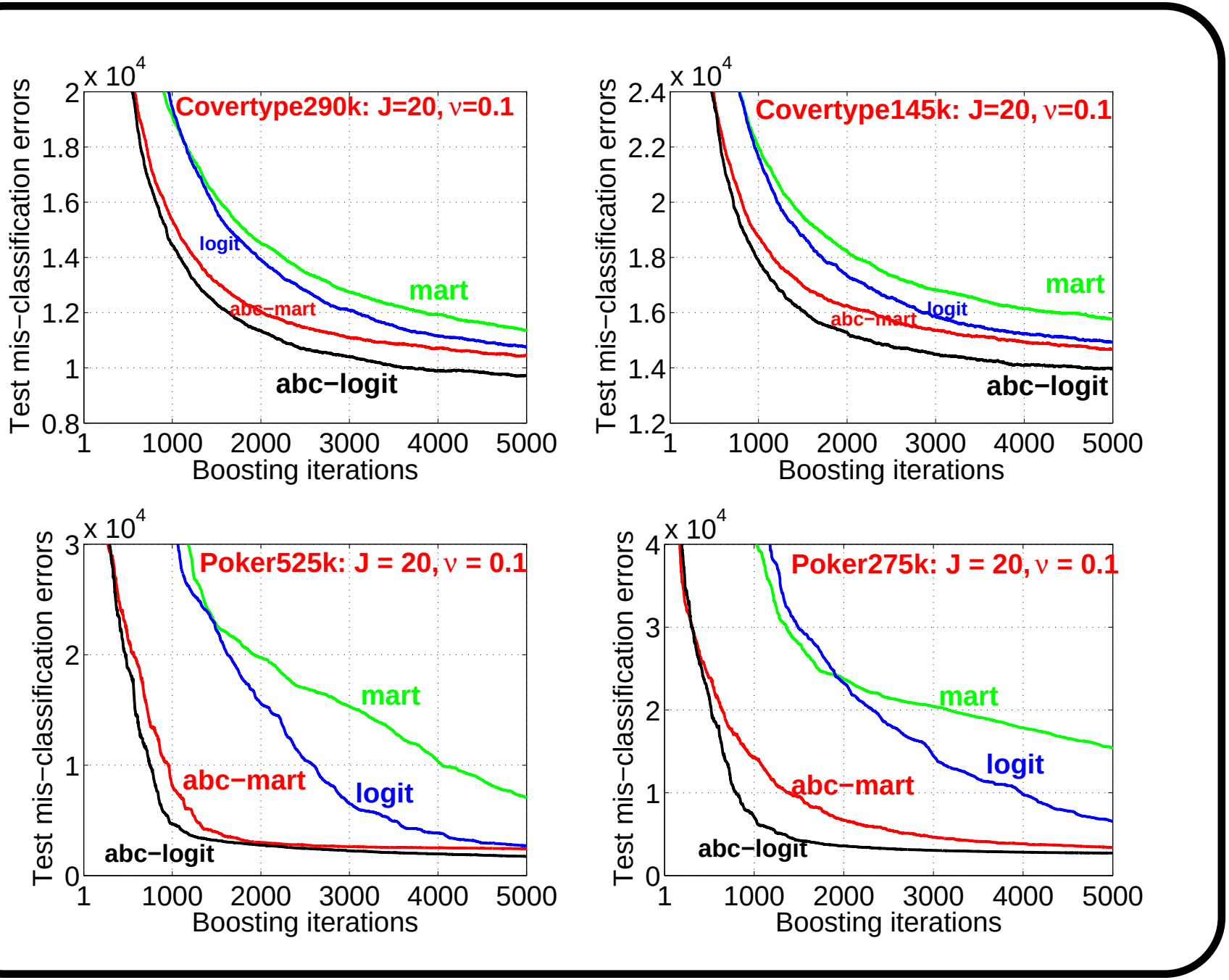
Test Errors Vs Boosting Iterations

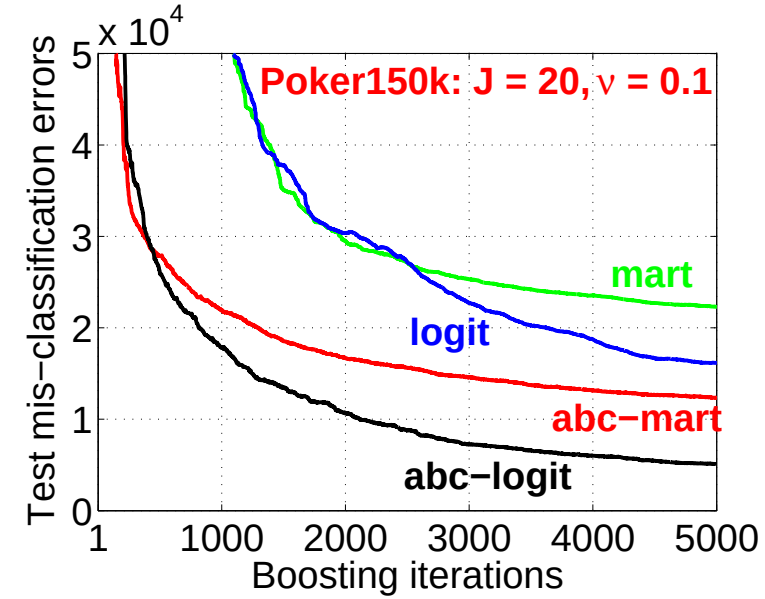


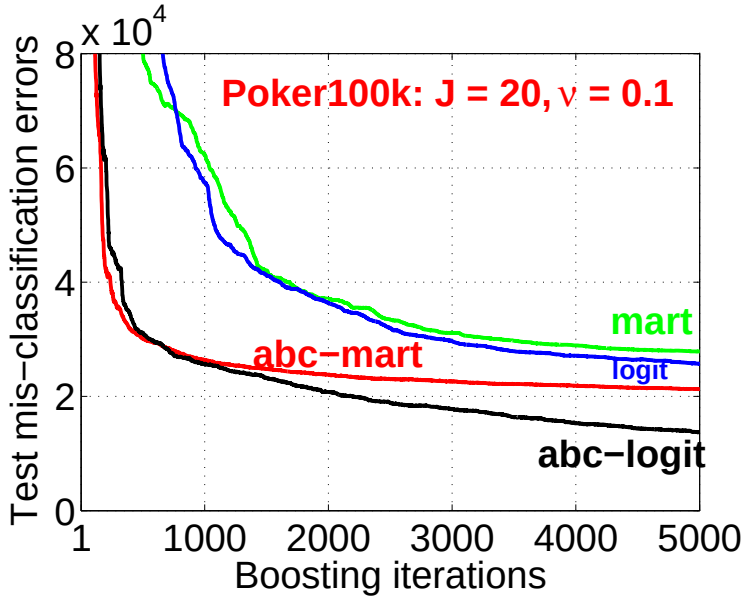


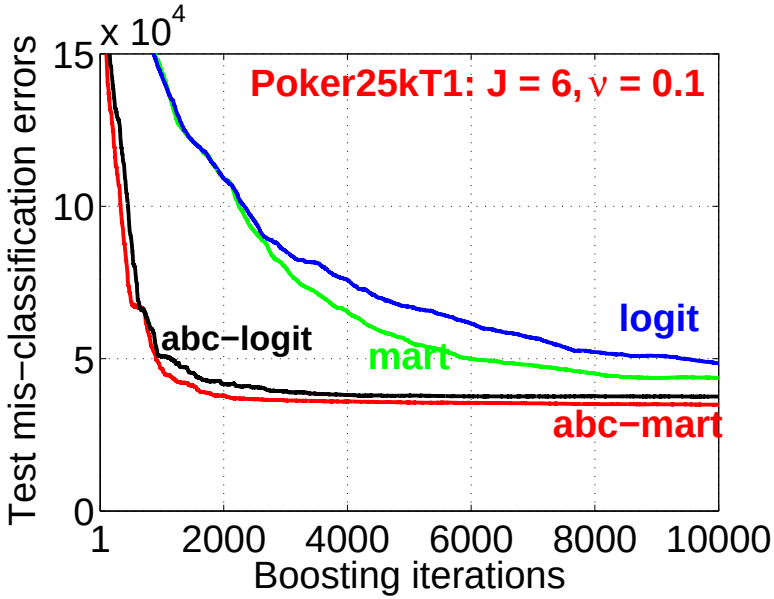

Letter4k: J = 20, ν = 0.1
Test mis−classification errors
2000
1800
1600
1400
1200
1000
1
200
400
600
800
1000
Boosting iterations
mart
logit
abc−mart
abc−logit

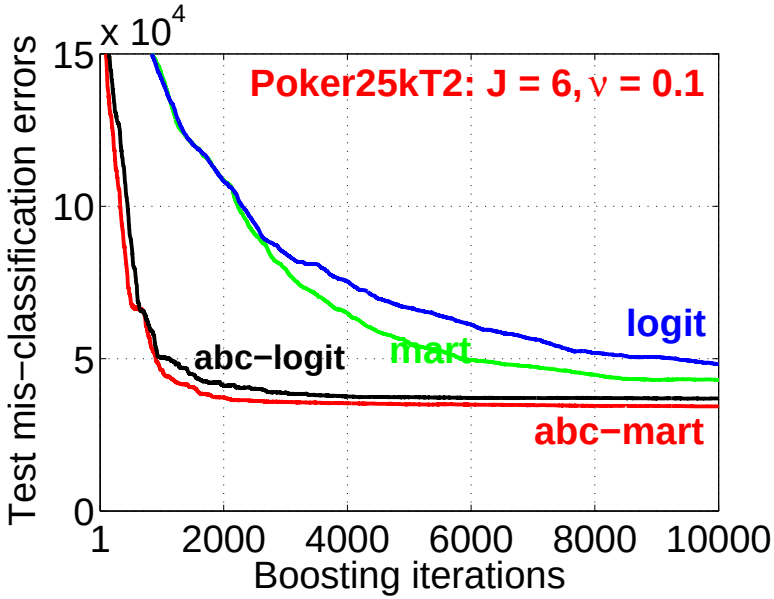

Letter2k: J = 20, ν = 0.1
Test mis−classification errors
3000
2800
2600
2400
2200
2000
1
200
400
600
800
Boosting iterations
mart
logit
abc−mart
abc−logit


Covertype290k: J=20, ν=0.1
Test mis−classification errors
x 10^4
Boosting iterations
logit
mart
abc−mart
abc−logit
Covertype145k: J=20, ν=0.1
Poker525k: J = 20, ν = 0.1
Poker275k: J = 20, ν = 0.1


x 10^4
Test mis−classification errors
Poker150k: J = 20, ν = 0.1
mart
logit
abc−mart
abc−logit
Boosting iterations


x 10^4
Test mis−classification errors
Poker100k: J = 20, ν = 0.1
mart
abc−mart
logit
abc−logit
Boosting iterations


x 10^4
Test mis−classification errors
Poker25kT1: J = 6, ν = 0.1
logit
abc−logit
mart
abc−mart
Boosting iterations


x 10^4
Test mis−classification errors
Poker25kT2: J = 6, ν = 0.1
logit
abc−logit
mart
abc−mart
Boosting iterations

Detailed Experiment Results on Mnist10k

$J \in \{4, 6, 8, 10, 12, 14, 16, 18, 20, 24, 30, 40, 50\}$

$\nu \in \{0.04, 0.06, 0.08, 0.1\}$.

The goal is to show the improvements at all reasonable combinations of J and ν.

	mart	**abc-mart**		
	$\nu = 0.04$	$\nu = 0.06$	$\nu = 0.08$	$\nu = 0.1$
$J = 4$	3356 **3060**	3329 **3019**	3318 **2855**	3326 **2794**
$J = 6$	3185 **2760**	3093 **2626**	3129 **2656**	3217 **2590**
$J = 8$	3049 **2558**	3054 **2555**	3054 **2534**	3035 **2577**
$J = 10$	3020 **2547**	2973 **2521**	2990 **2520**	2978 **2506**
$J = 12$	2927 **2498**	2917 **2457**	2945 **2488**	2907 **2490**
$J = 14$	2925 **2487**	2901 **2471**	2877 **2470**	2884 **2454**
$J = 16$	2899 **2478**	2893 **2452**	2873 **2465**	2860 **2451**
$J = 18$	2857 **2469**	2880 **2460**	2870 **2437**	2855 **2454**
$J = 20$	2833 **2441**	2834 **2448**	2834 **2444**	2815 **2440**
$J = 24$	2840 **2447**	2827 **2431**	2801 **2427**	2784 **2455**
$J = 30$	2826 **2457**	2822 **2443**	2828 **2470**	2807 **2450**
$J = 40$	2837 **2482**	2809 **2440**	2836 **2447**	2782 **2506**
$J = 50$	2813 **2502**	2826 **2459**	2824 **2469**	2786 **2499**

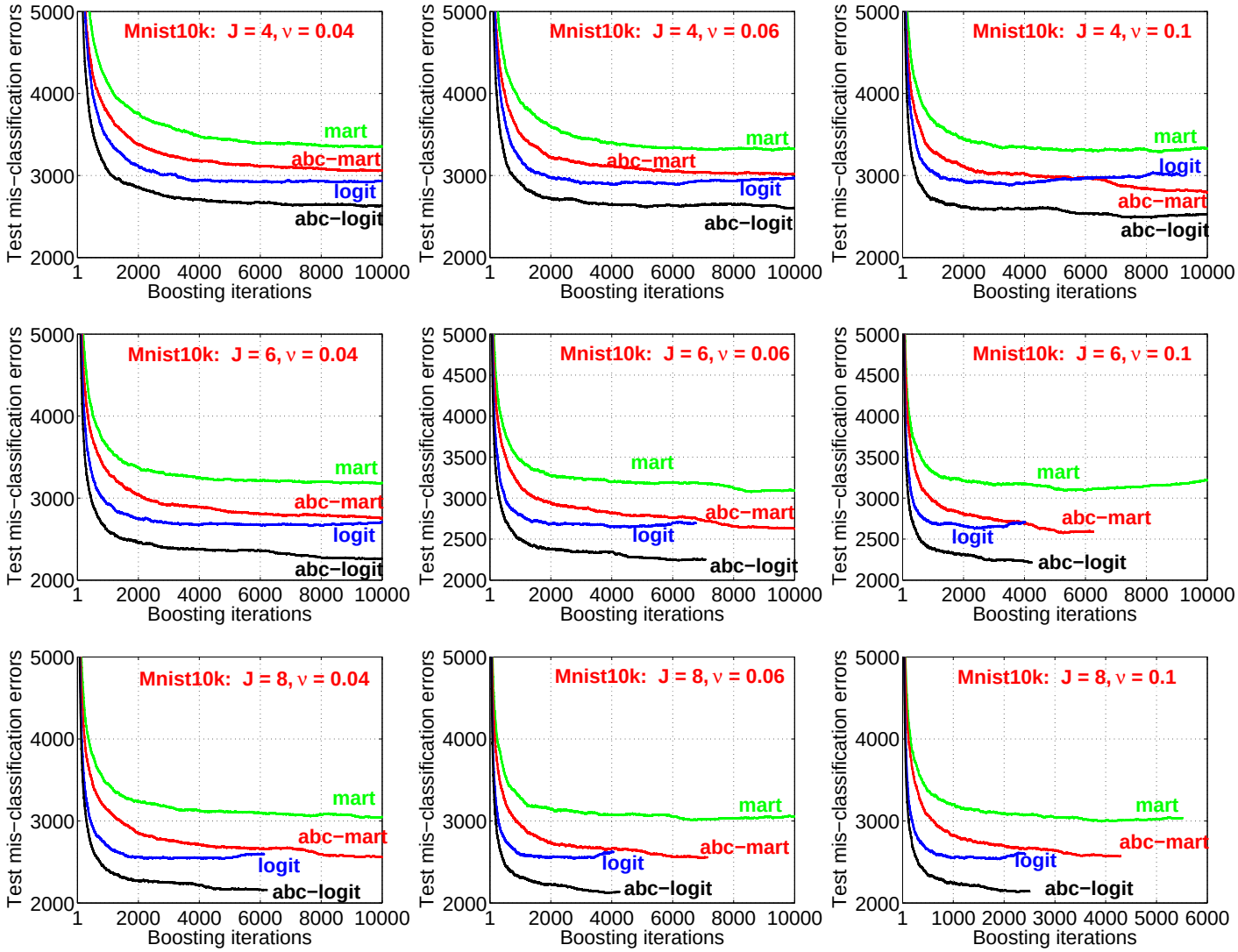
	logitboost	**abc-logit**		
	$\nu = 0.04$	$\nu = 0.06$	$\nu = 0.08$	$\nu = 0.1$
$J = 4$	2936 **2630**	2970 **2600**	2980 **2535**	3017 **2522**
$J = 6$	2710 **2263**	2693 **2252**	2710 **2226**	2711 **2223**
$J = 8$	2599 **2159**	2619 **2138**	2589 **2120**	2597 **2143**
$J = 10$	2553 **2122**	2527 **2118**	2516 **2091**	2500 **2097**
$J = 12$	2472 **2084**	2468 **2090**	2468 **2090**	2464 **2095**
$J = 14$	2451 **2083**	2420 **2094**	2432 **2063**	2419 **2050**
$J = 16$	2424 **2111**	2437 **2114**	2393 **2097**	2395 **2082**
$J = 18$	2399 **2088**	2402 **2087**	2389 **2088**	2380 **2097**
$J = 20$	2388 **2128**	2414 **2112**	2411 **2095**	2381 **2102**
$J = 24$	2442 **2174**	2415 **2147**	2417 **2129**	2419 **2138**
$J = 30$	2468 **2235**	2434 **2237**	2423 **2221**	2449 **2177**
$J = 40$	2551 **2310**	2509 **2284**	2518 **2257**	2531 **2260**
$J = 50$	2612 **2353**	2622 **2359**	2579 **2332**	2570 **2341**

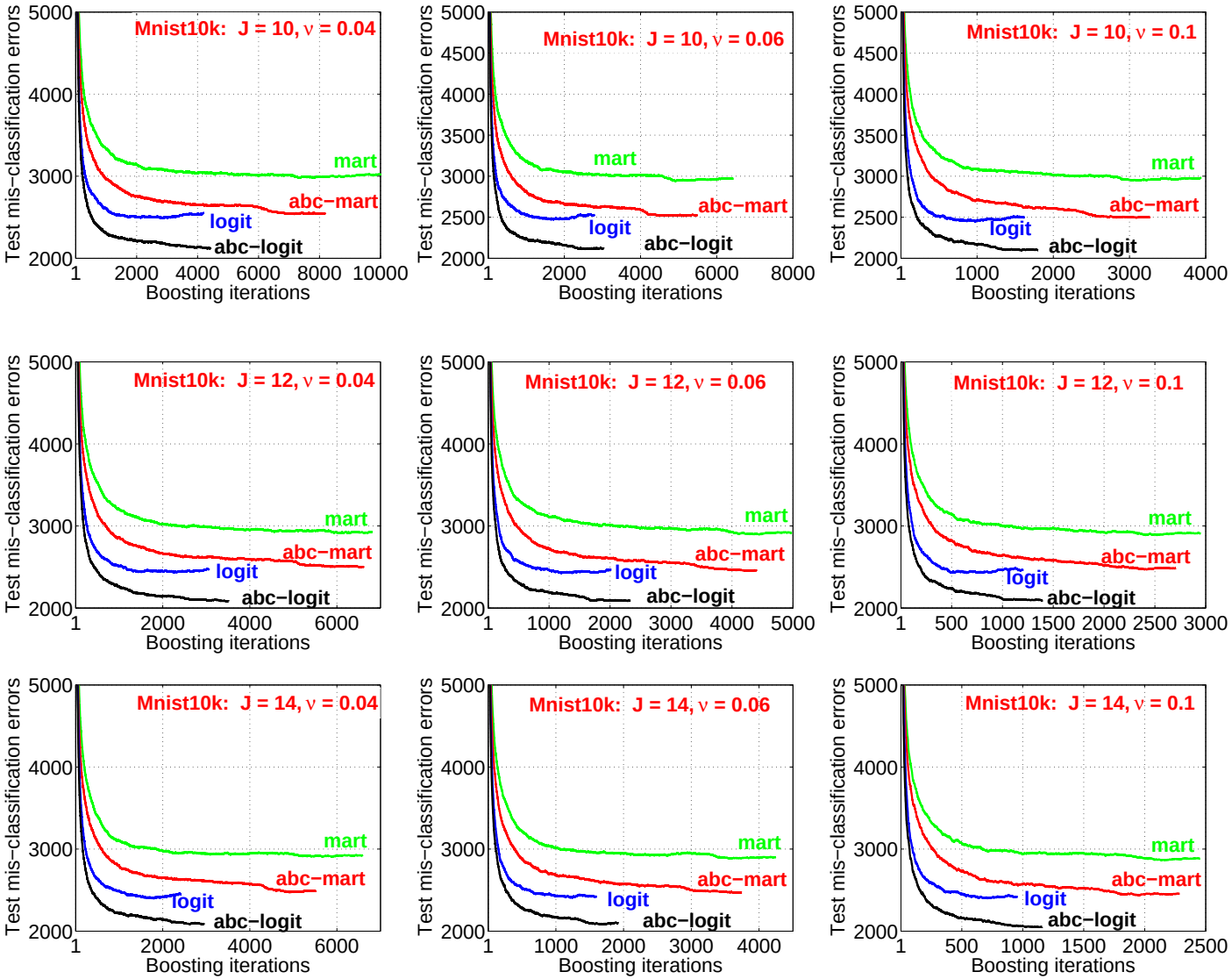
P1				
	$\nu = 0.04$	$\nu = 0.06$	$\nu = 0.08$	$\nu = 0.1$
$J = 4$	7×10^{-5}	3×10^{-5}	7×10^{-10}	1×10^{-12}
$J = 6$	8×10^{-9}	1×10^{-10}	9×10^{-11}	0
$J = 8$	9×10^{-12}	4×10^{-12}	5×10^{-13}	2×10^{-10}
$J = 10$	4×10^{-11}	2×10^{-10}	4×10^{-11}	3×10^{-11}
$J = 12$	1×10^{-9}	7×10^{-11}	1×10^{-10}	3×10^{-9}
$J = 14$	6×10^{-10}	1×10^{-9}	6×10^{-9}	9×10^{-10}
$J = 16$	2×10^{-9}	3×10^{-10}	6×10^{-9}	5×10^{-9}
$J = 18$	3×10^{-8}	2×10^{-9}	6×10^{-10}	9×10^{-9}
$J = 20$	2×10^{-8}	3×10^{-8}	2×10^{-8}	6×10^{-8}
$J = 24$	2×10^{-8}	1×10^{-8}	6×10^{-8}	2×10^{-6}
$J = 30$	1×10^{-7}	5×10^{-8}	2×10^{-7}	2×10^{-7}
$J = 40$	3×10^{-7}	1×10^{-7}	2×10^{-8}	5×10^{-5}
$J = 50$	6×10^{-6}	1×10^{-7}	3×10^{-7}	3×10^{-5}

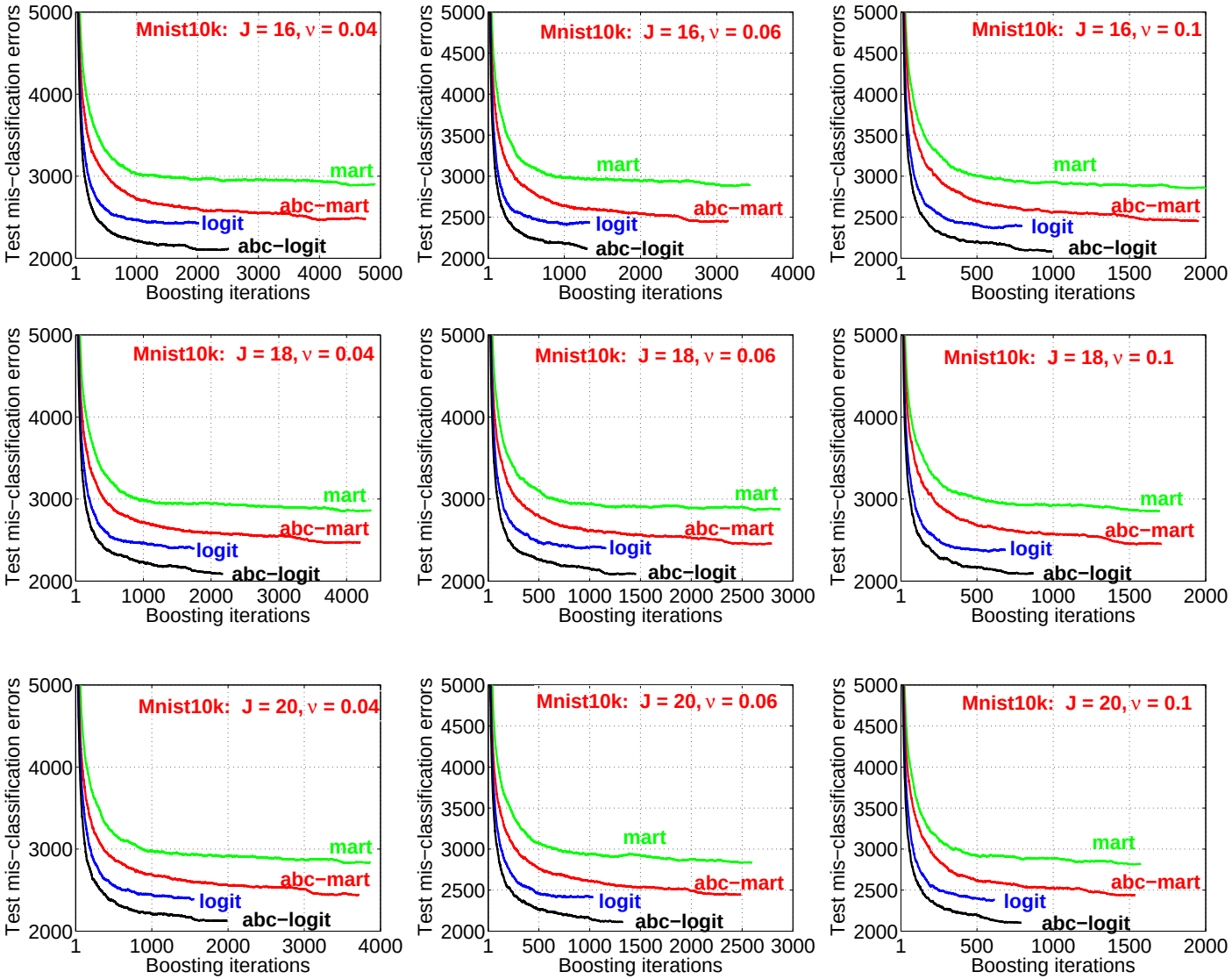
P2				
	$\nu = 0.04$	$\nu = 0.06$	$\nu = 0.08$	$\nu = 0.1$
$J = 4$	2×10^{-8}	2×10^{-6}	6×10^{-6}	3×10^{-6}
$J = 6$	1×10^{-10}	4×10^{-8}	9×10^{-9}	8×10^{-12}
$J = 8$	4×10^{-10}	2×10^{-9}	1×10^{-10}	1×10^{-9}
$J = 10$	7×10^{-11}	4×10^{-10}	3×10^{-11}	2×10^{-11}
$J = 12$	1×10^{-10}	2×10^{-10}	2×10^{-11}	3×10^{-10}
$J = 14$	2×10^{-11}	8×10^{-12}	2×10^{-10}	3×10^{-11}
$J = 16$	1×10^{-11}	8×10^{-11}	7×10^{-12}	3×10^{-11}
$J = 18$	5×10^{-11}	9×10^{-12}	6×10^{-12}	9×10^{-12}
$J = 20$	2×10^{-10}	2×10^{-9}	1×10^{-9}	4×10^{-10}
$J = 24$	1×10^{-8}	3×10^{-9}	3×10^{-8}	1×10^{-7}
$J = 30$	2×10^{-7}	2×10^{-8}	5×10^{-9}	2×10^{-7}
$J = 40$	3×10^{-5}	1×10^{-5}	4×10^{-6}	2×10^{-4}
$J = 50$	0.0026	0.0023	3×10^{-4}	0.0013

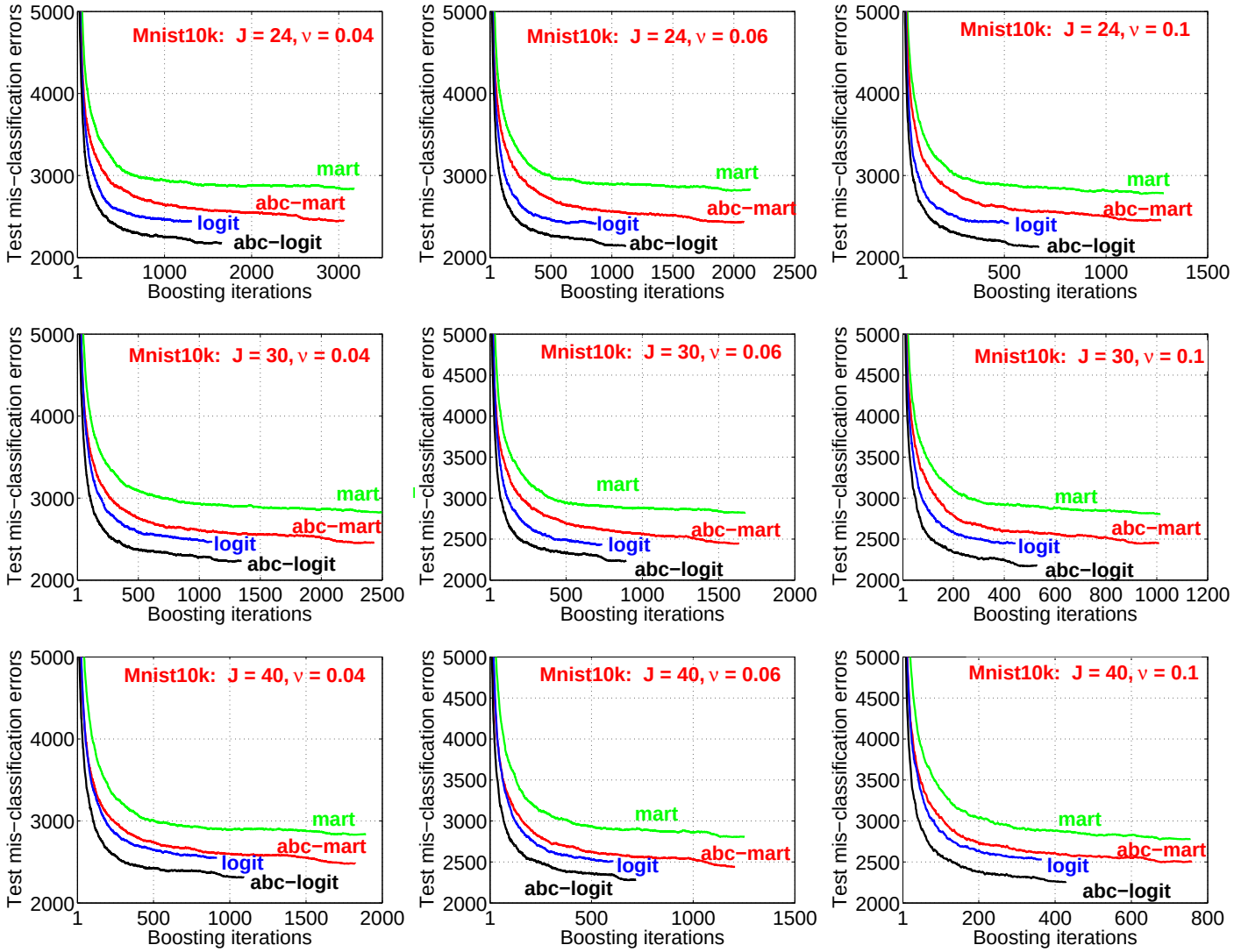
	P3			
	$\nu = 0.04$	$\nu = 0.06$	$\nu = 0.08$	$\nu = 0.1$
$J = 4$	3×10^{-9}	5×10^{-9}	4×10^{-6}	7×10^{-6}
$J = 6$	4×10^{-13}	2×10^{-8}	2×10^{-10}	3×10^{-8}
$J = 8$	2×10^{-9}	3×10^{-10}	3×10^{-10}	6×10^{-11}
$J = 10$	1×10^{-10}	8×10^{-10}	6×10^{-11}	4×10^{-10}
$J = 12$	2×10^{-10}	2×10^{-8}	1×10^{-9}	1×10^{-9}
$J = 14$	5×10^{-10}	6×10^{-9}	4×10^{-10}	4×10^{-10}
$J = 16$	2×10^{-8}	2×10^{-7}	1×10^{-8}	1×10^{-8}
$J = 18$	4×10^{-9}	8×10^{-9}	6×10^{-8}	3×10^{-8}
$J = 20$	1×10^{-6}	2×10^{-7}	6×10^{-8}	2×10^{-7}
$J = 24$	2×10^{-5}	9×10^{-6}	3×10^{-6}	9×10^{-7}
$J = 30$	5×10^{-4}	0.0011	1×10^{-4}	2×10^{-5}
$J = 40$	0.0056	0.0103	0.0024	1×10^{-4}
$J = 50$	0.0145	0.0707	0.0218	0.0102

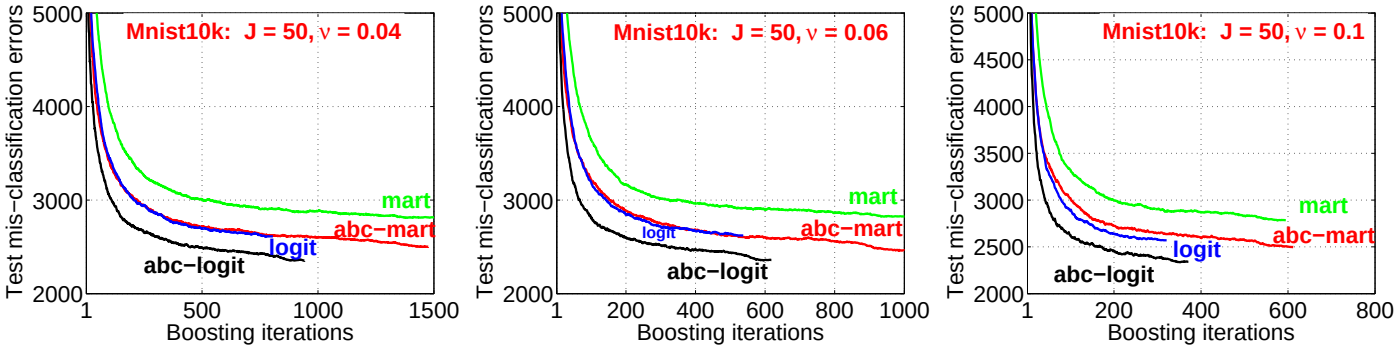
P4				
	$\nu = 0.04$	$\nu = 0.06$	$\nu = 0.08$	$\nu = 0.1$
$J = 4$	1×10^{-5}	2×10^{-7}	4×10^{-10}	5×10^{-12}
$J = 6$	5×10^{-11}	7×10^{-11}	1×10^{-12}	6×10^{-13}
$J = 8$	4×10^{-11}	5×10^{-13}	2×10^{-12}	8×10^{-12}
$J = 10$	6×10^{-11}	5×10^{-10}	8×10^{-11}	7×10^{-10}
$J = 12$	2×10^{-9}	6×10^{-9}	6×10^{-9}	1×10^{-8}
$J = 14$	1×10^{-8}	4×10^{-7}	1×10^{-8}	9×10^{-9}
$J = 16$	1×10^{-6}	5×10^{-7}	3×10^{-6}	9×10^{-7}
$J = 18$	1×10^{-6}	8×10^{-7}	2×10^{-6}	8×10^{-6}
$J = 20$	4×10^{-5}	2×10^{-6}	8×10^{-7}	1×10^{-5}
$J = 24$	3×10^{-5}	3×10^{-5}	7×10^{-6}	1×10^{-5}
$J = 30$	3×10^{-4}	0.0016	0.0012	2×10^{-5}
$J = 40$	2×10^{-4}	5×10^{-4}	6×10^{-5}	3×10^{-5}
$J = 50$	9×10^{-5}	7×10^{-5}	2×10^{-4}	4×10^{-4}


Mnist10k: J = 4, ν = 0.04
Mnist10k: J = 4, ν = 0.06
Mnist10k: J = 4, ν = 0.1
Mnist10k: J = 6, ν = 0.04
Mnist10k: J = 6, ν = 0.06
Mnist10k: J = 6, ν = 0.1
Mnist10k: J = 8, ν = 0.04
Mnist10k: J = 8, ν = 0.06
Mnist10k: J = 8, ν = 0.1
Test mis−classification errors
Boosting iterations
mart
abc−mart
logit
abc−logit


Mnist10k: J = 10, ν = 0.04
Mnist10k: J = 10, ν = 0.06
Mnist10k: J = 10, ν = 0.1
Mnist10k: J = 12, ν = 0.04
Mnist10k: J = 12, ν = 0.06
Mnist10k: J = 12, ν = 0.1
Mnist10k: J = 14, ν = 0.04
Mnist10k: J = 14, ν = 0.06
Mnist10k: J = 14, ν = 0.1
Test mis−classification errors
Boosting iterations
mart
abc−mart
logit
abc−logit


Mnist10k: J = 16, ν = 0.04
Mnist10k: J = 16, ν = 0.06
Mnist10k: J = 16, ν = 0.1
Mnist10k: J = 18, ν = 0.04
Mnist10k: J = 18, ν = 0.06
Mnist10k: J = 18, ν = 0.1
Mnist10k: J = 20, ν = 0.04
Mnist10k: J = 20, ν = 0.06
Mnist10k: J = 20, ν = 0.1
Test mis−classification errors
Boosting iterations
mart
abc−mart
logit
abc−logit


Mnist10k: J = 24, ν = 0.04
Mnist10k: J = 24, ν = 0.06
Mnist10k: J = 24, ν = 0.1
Mnist10k: J = 30, ν = 0.04
Mnist10k: J = 30, ν = 0.06
Mnist10k: J = 30, ν = 0.1
Mnist10k: J = 40, ν = 0.04
Mnist10k: J = 40, ν = 0.06
Mnist10k: J = 40, ν = 0.1
Test mis−classification errors
Boosting iterations
mart
abc−mart
logit
abc−logit



Want to See More?