Fast AdaBoost Training using Weighted Novelty Selection

Mojtaba Seyedhosseini[1,2], António R. C. Paiva[1] and Tolga Tasdizen[1,2]
[1] *Scientific Computing and Imaging Institute,* [2] *Dept. of Electrical and Computer Eng.,*
University of Utah, Salt Lake City, UT 84112
email: {mseyed,arpaiva,tolga}@sci.utah.edu

***Abstract*— In this paper, a new AdaBoost learning framework, called WNS-AdaBoost, is proposed for training discriminative models. The proposed approach significantly speeds up the learning process of adaptive boosting (AdaBoost) by reducing the number of data points. For this purpose, we introduce the weighted novelty selection (WNS) sampling strategy and combine it with AdaBoost to obtain an efficient and fast learning algorithm. WNS selects a representative subset of data thereby reducing the number of data points onto which AdaBoost is applied. In addition, WNS associates a weight with each selected data point such that the weighted subset approximates the distribution of all the training data. This ensures that AdaBoost can trained efficiently and with minimal loss of accuracy. The performance of WNS-AdaBoost is first demonstrated in a classification task. Then, WNS is employed in a probabilistic boosting-tree (PBT) structure for image segmentation. Results in these two applications show that the training time using WNS-AdaBoost is greatly reduced at the cost of only a few percent in accuracy.**

I. Introduction

BOOSTING is a general learning concept to train a single strong learner by combining a set of weak learners [1]. Based on this concept, many methods have been proposed in the literature to solve several problems, such as classification [2], clustering [3], recognition [4], etc. The first practical polynomial-time boosting algorithm was developed by Schapire in 1990 [5]. And, in 1995, Freund and Schapire proposed AdaBoost [6]. AdaBoost learns a strong classifier by linearly combining a set of weak classifiers. In order to focus the learning on the most difficult samples, it uses a sample weighting strategy. After the addition of each weak classifier, a sample weight is updated indicating the importance of the sample for classification by subsequent weak classifiers. The weights of the misclassified samples are increased, and the weights of correctly classified samples are decreased to lead the new weak classifiers to focus on the more difficult samples. Despite the simplicity of the method, AdaBoost has been shown to achieve good bounds on its training and generalization error [7].

Following the success of AdaBoost, several related approaches have been proposed. These include Gentle AdaBoost (GAB) [8], FloatBoost [4], robust alternating AdaBoost (RAAB) [9], Modest AdaBoost [10], AdaTree [11], probabilistic boosting tree (PBT) [2], among others. GAB uses Newton's algorithm instead of greedy steps for optimization, thereby improving generalization performance [8]. FloatBoost also improves the generalization performance by backtracing and pruning previously learnt weak classifiers deemed irrelevant. RAAB and modest AdaBoost use modified loss functions to reduce the effect of outliers in the optimization. AdaTree and PBT both use a tree structure and implement AdaBoost as the classifier on the nodes. The advantage of using a tree structure is that by focusing the training of AdaBoost on simpler subproblems, training of the classifiers is simpler and requires less iterations. This makes AdaTree and PBT particularly useful for learning with larger datasets. Another significant advantage of PBT is that it provides an approach for learning the posterior distribution which is useful in vision problems.

Although the AdaBoost learning algorithm is considered to be computationally efficient, training can be time consuming in some cases. An obvious example, is that of learning from large datasets, and this issue can aggravated depending on the learning complexity of the weak classifier. Convergence of the learning algorithm can also be slow for problems with a very complex decision boundary. In those problems the first weak classifiers influence the re-weighting process making it difficult for later weak classifiers to focus exclusively on the harder examples. As mentioned earlier, AdaTree and PBT are helpful in these cases because the strong classifier is obtained by combining simpler AdaBoost classifiers and because classifiers in lower levels of the tree learn from subsets of the original data [11], [2]. The tree structure is also helpful in the testing phase because the classification of some samples can be obtained without fully traversing the tree. Another approach for fast training proposed in the literature involves resampling the original data according to the distribution weights [12].

In this paper, we propose a new learning framework which speeds up the training of AdaBoost and any other boosting based algorithms, including all of the aforementioned methods. For this purpose, we introduce a novel sampling strategy, weighted novelty selection (WNS), and combine it with AdaBoost to obtain the WNS-AdaBoost framework. WNS is a sampling method which reduces the number of data points by selecting representative points from the dataset. It also determines a corresponding weight for each of these selected points which shows the importance of that point and aims at preserving the distribution of the original data. By reducing the number of training samples, the proposed framework significantly reduces the training time. The output of the WNS algorithm is then used by AdaBoost, or any of its variants, to learn a discriminative model. This is achieved by training AdaBoost on the representative set of data points and

initializing the weight distribution with the weights obtained from WNS after normalization.

II. Weighted Novelty Selection

Weighted novelty selection is the pre-processing sampling method in the WNS-AdaBoost framework. The main idea is to provide the boosting algorithm with a *concise* summary of the training dataset such that the learning algorithm can quickly and efficiently train the classifier. WNS achieves this by selecting representative points from the training dataset, and by deriving a corresponding set of weights such that the two pieces of information summarize the original data distribution.

WNS was inspired by Platt's work on resource-allocating networks [13]. Platt introduced a criterion to decide whether a given input point should be added to a growing radial basis function neural network in order to minimize network error. The point was added if the distance to the other points already in the network was larger than a threshold and the network error was above another threshold. Fundamentally, Platt's criterion aims to select a reduced set of data points that preserves the data structure relevant to the reduction of the modeling error.

Similarly, WNS picks a data point as a representative point if the smallest distance to all previous representative points is larger than a threshold δ. Hence, δ is a parameter that indirectly controls the number of representative points. Smaller δ increases the number of representative points and vice versa. This procedure ensures that enough points are picked to cover the whole space while keeping the number of them to a minimum.

The set of representative points provides a limited characterization, however, because it does not accurately reflect the density of the input data points. For example, in a classification problem with classes separated by a low density region, one wants to place the decision boundary between the two classes to minimize the error. However, the representative set alone would fail to properly provide this information, unless the clusters are clearly separated and the separation is larger than $\delta/2$. Hence, to more accurately capture the structure of the original training dataset, WNS associates a weight to each representative point. This weight corresponds to the number of input data points assigned to a representative data point, which captures information about the data distribution. Intuitively, since the weight states how many data points are summarized by a representative point, one can think that representative points with larger weights correspond to areas with higher density and thus are more relevant.

The WNS sampling strategy is quite simple and follows directly the ideas described above. Consider a set of N input data points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, and denote the representative set by $X^R = \{\mathbf{x}'_j\}$ and the corresponding set of weights by $W = \{w_j\}$, $j = 1, 2, \ldots$. In addition, denote by $I_X = \{j_1, \ldots, j_N\}$ the indices of the representative points in X^R for each $\mathbf{x}_i \in X$, such that $\mathbf{x}_i \in X$ is represented by $x'_{j_i} \in X^R$.

Accordingly, the WNS sampling algorithm proceeds as follows:

1) Initialization: set $X^R = \{\mathbf{x}_1\}$, $W = \{1\}$, $I_X = \{1, 0, \ldots, 0\}$, and $Y = \emptyset$;
2) For each $\mathbf{x}_i \in X \setminus X^R$,
   a) Compute the distance of $\mathbf{x}_i$ to all $\mathbf{x}'_l \in X^R$, $d(\mathbf{x}_i, \mathbf{x}'_l)$;
   b) Find $n = \underset{l}{\operatorname{argmin}}\, d(\mathbf{x}_i, \mathbf{x}'_l)$;
   c) If $d(\mathbf{x}_i, \mathbf{x}'_n) > \delta$,
      Add the point to X^R and set the corresponding weight in W to 1;
      else if $d(\mathbf{x}_i, \mathbf{x}'_n) \leq \delta/2$,
      Set $j_i = n$ and $w_n = w_n + 1$;
      else
      Add $\mathbf{x}_i$ to Y.
3) For each $\mathbf{x}_i \in Y$,
   a) Compute the distance of $\mathbf{x}_i$ to all $\mathbf{x}'_l \in X^R$, $d(\mathbf{x}_i, \mathbf{x}'_l)$;
   b) Find $n = \arg\min_l d(\mathbf{x}_i, \mathbf{x}'_l)$;
   c) Set $j_i = n$ and $w_n = w_n + 1$.

Note that even though the algorithm depicted here uses distances, the algorithm can be readily adapted to use similarities. This can be obtained simply by inverting the inequality comparisons. For dissimilarities, the distance metric can be relaxed to a semi-metric (which does not verify the triangle inequality) without affecting the outcome.

Computationally, the WNS sampling algorithm is fast and efficient since the algorithm proceeds in a single pass through the data. The computational complexity is $O(NM)$, where N is the number of points in the original dataset and M is the number of points in the representative set. Although in theory it is possible for the computationally complexity to be $O(N^2)$, this corresponds to the limiting case $\delta \to 0$, in which case the representative set equals the input data. Typically, M is much smaller than N. An additional advantage is that the WNS algorithm can be easily parallelized.

It is noteworthy that WNS is conceptually similar to the weighted Nyström approach proposed for kernel methods by Zhang and Kwok [14]. Both methods provide a weighted sampling strategy for summarizing the dataset. The weighting have slightly different roles however. In weighted Nyström, the weights are utilized to approximate the computation on large kernel systems by compressing the kernel matrix and expanding the eigendecomposition. In contrast, WNS explicitly summarizes the data distribution and passes that information directly to the boosting algorithm using the weight data distribution.

Clearly, many density-preserving data characterization methods exist in the literature, including: mixture models [15], [16], mean shift [17], vector quantization [18], etc. However, these methods have typically several data-dependent parameters that need to be carefully set. In comparison, WNS has only one parameter, δ, which is largely independent of the data if the data range is normalized. Moreover, WNS makes no assumptions on the data distribution.

We propose to employ WNS to speed up AdaBoost. In this regard, WNS is applied to each of the classes to reduce the number of points in them. Afterward, the selected points and corresponding weights are passed to AdaBoost. In addition to reducing the number of points, which can speed up any boosting algorithm, in this framework AdaBoost can take the advantage of the corresponding weights. In other words, not only the reduction of the number of data points improves the speed of the AdaBoost but also the corresponding weights help to keep the performance of the AdaBoost at reasonable rate. So unlike to the usual method that the weights are the same for all the input points, this time each input point has its specific weight which shows how important it is.

III. WNS-AdaBoost

The AdaBoost algorithm learns a strong classifier by linearly combining (simpler) weak classifiers according to,

$$H(x) = \sum_t \alpha_t h_t(x), \tag{1}$$

where $h_t(x)$ denotes a weak classifier. Different weak classifiers can be used in this framework. The key contribution of AdaBoost is to use a distribution, i.e. a set of weights, over the training samples. These weights are updated adaptively at each iteration of AdaBoost and play an important role in determining the combination factor for each weak classifier, i.e. $\{\alpha_t\}$ in equation 1.

At each iteration, AdaBoost selects the weak classifier that minimizes the weighted error,

$$\epsilon_t = \sum_i w_i [h_t(x_i) \neq y_i], \tag{2}$$

where w_i is the sample weight and y_i denotes the desired output for input x_i. This error is calculated with respect to the weights w_i on which the weak classifier is trained. The vote weight of each classifier is computed using this error

$$\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t}). \tag{3}$$

Accordingly, the weights are updated with

$$w_i^{t+1} = \frac{w_i^t \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \tag{4}$$

where w_i^t denotes the weight of training sample x_i at iteration t and Z_t is a normalization factor which is chosen so that w^{t+1} will be a probability distribution. This update rule increases the weights of the samples which are difficult to classify and decreases the weights of the samples which are easy to classify, so, the next weak classifier focuses on the more difficult samples.

Typically, the weights are initialized uniformly because prior knowledge about the importance of the training samples is not available. Put differently, AdaBoost is left to infer the distribution of the samples solely based on their relative amount. In the proposed method, however, WNS is used to explicitly capture and summarize the data distribution using a reduced number of training samples. This information is transferred to AdaBoost by setting the weights according to the values obtained from WNS. Hence, the weights are no longer initialized uniformly and each representative point has its own weight which can be different from the other points. Using this strategy, instead of a large number of training points with the same weights, AdaBoost is given a smaller number of training points together with prior information about the importance of them. To summarize, WNS speeds up AdaBoost in the training stage by reducing the number of training samples and maintains also the performance of AdaBoost at a good level by providing prior information about the importance of the selected representative points.

Given a training set $X = \{x_1, ..., x_N\}$ and the corresponding labels $L = \{l_1, ..., l_N\}$, $l_i \in \{-1, 1\}$, the WNS-AdaBoost training algorithm proceeds as follows:

1) Separate the classes and make two sets: $X_1 = \{x_i | l_i = -1\}$, $X_2 = \{x_i | l_i = 1\}$.
2) Choose a δ and run WNS for X_1 and X_2. The representative points and weights for each class are: $X_1 \rightarrow (Z_1, W_1)$, $X_2 \rightarrow (Z_2, W_2)$.
3) Construct a new training set $Z = \{Z_1, Z_2\}$ and $W = \{W_1, W_2\}$.
4) Normalize W so it will be a probability distribution.
5) Use Z, W to train an AdaBoost classifier.

Although we described the WNS-AdaBoost for training the AdaBoost, one can notice that it can be used also in other AdaBoost based frameworks, e.g. PBT, AdaTree, etc. This generalization can be described by considering the WNS as a pre-processing step. In other words, WNS gets the training set and provides a new training set with corresponding weights.

IV. Experimental Results

We illustrate the performance of WNS-AdaBoost in terms of accuracy and speed on two different problems: Poker hand classification and texture segmentation. In the first experiment we verify the effectiveness of WNS-AdaBoost in a simple AdaBoost structure while in the second experiment we show its performance in the probabilistic boosting tree (PBT) framework.

A. Poker hand classification

The poker hand dataset is available from the UCI Machine Learning Repository [19]. The dataset contains 25010 data points for training and 1000000 data points for testing distributed over 11 classes. This dataset was used in a two-class form where the first class represents the hands which are not a recognized poker hand and the second class contains the poker hands from one pair to royal flush. The size of the feature vector is ten, i.e. suit and rank for each card. A decision tree with 7 nodes was used as the weak classifier and boosting was run for 600 iterations.

Table I shows the classifier training time and its accuracy for different parameter values of δ. As we can see by using $\delta = 3$ the WNS-AdaBoost is more than two times faster than the conventional AdaBoost algorithm while its accuracy is

TABLE I

TRAINING TIME AND PERFORMANCE FOR THE "POKER HAND" DATASET.

Method	δ	Number of training samples	Time for applying WNS (s)	Time for training	Training error	Testing error	Speedup
WNS-AdaBoost	3	13396	8.85	135s	13%	20%	2.23
WNS-AdaBoost	2.7	18278	10.05	236.10s	11%	17%	1.3
AdaBoost	–	25010	–	320.19s	10.3%	16%	–

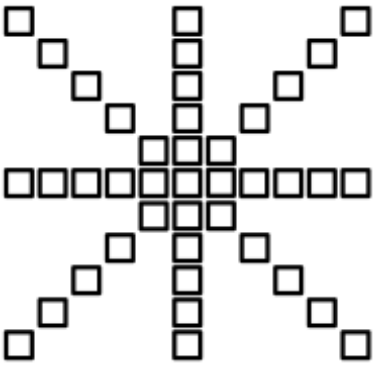

Fig. 1. The stencil which is used to sample the input image.

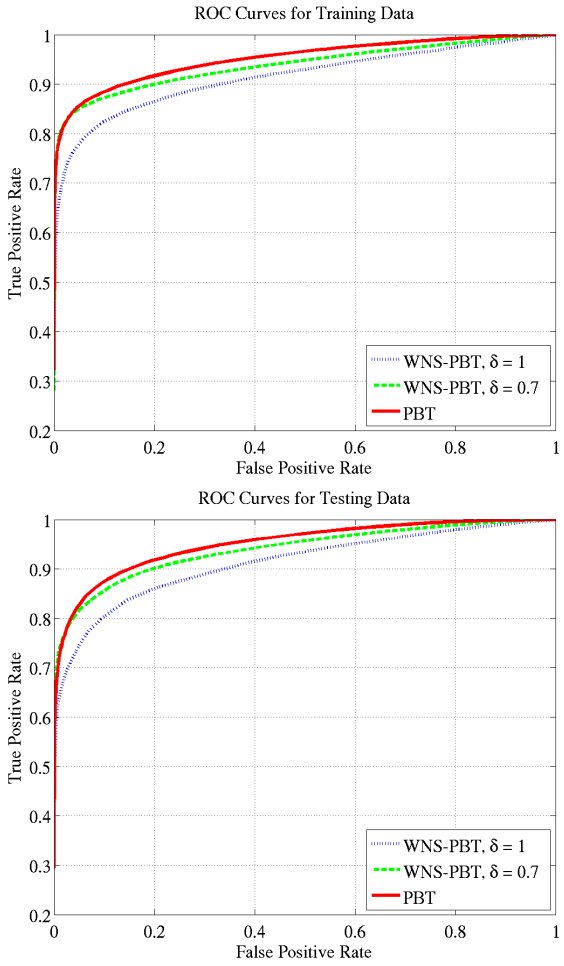

Fig. 2. ROC curves for the texture segmentation experiment.

4% less than the AdaBoost. As δ decreases the performance improves at the cost of speed, e.g. WNS-AdaBoost with parameter $\delta = 2.7$ performs almost the same as AdaBoost while it is 1.3 times faster than AdaBoost.

B. Texture segmentation

In order to show that proposed method is not restricted to regular AdaBoost algorithm and can be used in any AdaBoost based classifier, we adopted the probabilistic boosting tree (PBT) [2] together with WNS-AdaBoost for texture segmentation. The PBT learns a discriminative model in a hierarchical structure. At each level of the hierarchy, PBT learns some AdaBoost classifiers and use them to split data to smaller groups. The details can be found in [2]. In our experiment the depth of the tree in PBT is two and we used a decision tree with five nodes as our weak classifier.

The dataset used in this experiment contains 20 star images generated from five different textures for foreground and four different textures for background using textures from Brodatz database [20]. Eight of these images were used for training and the remaining 12 images were used for testing. The input feature vector to the PBT classifier was formed by sampling the input image at every pixel using an 11×11 stencil, Figure 1. The size of the feature vector is 41.

The ROC curves for training and testing images are shown in Figure 2. The accuracy performance of WNS-PBT is close to the AdaBoost while it is much faster. The classifier training time for PBT and WNS-PBT with different parameter values of δ is shown in Table II. One can notice that there is a trade-off between the accuracy and the speed of the classifier. In other words, we can make the classifier faster at the cost of accuracy. In this experiment it seems that $\delta = .7$ is a reasonable choice that makes the classifier much faster at the cost of few percents decreasing in the accuracy. It must be emphasized that speedup in this experiment is much higher compared to the previous experiment due to the size of the dataset, so, WNS-AdaBoost is more useful for the large dataset cases. The segmentation results on some test images for WNS-PBT and PBT are shown in Figure 3. The results for WNS-PBT with $\delta = 0.7$ and PBT are shown in third and fourth columns. The results are really close while the

TABLE II

TRAINING TIME FOR THE "TEXTURE SEGMENTATION" EXPERIMENT.

Method	δ	Number of training samples	Time for applying WNS (s)	Time for training	Speedup
WNS-PBT	1	1808	108.62	5.82s	38351
WNS-PBT	0.7	27206	11186.96	502.64s	444
AdaBoost	–	524288	–	62hours	–

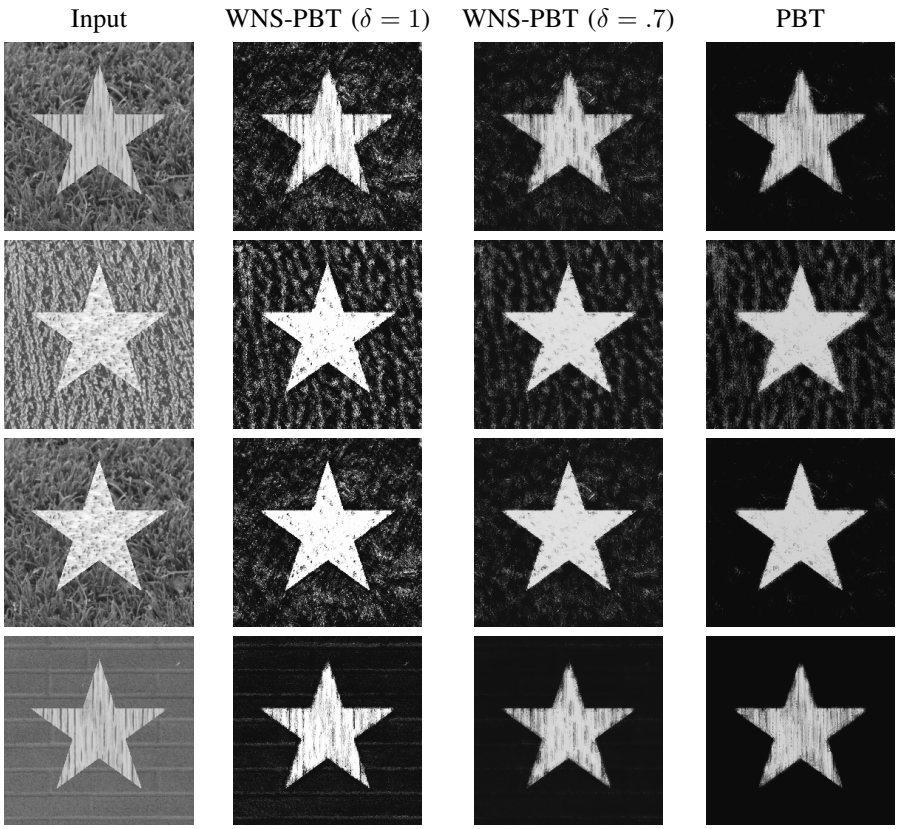

Fig. 3. Test results for the texture segmentation experiment. The first column shows the input image and the remaining columns show the output of WNS-PBT with $\delta = 1$, WNS-PBT with $\delta = 0.7$, and PBT classifiers respectively.

WNS-PBT method is 444 times faster than PBT in training.

V. CONCLUSIONS

This paper introduces a new framework WNS-AdaBoost for efficient learning of discriminative boosting models. The WNS-AdaBoost framework efficiently selects a reduced set of representative training points, thus reducing the overall computational complexity for training and increasing the speed of the training process. Moreover, by returning the weights for each of representative point, WNS provides a compact representation of the distribution of the training data in a way that is naturally amenable to AdaBoost. The combination of these two characteristics ensure faster training and with minimal loss of accuracy.

The improvement in training speed is achieved potentially at the expense of a small reduction in accuracy. This behavior is regulated by the sampling parameter δ. If δ is increased from zero, the size of the representative training set given to AdaBoost is reduced, thereby increasing the training speed but decreasing the accuracy because of the increasingly crude representation of the data. Conversely, as δ tends to zero, WNS outputs the original training data, which is equivalent to the direct use of AdaBoost. Still, the experiments show that by appropriately choosing δ, it is possible to achieve large improvements in training speed with negligible loss of accuracy.

It must be emphasized that the WNS-AdaBoost framework extends beyond AdaBoost alone to any other AdaBoost-based classifier. As an example, this generality was explicitly demonstrated in the application of the framework to the PBT classifier. Additionally, it is noted that, although the

algorithm was described here only for the two-class case for ease of presentation, the framework is not restricted to this case and can be generalized to multi-class problems in a straightforward way.

REFERENCES

[1] M. Kearns and L. G. Valiant, "Cryptographic limitations on learning boolean formulae and finite automata," *Journal of the Association for Computing Machinery*, vol. 41, no. 1, pp. 67–95, 1994.

[2] Z. Tu, "Probabilistic boosting-tree: learning discriminative models for classification, recognition, and clustering," *Proceedings of ICCV*, vol. 2, pp. 1589–1596, 2005.

[3] A. Demiriz, K. Bennett, and J. Shawe-Taylor, "Linear programming boosting via column generation," *Journal of Machine Learning Research*, vol. 46, pp. 225–254, 2002.

[4] S. Li and Z. Zhang, "Floatboost learning and statistical face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1112 –1123, Sep. 2004.

[5] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, 1990.

[6] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *European Conference on Computational Learning Theory*, pp. 23–37, 1995.

[7] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Proceeding of 11th Annual Conference on Computational Learning Theory*, pp. 80–91, 1998.

[8] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *Annals of Statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[9] H. Allende-Cid, R. Salas, H. Allende, and R. anculef, "Robust alternating adaboost," in *Progress in Pattern Recognition, Image Analysis and Applications*, vol. 4756/2007, 2007, pp. 427–436.

[10] A. Vezhnevets and V. Vezhnevets, "'modest adaboost' – teaching adaboost to generalize better," in *Proceedings of International Conference on Computer Graphics and Vision*, 2005.

[11] E. Grossmann, "Adatree: Boosting a weak classifier into a decision tree," in *CVPR workshop on learning in computer vision and pattern recognition*, 2004.

[12] H. chuan Wang and L. ming Zhang, "A novel fast training algorithm for adaboost," *Journal of Fudan University*, vol. 1, 2004.

[13] J. Platt, "Resource-allocating network for function interpolation," *Neural Computation*, vol. 3, no. 2, pp. 213–225, 1991.

[14] K. Zhang and J. T. Kwok, "Density-weighted nystrm method for computing large kernel eigensystems," *Neural Computation*, vol. 21, no. 1, pp. 121–146, 2009.

[15] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*, 2nd ed. Wiley Interscience, 2000.

[16] K. Zhang and J. T. Kwok, "Simplifying mixture models through function approximation," *IEEE Transactions on Neural Networks*, vol. 21, no. 4, pp. 644–658, Apr. 2010.

[17] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, May 2002.

[18] A. Gersho and R. M. Gray, *Vector Quantization and signal compression*. Kluwer Academic Publishers, 1992.

[19] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml

[20] P. Brodatz, *Textures: A Photographic Album for Artists and Designers*. Dover, 1966.