J. Intell. Syst. **20** (2011), 187–208
DOI 10.1515/JISYS.2011.011

Classification with NormalBoost

Erfan Davami and Hasan Fleyeh

Abstract. This paper presents a new boosting algorithm called NormalBoost which is capable of classifying a multi-dimensional binary class dataset. It adaptively combines several weak classifiers to form a strong classifier. Unlike many boosting algorithms which have high computation and memory complexities, NormalBoost is capable of classification with low complexity. Since NormalBoost assumes the dataset to be continuous, it is also noise resistant because it only deals with the means and standard deviations of each dimension. Experiments conducted to evaluate its performance shows that NormalBoost performs almost the same as AdaBoost in the classification rate. However, NormalBoost performs 189 times faster than AdaBoost and employs a very little amount of memory when a dataset of 2 million samples with 50 dimensions is invoked.

Keywords. Pattern recognition, classification, binary classifiers, boosting.

2010 Mathematics Subject Classification. 68T10.

1 Introduction

Boosting is a machine learning meta-algorithm to perform supervised learning which combines several weak learners to create a strong learner. A weak learner is a classifier which can classify samples with success rates a slightly more than randomly guessing the sample's class. A strong learner is a classifier that can classify data with rather high rates which means much better rates than what can be achieved by weak learners. It works based on a very basic principle "Pay more attention to the misclassified samples and next time try to classify them correctly". The original boosting algorithms were proposed by Schapire [1] and Freund [2]. The proposed algorithms could not take full advantage of the weak learners because they were not adaptive.

Boosting is unlimited to the algorithm constraints such as the number of training samples, the dimension of each sample, and the number of training rounds. After each round of learning, the weights (importance) of all the samples are updated based on the classification error of the weak learner of that round. The samples which are misclassified gain weight and samples which are correctly classified lose weight. Some boosting algorithms such as boost by majority [2] and BrownBoost [3] decrease the weight of misclassified samples. Thus, future weak learners

pay more attention to the samples which previous weak learners have misclassified.

The main differences between the different boosting algorithms are their hypotheses and the way the sample's weights are updated. AdaBoost [4] is a very popular boosting algorithm and perhaps the most significant; however, there are many other recent algorithms such as LPBoost [5], TotalBoost [6], BrownBoost [3], MadaBoost [7], and LogitBoost [8]. Most of these algorithms try to overcome the sensitivity of AdaBoost to noise.

Much research has been done on boosting algorithms but all of them share a number of drawbacks such as high computational cost and heavy memory consumption. To overcome these drawbacks, a new boost algorithm called NormalBoost is proposed in this paper. NormalBoost is designed in such a way that it could overcome the speed and memory complexity problems of common boost algorithms; in addition it is robust towards classifying noisy datasets.

NormalBoost assumes that there is a huge data set which has normal distribution in each of its dimensions. Using rather complex mathematics, the classification error of a weak classifier in any given point of the dimension scope can be estimated. The average weights of samples within a specific range can also be approximately determined and thus the weights could be altered using symbolic mathematics.

The reminder of the paper is organised as follows. In the next section, the relevant work is introduced. The AdaBoost is presented in Section 3. A full description of NormalBoost is given in Section 4. The experiments and results based on the proposed method are given in Section 5 and in Section 6, the conclusions are presented.

2 Relevant Work

The original Boosting methods were developed by Schapire and Freund [1, 2]. AdaBoost was the result of further improvements on the former boosting algorithms [4]. Adaptability is achieved in the way that current weak classifiers would be able to focus more on the data which is misclassified by the previous weak classifiers.

Domingo and Watanabe [7] suggested a boost algorithm called MadaBoost in which they modified AdaBoost to make it able to be used in filtering framework boosting and to be resistant to noise. LogitBoost was presented by Friedman et al. [8] which is very similar to AdaBoost except that it uses the logistic regression cost function. It can indeed be considered as a form of convex optimisation.

Freund [3] developed an adaptive version of 'Boost by Majority' which is called Brownboost. Since AdaBoost focuses on frequently misclassified data it does not

appear to work well on noisy datasets [9]. Furthermore, Brownboost stops giving priority to frequently misclassified samples and allows prioritising samples which have been correctly classified numerous times. Brownboost gives us the option to set a certain amount of tolerance for the classification error in the dataset.

Demiriz et al. [5] proposed LPBoost in which a strong classifier starts with an empty set of weak learners, a weak learner is added iteratively and the weights are updated until no weak learner remains. This property is known as the Totally-Corrective property. Boosting methods such as AdaBoost do not have this feature thus converge is slower.

SoftBoost [10] was designed to be suitable for datasets containing samples which are binary labelled and cannot necessarily be separable by convex combinations of base hypotheses. By capping the distribution of samples the algorithm gains robustness and reaches a convergence based on a logarithmic growth function.

FilterBoost [11] is based on a technique of logistic regression [12] and uses an oracle to draw samples instead of using a fixed training set which makes it efficient for large datasets. The algorithm is more resistant to overfitting and noise.

3 AdaBoost

A weak learner, which is illustrated in Figure 1, finds a separation boundary between negative and positive classes based on finding the location of the boundary *th* in the suitable dimension D and the location of the positive class with respect to the negative one $lp \in \{-1, +1\}$. Since this weak learner fails in most cases to separate the two classes, AdaBoost combines a number of weak learners to form a strong learner in order to achieve better separation between classes.

Let X be a finite set of training which is denoted by:

$$X = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathbb{R}^T, y_i \in \{+1, -1\}\}, \quad \text{for } i = 1, 2, \ldots, N, \tag{1}$$

where $\mathbf{x}_i$ is i-th training data, y_i is its corresponding target label, N is the number of training samples, and T is the space of the data set (number of features).

For any training round $k = 1, \ldots, K$, in each dimension $d = 1, \ldots, T$, a weak learner is described by four parameters $[th_d, lp_d, d, \alpha_d]$. The error generated by this weak learner in the current dimension is given by e_d as follows:

$$e_d = \frac{\sum_{i=1}^{N} w_i \cdot \text{abs}(y(i,t) - h_i(x_i))}{\sum_{i=1}^{N} w_i}. \tag{2}$$

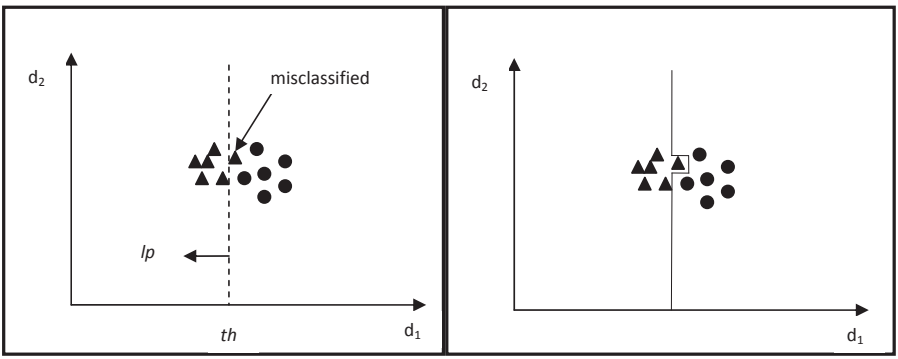


Figure 1. A weak learner (left) versus a strong learner (right).

The quality of classification in this dimension is measured by a parameter called α_d, which is a positive number in the range $[0, \infty]$ [5].

$$\alpha_d = \frac{1}{2} \cdot \log \frac{1 - e_d}{e_d} > 0. \tag{3}$$

The best weak learner among all weak learners in all dimensions is the one which has the least classification error e_k. This weak learner is described by $[th_k, lp_k, D_k, \alpha_k]$, where D_k is the dimension in which the classifier is defined

$$e_k = \min(e_d)_{d=1}^{T}. \tag{4}$$

In the beginning of the training, each sample $\mathbf{x}_i$ has a weight w_i which is initially $w_i = 1$. The weak learner classification result on sample $\mathbf{x}_i$ is given by $h_i(x_i) \in \{+1, -1\}$.

$$\mathrm{h}(\mathbf{x}_i) = \begin{cases} lp_k, & \mathbf{x}_{i,D_k} < th_k, \\ -1 \cdot lp_k, & \text{otherwise.} \end{cases} \tag{5}$$

The weight of sample $\mathbf{x}_i$ is updated by the following equations:

$$w_i^{k+1} = w_i^k \cdot e^{-\alpha_k \cdot y_i \cdot h_i(x_i)}. \tag{6}$$

Finally the strong classifier is updated by the following equation:

$$H_x = \mathrm{sign}\left(\sum_{k=1}^{K} \alpha_k \cdot h_k(x) \right). \tag{7}$$

The algorithm is based on K rounds where samples in round one are given an equal weight (importance). In order to focus more on the misclassified samples in the next rounds, the weights of misclassified samples are increased based on the minimum classification error of each round.

4 NormalBoost

NormalBoost is a type of classification algorithm which can be employed to achieve binary classification. It is applied on datasets which consist of two classes which are called positive and negative. Any dataset processed by the NormalBoost should have a *normal distribution*. Since NormalBoost assumes the dataset to be continuous, having a large number of samples is recommended to simulate the continuity of the data.

4.1 Computing the Best Weak Classifier

In order to separate the dataset into two different classes, NormalBoost finds the best linear separations in all dimensions and merges them to make a single nonlinear separator. This process starts by calculating the mean and standard deviation of the positive class $(\mu_{p,d}, \sigma_{p,d})$ and the negative class $(\mu_{n,d}, \sigma_{n,d})$ in each dimension, where $d \in \{1, \ldots, T\}$ and T represents the dimensionality of the problem.

Without loss of generality, a two dimensional dataset will be exploited for better clarity of the discussion. Figure 2 illustrates an example of two normally distributed datasets along two dimensions which fit the requirements of NormalBoost. The curves on the bottom and left of the Figure depict this distribution.

The weight representation in NormalBoost is based on a set of weight points wp and weight values wv in each dimension and class. Weight points are specific coordinates on the corresponding dimension in which the weights have specific values. Since there are positive and negative classes in each dimension, the weighs are represented by their positive and negative weight points and values $\{wp_d^+, wv_d^+\}$ and $\{wp_d^-, wv_d^-\}$, where $d \in \{1, \ldots, T\}$.

In the beginning of the boosting, the weights of the positive and negative classes in each dimension are initialised as follows:

$$wp^+ = [-\infty, \infty], \quad wv^+ = [1] \tag{8}$$

and

$$wp^- = [-\infty, \infty], \quad wv^- = [1], \tag{9}$$

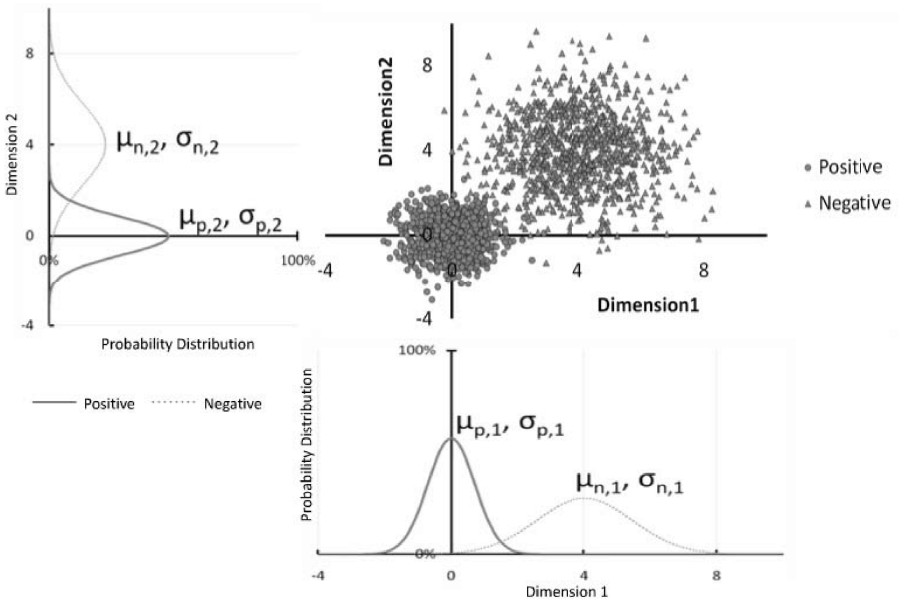


Figure 2. Probability distribution of dimensions.

which means that for each class, all the potential existing weights from $-\infty$ to $+\infty$ will have a value of 1. This means that the importance of the data in each dimension is equal.

In order to calculate the best weak learner in each dimension of the problem, it is necessary to find the location of a boundary *th* in each dimension and the location of the positive class with respect to the negative one *lp* as depicted in Figure 3. Following this discussion, the value of *lp* is specified by comparing the means of the two classes in each dimension. When $\mu_{n,d} < \mu_{p,d}$ then the positive class is located to the right with respect to the negative one, and therefore $lp = -1$, otherwise $lp = +1$. When $\mu_{n,d} = \mu_{p,d}$, then a very bad weak learner will be generated which will be rejected by the other weak learners of the other dimensions. However, by default the value of *lp* is set to $+1$.

The threshold *th* is defined as the position of a line which separates the two classes provided that this line maximises the classification rate (minimises the classification error). Figure 3 illustrates the idea of the threshold *th*.

In a certain dimension d, the classification error of the weak learner is given as follows:

$$\text{Classification Error } = \frac{\text{FPA} + \text{FNA}}{\text{APC} + \text{ANC}}, \tag{10}$$

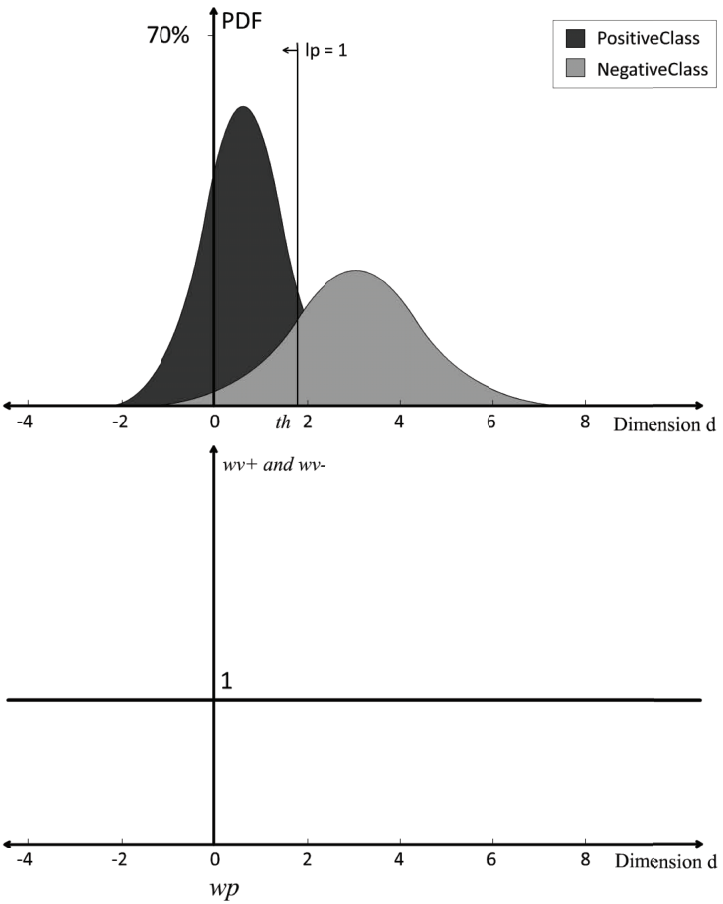


Figure 3. The threshold *th* of a weak learner and the initialisation of weights in NormalBoost.

where FPA is the area of the false positive, FNA is the area of the false negative, APC is the area under the positive curve and ANC is the area under the negative curve. The numerator of Equation 10 represents the misclassified area while the denominator represents the summation of the two areas under the two curves. Consider x as any position on any dimension as shown in Figure 4, when $lp = 1$, FPA represents the area under the negative curve between $-\infty$ and x while FNA represents the area under the positive curve between x and $+\infty$. When $lp = -1$, FPA represents the area under the negative curve between $-\infty$ and x while FNA represents the area under the positive curve between x and $+\infty$.

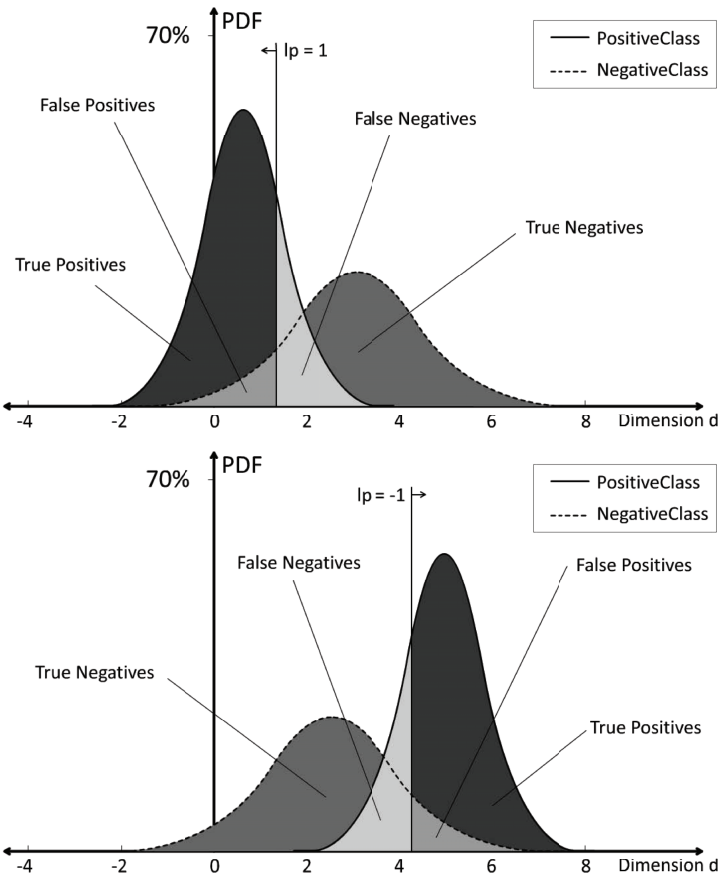


Figure 4. Error regions.

Since a Gaussian function, which is represented by Equation 11, is invoked here

$$p(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{11}$$

$$\text{APC} = \int_{-\infty}^{\infty} p(x, \mu_{p,d}, \sigma_{p,d}) dx, \tag{12}$$

$$\text{ANC} = \int_{-\infty}^{\infty} p(x, \mu_{n,d}, \sigma_{n,d}) dx. \tag{13}$$

The misclassified area is then given by

$$\text{misclassified Area}(x) = \begin{cases} \int_{-\infty}^{x} p(t, \mu_{n,d}, \sigma_{n,d})dt + \int_{x}^{\infty} p(t, \mu_{p,d}, \sigma_{p,d})dt, & lp = 1, \\ \int_{-\infty}^{x} p(t, \mu_{p,d}, \sigma_{p,d})dt + \int_{x}^{\infty} p(t, \mu_{n,d}, \sigma_{n,d})dt, & lp = -1 \end{cases} \tag{14}$$

and the misclassification error generated by choosing x as a threshold is then given by:

$$\text{Error}_d(x) = \frac{1}{\text{APC} + \text{ANC}} \cdot \text{misclassified Area}(x). \tag{15}$$

The optimum value of *th* is calculated by finding the first derivative of Equation 15 with respect to x and calculating its root. Equation 16 shows the optimum threshold *th*.

$$th = \mu_{p,d} \cdot \sigma_{n,d}^2 - \mu_{n,d} \cdot \sigma_{p,d}^2 + \sigma_{p,d} \cdot \sigma_{n,d} \tag{16}$$
$$\times \frac{\sqrt{-2\mu_{p,d} \cdot \mu_{n,d} - 2\sigma_{p,d}^2 \cdot \ln\left(\frac{\sigma_{n,d}}{\sigma_{p,d}}\right) + 2\sigma_{n,d}^2 \cdot \ln\left(\frac{\sigma_{n,d}}{\sigma_{p,d}}\right) + \mu_{p,d}^2 + \mu_{n,d}^2}}{\sigma_{n,d}^2 - \sigma_{p,d}^2}.$$

Computation of Equation 16 is only based on the presence of four parameters which are $\mu_{p,d}$, $\sigma_{p,d}$, $\mu_{n,d}$ and $\sigma_{n,d}$, thus it can be calculated very fast.

In analogy to AdaBoost, classification quality α_d can be calculated from Equation 15 as follows

$$\alpha_d = \frac{1}{2} \cdot \log \frac{1 - \text{Error}_d(x)}{\text{Error}_d(x)} > 0 \tag{17}$$

and thus the parameters of the weak learner in dimension d is described by $[th_d, lp_d, d, \alpha_d]$. The weak classifier with the highest α in all dimensions will be selected as the best weak learner in this specific round of training. This weak learner is then specified by $[th_{D,k}, lp_{D,k}, D_k, \alpha_{D,k}]$, where D is the dimension in which the best weak learner is located and $k \in \{1, \ldots, K\}$ is the specific round of training.

4.2 Updating the Weights

The weights of each dimension d of the problem in a certain round k are denoted by $\{wp_{d,k}^+, wv_{d,k}^+\}$ and $\{wp_{d,k}^-, wv_{d,k}^-\}$, where $d \in \{1, \ldots, T\}$ and $k \in \{1, \ldots, K\}$.

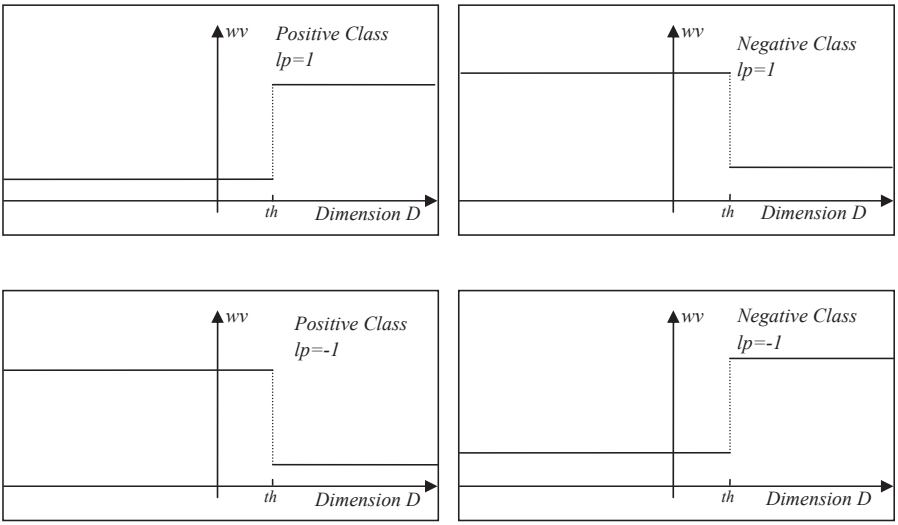


Figure 5. Updated weights of the positive and negative classes in Dimension D.

Based on Equations 8 and 9, the graphical representation of the weights of dimension d when $k = 1$ is illustrated in Figure 5. As indicated in this Figure, in between all weight points the weight values equal 1.

In a specific round k, the weights of dimension D is updated in a way that the weight of the correctly classified area is decreased and the weight of the misclassified area is increased. This is done in order to force NormalBoost to pay more attention to misclassified areas in the incoming rounds. For any position x on dimension D, the weights will be updated as follows [4]:

$$wv^{\pm}_{D,k}(x) = \begin{cases} wv^{\pm}_{D,k-1} \cdot e^{\alpha_{D,k} \cdot \pm lp_{D,k}}, & x \leq th_{D,k}, \\ wv^{\pm}_{D,k-1} \cdot e^{\alpha_{D,k} \cdot \pm lp_{D,k}}, & x > th_{D,k}. \end{cases} \tag{18}$$

For all other dimensions except D, updating the weight is based on two parameters κ^+ and κ^- which represent the fraction of the positive and negative samples, respectively, which were wrongly classified by the weak learner in dimension D.

Consider the Cumulative Distribution Function (CDF) of the normal distribution which is defined by:

$$\mathrm{CDF}(\mu, \sigma, x) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x - \mu}{\sigma\sqrt{2}}\right)\right], \tag{19}$$

where erf() is the error function of normal distribution.

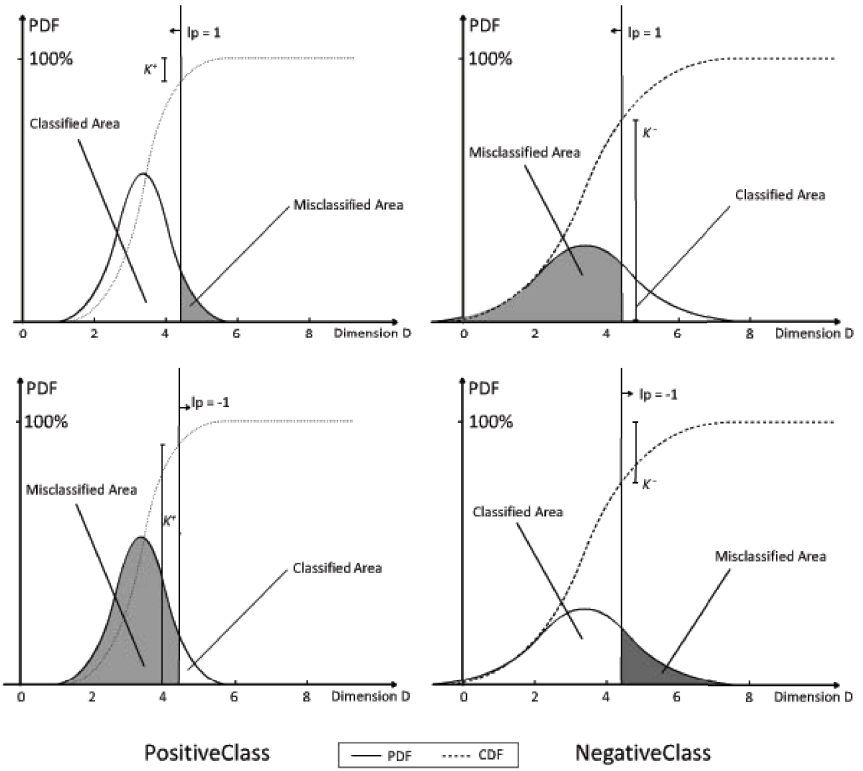


Figure 6. Calculating κ.

Equation 19 represents the area under the curve of any of the positive or negative classes in the interval $[-\infty, x]$. When a threshold *th* is specified for dimension D, this *th* divides the area under the curve into classified and misclassified areas, as shown in Figure 6. Since CDF represents the area under the curve of any of the two classes, it follows that κ is the percentage of data which is misclassified in dimension D. For the positive and negative curves, and different lp, κ has different values. Table 1 summarises the way by which κ is computed.

The values of κ will be in the range [0, 1]. For the two classes they are given as:

$$\kappa^{+} = \frac{1}{2}\left[1 - \operatorname{erf}\left(\frac{th - \mu_{p,D}}{\sigma_{p,D}\sqrt{2}}\right)\right], \tag{20}$$

$$\kappa^{-} = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{th - \mu_{n,D}}{\sigma_{n,D}\sqrt{2}}\right)\right]. \tag{21}$$

Class	***lp***	**Misclassified Area**	κ
Positive	1	Right side	$\kappa^+ = 1-$ CDF
Positive	-1	Left side	$\kappa^+ =$ CDF
Negative	1	Left side	$\kappa^- =$ CDF
Negative	-1	Right side	$\kappa^- = 1-$ CDF

Table 1. The method to calculate κ. In this Table, the value of the CDF of the positive curve is different from CDF of the negative curve.

Updating the weights of positive and negative samples of any dimension $D \neq d$ takes place by calculating the two limit points p_1 and p_2 illustrated in Figure 7 as follows

$$p_1 = \mathrm{CDF}^{-1}(\mu, \sigma, \kappa), \tag{22}$$

$$\mathrm{CDF}^{-1}(\mu, \sigma, x) = \sigma\sqrt{2} \cdot \mathrm{erf}^{-1}(2 \cdot x - 1) + \mu, \tag{23}$$

$$p_2 = \begin{cases} p_1, p_1 = \mu - |p_1 - \mu| & \text{if } p_1 > \mu, \\ \mu + |p_1 - \mu| & \text{if } p_1 \leq \mu. \end{cases} \tag{24}$$

And the weight values will be updated as follows:

$$wv^{\pm}_{d,k}(x) = \begin{cases} wv^{\pm}_{d,k-1} \cdot e^{\alpha_{d,k} \cdot -lp_{d,k}}, & x < p_1 \text{ or } x > p_2, \\ wv^{\pm}_{d,k-1} \cdot \kappa^{\pm}_{d,k} \cdot e^{\alpha_{d,k} \cdot lp_{d,k}} + (1 - \kappa^{\pm}_{d,k}) \cdot e^{\alpha_{d,k} \cdot -lp_{d,k}}, & x \geq p_1 \text{ and } x \leq p_2. \end{cases} \tag{25}$$

When the best weak learner updates the weight values of its own dimension D, it also influences the weight values of other dimensions d. This influence is the projection of the weight update in D on the other dimensions d. The result of this influence is that the distance between p_1 and p_2 is directly proportional to κ, while the amount of the weight value is inversely proportional to κ. This means that the better classification is achieved in dimension D the closer p_1 and p_2 are in dimension d. This is depicted in Figure 8.

Once the weights are updated, the importance of different areas under the positive and negative curves changes depending on the weight values as shown in Figure 9. This will affect the calculation of the next threshold value in the coming rounds. The next section describes this thoroughly.

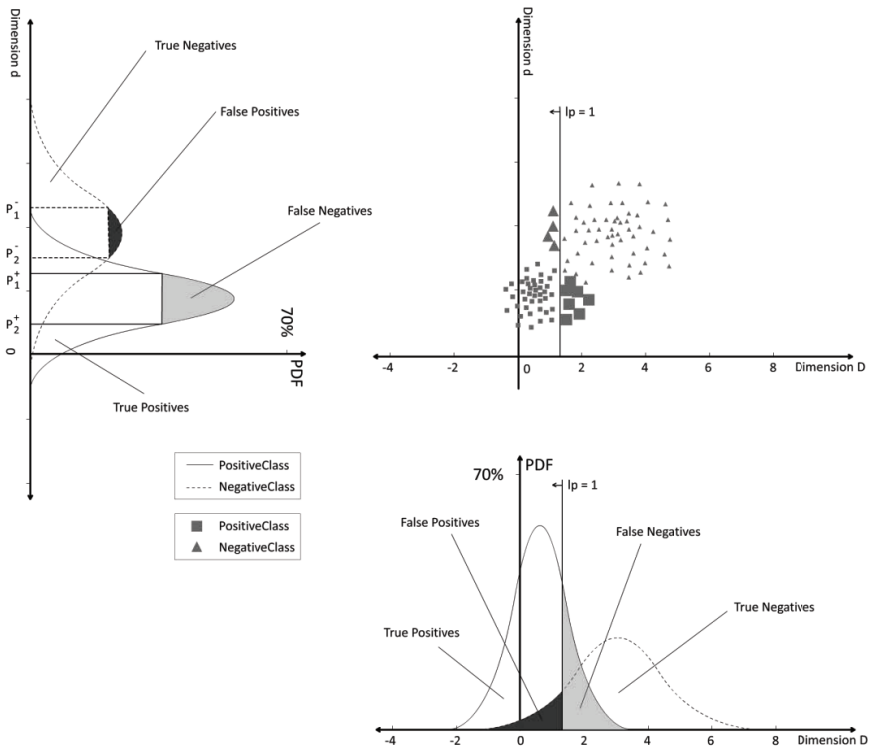


Figure 7. Calculating the limit points p_1 and p_2.

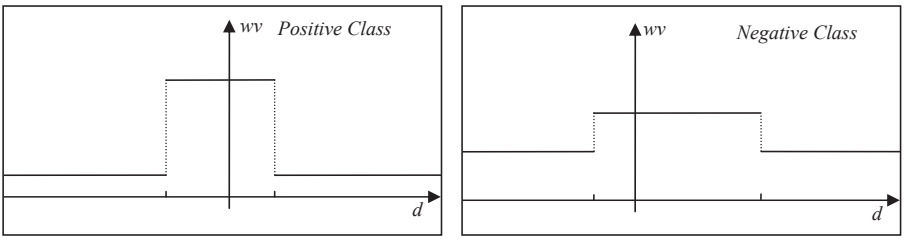


Figure 8. Updated weights of the positive and negative classes in Dimension d with low κ (left) and high κ (right).

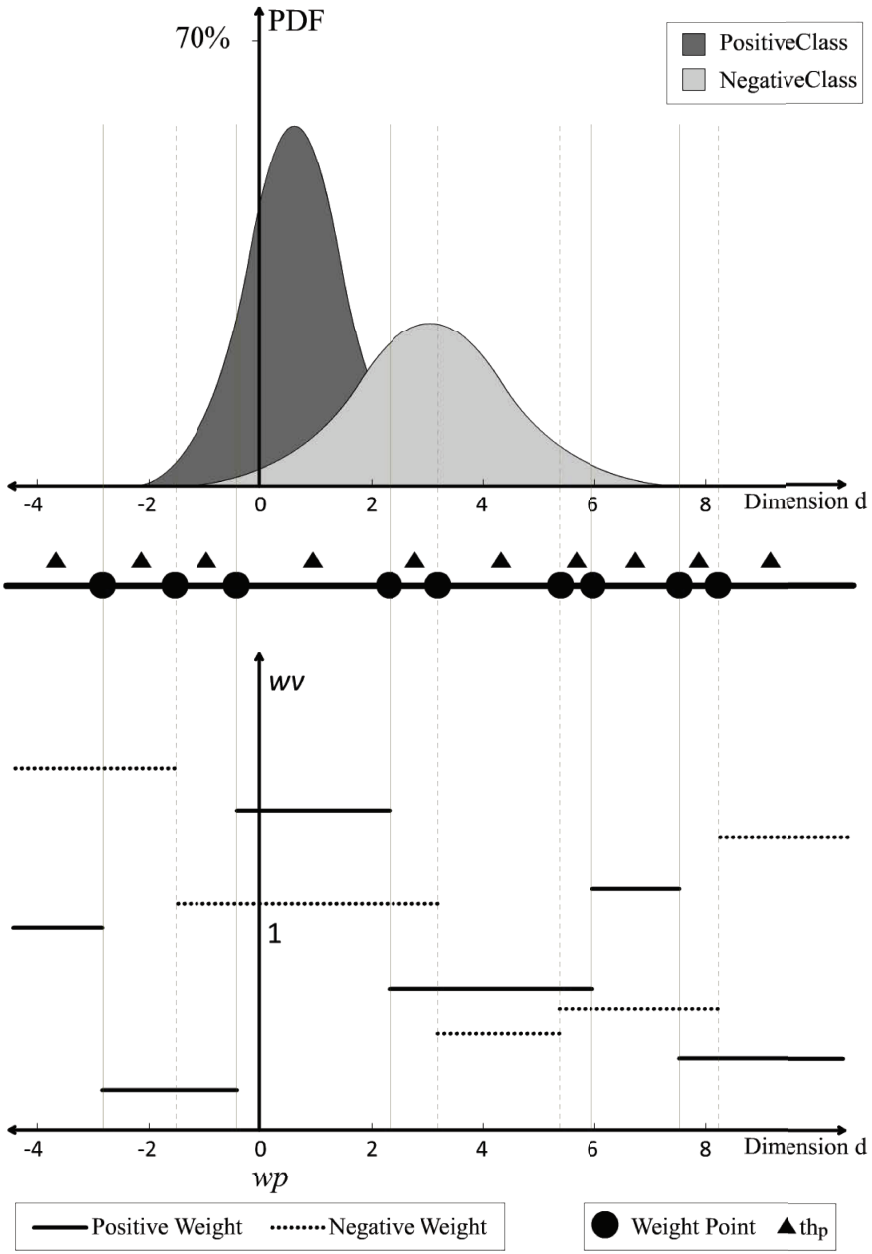


Figure 9. Weights of Positive and Negative classes after a number of rounds in dimension d.

4.3 Generalisation of Finding the Best Weak Learner

As boosting continues, the optimum threshold for the best weak learner of the corresponding dimension in the current round could be any member of the set of weight points (wp^+ and wp^-) and the set of specific points th_p in between a couple of weight points.

As the weights of the different points on the positive and negative curves change from the former round, the importance of the different regions under the curves of the two classes would become different. Therefore, the summation of weights for all classes would be the multiplication of the weights by the PDF portion in the region under consideration. Hence, the summation of the weights is given by:

$$\begin{aligned} \text{sum}_{\text{weights}} = &\sum_{i}^{C} \int_{C_i}^{C_{i+1}} wv_i^+(t) \cdot p(t, \mu^+, \sigma^+)dt \\ &+ \sum_{i}^{C} \int_{C_i}^{C_{i+1}} wv_i^-(t) \cdot p(t, \mu^-, \sigma^-)dt. \end{aligned} \tag{26}$$

The summation of the misclassified weights in any specific point x in dimension d is calculated in the same manner described above as follows:

$$\text{missed}_{\text{weights}}(x) = \begin{cases} \sum_i^{C \le x} \int_{C_i}^{C_{i+1}} wv_i^-(t) \cdot p(t, \mu^-, \sigma^-)dt \\ \quad + \int_{C \le x}^{x} wv_i^-(t) \cdot p(t, \mu^-, \sigma^-)dt \\ \quad + \sum_i^{C > x} \int_{C_i}^{C_{i+1}} wv_i^+(t) \cdot p(t, \mu^+, \sigma^+)dt, \quad lp = 1 \\ \sum_i^{C \le x} \int_{C_i}^{C_{i+1}} wv_i^+(t) \cdot p(t, \mu^+, \sigma^+)dt \\ \quad + \int_{C \le x}^{x} wv_i^+(t) \cdot p(t, \mu^+, \sigma^+)dt \\ \quad + \sum_i^{C > x} \int_{C_i}^{C_{i+1}} wv_i^-(t) \cdot p(t, \mu^-, \sigma^-)dt, \quad lp = -1. \end{cases} \tag{27}$$

Therefore, the error in position x of dimension d is calculated as follows:

$$\text{Error}(x) = \frac{1}{\text{sum}_{\text{weights}}} \cdot \text{missed}_{\text{weights}}(x). \tag{28}$$

Thus, by deriving Equation 28 and minimising the error, th_p will be calculated as follows:

$$\text{Error}'(x) = \frac{d}{dx}\text{missed}_{\text{weights}}(x) \rightarrow \text{Error}'(th_p) = 0, \tag{29}$$

$$th_p = \frac{\mu_{p,d} \cdot \sigma_{n,d}^2 - \mu_{n,d} \cdot \sigma_{p,d}^2 \pm \sigma_{p,d} \cdot \sigma_{n,d} \cdot \Delta}{\sigma_{n,d}^2 - \sigma_{p,d}^2}. \tag{30}$$

Where Δ is given by the following equation:

$$\Delta = \sqrt{\begin{array}{l} -2\mu_{p,d} \cdot \mu_{n,d} - 2\sigma_{p,d}^2 \cdot \ln\left(\frac{\sigma_{n,d}}{\sigma_{p,d}}\right) + 2\sigma_{n,d}^2 \cdot \ln\left(\frac{\sigma_{n,d}}{\sigma_{p,d}}\right) \\ +2\sigma_{p,d}^2 \cdot \ln\left(\frac{wv_{n,d}}{wv_{p,d}}\right) - 2\sigma_{n,d}^2 \cdot \ln\left(\frac{wv_{n,d}}{wv_{p,d}}\right) + \mu_{p,d}^2 + \mu_{n,d}^2 \end{array}}. \quad (31)$$

If the value of th_p produced by Equation 30 is a complex number, then this th_p will be rejected because the separator of two real datasets should be a real number. Furthermore, if the location of th_p is outside the range of the two weight points used to produce the th_p then this th_p is also rejected. The remaining candidates in the list are the weight points and the th_p which pass the two rejection criteria. To choose the optimum threshold among these candidates, Equation 28 is applied on all of these candidates and the optimum threshold is the one which produces the minimum error.

The four parameters specifying the new weak learner are specified as follows:

- The new $th_{d,k}$ will be used to specify the threshold of the new weak learner.
- $lp_{d,k}$ will remain the same as before.
- The classification error is calculated by applying Equation 28 to the optimum threshold. Thus, $\alpha_{d,k}$ is calculated by applying this classification error to Equation 17.
- After all the weak learners of all dimensions are specified, D would be the dimension of the weak learner with the maximum $\alpha_{d,k}$.

Therefore, this weak learner is then specified by $[th_{D,k}, lp_{D,k}, D_k, \alpha_{D,k}]$.

5 Experiments and Discussions

NormalBoost was tested using a dataset consisting of 4000 gray images of human faces which were normalised to 20×20 pixels. These images were exploited as the positive class. Images in the negative class were created from 3019 gray images of size 480×640 pixels of different scenes which do not contain any human faces. Each image was scanned to create blocks of different sizes in different locations in the image. The minimum size of the blocks was 20×20 pixels while the maximum size was 480×80 pixels. This process produced 5 million gray images which were normalised to 20×20 pixels. Following the process described by Viola and Jones [13], all normalised images in both positive and negative classes were converted into integral images and the Haar-like features are then computed. A total of 477 650 Haar-like features were calculated for each integral image.

No. of Dimensions (T)	No. of samples (N)	NormalBoost		AdaBoost	
		FPRate	TPRate	FPRate	TPRate
2	2×10^3	0.285	0.976	0.124	0.903
2	2×10^4	0.011	0.955	0.008	0.957
10	2×10^5	0.005	0.935	0.020	0.9707
20	2×10^5	0.127	0.991	0.016	0.981
50	2×10^5	0.027	0.993	0	1
50	2×10^6	0.002	1	0	1

Table 2. False Positive and True Positive rates of the strong classifiers generated by NormalBoost and AdaBoost in different testing datasets.

A total of six training datasets were created using the method described above. For testing, another six datasets were created in which the data points on each dimension of each class of one dataset are created using a normal distribution based on the mean and standard deviation of the corresponding dimension and class of the training dataset.

NormalBoost was trained and tested for 25 rounds using six different datasets. Each dataset consisted of different dimensions and number of samples as shown in Table 2. Figure 10 illustrates the classification rates of the six testing datasets.

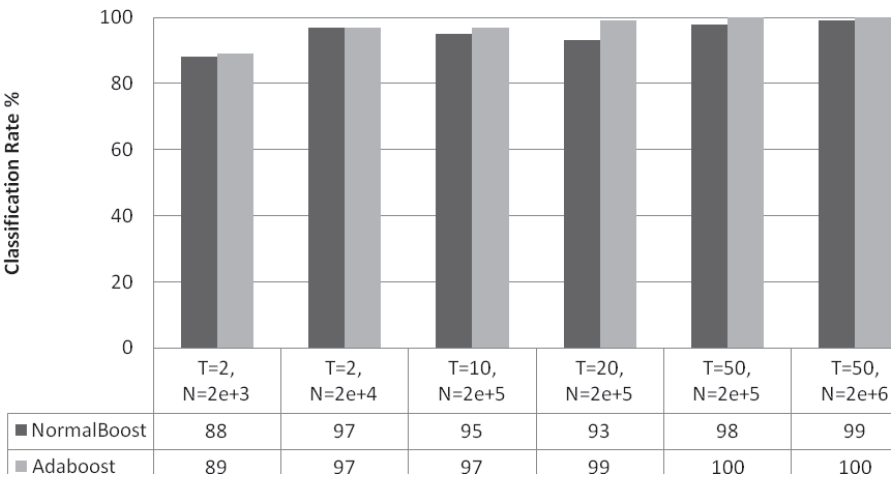
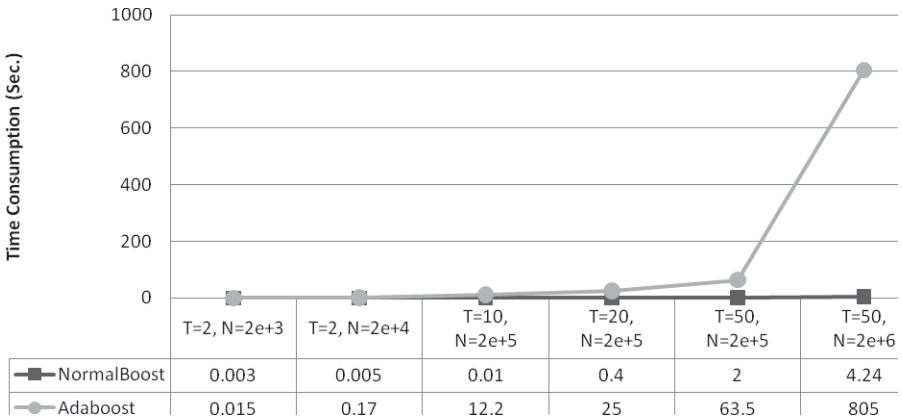


Figure 10. Classification rates of the strong classifier in each round of NormalBoost and AdaBoost.



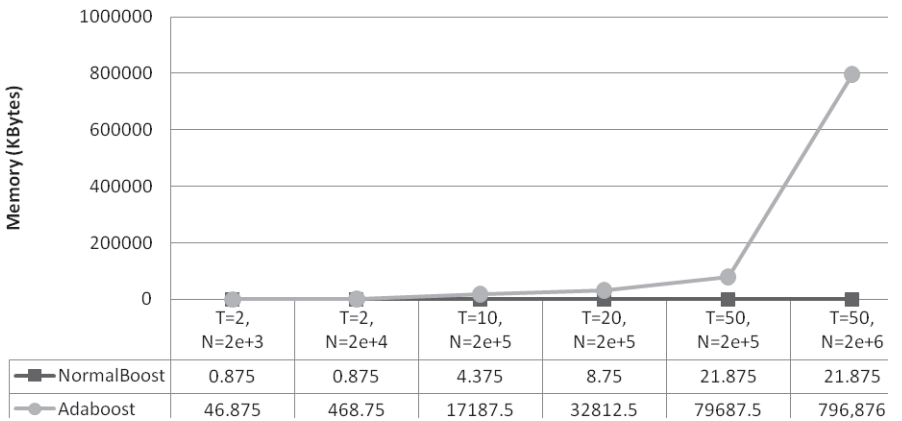


Figure 11. Computational time (top) and memory consumption (bottom) of training the strong classifier in each round of AdaBoost and NormalBoost.

For the purpose of comparison, AdaBoost was trained and tested using exactly the same datasets and experiment parameters. The performance of NormalBoost is almost identical to that of AdaBoost. A small amount of performance difference can be noticed between the two algorithms.

Figure 11 represents the time and memory consumption for training a strong classifier from the training datasets using both AdaBoost and NormalBoost. The time consumption of AdaBoost increases rapidly when the dimensionality of the problem increases while NormalBoost is almost constant in comparison to AdaBoost. The same behaviour applies to memory consumption. Figure 12 illustrates

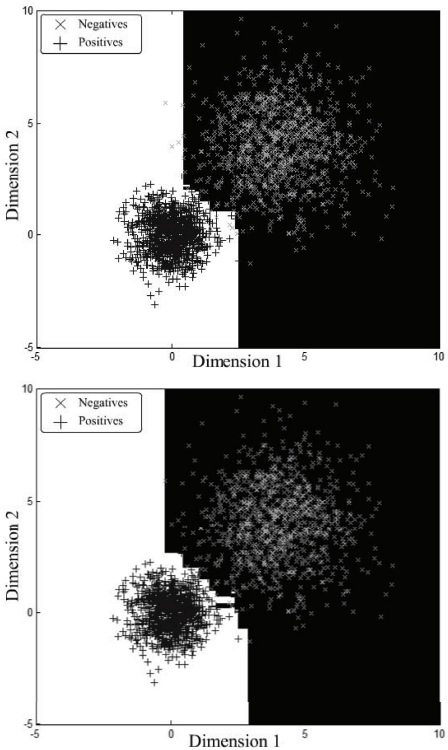


Figure 12. Visualisation of classification using the trained strong learner of NormalBoost (top) and AdaBoost (bottom) for $T = 2$ and $N = 2000$.

the visualisation of the strong classifier produced by the NormalBoost compared with AdaBoost.

The processing complexity of AdaBoost is $\mathrm{O}(K \cdot T \cdot N \cdot \log N)$ because in K rounds, data in T dimensions need to be processed by sorting and searching for the best weak learner. On the other hand, memory complexity is $\mathrm{O}(T \cdot N)$ because all the data which consists of N samples with T dimensions each should be resident in the main memory. Unlike AdaBoost, NormalBoost requires an initial pre-processing on data to extract the means and standard deviations of each of the two classes in each of their dimensions thus the complexity of the pre-processing phase would be $\mathrm{O}(T \cdot N)$.

NormalBoost initialises a total of $2 \cdot K$ weight points and weight values for each class in all T dimensions and then pre-processed data in K rounds and on T dimensions and a maximum of $4 \cdot K$ candidate points per dimension will be trained. After each round or training session is finished the weights would be updated in

	AdaBoost	NormalBoost
Processing Complexity	$O(K \cdot T \cdot N \cdot \log N)$	$O(T(N + K^2))$
Memory Complexity	$O(T \cdot N)$	$O(K \cdot T)$

Table 3. Complexity comparison of AdaBoost and NormalBoost.

all T dimensions thus the computational complexity of NormalBoost would be $O(T(N + K^2))$. In any given round the means and standard deviations of every T dimension and also all the weight points and values would be present in the main memory thus the memory complexity of NormalBoost would be $O(K \cdot T)$. Table 3 shows a comparison of the complexity between AdaBoost and NormalBoost.

Figure 13 shows the ROC diagram of the NormalBoost for the experiment in which a dataset of 10 dimensions and 2×10^5 samples are used. The strong classifier is trained on the training set while the threshold of the first weak learner is

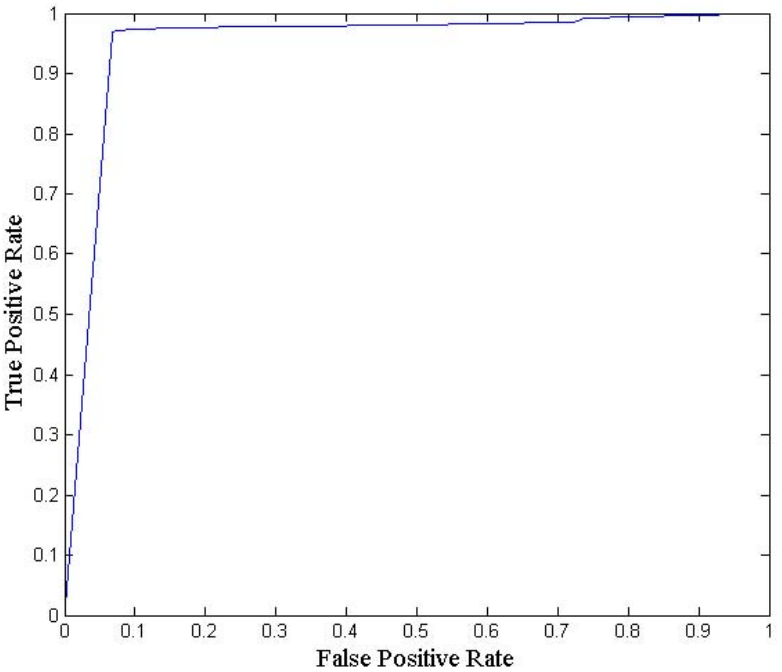


Figure 13. The ROC diagrams of NormalBoost.

gradually increased from $-\infty$ to $+\infty$. The resulting classifier is used to classify the test dataset and the values of the false positive rates and the true positive rates are computed for every threshold value.

6 Conclusions

NormalBoost, which is presented in this paper, is a novel binary classifier which adaptively combines several weak classifiers to form a strong classifier. One of the major problems of boosting is its high computational and memory complexity and NormalBoost overcomes this problem. It is noise resistant because it only deals with the means and standard deviations of the data within a specific dataset and not the discrete values of training data. However, a small amount of accuracy drop (usually less than 5%) is expected. NormalBoost has a linear computation growth compared to the logarithmic computation growth of AdaBoost thus is much less complex in terms of computation and memory complexity.

NormalBoost can be used to classify any dataset with a normal distribution among its dimensions thus this makes it applicable in rapid object detection, voice and fingerprint recognition, and statistical analysis of various datasets.

Bibliography

[1] R. Schapire, Strength of Weak Learnability. Journal of Machine Learning 5 (1990), 197–227.

[2] Y. Freund, Boosting a weak learning algorithm by majority, in, The Third Annual Workshop on Computational Learning Theory (COLT '90), 1990.

[3] Y. Freund, An adaptive version of the boost by majority algorithm, in, 14th Annual Conference on Computational Learning Theory Amsterdam, The Netherlands, 2001, pp. 102–113.

[4] Y. Freund, R. Schapire, A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting. Journal of Computer and System Sciences 55 (1997), 119–139.

[5] A. Demiriz, K. Bennett, J. Shawe-Taylor, Linear Programming Boosting via Column Generation. Machine Learning 46 (2002), 225–254.

[6] M. Warmuth, J. Liao, G. Rätsch, Totally Corrective Boosting Algorithms that Maximize the Margin, in, The 23rd International Conference on Machine Learning, Pittsburg, USA, 2006, pp. 1001–1008.

[7] C. Domingo, O. Watanabe, MadaBoost: A Modification of AdaBoost, in, 13th Annual Conference on Computational Learning Theory, Palo Alto, USA, 2000, pp. 180–189.

[8] J. Friedman, T. Hastie, R. Tibshirani, Additive logistic regression: a statistical view of boosting. Annuals of Statistics 28 (2000), 337–407.

[9] T. Dietterich, An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. Machine Learning 40 (2000), 139–158.

[10] M. Warmuth, K. Glocer, G. Raetsch, Boosting Algorithms for Maximizing the Soft Margin, in, The 21st Annual Conference on Neural Information Processing Systems, Vancouver, B.C., Canada, 2007.

[11] J. Bradley, R. Schapire, FilterBoost: Regression and classification on large datasets, in, Advances in Neural Information Processing Systems 20, Cambridge, MA, USA, 2008.

[12] M. Collins, R. Schapire, Y. Singer, Logistic regression, AdaBoost and Bregman distances. Machine Learning 48 (2002), 253–285.

[13] P. Viola, M. Jones, Rapid object detection using a boosted cascade of simple features, in, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Kauia, HI, USA, 2001, pp. 511–518.

Received May 12, 2011.

Author information

Erfan Davami, Computer Science Department, School for Technology and Business Studies, Dalarna University, Rödavägen 3, 78188 Borlänge, Sweden.
E-mail: v09erfda@du.se

Hasan Fleyeh, Computer Science Department, School for Technology and Business Studies, Dalarna University, Rödavägen 3, 78188 Borlänge, Sweden.
E-mail: hfl@du.se