TR-IIS-07-014

Boosting Multiclass Learning with Repeating Codes

Yu-Shi Lin and Chun-Nan Hsu

Oct. 17, 2007 || Technical Report No. TR-IIS-07-014
http://www.iis.sinica.edu.tw/page/library/LIB/TechReport/tr2007/tr07.html

Boosting Multiclass Learning with Repeating Codes

Yu-Shi Lin
Institute of Information Science
Academia Sinica
Taipei, Taiwan
bathroom@iis.sinica.edu.tw

Chun-Nan Hsu
Institute of Information Science
Academia Sinica
Taipei, Taiwan
chunnan@iis.sinica.edu.tw

Abstract

A long-standing goal of machine learning is to build a system which can detect a large number of classes with accuracy and efficiency. Some relationships between classes would become a scale-free network in which we can classify the assigned class very fast. Many available methods for multiclass problems have been proposed in the literatures, such as AdaBoost.ECC [4], AdaBoost.ERP, [7] and JointBoost [12]. However, many of them are inaccurate or time-consuming on training. In this paper, we propose a new algorithm, called AdaBoost.ERC, which combines the approach of Dietterich and Bakiri [2] based on error correcting output codes (ECOC) and Shapire's boosting algorithm [3] [10]. With advantages of both concepts, our new approach achieves better performance compared to AdaBoost.ECC, AdaBoost.ERP, and JointBoost.

Keywords: multiclass learning, scale-free network, ECOC, AdaBoost

1. Introduction

In our previous research and experiences in machine learning, we found that it is difficult to learn a large number of classes with accuracy and efficiency. For example, in a recommend system, the underlying concept would detect potential customers for cold sellers [6]; however, a large proportion of all products are cold sellers. Other examples such as applications include large vocabulary speech categorization [8], natural language processing tasks [5], and object recognition in cluttered scenes [12] still have the same problem. Obviously, traditional methods require to train a classifier for each class independently. When classes are increasing, we need applying a large number of different classifiers to those classes. As a result, the computational complexity and the sample complexity scales linearly with the number of classes to be detected.

In many practical applications, there are some relationships among classes and these relationships would become a scale-free network, where a small number of nodes have a very high degree of connections and most of the other nodes have a very small degree. In other words, some nodes are very important while a large majority of nodes are less important. It turns out that the degrees of nodes have a power-law distribution. Therefore, suppose there is an application where we have a large number of classes to classify, and they form a scale-free network. Then potentially we can learn to classify those important classes first.

In this paper, we want to take advantage of the scale-free network, we do not need to train a classifier independently. We propose sharing classifiers among different categories for constructing a scale-free network of classifiers. The key idea is to explore the property of short characteristic path length of a scale-free network. Therefore, a small number of related classifiers can be associated together to accurately classify a piece of data. Another important property of scale-free networks is robustness. It means that even if some nodes have breakdowns, the network still works well. That is, if our classifiers constitute a scale-free network, even if some of the classifiers fail, the network will still be robust and quickly find a route to complete the classification.

If we can reuse the learned classifiers, then the resources (i.e., time, training examples, etc.) require to learning a classifier for the majority of other classes will be decreased significantly. In this way, we can scale up the learning algorithm to a large number of classes. This is also more similar to human learning curves: initially, a baby takes a long time to learn to say a few words. But later, the baby can learn a lot of words very quick. Currently, no machine learning can achieve that kind of feat.

2. Related Work

Given training data $D = \{(x_i, y_i)\}_{i=1}^{n} \in \mathcal{X} \times \mathcal{Y}$ where $\mathcal{X}$ is the input space and $\mathcal{Y} = \{1, ..., K\}$ is the label space. Multiclass learning problems involve finding an unknown function H such that the value of $E_{(x,y)\in D}[H(x) \neq y]$ is minimun. The complexity of multiclass learning problems is increasing with the number of clsses. Because of good performance of binary classifiers, most muticlass learning tasks are solved by reducing to multiple binary classification problems [1]. There are many available algorithms for learning binary classification problems. The AdaBoost algorithm introduced in 1995 by Freund and Shapire is the most popular one of them [3]. AdaBoost works by repeatedly reweighting examples and rerunning the weak learner whose accuracy is only better than random guessing on these reweighted examples. Finally, the combined ensemble is a weighted vote of the weak classifiers. Because AdaBoost focuses on those hardest examples, the error rate of AdaBoost can be achieved arbitrarily small.

For dealing with multiclass problems, we investigate the approach of Dietterich and Bakiri [2] based on error correcting output codes (ECOC), which is designed to handle multiclass problems using only a binary learning algorithm. Given an error correcting output coding matrix $M \in \{-1, +1\}^{K\times T}$, where T is the length of a codeword and K is the number of classes. Here, $M(k)$ denotes the $k - th$ row of the matrix M, that is, a codeword for class k. Each column of M defines a binary partition of K classes over data, then a binary classifier is trained. After T training steps, it produces T classifiers $h_1, ..., h_T$. A given new example is then classified by choosing the class whose associated codeword is the closest in Hamming distance to the sequence of the predictions generated by $h_1, ..., h_T$.

The most fascinating property of ECOC is that the code can correct some bit errors. It means that even some bit positions of codeword predict incorrectly, the nearest codeword will still be the correct codeword. A review of the literature indicate that combining AdaBoost algorithm with the method of ECOC is faster and better in experiment and in theory than other multiclass algorithms. Those variants of ECOC include AdaBoost.ECC, AdaBoost.ERP, and JointBoost. Their algorithms are depicted in Figure 1, 2, and 3. (Note: In these figures, $I(\phi)$ is an indicator function defined to be 1 if proposition ϕ holds and 0 otherwise.)

3. Proposed Approach

The two primary properties of scale-free network: (1) average short path length between any two nodes; (2) robustness of network. In our research, we want to construct classifiers to achieve those properties of scale-free network. We think using ECOC may be a good start. The error correcting property of the code is like the robustness of scale-free network, that is, even the system make a little breakdown it will still work well. Furthermore, binary partition of each column of code can be regard as sharing knowledge among different classes. Then if we want to classify some assigned class, we may pick partial columns of codes which are associated about assigned class to classify. That is like the average short path length of scale-free network. Therefore, using ECOC is our proposal approach. For making ECOC to achieve better performance, our proposal algorithm also combine with boosting. Our new method differs from the above mentioned algorithms.

More specifically, given an input x, the ensemble output $H(x) = (h_1(x), ..., h_T(x))$ is computed, and the Hamming distance is used to predict the class of x. The Hamming distance between $H(x)$ and $k - th$ row $M(k)$ is defined as

$$\triangle(M(k), H(x)) = \sum_{t=1}^{T}(1 - M(k,t)h(x))/2. \quad (1)$$

Then the predicted class $\hat{y}$ is

$$\hat{y} = \arg\min \triangle(M(k), H(x)). \quad (2)$$

If we want to classify an example (x, y) correctly, $\triangle(M(y), H(x))$ must be smaller than $\triangle(M(k), H(x))$ for any $k \neq y$. Like AdaBoost.ECC, we adopt their definition *margin* of the example (x, y) for class k. That is,

$$\rho_k(x, y) = \triangle(M(k), H(x)) - \triangle(M(y), H(x)). \quad (3)$$

[11]. Intuitionally, we want to design a learning algorithm which can let the margins of the training examples be as large as possible. As AdaBoost.ECC and AdaBoost.ERP, they optimize an exponential objective function based on the margins

$$C_M(H) = \sum_{n=1}^{N}\sum_{k\neq y_n} e^{-\rho_k(x_n,y_n)}. \quad (4)$$

Given a matrix M, we want to find a optimal classifier H such that objective function $C_M(H)$ minimum. Using the negative gradient method to minimize the objective function. The negative gradient can be reduced to

$$U_t \sum_{n=1}^{N} D_t(n)M(y_n, t)h_t(x_n) = U_t(1 - 2\epsilon_t), \quad (5)$$

```
**Input**: A training set $\{(x_n, y_n)\}_{n=1}^{N}$;length of codeword $T$
**Initialize**: $\tilde{D}_1(n,k) = I(k \neq y_n)/N(K-1)$
**For**: $t = 1, 2, ..., T$
 · Choose the $t-th$ column $M(\cdot,t) \in \{-1,+1\}^K$
 · $U_t = \sum_{n=1}^{N} \sum_{k=1}^{K} \tilde{D}_t(n,k) I(M(k,t) \neq M(y_n,t))$
 · $D_t(n) = U_t^{-1} \sum_{k=1}^{K} \tilde{D}_t(n,k) I(M(k,t) \neq M(y_n,t))$
 · Train a decision stump $h_t$ with distribution $D_t$
 · $\tilde{D}_{t+1}(n,k) = \tilde{Z}_t^{-1} \cdot \tilde{D}_t(n,k) \cdot exp^{-h_t(x_n)(M(y_n,t)-M(k,t))/2}$
   where $\tilde{Z}_t$ is a normalization factor.
**End**
**Output** $H(x) = \arg\max_{k \in Y} \sum_{t=1}^{T} h_t(x) M(k,t)$
```

Figure 1: A description of AdaBoost.ECC

where U_t and $D_t(n)$ are defined in Figure 1 and ϵ_t is the error rate of classifier h_t. Both AdaBoost.ECC and AdaBoost.ERP try to maximize this negative gradient, then along the negative gradient we can get minimization of objective function $C_M(H)$. Let's review negative gradient again. It can be separated into two parts : one is U_t and the other is ϵ_t. The larger U_t is, the stronger the error-correcting ability is. The ϵ_t is the error rate of weak learners trained based on each column. It seems that in order to optimize the objective function, it should both maximize U_t and minimize ϵ_t. Unfortunately, in empirical studying, there is a trade-off between U_t and ϵ_t. However, U_t can be rewritten as

$$U_t = \sum_{k,k' \in 1,\ldots,K} I(M(k,t) \neq M(k',t)) \omega_t(k,k'), \tag{6}$$

where $\omega_t(k,k') = \sum_{n=1}^{N} \tilde{D}_t(n,k) I(k' = y_n)$ [9]. It is obvious to show that maximizing U_t is a special case of the ”Max-Cut” problem, which is known to be NP-complete. Li [7] shows that even using max-cut for getting maximized U_t, the performance is not as good as expected. One probable reason is that the binary classification problems from max-cut are usually much ”harder” for the base learner. Therefore, Li proposed repartition columns of code to make base learner easier to train in every iteration.

In our new method, however, we propose more a straightforward approach to make base learners get small error. Unlike Li's repartition code approach, we just repeat columns of code iteratively.Then we apply AdaBoost algorithm for each repeating column. Therefore, even for some ”harder” columns which are produced by max-cut approach, as long as more repeating columns produced we still could get low error rate of these columns. We call our new algorithm AdaBoost.ERC (AdaBoost with Error-correcting Repeat Code). Its algorithm is depicted in Figure 4.

In ECOC method, the fewer error bits we make, the shorter length of codeword we need. If we can get more accuracy in each position of codeword, the length of codeword is to scale approximately logarithmically with the number of classes. In order to achieve this goal, we apply AdaBoost for each column of code repeatedly until some specified number of repeats or specified error rate. In the next section, we will compare our new method with AdaBoost.ECC, AdaBoost.ERP, and JointBoost. Our experimental results show that AdaBoost.ERC performs significantly better than theirs.

4. Results

We tested our method experimentally on four multi-class benchmark problems available from the machine learning repository at UCI. These data sets have both training data and test data. The characteristics of the benchmarks used are summarized in Table 1. For the weak learner of AdaBoost, we used the decision stump which makes its prediction based on the result of a single test comparing one of the attributes to a threshold value. As mentioned above, maximizing U_t will make base learners hard to learn, so we choose the simplest option to produce each column of code uniformly and independently at random from $\{-1,+1\}$ but ensuring half of column mapping to 1. We apply the same code to AdaBoost.ECC, AdaBoost.ERP, and AdaBoost.ERC except JointBoost. For JointBoost framework, its algorithm picks the subset of classes to share a classifier that maximally reduces the error on the weighted training set for all the classes. Other algorithms in our

```
**Input**: A training set $\{(x_n, y_n)\}_{n=1}^{N}$;length of codeword $T$
**Initialize**: $\tilde{D_1}(n,k) = I(k \neq y_n)/N(K-1)$
**For**: $t = 1, 2, ..., T$
 · Choose the $t-th$ column $M(\cdot, t) \in \{-1, +1\}^K$
 · **repeat** Alternate learning and repartitioning
   · $U_t = \sum_{n=1}^{N} \sum_{k=1}^{K} \tilde{D}_t(n,k) I(M(k,t) \neq M(y_n, t))$
   · $D_t(n) = U_t^{-1} \sum_{k=1}^{K} \tilde{D}_t(n,k) I(M(k,t) \neq M(y_n, t))$
   · Train a decision stump $h_t$ with distribution $D_t$
   · $M(k,t) = sign[\sum_{n:y_n=k} \sum_{l=1}^{K} \tilde{D}_t(n,l) h_t(x_n) - \sum_{n=1}^{N} \tilde{D}_t(n,k) h_t(x_n)]$
 ·**until** convergence or some specified steps
 · $\tilde{D}_{t+1}(n,k) = \tilde{Z_t}^{-1} \cdot \tilde{D}_t(n,k) \cdot exp^{-h_t(x_n)(M(y_n,t)-M(k,t))/2}$
   where $\tilde{Z_t}$ is a normalization factor.
**End**
**Output** $H(x) = \arg\max_{k \in Y} \sum_{t=1}^{T} h_t(x) M(k,t)$
```

Figure 2: A description of AdaBoost.ERP

experiment are all based on the ECOC method so we specify the same length of the codeword for each algorithm. In order to make the results comparable, we set up the number of repartitions of AdaBoost.ERP equal to the number of repeats of AdaBoost.ERC. Figure 5 and Figure 6 show the training error and test error curves with the same length of the codeword for different data sets. It is obvious that AdaBoost.ERC gets the lowest training error and test error on each data set. And its performance continues to improve with the increasing of the number of repeats. We also found that the repartitions of AdaBoost.ERP converge quickly. It means that the repartitions would not change after a few steps, hence its performance is no longer improving. These phenomenons are displayed in Figure 7. (Note: In figures, the digit in the bracket is the number of repartitions/repeats of AdaBoost.ERP/AdaBoost.ERC.)

data set	♯ train	♯ test	♯ attribute	♯ class
segment	2100	210	36	7
optdigits	3823	797	64	10
pendigits	7494	3498	16	10
letter	16000	4000	16	64

Table 1:MultiClass DataSet from UCI

5 Conclusion

We have introduced a new multiclass boosting algorithm, AdaBoost.ERC, with error-correcting output codes and repeating. The repeating is meant to find a better prediction associated each position of the codeword. Our experimental results show that, compared with AdaBoost.ECC, AdaBoost.ERP, and JointBoost achieved the lowest training error and test error on the data set we used. Furthermore, we find out that repartition can improve the performance of base learner, however, after a few steps, the repartition will not change any more. Consequently, the performance of AdaBoost.ERP will not keep improving with the increasing number of repartitions. AdaBoost.ERC can conquer this drawback. In our future work, we hope that we can apply this new approach to constructing a scale-free network of classifiers for learning a large number of classes with accuracy and efficiency.

Acknowledgement

The research reported here is supported in part by the National Science Council, Taiwan under grant No. NSC 95-2221-E-001-038 and No. NSC 95-3112-B-001-017.

References

[1] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. In *Proc. 17th International Conf. on Machine Learning*, pages 9–16. Morgan Kaufmann, San Francisco, CA, 2000.

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

```
**Input**: A training set $\{(x_n, y_n)\}_{n=1}^{N}$
**Initialize**: $\tilde{D_1}(n,k) = 1/NK$
**For**: $t = 1, 2, ..., T$
 · For $m = 1, 2, ..., 2^K - 1$
   · Fit shared stump :
     $h_t(x_n, k) = \{ \begin{array}{ll} aI(x_n^f > \theta) + b & if\ k \in S(n) \\ c^k & if\ k \notin S(n) \end{array}$
   · Evaluate error
     $J_{wse}(m) = \sum_{k=1}^{K} \sum_{n=1}^{N} D_t(n,k)(z_n^k - h_t(x_n, k))^2$
     where $z_n^k = \{ \begin{array}{ll} 1 & if\ y_n = k \\ -1 & if\ y_n \neq k \end{array}$
 · Find best subset: $m^\star = \arg\min_m J_{wse}(m)$
 · Update:
       $H(x,k) = H(x,k) + h_t^{m^\star}(x,k)$
       $D_{t+1}(x_n,k) = Z_t^{-1} \cdot D_t(x_n,k) exp^{-z_n^k h_t^{m^\star}(x_n,k)}$
       where $Z_t$ is a normalization
```

Figure 3: A description of JointBoost

[3] 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*, pages 23–37, 1995.

[4] V. Guruswami and A. Sahai. Multiclass learning, boosting, and error-correcting codes. In *Proc. 12th Annual Conference on Computational Learning Theory*, pages 145–155, 1999.

[5] V. Hoste and W. Daelemans. Comparing bagging and boosting for natural language processing tasks: A typicality approach. In *Proceedings of the Tenth Belgian-Dutch Conference on Machine Learning*, Benelearn 2000, pages 101–108, 2000.

[6] H.-S. Huang, K.-L. Lin, J. Y. jen Hsu, and C.-N. Hsu. Item-triggered recommendation for identifying potential customers of cold sellers in supermarkets. In *A Workshop on the Next Stage of Recommender Systems Research, in conjunction with the 2005 International Conference on Intelligent User Interfaces (IUI 2005)*, 2005.

[7] L. Ling. Multiclass boosting with repartitioning. In *Proc. 23rd International Conference on Machine Learning*, 2006.

[8] R.-Y. Lyu, D.-C. Lyu, M.-S. Liang, M.-H. Wang, Y.-C. Chiang, and C.-N. Hsu. A unified framework for large vocabulary speech recognition of mutually unintelligible chineseregionalects.. In *In Proceedings of the 8th International Conference on Spoken Language Processing (ICSLP2004)*, 2004.

[9] R. E. Schapire. Using output codes to boost multiclass learning problems. In *Proc. 14th International Conference on Machine Learning*, pages 313–321. Morgan Kaufmann, 1997.

[10] R. E. Schapire. A brief introduction to boosting. In *IJCAI*, pages 1401–1406, 1999.

[11] Y. Sun, S. Todorovic, J. Li, and D. Wu. Unifying the error-correcting and output-code adaboost within the margin framework. In *Proceedings of the 22nd international conference on Machine learning*, pages 872–879, 2005.

[12] A. Torralba, K. P. Murphy, and W. T. Freeman. Sharing features: efficient boosting procedures for multiclass object detection, 2004.

```
**Input**: A training set $\{(x_n, y_n)\}_{n=1}^N$;length of codeword $T$
**Initialize**: $\tilde{D}_1(n,k) = I(k \neq y_n)/N(K-1)$
**For**: $t = 1, 2, ..., T$
 · Choose the $t-th$ column $M(\cdot, t) \in \{-1, +1\}^K$
 · $U_t = \sum_{n=1}^N \sum_{k=1}^K \tilde{D}_t(n,k) I(M(k,t) \neq M(y_n,t))$
 · $D_t(n) = U_t^{-1} \sum_{k=1}^K \tilde{D}_t(n,k) I(M(k,t) \neq M(y_n,t))$
 · **repeat** $m = 1, ..., M$
   · Train a decision stump $h_t^m$ with distribution $D_t$
   · $D_t(n) = Z_t^{-1} \cdot D_t(n) exp^{-h_t^m(x_n) M(y_n,t)}$
     where $Z_t$ is a normalization factor
   · $h'_t = h'_t + h_t^m$
 · **until** some specified repeats or specified error rate
 · $\tilde{D}_{t+1}(n,k) = \tilde{Z}_t^{-1} \cdot \tilde{D}_t(n,k) \cdot exp^{-h'_t(x_n)(M(y_n,t) - M(k,t))/2}$
   where $\tilde{Z}_t$ is a normalization factor.
**End**
**Output** $H(x) = \arg\max_{k \in Y} \sum_{t=1}^T h'_t(x) M(k,t)$
```

Figure 4: A description of AdaBoost.ERC

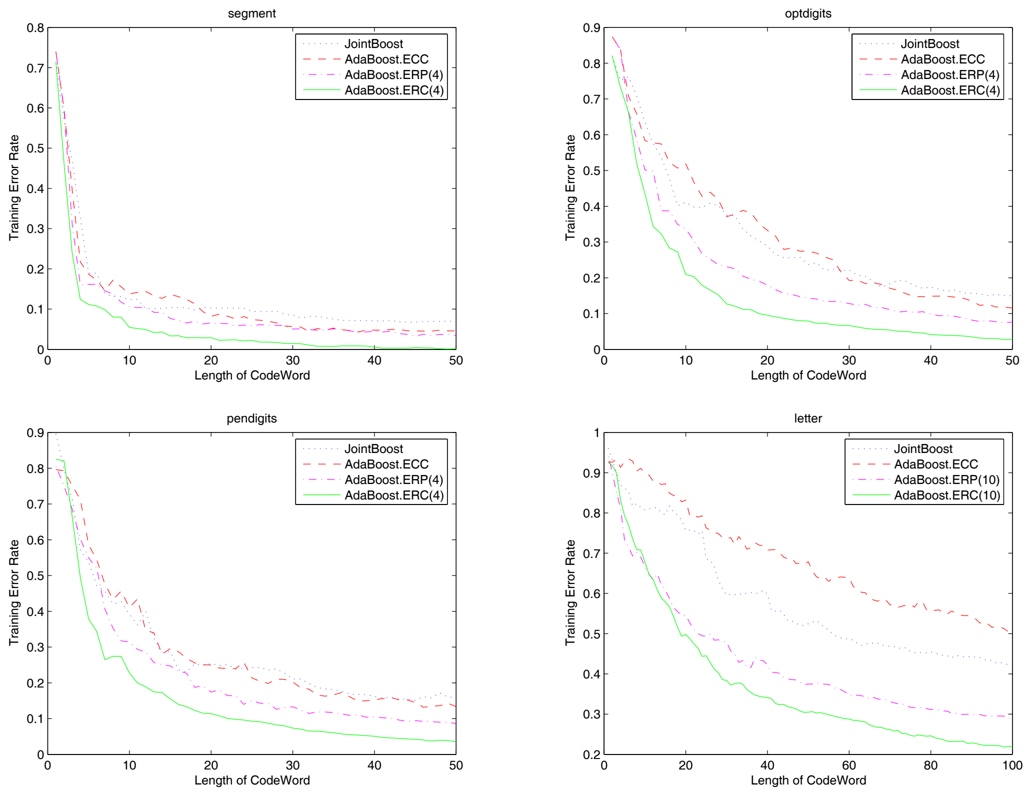

Figure 5: Training Error on UCI DataSet

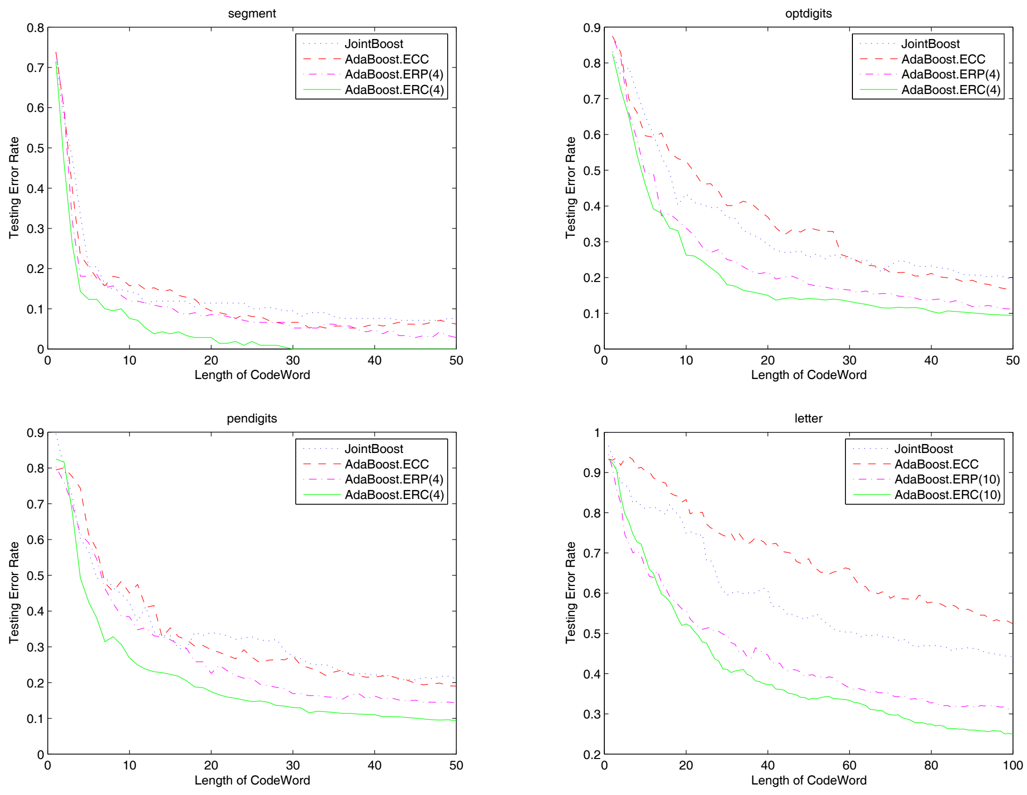

Figure 6: Testing Error on UCI DataSet

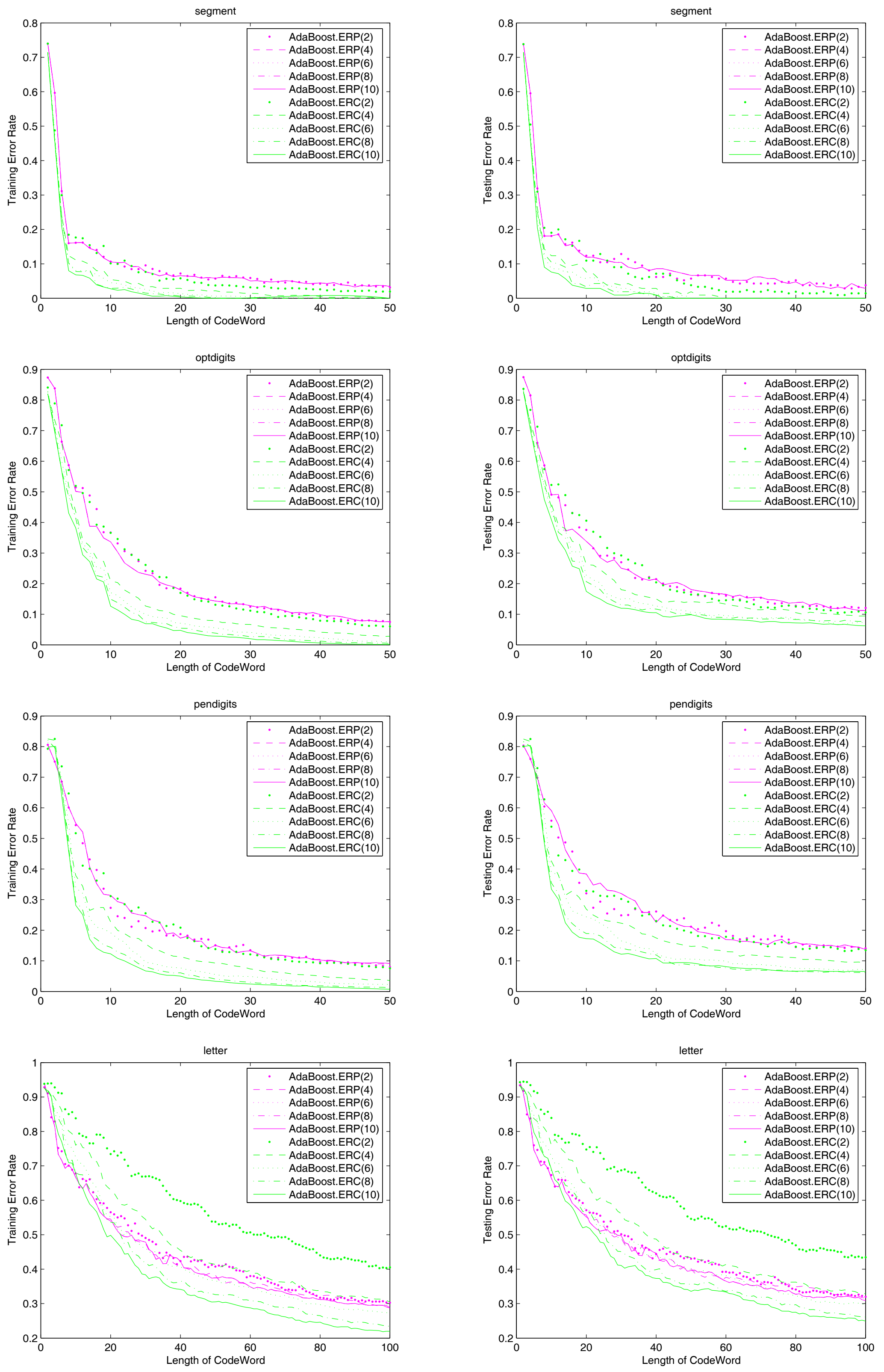

Figure 7: AdaBoost.ERP VS. AdaBoost.ERC