Multi-class Boosting with Class Hierarchies

Goo Jun and Joydeep Ghosh

Department of Electrical and Computer Engineering
The University of Texas at Austin, Austin TX-78712, USA
{gjun, ghosh}@ece.utexas.edu

Abstract. We propose AdaBoost.BHC, a novel multi-class boosting algorithm. AdaBoost.BHC solves a C class problem by using $C-1$ binary classifiers defined by a hierarchy that is learnt on the classes based on their closeness to one another. It then applies AdaBoost to each binary classifier. The proposed algorithm is empirically evaluated with other multi-class AdaBoost algorithms using a variety of datasets. The results show that AdaBoost.BHC is consistently among the top performers, thereby providing a very reliable platform. In particular, it requires significantly less computation than AdaBoost.MH, while exhibiting better or comparable generalization power.

1 Introduction

Adaptive reweighting and combining methods such as boosting have become very popular because of their remarkable ability to reduce both model bias and variance as compared to a base learner. In particular, AdaBoost [1] has been successfully applied to vast range of machine learning applications. AdaBoost is an ensemble learning method for binary classification problems based on a set of weak learners trained under different distributions. There is one baseline requirement for the boosting procedure to work: the weak learner should be at least 50% accurate.

AdaBoost.M1 [2] is a direct extension of the binary AdaBoost algorithm with a multi-class weak learner. The problem of AdaBoost.M1 is that the multi-class weak learner needs to be much stronger, since a random guess would yield only $1/C$ accuracy for a C class problem. This observation has prompted a plethora of approaches that convert a multi-class problem into multiple binary classification problems thus omitting the necessity of directly employing a multi-class classifier. These algorithms either change a single multi-class problem into multiple binary class problems [3], into one big binary class problem with increased number of examples[4], or into a sequence of binary class problems with output codes [5][6].

We take a different approach by employing a binary hierarchical classifier (BHC), which converts a multi-class problem into a set of binary problems based on affinity between classes. In BHC, similar classes are clustered together into

meta-classes so that the resulting binary problems are simpler to solve than in other algorithms mentioned above. In this paper, AdaBoost.BHC, a novel method that combines binary AdaBoost with BHC, is proposed and applied to multi-class classification problems, and the results are compared to the results from existing multi-class AdaBoost algorithms. The results show that the performance of AdaBoost.BHC is always among the best, and it runs much faster than AdaBoost.MI and AdaBoost.MH.

2 Dealing with Multi-class Problems

In binary classification, a classifier maps the input space onto a binary output space, $\{+1, -1\}$. In many cases, however, we have to deal with $C > 2$ classes, so the output space is defined as $\{1, 2, ..., C\}$. There are classification algorithms that can directly handle multiple classes, such as decision trees or multi-layer perceptrons. Producing a non-binary output, however, is not possible or less natural for other approaches such as SVMs. In such a case, we can model a multi-class problem using a fusion of binary classifiers.

One way to employ binary classifiers for a multi-class problem is using binary codes to decompose the problem's output space. One-versus-all method and "all-pairs" method are examples of solving multi-class problem by decomposing output spaces. The error-correcting output code (ECOC) [7] is another example of a binary code approach combined with robust error-correcting coding. All of these algorithms can be considered as two-stage approaches in the sense that multiple binary classifiers are trained independently and combined to produce the final class label at the second level. These methods have another common aspect that the binary classification problems are specified without considering similarity between classes. Therefore, the resulting binary problems can become more problematic, e.g. highly unbalanced in one-vs-all approaches, and leading to complicated, multimodal decision boundaries in ECOCs.

An alternative approach is the binary hierarchical classifier (BHC) [8] that was developed for hyperspectral remote-sensing applications where classes (land cover types) have certain natural groupings, i.e. some classes are more similar to one another than to others. BHC recursively decomposes a multi-class problem into $C - 1$ binary (meta-)class problems, arranged as a binary hierarchical tree. In a BHC tree, similar classes are grouped together to form meta-classes at each inner level of the tree. Since the resulting structure of the BHC tree is determined by affinity between classes, the hierarchy often provides useful insights on the problem domain. More importantly, the resulting binary classification problems tend to be simpler, thereby facilitating both feature extraction/selection and classifier design, and making it easier to satisfy the baseline requirement for boosting weak learners.

At the root node of a BHC tree, the given set of classes is first partitioned into two disjoint sets or meta-classes. The meta-classes are recursively partitioned until the leaves of the decomposition tree are reached where each leaf corresponds to only one of the original classes. Consequently, the number of leaf nodes in

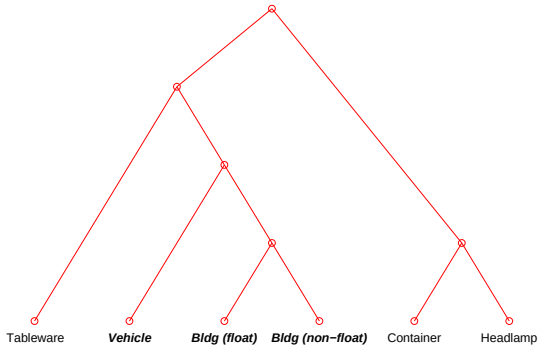

Fig. 1. BHC tree for Glass dataset. Class names in bold italic font are window classes, and others are non-window classes.

Algorithm 1 Outline of PARTITION NODE algorithm

1. Initialize associations as $a_1 = 1$, and $a_i = 0.5, \forall i > 1$. $a_i = P(T_l|c_i) = 1 - P(T_r|c_i)$.
2. Find an optimal projection by Fisher's discriminant analysis with soft assignments.
3. Calculate the mean log-likelihood of T_r and T_l in the Fisher-projected space.
4. Update a_i's, using the mean log-likelihoods and a pre-defined step size T.
5. Repeat Steps 2 through 4 until increase in Fisher's discriminant is insignificant.
6. Compute the entropy: $\mathcal{H} = -\frac{1}{M}\sum_i [a_i \log_2 a_i + (1 - a_i)\log_2(1 - a_i)]$.
7. Stop if $\mathcal{H} < \theta_H$. Otherwise, decrease T and repeat Steps 2 through 6.

the BHC tree equals to the number of classes. An outline of the partitioning algorithm is given in Algorithm 1. As a result of the partitioning process, classes that are similar in the input feature space tend to get lumped into the same meta-class in the tree. In the Glass data, for example, all glass types can be divided into two groups: window and non-window. Fig. 1 shows a BHC tree for the Glass data, and we can observe that all classes that belong to the window meta-class are located under the same sub-tree. An empirical study [9] has shown that BHC offers comparable classification accuracies with that of the ECOC with fewer number of classifiers.

Recently, another tree-based approach, margin tree, was proposed [10]. The margin tree algorithm employs the margin between classes as a distance measure for the hierarchical agglomerative clustering of classes. Both BHC and margin tree produce classification trees, but differ in how this tree is built. In the margin tree algorithm, it is assumed that the dimensionality of data is higher than the number of samples, so that the classes are always linearly separable. On the other hand, in BHC we need at least as many samples as number of dimensions, which enables the Fisher's discriminant analysis. In this paper, we chose BHC since all datasets in our experiments have more number of samples than number of features, but our framework is easily applicable to margin tree or other tree-based multi-class decomposition algorithms as well.

3 Multi-class AdaBoost

There are many variations of AdaBoost for multi-class problems. AdaBoost.M1 [2] is a direct extension of the binary AdaBoost algorithm. In AdaBoost.M1, the weak-learner should be able to produce multi-class output. The problem of AdaBoost.M1 is that even weak learning may not be easily obtainable for challenging multi-class problems. Since we are giving more weight to the samples on which our hypotheses fail, the distribution often gets harder to learn for weak learners as we keep boosting, making it much more difficult to produce at least 50% accuracy for multi-class classifiers.

AdaBoost.MH [4] transforms a multi-class problem into a binary classification problem by replicating examples with attached class labels, based on the Hamming loss. AdaBoost.MH can handle both multi-class and multi-label problems. AdaBoost.MH is the most popular multi-class version of AdaBoost in practice [11], and it shows good generalization ability even for relatively hard multi-class problems. One of the problems with AdaBoost.MH is that it converts a multi-class problem into one huge binary problem that requires $N \cdot C$ examples. An alternative approach is AdaBoost.MI [3], a direct application of the one-vs-all method. In AdaBoost.MI, we have C independent weak learners for a C-class problem, and each binary weak learner is dedicated to one class. Each weak learner is trained separately, with independently managed distributions. AdaBoost.MI has similar computational complexity as AdaBoost.MH, but requires a smaller memory footprint because the algorithm can be easily modularized.

Another approach, AdaBoost.OC [5] combines the output code algorithm with AdaBoost. It requires only one binary classifier with N examples, which makes it much faster than other algorithms. AdaBoost.ECC [6] is based on AdaBoost.OC, and it has been shown that AdaBoost.OC is actually a shrinkage version of AdaBoost.ECC [12]. In AdaBoost.ECC, a coloring $\mu_t : Y \rightarrow \{-1, +1\}$ is computed at t-th round of boosting, mapping the output space onto a binary space. A new coloring is obtained for each round, hence we have a sequence of colorings $(\mu_1, \mu_2, ..., \mu_T)$ from T rounds of boosting, which makes each class label correspond to a unique codeword, *e.g.*, (+1, +1, -1...). The codeword from each class labels is multiplied by the outputs of hypotheses and summed up, and the class label which maximizes the value is selected as the final output.

Recently, a different approach that employs a multi-class weak learner was also proposed, where the minimum accuracy requirement for the multi-class weak learner is $1/C$ rather than $1/2$ [11], which is not included in our experiments since we focus on algorithms that work with binary weak learners. We compare performances of MH, MI, and ECC algorithms with that of the proposed AdaBoost.BHC.

4 AdaBoost.BHC

In the AdaBoost.BHC algorithm, a standard binary AdaBoost algorithm is applied to each internal node of the BHC tree, with separately updated distributions. The final hypothesis of each node, H_k, is the weighted sum of hypotheses,

Algorithm 2 AdaBoost.BHC

Given $(x_1, y_1), ..., (x_N, y_N)$ and a BHC tree T, where $x_i \in X,\ y_i \in Y = \{1, 2, ..., C\}$.

- T_1 is the root node, and $T_k.C \subset Y$ is a set of classes belong to T_k.
- $T_k.L(= l)$ and $T_k.R(= r)$ are indices of left and right child of T_k.
- If T_k is a leaf node, $l = r = 0$ and $|T_k.C| = 1$.

1. For each internal node T_k,
   (a) Construct $(X_k, Y_k) = \{(x_i, y_i) | y_i \in T_k.C\}$,
   (b) Map $(X_k, Y_k) \rightarrow (X_k, Z_k)$, $z_i \in Z_k = \{+1, -1\}$.

$$z_i(x_i) = \begin{cases} +1 & \text{if } \ y_i \in T_l.C \\ -1 & \text{if } \ y_i \in T_r.C \end{cases}$$

   (c) Run AdaBoost on with (X_k, Z_k) to obtain H_k.
2. Get $H_{final}(x_i)$, starting from $k = 1$
   (a) If $|T_k.C| = 1$, output $H_{final}(x_i) = y \in T_k.C$ and finish.
   (b) If $H_k(x_i) = +1$, $k = T_k.L$. Otherwise $k = T_k.R$. Return to step 2-(a).

and the final multi-class output, $H_{final}(x_i)$ is determined from binary labels generated at each node, $H_k(x_i)$ of the BHC tree. A detailed description of the AdaBoost.BHC algorithm is given in Algorithm 2.

One of the main differences of AdaBoost.BHC from other multi-class AdaBoost algorithms is that the binary decomposition of AdaBoost.BHC is determined from the distribution of the input data. As a result, AdaBoost.BHC produces more separable binary classification problems than other approaches, hence supporting good generalization behavior. Another notable advantage of AdaBoost.BHC is that it requires less number of computations per round than AdaBoost.MH or AdaBoost.MI. The weak learner at the root node is trained with N examples, and the left and the right child nodes of the root are trained with $N/2$ examples on average. Assuming a full balanced binary tree, computational complexity of AdaBoost.BHC algorithm is:

$$\sum_{k=0}^{log_2 C - 1} \frac{1}{2^k} f\left(\frac{N}{2^k}\right) ,$$

where $f(\cdot)$ is the complexity of the weak learning algorithm. Table 1 shows computational requirements of MH, MI, ECC, and BHC algorithms for $O(N)$ and $O(N^2)$ weak learning algorithms.

5 Experiments and Results

Empirical comparisons of existing multi-class AdaBoost algorithms and AdaBoost.BHC are made using datasets from the UCI machine learning repository [13] as in Table 2. Seven different multi-class datasets from the repository are

	AdaBoost.MH	AdaBoost.MI	AdaBoost.ECC	AdaBoost.BHC
$O(N)$ weak learner	$C \cdot N$	$C \cdot N$	N	$\log_2 C \cdot N$
$O(N^2)$ weak learner	$C^2 \cdot N^2$	$C \cdot N^2$	N^2	$2(1 - \frac{1}{C})N^2$

Table 1. Time complexity comparison (per round) between different multi-class AdaBoost algorithms for N examples from C classes.

Name	# Train	# Test	# Attributes	# Classes
Iris	150	4-CV	4	3
Glass	214	4-CV	10	6
Yeast	1484	4-CV	8	10
Page blocks	5473	4-CV	10	5
Landsat	4435	2000	36	6
Optical digits	3823	1797	64	10
Pen-based digits	7494	3498	16	10

Table 2. Datasets used in experiments from UCI repository

employed. 4-fold cross validation (CV) is done 10 times for Landsat, Optical digits, and Pen-based digits datasets. For the other four datasets with pre-specified test sets, each test is repeated 40 times and the results are averaged. 100 rounds of boosting are done for each algorithm. MATLAB's built-in classification and regression tree (CART) is used as the base learner. Four different algorithms are tested: AdaBoost.MH, AdaBoost.MI, AdaBoost.ECC, and AdaBoost.BHC.

Figs. 2 to 8 show training error and test error of each algorithm for seven datasets. All algorithms except AdaBoost.MI generally show good performance on all data. AdaBoost.BHC and AdaBoost.MH display the best generalization behaviors for most datasets. Note that AdaBoost.BHC achieves low error rates much faster than AdaBoost.MH in most experiments. AdaBoost.MI shows problems with Yeast and Page blocks datasets as shown in Figs. 4 and 5, where both training error and test error increase after some rounds. One probable reason is the fact that the class distributions of Yeast and Page blocks datasets are highly unbalanced. The smallest class of the Yeast data has only 5 examples from a total of 1484, and the largest class of the Page blocks dataset has 4913 examples from a total of 5473. Because AdaBoost.MI makes C different binary classification problems, the highly skewed prior distribution makes the problem much more difficult for weak learners. AdaBoost.MH is less affected by unbalanced distributions, because it converts the hypothesis space from $h : X \rightarrow Y$ to $h : (X, Y) \rightarrow (correct, incorrect)$ domain, hence it always yields the same fraction of positive and negative examples. AdaBoost.ECC and Adaboost.BHC also perform better than AdaBoost.MI under skewed distributions because they aggregate several classes together based on the coding scheme or the affinity between classes. There are more systematic approaches to evaluate performances

Algorithm	Yeast	Page blocks	Landsat	Optical	Pen digits
AdaBoost.MH	32.0	156.9	311.8	698.9	866.8
AdaBoost.MI	43.5	128.1	331.0	731.4	688.4
AdaBoost.ECC	5.5	15.8	42.5	60.0	33.9
AdaBoost.BHC	24.8	49.3	167.1	198.7	138.1

Table 3. Average training time(seconds) for 100 rounds

of classifiers with respect to the complexity of the problem [14], which is left as a future work.

AdaBoost.ECC generally shows relatively higher generalization error except for the Yeast dataset. One possible reason for this is the sub-optimal output space partitioning of the AdaBoost.ECC algorithm. AdaBoost.ECC require mappings from Y to $\{+1, -1\}$ for each round, and a recent study [15] has shown that appropriate partitioning of the output space is important for the algorithm's performance. Here we employed a random partitioning, as suggested in [5]. It is also shown that random partitioning is generally better than the optimized max-cut algorithm, but it still has room for improvement [15]. Our empirical results suggests that Adaboost.BHC provides better output decomposition than AdaBoost.ECC, because it decomposes the output space based on the class-conditional distributions, producing simpler decision boundaries for binary weak learners.

Table 3 shows average training time of different algorithms for large ($N \geq 1000$) datasets. AdaBoost.ECC is clearly the fastest algorithm. AdaBoost.BHC is slower than AdaBoost.ECC, but is significantly faster than AdaBoost.MI and AdaBoost.MH.

6 Conclusions

In this paper, the AdaBoost.BHC algorithm incorporating multiple binary classifiers, a class hierarchy, and boosting was proposed and tested. AdaBoost.BHC is always among the best performers if not the very best, thus providing a more reliable solution. Moreover it is faster than all algorithms other than AdaBoost.ECC. AdaBoost.ECC however typically does not generalize as well as AdaBoost.BHC.

Acknowledgements: This research was supported by NSF Grant IIS-0705815.

References

1. Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *European Conference on Computational Learning Theory*, pp. 23–37, 1995.
2. Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting,," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

3. S. Abney, R. Schapire, and Y. Singer, "Boosting applied to tagging and pp attachment," 1999.
4. R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, no. 3, pp. 297–336, 1999.
5. R. E. Schapire, "Using output codes to boost multiclass learning problems," in *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 313–321, 1997.
6. V. Guruswami and A. Sahai, "Multiclass learning, boosting, and error-correcting codes," in *COLT '99: Proceedings of the twelfth annual conference on Computational learning theory*, (New York, NY, USA), pp. 145–155, ACM, 1999.
7. T. G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," *Journal of Artifical Intelligence Research*, vol. 2, p. 263, 1995.
8. S. Kumar, J. Ghosh, and M. M. Crawford, "Hierarchical fusion of multiple classifiers for hyperspectral data analysis," *Pattern Analysis & Applications*, vol. V5, no. 2, pp. 210–220, 2002.
9. S. Rajan and J. Ghosh, "An empirical comparison of hierarchical vs. two-level approaches to multiclass problems," *Multiple Classifier Systems*, pp. 283–292, 2004.
10. R. Tibshirani and T. Hastie, "Margin trees for high-dimensional classification," *J. Mach. Learn. Res.*, vol. 8, pp. 637–652, 2007.
11. J. Zhu, S. Rosset, H. Zou, and T. Hastie, "Multi-class adaboost," tech. rep., Department of Statistics, University of Michigan, Ann Arbor, MI 48109, 2006.
12. Y. Sun, S. Todorovic, and J. Li, "Unifying multi-class adaboost algorithms with binary base learners under the margin framework," *Pattern Recogn. Lett.*, vol. 28, no. 5, pp. 631–643, 2007.
13. A. Asuncion and D. J. Newman, "UCI machine learning repository http://www.ics.uci.edu/∼mlearn/MLRepository.html," 2007.
14. T. K. Ho and M. Basu, "Complexity measures of supervised classification problems," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, pp. 289–300, Mar 2002.
15. L. Li, "Multiclass boosting with repartitioning," in *ICML '06: Proceedings of the 23rd international conference on Machine learning*, (New York, NY, USA), pp. 569–576, ACM, 2006.

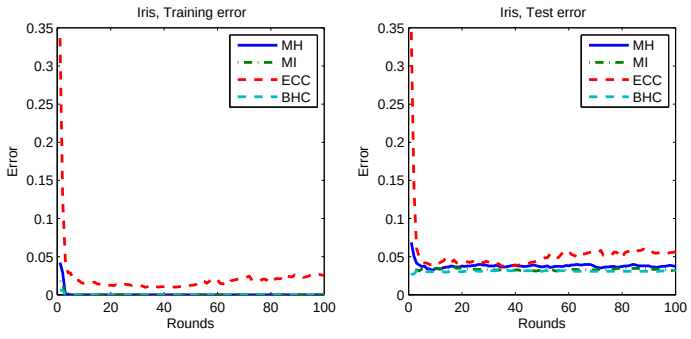

Fig. 2. Training and test errors for Iris data

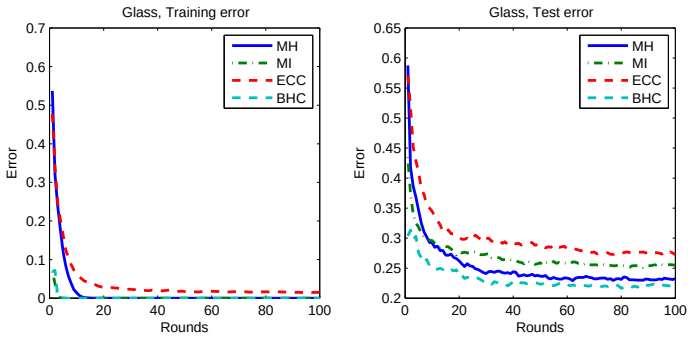

Fig. 3. Training and test errors for Glass data

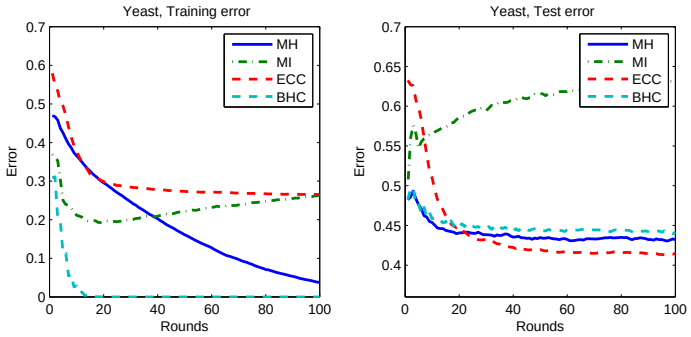

Fig. 4. Training and test errors for Yeast data

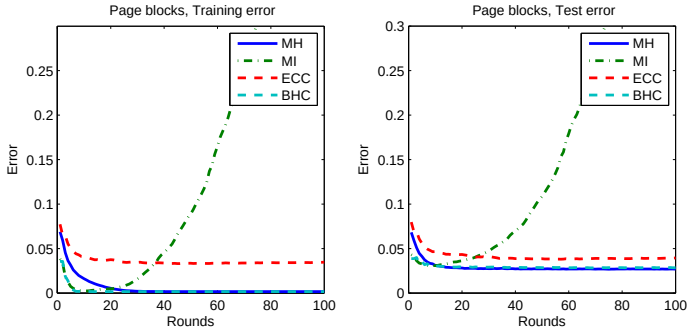

Fig. 5. Training and test errors for Page blocks data

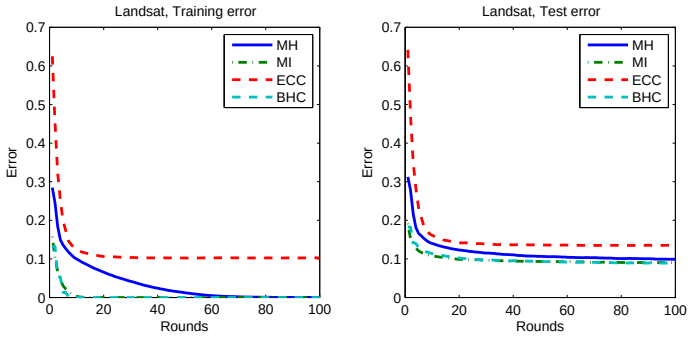

Fig. 6. Training and test errors for Landsat data

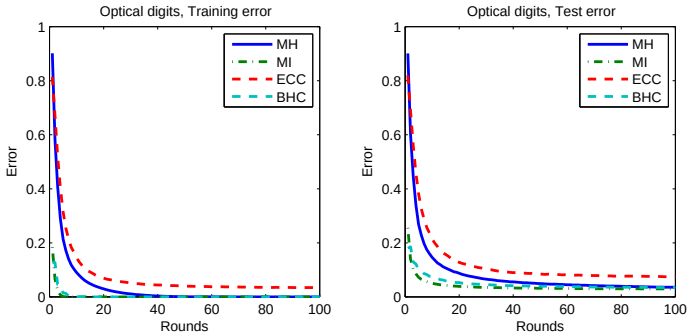

Fig. 7. Training and test errors for Optical digits data

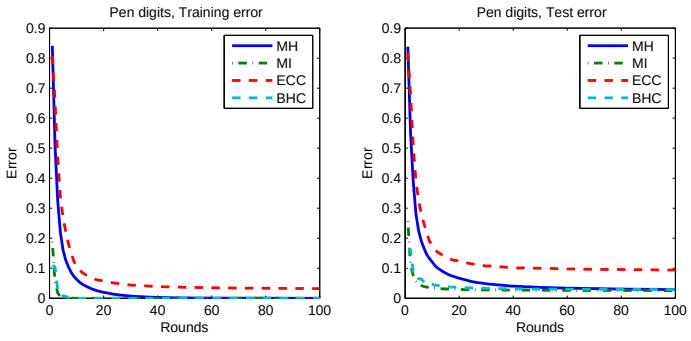

Fig. 8. Training and test errors for Pen digits data