Generalized Multiclass AdaBoost and Its Applications to Multimedia Classification

Wei Hao Jiebo Luo
Kodak Research Labs
Eastman Kodak Company
{wei.hao, jiebo.luo}@kodak.com

Abstract

AdaBoost has received considerable attention in the vision and multimedia research community in recent years. It is originally designed for two-class classification problems. To handle multiple classes, many AdaBoost extensions have been developed primarily based on various schemes for reducing multiclass classification to multiple two-class problems. From a statistical prospective, AdaBoost can be viewed as a forward stepwise additive model using an exponential loss function. In this paper, we derive a generalized form of AdaBoost for multiclass classification based on a multiclass exponential loss function. To prove its effectiveness, we benchmarked a number of multimedia problems of different nature. Experimental results show that the new boosting algorithm outperforms other multiclass alternatives. In addition, the generalized boosting algorithm can be used to either boost a multiclass classifier, or build a multiclass classifier from a binary one.

1. Introduction

Boosting is a general methodology for improving the performance of any given learning algorithm. AdaBoost, introduced by Freund & Schapire [1], is the perhaps the most popular boosting algorithm. It achieves boosting by combining many "weak" classifiers (h_t) to produce a "strong" classifier (H):

$$H(x) = \sum_t \alpha_t h_t(x)$$

However, originally designed for two-class classification, AdaBoost is not directly applicable to multiclass. Various methods have been developed in order to extend the AdaBoost algorithm to multiclass classification [1–5]. Most of them are based on reducing the multiclass classification to multiple binary problems. Friedman, Hastie & Tibshirani [2] have provided a statistical perspective on AdaBoost. They showed that AdaBoost could be viewed as a forward stepwise additive model using an exponential loss function.

In this paper, we present a generalized form of AdaBoost designed for multiple classes based on a multiclass exponential loss function. Many researchers have so far tried to devise various extension schemes to reduce a multiclass problem to multiple two-class problems, without providing a framework that is fundamentally capable of handling multiple classes. We differentiate our work from these *extensions* by referring to it as *generalization*. Furthermore, the proposed generalized AdaBoost (AdaBoostK) procedure can be used to either boost a multiclass classifier, or build a multiclass classifier gradually from a binary one.

To prove the effectiveness of the proposed algorithm, a diverse collection of multimedia classification problems were used to conduct benchmark experiments. These include multiclass datasets from the UCI machine learning repository [6], and an outdoor scene dataset [12]. The final results are compared with other multiclass alternatives [8]-[12].

2. AdaBoost: From Binary to Multiclass

2.1 Classic AdaBoost

AdaBoost was first introduced by Freund and Schapire [1]. In a two-class classification setting, we have training samples $\{(x_i, y_i), i = 1,...,m\}$ with x_i belongs to a feature domain and $y_i \in \{-1,1\}$. The classic AdaBoost algorithm corresponds to the following boosting procedure:

---

1 *Initialize the weight* $w_i=1/m$.

2 *Repeat for* $t=1,2,...,T$:

(a). Fit the classifier $h_t(x) \in \{-1,1\}$ *using weights* w_i *on the training data.*

(b). Compute the $\varepsilon_t=Prob(h_t(x_i) \neq y_i)$, $\alpha_t=ln(1/\varepsilon_t-1)$.

(c). Update the weights:

$W_i \leftarrow W_i exp(-\alpha_t y_i h_t(x_i))/z_t$

where z_t *is a normalization factor.*

3 *Output the final classifier:*

$H(x)=\Sigma_t \alpha_t h_t(x)$

Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)
0-7695-2646-2/06 $20.00 © 2006 **IEEE**

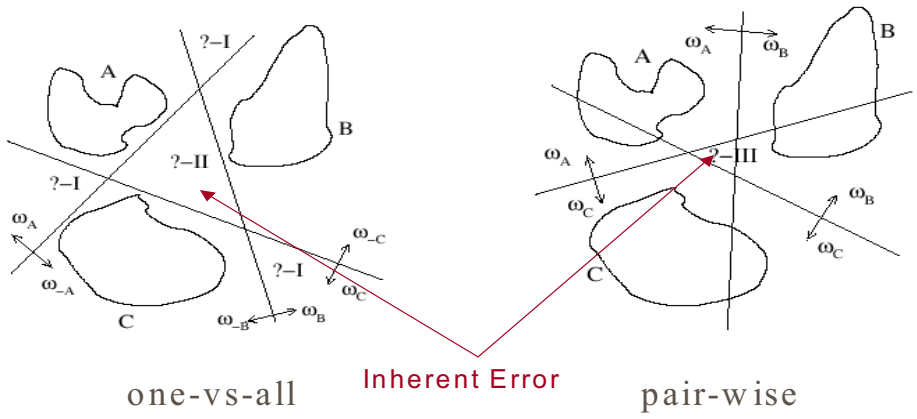

Figure 1. Using binary classifiers for multiclass problems.

2.2 Multiclass extensions for binary classifiers

Many powerful classifiers, including supporting vector machines (SVMs) [13] and AdaBoost, are binary classifiers by design. Therefore, it is desirable to extending them to solve multiclass problems. A meta-classifier composed of SVMs may be designed for general n-class problems. Two straightforward combination schemes are the one-vs-all classifier and the pairwise classifier [13][14], as illustrated in Figure 1. With the one-vs-all classifier, n SVMs are trained, each of which is able to distinguish one class from all of the others. At the end, the test vector is assigned the class corresponding to that of the machine producing the largest positive score. The pairwise classifier uses (n)(n-1)/2 binary classifiers to separate each class from each other class. A voting scheme is then used at the end to determine the correct classification. However, each technique leaves some areas of confusion. Let *i* represent class *i*. In Figure 1, the areas labeled with question marks would have inconsistent output. For example, points in region *I* on the left-hand side of Figure 1 would be classified as in class *A* and in class *C* (both would have positive output), necessitating taking the maximum score. However, this may not always be ideal, since the SVM is a boundary classifier, and its output's magnitude may not increase monotonically as one approaches the center of the distribution. Likewise, points in the center regions in Figure 1 would be classified as none of the three classes, since all SVM outputs would be negative. Each of the approaches, as well as other, more general approaches, have been discussed in the literature [13]. We will use the one-vs-all technique in this study due to its simplicity, where an image is classified with the class that produces the maximum SVM output.

Existing AdaBoost multiclass extensions include AdaBoost.M1, AdaBoost.M2 and AdaBoost.MH in [1], AdaBoost.OC in [4], and AdaBoost.ECC in [5].

3. Generalized AdaBoost (AdaBoostK)

For a K-class classification problem, we assume that a set of training samples $\{(x_i, y_i),\ i = 1,\ldots,m\}$ are given, where x_i belongs to a domain and y_i is the label of instance i. Let $\boldsymbol{U}^K = \{u_1,\ldots,u_K\}$ be the unit base vectors of a K-dimension space $\boldsymbol{R}^K$. The labels can be represented by one of the base vectors, i.e., $y_i = u_n$ if the instance i is in class n. The classifier $h(x)$ produces a vector belonging to $\boldsymbol{R}^K$, or $h(x)\in\boldsymbol{U}^K$ if $h(x)$ is discrete. We propose a generalized multiclass AdaBoost algorithm as follows:

---

1 *Initialize the weight* w_{ij} $(i = 1,\ldots,m$ *and* $j = 1,\ldots,K)$*:*

$$w_{ij} = \begin{cases} 0 & y_i = u_j \\ 1 & otherwise \end{cases}$$

2 *For* $t = 1,\ldots,T$*:*

a. *Normalize* w_{ij}.

b. *Train* $h_t(x)$ *by minimizing loss function:*

$$L = \sum_{i=1}^{m}\sum_{j=1}^{K} w_{ij}\exp((u_j - y_i)\cdot h_t(x_i)) \qquad (1)$$

c. *Update the weight matrix* w_{ij}*:*

$$w_{ij} \leftarrow w_{ij}\exp((u_j - y_i)\cdot h_t(x_i)) \qquad (2)$$

3 *Final classifier:*

$$H(x) = \sum_{t=1}^{T} h_t(x)$$

The final classifier $H(x)$ is a K-dimensional vector $(H_k,\ k = 1,\ldots,K)$. And the final output is decided by $arg\max_k(H_k)$.

Compared with the original AdaBoost algorithm, one major difference we introduce is to let the coefficient α_t be a part of the classifier $h_t(x) \leftarrow \alpha_t h_t(x)$. Therefore, the process of calculating α_t can be viewed as part of the $h_t(x)$ training process. This change would allow more freedom in training for potentially better results in most cases, and more importantly, provide an elegant theoretical framework for handling both binary and multiple classes.

Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)
0-7695-2646-2/06 $20.00 © 2006 IEEE

We first assume a current classifier $H(x)$ is obtained by minimizing the loss function in (1), and seek an improved classifier $H(x) + h(x)$ in the next iteration. The loss function becomes:

$$L = \sum_{i=1}^{m} \sum_{j=1}^{K} w_{ij} \exp((u_j - y_i) \cdot H(x_i)) \exp((u_j - y_i) \cdot h(x_i))$$

If the weights are updated as $w_{ij} \leftarrow w_{ij} exp((u_j - y_i) \cdot H(x_i))$, the loss function is brought back as it was in Eq. (1). Thus, $h(x)$ can be trained by repeating the exactly same procedure using the updated weights.

4. Boosting multiclass classifier

In this study, we use the decision tree as the base classifier to demonstrate how to apply AdaBoostK. To boost a decision tree, two questions need to be answered: (1) what is the tree-splitting criterion, and (2) how to minimize the loss function? The first question does not have a simple answer because of the complication introduced to the weight matrix, and it is related to the second question. We will address the second one for now and figure out the first one later.

Apparently, the loss function can be decomposed into sub-loss functions for each terminal node. And the sub-loss function can be minimized within the node locally. In the rest of this section, for convenience, the symbol L will be used to represent sub-loss function of a terminal node and the summation Σ_i only collects instances within the terminal node.

4.1. Discrete tree

Assume the output of a node is class c, and then the output can be formatted as αu_c, where u_c is one of the base vectors and α is the magnitude. The loss function becomes:

$$L = P_1(c)e^{-\alpha} + P_2(c)e^{\alpha} + P_3(c)$$

where $P_1(c)=(\Sigma_{i,j} w_{ij} | y_i = u_c)$, $P_2(c)=(\Sigma_i w_{ic} | y_i \neq u_c)$ and $P_3(c)=(\Sigma_{i,j} w_{ij} | y_i \neq u_c, j \neq c)$.

Therefore, the loss function is minimized when:

$$\alpha = 1/2 \cdot \log(P_1(c) / P_2(c)) \quad (4)$$

Compared with the AdaBoost.M1 algorithm in [1], α in (4) does not have to meet the requirement of accuracy better than ½ for α to be positive, as $P_2(c)$ takes only a fraction of the total weights from the misclassified instances. However, α still has a chance to be less than zero. Intuitively, one may choose $c = arg\max_k(P_1(k))$ as $P_1(k)$ is the summation of the weights from all the instances for class k. However, α will be negative if $P_2(c)$ is greater than $P_1(c)$. To avoid this problem, we can choose $c = arg\max_k(Q(k))$ with $Q(k)$ defined as:

$$Q(k) = P_1(k) - P_2(k) \quad (5)$$

It can be easily proven that $\sum_k Q(k) = 0$. Therefore, we can guarantee that $P_1(c)$ is greater than $P_2(c)$ for a positive α by choosing $c = arg\max_k(Q(k))$.

4.2. Non-discrete output

In the case of a non-discrete classifier, $h(x)$ can be any vector that belongs to $\boldsymbol{R}^K$ and there is no analytical solution for minimizing the loss function. We provide two approximation methods in this subsection.

4.2.1. Approximation 1. We assume the output of a terminal node has the form of αv, where α is the magnitude, and v is a K-dimensional vector. We need to choose v properly so that the loss L can reach its minimum at a positive α. It can be proven that $\partial^2 L/\partial^2 \alpha$ is always greater than 0. Therefore, if we choose v such that $L'(0) = (\partial L/\partial \alpha | \alpha = 0)$ is negative, the point of $\partial L/\partial \alpha = 0$ will be somewhere at $\alpha > 0$. In general, the smaller $L'(0)$ is, the bigger α is for $\partial L/\partial \alpha = 0$, and the smaller L will be. Therefore, we can determine the vector v by minimizing $L'(0)$:

$$(\frac{\partial L}{\partial \alpha} | \alpha = 0) = \sum_{i,j} w_{ij} (u_j - y_i) \cdot v = -Q \cdot v$$

where the vector Q is as defined in Equation (5). The obvious choice for minimum $L'(0)$ is $v = Q$, since the magnitude of v is not relevant. Then α can be obtained using a second-order Taylor series approximation, i.e.:

$$L(\alpha + \Delta\alpha) = L(\alpha) + L'(\alpha)\Delta\alpha + L''(a)\Delta\alpha^2 / 2$$

The loss function is minimized when $\Delta\alpha = L'(\alpha)/L''(\alpha)$. Starting with $\alpha = 0$, one can iterate the calculation until it converges.

4.2.2. Approximation 2. The loss function can be minimized directly through regression. Assuming the output is a K-dimensional vector h, one can minimize the loss function for each of the components:

$$\frac{\partial L}{\partial h_c} = (\sum_i w_{ic} e^{-h(i)} | y_i \neq u_c) e^{h_c} - (\sum_{i,j} w_{ij} e^{h_j} | y_i = u_c) e^{-h_c} = 0$$

where $h(i) = h \cdot y_i$. Therefore:

$$h_c = 1/2 \cdot \frac{\log(\sum_{i,j} w_{ij} e^{h_j} | y_i = u_c)}{\log(\sum_i w_{ic} e^{-h(i)} | y_i \neq u_c)} \quad (7)$$

Starting with $v = 0$, we can repeat the calculation (7) for components $c=2, ..., K$ until they converge.

4.3. Quantity $Q(c)$ and tree splitting criteria

From the previous two subsections, it is clear that the quantity $Q(c)$ defined in Equation (5) plays a very important role. $Q(c)$ functions as the probability of class c. Unlike a real probability, $Q(c)$ not only takes into account the number of instances of class c and their weights $P_1(c)$, it also takes into account the influence $P_2(c)$ of non-class-c instances in terms of the cost being classified as class c. Therefore, $Q(c)$ can be viewed as "pseudo probability" and used to replace the probability in the splitting criteria for the decision tree.

The most popular binary tree-splitting criteria are Entropy $-\sum P \log(P)$ and the Gini index $-\Sigma P^2$ [1]. Clearly, the entropy criterion cannot be used because

Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)
0-7695-2646-2/06 $20.00 © 2006 IEEE

Q can be negative. Therefore, the trees should be split using the Gini index, i.e., minimizing $-\Sigma Q^2$.

4.4. Boosting binary classifier

In this section, we explain how to build a boosted multiclass classifier starting from a binary classifier. The idea is to build K binary classifiers and then use the K outputs to form the output of the multiclass classifier. We then use AdaBoostK to determine what these classifiers should be and how to train them.

We assume the multiclass classifier's output is in the form of $\alpha h_k(x)$ $(k = 1,...,K)$, where $h_k(x) \in [0,1]$ is the output of kth classifier and α is the magnitude. Consequently, we can use the approximation 1 in section 3.2 to determine $h_k(x)$, i.e., minimizing $L'(0)$:

$$\begin{aligned} L'(0) &= \sum_{i,j} w_{ij}(u_j - y_i)\cdot h(x_i) \\ &= \sum_{k=1}^{K}\left(\left(\sum_i w_{ik}h_k(x_i)\middle|y_i \neq u_k\right) - \left(\sum_{i,j} w_{ij}h_k(x_i)\middle|y_i = u_k\right)\right) \end{aligned} \quad (8)$$

It can be seen that we would like $h_k(x_i)$ to be 1 if $y_i = u_k$ and 0 otherwise. Therefore, h_k should be a typical binary one-vs-all classifier for class k, which is as expected. Moreover, Equation (8) suggests that the instance i should have weight d_{ik} defined as:

$$d_{ik} = \begin{cases} \sum_j w_{ij} & y_i = u_k \\ w_{ik} & y_i \neq u_k \end{cases} \quad (9)$$

Thus a K-class classifier can be built as follows:

1. Build K one-vs-all binary classifiers.
2. Train the kth classifier for class k. The weight of instance i should be d_{ik} as defined in Equation (9).
3. Compute the output of the multiclass classifier in the form of αh_k, where h_k is the output of the kth classifier and α is a positive number. Determine α by minimizing the loss function. We can use the same Taylor series approximation in section 4.2.1.

5. Choosing the best boosting scheme

To understand the performance of various schemes in AdaBoostK, several datasets from the UC-Irvine machine learning repository [6] are used. In the case of two-class, the results show no significant difference compared with other versions of AdaBoost [1–3].

For multiple classes, decision trees are used as the base classifiers. The trees are split by minimizing $-\Sigma Q^2$. For binary base classifiers, a nondiscrete binary tree was used, and the tree is split per entropy. All trees are pruned by cutting the node with the smallest total weight. Four boosting schemes are tested:

DT: AdaBoostK + Discrete Tree
RT1: AdaBoostK + Non-discrete: Approximation 1
RT2: AdaBoostK + Non-discrete: Approximation 2
BT: AdaBoostK + Binary Non-discrete Tree

To compare the performances of the above schemes, four datasets are used. For example, the "Waveform" dataset contains three classes (see [7] for details). The training sample size is 100 instances per class and 300 in total. A test set of 5000 instances was generated independently for each training set. In order to ensure that the statistical errors are small enough, the averaged result over twenty such independently drawn sets (combinations of training-test set) was used to estimate the final error rates.

The results of the method RT1 and BT are shown in Tables 1 and 2, respectively. As expected, the best result from RT1 (15.7%) is better than that from BT (16.6%) since RT1 handles all classes simultaneously.

Table 1 lists the results from all four of the methods with all the tree sizes set to six terminal nodes per class. RT1 outperforms the other schemes. Note that RT2 is the most "greedy" method. In theory, one can prove that RT1 reduces the loss function the most among the testing methods, but RT2 is worse than even DT. This suggests that the most "greedy" method might not produce the best result, and a "gentler" approach might do better.

Table 1. The average test error rates (%) of the simlated data waveform. (The statistical uncertainties of those rates are around 0.04%)

Boosting Iterations	Boosting Schemes			
	DT	RT1	RT2	BT
100	17.46	17.44	17.67	16.96
200	16.89	16.54	16.95	16.85
500	16.39	16.14	16.57	16.79
1000	16.18	15.95	16.35	16.77

All the tree sizes are set to 6 terminal nodes per class, and 1000 boosting iterations are used in all the cases. The final testing error rates are listed in Table 2. The best results from reference [2] are quoted in the table as FHT for comparison. FHT used the algorithm AdaBoost.MH and the base classifiers were also decision trees. It can be seen that AdaBoostK is significantly better than AdaBoost.MH, as well as BT. In almost all the tests, RT1 outperformed all the other methods, except for the vowel dataset where the difference is probably statistically insignificant.

Table 2. Testing error rates (%). "FHT" is the result from [2], which used the AdaBoost.MH algorithm.

Dataset	MultiClass AdaBoost				FHT [2]
	DT	RT1	RT2	BT	
Vowel	44	45	40	47	50
Waveform	15.9	15.6	16.9	16.6	18.2
Satimage	7.6	7.3	8.3	7.7	8.8
Letter	2.6	2.3	3.1	2.8	2.8

6. Multimedia Applications

In this study, we use a variety of multimedia classification problems to benchmark the empirical effectiveness of the generalized AdaBoost algorithm, given its theoretically proved advantages.

We use the decision tree as the base classifier for boosting in all the experiments, and the RT1 scheme for non-discrete output cases. The size of the tree will depend on the number of classes and the number training samples. We always use 1000 iterations.

In [8], the base classifier is nearest neighbor (NN). In [9] and [11], the methods of one-vs-all ("OVA") and all-pairs ("Pair") are used to extend the binary classifier (SVM) to a multiple classifier, respectively. In [10], the results are obtained using several AdaBoost multiclass versions. In [12], SVMs are trained to classify the images in the one-vs-all fashion.

6.1 Recognition of Handwritten Digits (UCI)

This is a handwritten digit dataset consisting of 250 samples collected from 44 writers. The samples written by 30 writers are used for training, and the digits written by the other 14 are used for writer-independent testing. A total of 16 features were extracted from each handwriting. The datasets have 10 classes (for numbers 0 to 9) with 7479 samples for training and 3498 for testing.

The decision tree used in AdaBoostK has 50 terminal nodes. The results in terms of error rates are shown in the table below.

AdaBoostK	[8] Boost-NN	[9] SVM Pair/OVA	[11] SVM OVA
2.1	3.9	1.9/2.0	2.9

For this dataset, AdaBoostK produced essentially the same best result as in [9], where the baseline classifier SVM is more powerful than decision trees (and the pairs did slightly better than one-vs-all). Note that the one-vs-all scheme using a SVM baseline did much worse in [11].

6.2 Vowel Recognition (UCI)

This dataset is intended for speaker-independent recognition of the eleven steady state vowels of British English. Overall, 528 samples are collected from 15 speakers and 462 samples are collected from different speakers for testing. Each sample has 11 input features and there are 11 classes in the dataset.

The decision tree used in AdaBoost has 70 terminal nodes. The error rates are compared in the table below.

AdaBoostK	[8] Boost-NN	[11] SVM OVA
42	42	50

For this dataset, AdaBoostK produced the same best result as in [8], while the one-vs-all scheme using a SVM baseline in [11] again did much worse.

6.3 Image Segmentation (UCI)

In this dataset, samples were drawn randomly from a database of 7 outdoor images (brick, sky, foliage, cement, window, path, grass). The images were hand-segmented to create a classification for every pixel. Each sample is a 3x3 region. There are 210 samples for training and 2100 for testing. Each sample has 19 attributes, including position of the center pixel of the region, number of pixels in a region, lines through the region, contrast measures of adjacent pixels in the region, average intensity and color measures over the region, and so on.

The decision tree used in AdaBoostK has 12 terminal nodes. The results are compared below.

AdaBoostK	[9] SVM Pair/OVA	[10] C4.5
3.7	4.7/5.2	4.2*

In this experiment, the error rate generated by AdaBoostK is the lowest. Again, we noticed that the pairs did better than one-vs-all in [9], probably due to a higher degree of specification (more classifiers are trained in a pair-wise scheme). The baseline classifier in [10] was C4.5.

6.4 Outdoor Scene Classification (ISC)

Color was found to be salient for the problem of outdoor scene classification [12]. Furthermore, spatial information need to be incorporated to distinguish scenes containing similar colors. To that end, spatial color moments are computed by first dividing the image into 49 regions using a 7 x 7 grid and calculating the mean and variance of each band of an Luv-transformed image. This yields 49 x 2 x 3 = 294 features. It is usually advantageous to use a more perceptually uniform color space than RGB such that perceived color differences correspond closely to Euclidean distances in the color space selected for representing the features.

These features are used to distinguish between six types of outdoor scenes (beach, sunset, fall foliage, field, mountain, and urban). These categories are important to multimedia applications using consumer and professional photographs. The images used for training and testing include 2400 Corel and consumer images (400 per class). SVM classifiers are extended to multiple classes by using a one-vs-all approach.

The decision tree used in AdaBoostK has 20 terminal nodes. The results are compared in the following table, where AdaBoostK showed a clear advantage even when the baseline classifies are simple decision trees.

AdaBoostK	[12] SVM (OVA)
18.8%	21.3%

7. Discussions and conclusions

In this paper, a generalized AdaBoost algorithm is presented for multiclass classification. AdaBoostK extends the original two-class AdaBoost procedure to multiple classes through a generalized *multiclass* loss function. Such generalization provides theoretical advantages over the pair-wise and one-vs-all extensions, which are inherently based on binary loss functions. The *inherent errors* illustrated in Figure 1 are eliminated, thanks to the multiclass loss function. As demonstrated using a wide variety of multimedia classification problems, it is capable of producing improved classification results.

The following important features are worth noting:

1. In the AdaBoostK algorithm, a weight matrix w_{ij} is introduced instead of one weight per instance, and w_{ij} can be understood as the cost of sample i being classified as class j. If the training data carry their own weight, the matrix w_{ij} should be initialized accordingly instead of using step 1 of the algorithm.
2. In the case of two classes, AdaBoostK regresses to the original AdaBoost (DT is equivalent to the original AdaBoost). RT2 is equivalent to "Real AdaBoost" in [2]. For RT1, one gets $\alpha = 1$ with only one round of Taylor series approximation, and that would make RT1 equivalent to "Gentle AdaBoost" in [2].
3. Similar to the original AdaBoost algorithm, AdaBoostK is also resistant to overfitting.
4. The final output of the AdaBoostK $H(x)$ cannot be considered directly as a probability distribution. By definition, the result will not make any difference if the output is subtracted by a constant. If properly normalized, $H(x)$ does act like the pseudo-probability as defined in Equation (5).
5. In general, the base classifiers do not have to be a single type of classifier. Different types of classifiers can be used for each step of boosting, as long as the loss function is reduced. It is very intriguing to use hybrid classifiers in boosting to improve the accuracy or to reduce the number of boosting iterations.

Future work includes investigating the use of different baseline classifiers, and extending the boosting procedure to incremental learning. As mentioned earlier, the proposed AdaBoostK algorithm is amenable to building a boosted multiclass classifier starting from a binary classifier or from fewer classes.

References

[1] Y. Freund and R. Schapire, "A decision-theoretic generation of on-line learning and an application to boosting," *Journal of Computer and System Science,*. 55(1): 119–139, 1997.

[2] J. Friedman, T. Hastie, and R. Tibshirani, "Additive Logistic Regression: a Statistical View of Boosting," *Annals of Statistics,* 28(2): 337–407, 2000.

[3] R. Schapire and Y. Singer, "Improved Boosting Algorithm Using Confidence-rated Prediction," *Machine Learning,* 37(3): 297–336, 1999.

[4] R. Schapire, "Using Output Codes too Boost Multiclass Learning Problems," *Proc. 14th International Conference on Machine Learning,* 313–321, 1997.

[5] V. Guruswami and A. Sahai, "Multiclass learning, boosting, and error-correcting codes," *Proc. 12th Annual Conf. Computational Learning Theory*, 145–155, 1999.

[6] D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz, 1998. UCI Repository of machine learning databases.

[7] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Tree*, Wadsworth, Belmont, CA, 1984.

[8] V. Athitsos and S. Sclaroff, "Boosting Nearest Neighbor Classifiers for Multiclass Recognition," *Boston University Computer Science Tech. Report,* No. 2004-006.

[9] A. Passerini, M. Pontil, and P. Frasconi, "From Margins to Probabilities in Multiclass Learning Problems," *Proc. 15th European Conf. On Artificial Intelligence*, 2002.

[10] Y. Sun, S. Todorovic, and J. Li, "Unifying Multi-Class AdaBoost Algorithms with Binary Base Learners under the Margin Framework," *Proc. 22nd international conference on Machine learning*, 872 - 879, 2005.

[11] E. L. Allwein, R. E. Schapire, and Y. Singer, "Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers," *Journal of Machine Learning Research*, 113-141, 2001.

[12] J. Luo, M. Boutell, R. T. Gray, and C. Brown, "Image Transform Bootstrapping and Its Applications to Semantic Scene Classification," *IEEE Trans. Systems, Man, and Cybernetics - Part B* 35(3), 2005.

[13] B. Scholkopf, C. Burges, and A. Smola. Advances in Kernel Methods: Support Vector Learning. MIT Press, Cambridge, MA, 1999.

[14] D.M.J. Tax and R.P.W. Duin. Using two-class classifiers for multiclass classification. Proceedings of Int. Conf. on Pattern Recognition, Quebec City, Canada, August 2002.