

# Survey on Boosting Algorithms for Supervised and Semi-supervised Learning

Artur Ferreira  
Instituto de Telecomunicações

12 October 2007

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Ensemble of classifiers and boosting</b>	<b>1</b>
<b>3</b>	<b>The origins of Boosting and Adaptive Boosting</b>	<b>1</b>
3.1	Bootstrap . . . . .	2
3.2	Bagging . . . . .	2
3.3	Boosting . . . . .	3
<b>4</b>	<b>The AdaBoost algorithm and its variants</b>	<b>4</b>
4.1	The AdaBoost Algorithm . . . . .	4
4.2	Variants for supervised learning . . . . .	5
4.2.1	Real AdaBoost . . . . .	5
4.2.2	Gentle AdaBoost . . . . .	6
4.2.3	Modest AdaBoost . . . . .	7
4.2.4	LogitBoost . . . . .	8
4.2.5	FloatBoost . . . . .	8
4.2.6	Emphasis Boost . . . . .	9
4.2.7	Reweight Boost . . . . .	10
4.2.8	AnyBoost and MarginBoost . . . . .	10
4.2.9	List of variants . . . . .	11
4.3	Variants for semi-supervised learning . . . . .	13
4.3.1	MixtBoost . . . . .	13
4.3.2	SSMarginBoost . . . . .	13
4.3.3	SemiBoost . . . . .	13
<b>5</b>	<b>Experimental results and discussion</b>	<b>13</b>
5.1	Successful applications of boosting algorithms . . . . .	13
5.2	Discussion on the performance of boosting . . . . .	14

# 1 Introduction

This survey presents an overview of boosting algorithms to build ensembles of classifiers. The basic boosting technique and its variants are addressed and compared for supervised learning. The extension of these techniques for semi-supervised learning is also addressed. Section 2 presents the concept of ensemble of classifiers and the general idea of boosting. Section 3 addresses the basic boosting technique, and its genesis on bootstrap and bagging techniques. Section 4 presents the basic adaptive boosting algorithm and its variants for supervised and semi-supervised learning. On section 5, some experimental results on standard datasets are presented. Finally, section 6 closes the document with a generic appreciation of these techniques and some intentions of future work and developments.

## 2 Ensemble of classifiers and boosting

The idea of building ensembles of classifiers has gained interest in the last decade. Instead of building a single complex classifier, one aims at designing a combination of several simple (weak) classifiers. For instance, instead of training a large neural network (NN), we train several simpler NN and combine their individual output in order to produce the final output. This allows us to have faster training and to focus each NN on a given portion of the training set. Figure 1 illustrates the concept of ensemble of classifiers. The input pattern  $x$  is classified by each weak learner. The outputs of these weak learners are then combined in order to establish the final classification decision. Assuming that the individual classifiers are uncorrelated, majority voting of an ensemble of classifiers should lead to better results than using one individual classifier.

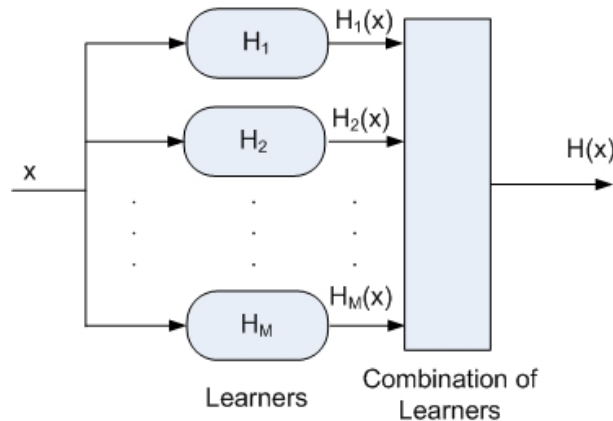


Figure 1: The concept of ensemble of classifiers. The outputs of the weak learners (classifiers) are combined to produce the output of the ensemble of classifiers.

Considering a linear combination of the outputs of the weak classifiers, the output of the ensemble and the final decision of classification is given by

$$H(x) = \text{sign} \left( \sum_{m=1}^M \alpha_m H_m(x) \right), \quad (1)$$

in which  $\alpha_m$  is the weight of each weak classifier  $H_m$ . Although ensembles of classifiers can be learned in several ways, boosting techniques are well suited for this purpose [14]. Boosting consists in building sequentially a linear combination of base classifiers that focus on difficult examples. It has been shown that ensembles of classifiers perform quite well, as long as each classifier is just a little better than random guessing [14].

## 3 The origins of Boosting and Adaptive Boosting

This section describes the Bootstrap and Bagging techniques and how they are related with the primary idea of (non-adaptive) boosting. These statistical techniques have been around for quite some time.

### 3.1 Bootstrap

The Bootstrap procedure [4, 3] is a general purpose sample-based statistical method which consists of drawing randomly with replacement from a set of data points. It has the purpose of assessing the statistical accuracy of some estimate, say  $S(Z)$ , over a training set  $Z = \{z_1, z_2, \dots, z_N\}$ , with  $z_i = (x_i, y_i)$ . To check the accuracy, the measure is applied over the  $B$  sampled versions of the training set. The Bootstrap algorithm is as follows:

---



---

<b>Bootstrap Algorithm</b>	
Input:	$Z = \{z_1, z_2, \dots, z_N\}$ , with $z_i = (x_i, y_i)$ as training set. B, number of sampled versions of the training set.
Output:	$S(Z)$ , statistical estimate and its accuracy.

---

1. for n=1 to B
  - a) Draw, with replacement,  $L \leq N$  samples from the training set  $Z$ , obtaining the  $n^{th}$  sample  $Z^{*n}$ .
  - b) For each sample  $Z^{*n}$ , estimate a statistic  $S(Z^{*n})$ .
2. Produce the bootstrap estimate  $S(Z)$ , using  $S(Z^{*n})$  with  $n = \{1, \dots, B\}$ .
3. Compute the accuracy of the estimate, using the variance or some other criterion.

---



---

Figure 2 shows the graphical idea of the Bootstrap algorithm. We start with the training set  $Z$ , obtaining several versions  $Z^{*n}$  (bootstrap samples). For each sampled version, we compute the desired statistical measure  $S(Z^{*n})$ .

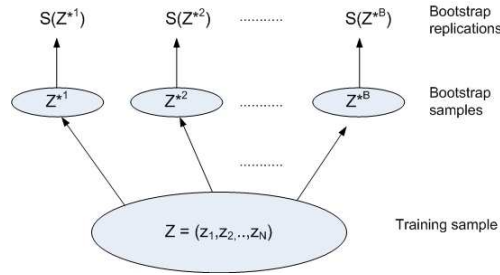


Figure 2: The Bootstrap procedure (adapted from [14]).

### 3.2 Bagging

The Bagging technique [2, 14, 3] consists of Bootstrap aggregation. Let us consider a training set  $Z = \{z_1, z_2, \dots, z_N\}$ , with  $z_i = (x_i, y_i)$  for which we intend to fit a regression model, obtaining a prediction  $\hat{f}(x)$  at input  $x$ . Bagging averages this prediction over a collection of bootstrap samples, thereby reducing its variance. For each bootstrap sample  $Z^{*b}$  with  $b \in \{1, \dots, B\}$ , we fit our model, giving prediction  $\hat{f}^{*b}(x)$ . The Bagging estimate is defined by

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(x). \quad (2)$$

For classification purposes, the Bagging algorithm is as follows.

---



---

<b>Bagging Algorithm for Classification</b>	
Input:	$Z = \{z_1, z_2, \dots, z_N\}$ , with $z_i = (x_i, y_i)$ as training set. B, number of sampled versions of the training set.
Output:	$H(x)$ , a classifier suited for the training set.

---

1. for n=1 to B
  - a) Draw, with replacement,  $L \leq N$  samples from the training set  $Z$ , obtaining the  $n^{th}$  sample  $Z^{*n}$ .
  - b) For each sample  $Z^{*n}$ , learn classifier  $H_n$ .

2. Produce the final classifier as a vote of  $H_n$  with  $n = \{1, \dots, B\}$

$$H(x) = \text{sign} \left( \sum_{n=1}^B H_n(x) \right).$$

As compared to the process of learning a classifier in a conventional way, that is, strictly from the training set, the Bagging approach increases classifier stability and reduces variance.

### 3.3 Boosting

The Boosting procedure is similar to Bootstrap and Bagging. The first was proposed in 1989 by Schapire and is as follows.

#### Boosting Algorithm for Classification

Input:  $Z = \{z_1, z_2, \dots, z_N\}$ , with  $z_i = (x_i, y_i)$  as training set.

Output:  $H(x)$ , a classifier suited for the training set.

1. Randomly select, without replacement,  $L_1 < N$  samples from  $Z$  to obtain  $Z_1$ ; train weak learner  $H_1$  on it.
2. Select  $L_2 < N$  samples from  $Z$  with half of the samples misclassified by  $H_1$  to obtain  $Z_2$ ; train weak learner  $H_2$  on it.
3. Select all samples from  $Z$  that  $H_1$  and  $H_2$  disagree on; train weak learner  $H_3$ .
4. Produce final classifier as a vote of weak learners  $H(x) = \text{sign} \left( \sum_{n=1}^3 H_n(x) \right)$ .

Figure 3 shows the connection between bootstrap, bagging and boosting. This diagram emphasizes the fact that these three techniques are built on random sampling, being that bootstrapping and bagging perform sampling with replacement while boosting does not. The bagging and boosting techniques have in common the fact that they both use a majority vote in order to perform the final decision.

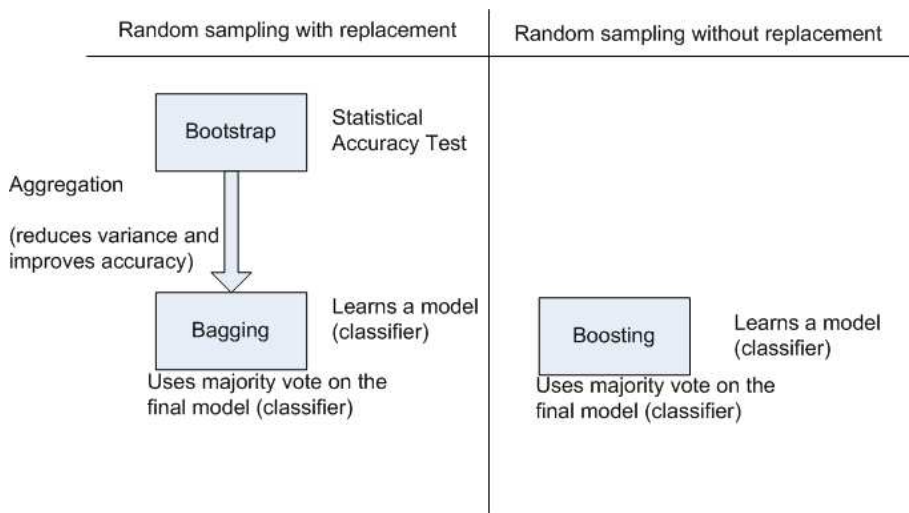


Figure 3: Connection between bootstrap, bagging and boosting.

## 4 The AdaBoost algorithm and its variants

The same researchers that proposed the Boosting algorithm, Freund and Schapire, also proposed in 1996, the AdaBoost (Adaptive Boosting) algorithm [7]. The idea behind adaptive boosting is to weight the data instead of (randomly) sampling it and discarding it. In recent years, ensemble techniques, namely boosting algorithms, have been a focus of research [1, 7, 8, 11]. The AdaBoost algorithm [7, 14] is a well-known method to build ensembles of classifiers with very good performance [14]. It has been shown empirically that AdaBoost with decision trees has excellent performance [1], being considered the best off-the-shelf classification algorithm [14].

### 4.1 The AdaBoost Algorithm

The AdaBoost algorithm, proposed in [7], learns a combination of the output of  $M$  (weak) classifiers  $H_m(x)$  in order to produce the final decision of classification given by

$$H(x) = \text{sign} \left( \sum_{m=1}^M \alpha_m H_m(x) \right), \quad (3)$$

where  $\alpha_m$  is the weight (contribution) of each classifier. The weak classifiers are trained sequentially. The weight distribution of the training set patterns is updated between iterations according to the accuracy of classification, of the previous classifiers. The weight of the misclassified patterns is increased for the next iteration, whereas the weight of the correctly classified patterns is decreased. The next classifier is trained with a re-weighted distribution. The amount of change on the weight of each pattern is proportional to the classification error of the pattern.

On figure 4 we depict the idea of adaptive boosting. The training set is always the same in each iteration and each input pattern is given a weight according to its (mis)classification by the previous classifiers. On subsequent iterations, the weight of the misclassified patterns by the ensemble is increased, whereas the weight of the correctly classified patterns decreases. This allows for the current classifier to be focused on the most difficult patterns, that is, the ones that were not well classified by the previous classifiers.

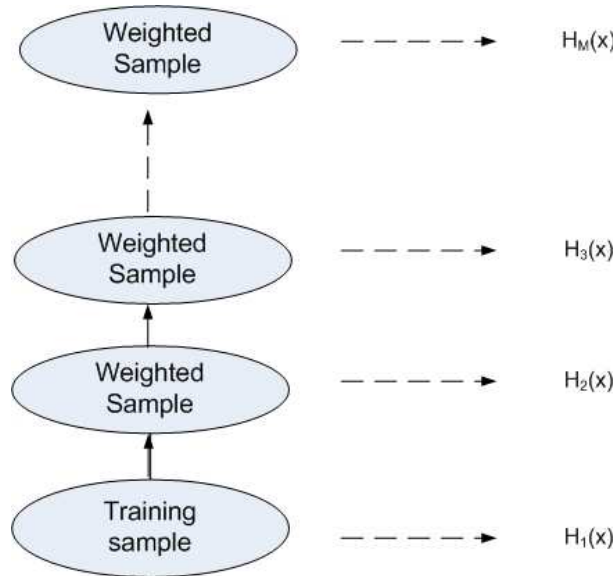


Figure 4: Graphical idea of the adaptive boosting algorithm (adapted from [14]).

Table 1 shows the connection between the boosting algorithm and its adaptive version (AdaBoost). We compare these algorithms in terms of how the training data is processed, the number of classifiers and how the final decision is produced.

Considering that we have  $M$  weak classifiers and  $N$  patterns ( $x_i$ ) and its corresponding class label  $y_i$  on the training set, the discrete AdaBoost algorithm is as follows.

Feature	Boosting	Adaptive Boosting
Data processing	Random sampling without replacement	Weighting (no sampling)
Num. of classifiers	Three	Up to M
Decision	Majority vote	Weighted vote

Table 1: Comparison of Boosting and AdaBoost algorithms.

---



---

### Discrete AdaBoost Algorithm for Classification

---

Input:  $Z = \{z_1, z_2, \dots, z_N\}$ , with  $z_i = (x_i, y_i)$  as training set.

$M$ , the maximum number of classifiers.

Output:  $H(x)$ , a classifier suited for the training set.

---

1. Initialize the weights  $w_i = 1/N$ ,  $i \in \{1, \dots, N\}$ .
  2. For  $m=1$  to  $M$ 
    - a) Fit a classifier  $H_m(x)$  to the training data using weights  $w_i$ .
    - b) Let  $\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq H_m(x_i))}{\sum_{i=1}^N w_i}$ .
    - c) Compute  $\alpha_m = 0.5 \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$ .
    - d) Set  $w_i \leftarrow w_i \exp(-\alpha_m I(y_i \neq H_m(x_i)))$  and renormalize to  $\sum_i w_i = 1$ .
  3. Output  $H(x) = \text{sign}\left(\sum_{m=1}^M \alpha_m H_m(x)\right)$ .
- 
- 

The function  $I(c)$  used in steps 2.b and 2.d is an indicator function  $I(c) = 1$ , if  $c = \text{“true”}$  and  $I(c) = 0$  if  $c = \text{“false”}$ . It’s worth mentioning that step 2.c can also be performed with  $\alpha_m = 0.5 \log((1 + \text{err}_m)/(1 - \text{err}_m))$ . The algorithm stops when  $m = M$  or if  $\text{err}_m > 0.5$ ; this last condition means that it is impossible to build a better ensemble using these weak classifiers, regardless of the increase of their number. This way,  $M$  represents the maximum number of classifiers to accommodate in the learning ensemble.

The adaptive boosting techniques can be considered as a greedy optimization method for minimizing exponential error function

$$\sum_{i=1}^N \exp(-y_i H(x_i)), \quad (4)$$

with  $H$  being the ensemble of classifiers.

## 4.2 Variants for supervised learning

After the original AdaBoost was introduced, several modified versions (variants) have been proposed, developed and compared with AdaBoost. This section addresses some of these variants for supervised learning as shown in figure 5.

### 4.2.1 Real AdaBoost

The first variant to consider is Real AdaBoost [25], which optimizes the cost function

$$E[\exp(-y(H(x) + H_m(x)))], \quad (5)$$

with respect to  $H_m(x)$ . The term “Real” refers to the fact that the classifiers produce a real value. This real value is the probability that a given input pattern belongs to a class, considering the current weight distribution for the training set. This way, the Real AdaBoost algorithm can be seen as a generalization of Discrete AdaBoost.

---



---

### Real AdaBoost Algorithm for Classification

---

Input:  $Z = \{z_1, z_2, \dots, z_N\}$ , with  $z_i = (x_i, y_i)$  as training set.

$M$ , the maximum number of classifiers.

Output:  $H(x)$ , a classifier suited for the training set.

---

Boosting (1989)	Real AdaBoost (1999)
AdaBoost (1996)	Margin Boost (2000)
	Modest AdaBoost (2000)
	Gentle AdaBoost (2000)
	AnyBoost (2000)
	MarginBoost (2000)
	KLBoost (2003)
	FloatBoost (2004)
	ActiveBoost (2004)
	JensenShannon Boost (2005)
	Infomax Boost (2005)
	Emphasis Boost (2006)
	Entropy Boost (2007)
	Reweight Boost (2007)
	LogitBoost (????)
	Brown Boost (????)
	Weight Boost (????)

Figure 5: AdaBoost and its variants for supervised learning (two-class case).

1. Initialize the weights  $w_i = 1/N$ ,  $i \in \{1, \dots, N\}$ .
2. For  $m=1$  to  $M$ 
  - a) Fit the class probability estimate  $p_m(x) = \hat{P}_w(y = 1|x) \in [0, 1]$ , using weights  $w_i$  on the training data.
  - b) Set  $H_m = \frac{1}{2} \log \left( \frac{1-p_m(x)}{p_m(x)} \right) \in \mathcal{R}$ .
  - c) Set  $w_i \leftarrow w_i \exp(-y_i H_m(x_i))$  and renormalize to  $\sum_i w_i = 1$ .
3. Output  $H(x) = \text{sign} \left( \sum_{m=1}^M H_m(x) \right)$ .

---

Comparing this algorithm with Discrete AdaBoost, we see that the most important differences are in steps 2a) and 2b). On the Real AdaBoost algorithm, these steps consist on the calculation of the probability that a given pattern belongs to a class. The AdaBoost algorithm classifies the input patterns and calculates the weighted amount of error.

#### 4.2.2 Gentle AdaBoost

The Real AdaBoost algorithm performs exact optimization with respect to  $H_m$ . The Gentle AdaBoost [10] algorithm improves it, using Newton stepping, providing a more reliable and stable ensemble. Instead of fitting a class probability estimate, the Gentle AdaBoost algorithm uses weighted least-squares regression to minimize the function

$$E[\exp(-yH(x))]. \quad (6)$$

---

---

**Gentle AdaBoost Algorithm for Classification**

---

Input:  $Z = \{z_1, z_2, \dots, z_N\}$ , with  $z_i = (x_i, y_i)$  as training set.  
 $M$ , the maximum number of classifiers.  
Output:  $H(x)$ , a classifier suited for the training set.

---

1. Initialize the weights  $w_i = 1/N$ ,  $i \in \{1, \dots, N\}$ .
  2. For  $m=1$  to  $M$ 
    - a) Train  $H_m(x)$  by weighted least-squares of  $y_i$  to  $x_i$ , with weights  $w_i$ .
    - b) Update  $H(x) = H(x) + H_m(x)$ .
    - c) Update  $w_i \leftarrow w_i \exp(-y_i H_m(x_i))$  and renormalize to  $\sum_i w_i = 1$ .
  3. Output  $H(x) = \text{sign}\left(\sum_{m=1}^M H_m(x)\right)$ .
- 

Gentle AdaBoost differs from Real AdaBoost in steps 2a) and 2b). Gentle is because....

### 4.2.3 Modest AdaBoost

The Modest AdaBoost algorithm [26] is known to have less generalization error and higher training error, as compared to the previous variants.

---

---

**Modest AdaBoost AdaBoost Algorithm for Classification**

---

Input:  $Z = \{z_1, z_2, \dots, z_N\}$ , with  $z_i = (x_i, y_i)$  as training set.  
 $M$ , the maximum number of classifiers.  
Output:  $H(x)$ , a classifier suited for the training set.

---

1. Initialize the weights  $w_i = 1/N$ ,  $i \in \{1, \dots, N\}$ .
  2. For  $m=1$  to  $M$  and while  $H_m \neq 0$ 
    - a) Train  $H_m(x)$  by weighted least-squares of  $y_i$  to  $x_i$ , with weights  $w_i$ .
    - b) Compute “inverted” distribution  $\bar{w}_i = (1 - w_i)$  and renormalize to  $\sum_i \bar{w}_i = 1$ .
    - c) Compute:  
$$P_m^{+1} = P_w(y = +1, H_m(x))$$
$$\bar{P}_m^{+1} = P_{\bar{w}}(y = +1, H_m(x))$$
$$P_m^{-1} = P_w(y = -1, H_m(x))$$
$$\bar{P}_m^{-1} = P_{\bar{w}}(y = -1, H_m(x))$$
    - d) Set  $H_m(x) = (P_m^{+1}(1 - P_m^{-1}) - P_m^{-1}(1 - P_m^{+1}))$
    - e) Update  $w_i \leftarrow w_i \exp(-y_i H_m(x_i))$  and renormalize to  $\sum_i w_i = 1$ .
  3. Output  $H(x) = \text{sign}\left(\sum_{m=1}^M H_m(x)\right)$ .
- 

The standard distribution  $w_i$  assigns high weights to training samples misclassified by earlier stages. On the contrary,  $\bar{w}_i$  gives higher weights to samples that are already correctly classified by earlier steps.

On step 2c), we have the expressions  $P_m^{+1} = P_w(y = +1, H_m(x))$  and  $P_m^{-1} = P_w(y = -1, H_m(x))$ . These expressions compute how good is our current weak classifier at predicting class labels. On the other hand, the expressions  $\bar{P}_m^{+1} = P_{\bar{w}}(y = +1, H_m(x))$  and  $\bar{P}_m^{-1} = P_{\bar{w}}(y = -1, H_m(x))$  estimate how well our current classifier  $H_m(x)$  is working on the data that has been correctly classified by previous steps.

The update  $H_m(x) = (P_m^{+1}(1 - P_m^{-1}) - P_m^{-1}(1 - P_m^{+1}))$  decreased weak classifiers contribution, if it works “too well” on data that has been already correctly classified with high margin. This way, the algorithm is named *Modest* because the classifiers tend to work only in their domain, as defined by  $w_i$ .

#### 4.2.4 LogitBoost

The LogitBoost algorithm consists of using adaptive Newton steps to fit an adaptive symmetric logistic model.

---



---

##### LogitBoost Algorithm for Classification

---

Input:  $Z = \{z_1, z_2, \dots, z_N\}$ , with  $z_i = (x_i, y_i)$  as training set.

$M$ , the maximum number of classifiers.

Output:  $H(x)$ , a classifier suited for the training set.

---

1. Initialize the weights  $w_i = 1/N$ ,  $i \in \{1, \dots, N\}$ .
  2. For  $m=1$  to  $M$  and while  $H_m \neq 0$ 
    - a) Compute the working response  $z_i = \frac{y_i^* - p(x_i)}{p(x_i)(1-p(x_i))}$  and weights  $w_i = p(x_i)(1-p(x_i))$ .
    - b) Fit  $H_m(x)$  by a weighted least-squares of  $z_i$  to  $x_i$ , with weights  $w_i$ .
    - c) Set  $H(x) = H(x) + \frac{1}{2}H_m(x)$  and  $p(x) = \frac{\exp(H(x))}{\exp(H(x)) + \exp(-H(x))}$ .
  3. Output  $H(x) = \text{sign}\left(\sum_{m=1}^M H_m(x)\right)$ .
- 
- 

#### 4.2.5 FloatBoost

The FloatBoost [18, 19] learning algorithm is composed of the following parts: 1-initialization; 2-forward inclusion; 3-conditional exclusion; 4-output. All these steps, with the exception of step 3, are similar to those of AdaBoost. The novelty is the conditional exclusion step, in which the least significant weak classifier is removed from the set of classifiers, subject to the condition that the removal leads to a lower error than some threshold. The FloatBoost algorithm is as follows.

---



---

##### FloatBoost Algorithm for Classification

---

Input:  $Z = \{z_1, z_2, \dots, z_N\}$ , with  $z_i = (x_i, y_i)$  as training set,  
with  $N=a+b$ ;  $a$  examples have  $y_i = +1$  and  $b$  examples have  $y_i = -1$ .

$M_{\max}$ , the maximum number of classifiers.

$J(H_M)$ , the cost function and the maximum acceptable cost  $J^*$ .

Output:  $H(x)$ , a classifier suited for the training set.

---

1. Initialization.
  - a) Initialize the weights  $w_i^{(0)} = 1/2a$ , for those examples with  $y_i = +1$ .
  - b) Initialize the weights  $w_i^{(0)} = 1/2b$ , for those examples with  $y_i = -1$ .
  - c)  $J_m^{\min} = J^*$   $m = \{1, \dots, M_{\max}\}$ .
  - d)  $M = 0$ ,  $\mathcal{H}_0 = \{\}$ .
2. Forward inclusion.
  - a)  $M \leftarrow M + 1$ .
  - b) Learn  $H_m(x)$  and  $\alpha_M$ .
  - c) Update  $w_i^{(M)} \leftarrow w_i^{(M-1)} \exp(-y_i \alpha_M H_M(x_i))$  and renormalize to  $\sum_i w_i = 1$ .
  - d)  $\mathcal{H}_M = \mathcal{H}_{M-1} \cup \{H_M\}$ .  
If  $J_M^{\min} > J(H_M)$  then  $J_M^{\min} = J(H_M)$ .
3. Conditional exclusion.
  - a)  $h' = \arg \min_{h \in \mathcal{H}_M} J(H_M - h)$ .
  - b) If  $J(H_M - h') < J_{M-1}^{\min}$ , then

- b1)  $\mathcal{H}_{M-1} = \mathcal{H}_M - h'$ .  
 $J_{M-1}^{\min} = J(H_M - h'); M = M - 1$ .  
b2) If  $h' = h'_m$ , then recalculate  $w_i^{(j)}$  and  $h_j$  for  $j = \{m', \dots, M\}$ .  
b3) Go to 3a).  
c) Else  
c1) If  $M = M_{\max}$  or  $J(\mathcal{H}_M) < J^*$ , then go to 4  
c2) Go to 2a).

4. Output the classification function  $H_M(x) = \text{sign} \left( \frac{\sum_{m=1}^M \alpha_m H_m(x)}{\sum_{m=1}^M \alpha_m} \right)$ .

#### 4.2.6 Emphasis Boost

On this tutorial, we name Emphasis Boost to the Real AdaBoost variant that uses a weighted emphasis function [11]. Each input pattern is weighted according to a criterion, defined by a parameter  $\lambda$ , through an emphasis function, in such a way that the training process focuses on the “critical” patterns (near the classification boundary) or on the quadratic error of each pattern.

##### EmphasisBoost Algorithm for Classification

Input:  $Z = \{z_1, z_2, \dots, z_N\}$ , with  $z_i = (x_i, y_i)$  as training set.  
 $M$ , the maximum number of classifiers.  
 $\lambda$ , weighting parameter ( $0 \leq \lambda \leq 1$ ).

Output:  $H(x)$ , a classifier suited for the training set.

1. Initialize the weights  $w_i = 1/N$ ,  $i \in \{1, \dots, N\}$ .
2. For  $m=1$  to  $M$ 
  - a) Fit a classifier  $H_m(x)$  to the training data using weights  $w_i$ .
  - b) Let  $\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq H_m(x_i))}{\sum_{i=1}^N w_i}$ .
  - c) Compute  $\alpha_m = 0.5 \log \left( \frac{1 + \text{err}_m}{1 - \text{err}_m} \right)$ .
  - d) Set  $w_i = \exp \left( \lambda \left( \sum_{j=1}^m (\alpha_j H_j(x_i) - y_i)^2 \right) - (1 - \lambda) \left( \sum_{j=1}^m H_j(x_i) \right)^2 \right)$  and renormalize to  $\sum_i w_i = 1$ .
3. Output  $H(x) = \text{sign} \left( \sum_{m=1}^M \alpha_m H_m(x) \right)$ .

The emphasis function

$$w_i = \exp \left( \lambda \left( \sum_{j=1}^m (\alpha_j H_j(x_i) - y_i)^2 \right) - (1 - \lambda) \left( \sum_{j=1}^m H_j(x_i) \right)^2 \right) \quad (7)$$

on step 2d) controls where the emphasis is placed. This flexible formulation allows us to choose how much to consider the “proximity” terms by means of a weighting parameter ( $0 \leq \lambda \leq 1$ ). This way, we have a “boosting by weighting boundary and erroneous samples”. Regarding the value of  $\lambda$ , three particular cases are interesting enough to be considered:

- $\lambda = 0$ , focus on the “critical” patterns because only the “proximity” to the boundary is taken into account

$$w_i = \exp \left[ - \left( \sum_{j=1}^m H_j(x_i) \right)^2 \right]. \quad (8)$$

- $\lambda = 0.5$ , we get the classical Real AdaBoost emphasis function

$$w_i = \exp \left[ \left( \frac{\sum_{j=1}^m (H_j(x_i) - y_i)^2}{2} \right) - \frac{(\sum_{j=1}^m H_j(x_i))^2}{2} \right]. \quad (9)$$

- $\lambda = 1$ , the emphasis function only pays attention to the quadratic error of each pattern

$$w_i = \exp \left[ \sum_{j=1}^m (H_j(x_i) - y_i)^2 \right]. \quad (10)$$

The key issue with this algorithm is how to choose the  $\lambda$  parameter. It depends on the training set and how the training error evolves.

#### 4.2.7 Reweight Boost

In the Reweight AdaBoost variant [23], the weak classifiers are stumps (decision trees with a single node). This way, the combination of these weak classifiers produces a decision tree. The main idea is to consider as base classifier for boosting, not only the last weak classifier, but a classifier formed by the last  $r$  selected weak classifiers ( $r$  is a parameter of the method). This algorithm is an AdaBoost variant, with classifiers reuse.

---

#### Reweight Boost Algorithm for Classification

---

Input:  $Z = \{z_1, z_2, \dots, z_N\}$ , with  $z_i = (x_i, y_i)$  as training set.  
 $M$ , the maximum number of classifiers.  
 $r$ , the last  $r$  selected weak classifiers .

Output:  $H(x)$ , a classifier suited for the training set.

---

1. Initialize the weights  $w_i = 1/N$ ,  $i \in \{1, \dots, N\}$ .
  2. For  $m=1$  to  $M$ 
    - a) Fit a classifier  $H_m(x)$  to the training data using weights  $w_i$ .
    - b) Get combined classifier  $H_t^r$  from  $H_t, H_{t-1}, \dots, H_{\max(t-r, 1)}$ .
    - c) Let  $\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq H_m(x_i))}{\sum_{i=1}^N w_i}$ .
    - d) Compute  $\alpha_m = 0.5 \log \left( \frac{1 - \text{err}_m}{\text{err}_m} \right)$ .
    - e) Set  $w_i \leftarrow w_i \exp(-\alpha_m I(y_i \neq H_t^r(x_i)))$  and renormalize to  $\sum_i w_i = 1$ .
  3. Output  $H(x) = \text{sign} \left( \sum_{m=1}^M \alpha_m H_m(x) \right)$ .
- 

#### 4.2.8 AnyBoost and MarginBoost

The MarginBoost algorithm [22] is a variant of the more general algorithm AnyBoost [22]. MarginBoost is also a general algorithm. It chooses a combination of classifiers to optimize the sample average of any cost function of the margin. MarginBoost performs gradient descent in function space, at each iteration choosing a base classifier to include in the combination so as to maximally reduce the cost function. The idea of performing gradient descent in function space in this way is due to Breiman (1998). It turns out that, as in AdaBoost, the choice of the base classifier corresponds to a minimization problem involving weighted classification error. That is, for a certain weighting of the training data, the base classifier learning algorithm attempts to return a classifier that minimizes the weight of misclassified training examples.

The general class of algorithms named AnyBoost consists of gradient descent algorithms for choosing linear combinations of elements of an inner product space so as to minimize some cost functional. Each component of the linear combination is chosen to maximize a certain inner product. In MarginBoost, this inner product corresponds to the weighted training error of the base classifier.

---

#### AnyBoost Algorithm for Classification

---

---

Input:  $Z = \{z_1, z_2, \dots, z_N\}$ , with  $z_i = (x_i, y_i)$  as training set.  
 $M$ , the maximum number of classifiers.  
Output:  $H(x)$ , a classifier suited for the training set.

---

1. Initialize the weights  $w_i = 1/N$ ,  $i \in \{1, \dots, N\}$ .
  2. For  $m=1$  to  $M$ 
    - a) Fit a classifier  $H_m(x)$  to the training data using weights  $w_i$ .
    - b) Get combined classifier  $H_t^r$  from  $H_t, H_{t-1}, \dots, H_{\max(t-r, 1)}$ .
    - c) Let  $\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq H_m(x_i))}{\sum_{i=1}^N w_i}$ .
    - d) Compute  $\alpha_m = 0.5 \log\left(\frac{1 - \text{err}_m}{\text{err}_m}\right)$ .
    - e) Set  $w_i \leftarrow w_i \exp(-\alpha_m I(y_i \neq H_t^r(x_i)))$  and renormalize to  $\sum_i w_i = 1$ .
  3. Output  $H(x) = \text{sign}\left(\sum_{m=1}^M \alpha_m H_m(x)\right)$ .
- 

---

#### MarginBoost Algorithm for Classification

---

Input:  $Z = \{z_1, z_2, \dots, z_N\}$ , with  $z_i = (x_i, y_i)$  as training set.  
 $M$ , the maximum number of classifiers.  
Output:  $H(x)$ , a classifier suited for the training set.

---

1. Initialize the weights  $w_i = 1/N$ ,  $i \in \{1, \dots, N\}$ .
  2. For  $m=1$  to  $M$ 
    - a) Fit a classifier  $H_m(x)$  to the training data using weights  $w_i$ .
    - b) Get combined classifier  $H_t^r$  from  $H_t, H_{t-1}, \dots, H_{\max(t-r, 1)}$ .
    - c) Let  $\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq H_m(x_i))}{\sum_{i=1}^N w_i}$ .
    - d) Compute  $\alpha_m = 0.5 \log\left(\frac{1 - \text{err}_m}{\text{err}_m}\right)$ .
    - e) Set  $w_i \leftarrow w_i \exp(-\alpha_m I(y_i \neq H_t^r(x_i)))$  and renormalize to  $\sum_i w_i = 1$ .
  3. Output  $H(x) = \text{sign}\left(\sum_{m=1}^M \alpha_m H_m(x)\right)$ .
- 

#### 4.2.9 List of variants

In this section, we outline the variants of AdaBoost, producing a list as extensive as possible. We consider some other variants of Discrete AdaBoost, that are not well documented. We briefly point out their main features. We also state the multi-class extension of the Discrete AdaBoost algorithm. For binary classification, we have:

- Real AdaBoost [25], Gentle AdaBoost [10], Modest AdaBoost [26], Logit Boost, Float Boost [18, 19], Emphasis Boost [11], Reweight Boost [23], and MarginBoost [22]. These algorithms were described in the previous sections.
- KLBoost [20] - uses Kullback-Leibler divergence and operates as follows. First, classification is based on the sum of histogram divergences along corresponding global and discriminating linear features. Second, these linear features, called KL features, are iteratively learnt by maximizing the projected Kullback-Leibler divergence in a boosting manner. Third, the coefficients to combine the histogram divergences are learnt by minimizing the recognition error once a new feature is added to the classifier. This contrasts conventional AdaBoost where the coefficients are empirically set. Because of these properties, KLBoosting classifier generalizes very well. KLBoosting has been applied to high-dimensional image space.

- ActiveBoost [28] - AdaBoost can not improve the performance of Naive Bays as expected. ActiveBoost, applies active learning to mitigate the negative effect of noise data and introduce instability into boosting procedure. Empirical studies on a set of natural domains show that ActiveBoost has clear advantages with respect to the increasing of the classification accuracy of Naive Bayes when compared against Adaboost.
- Jensen-Shannon Boosting [15] - incorporates Jensen-Shannon (JS) divergence into AdaBoost. JS divergence is advantageous in that it provides more appropriate measure of dissimilarity between two classes and it is numerically more stable than other measures such as Kullback-Leibler (KL) divergence.
- Infomax Boosting [21]. It is an efficient feature pursuit scheme for boosting. The proposed method is based on the infomax principle, which seeks optimal feature that achieves maximal mutual information with class labels. Direct feature pursuit with infomax is computationally prohibitive, so an efficient gradient ascent algorithm is further proposed, based on the quadratic mutual information, nonparametric density estimation and fast Gauss transform. The feature pursuit process is integrated into a boosting framework as infomax boosting. It is similar to RealAdaBoost, but with the following exceptions:
  - features are general linear projections;
  - generates optimal features;
  - uses KL divergence to select features;
  - finer tuning on coefficients.
- Ent-Boost [16] - uses entropy measures. Class entropy information is used to automatically subspace splitting and optimal weak classifier selection. The number of bins is estimated through a discretization process. Kullback-Leibler divergence is applied to probability distribution of positive and negative samples, to select the best weak classifier in the weak classifier set.
- Brown Boost - uses a non-monotonic weighting function such as examples far from the boundary decrease in weight, trying to achieve a given target error rate.
- Weight Boost - uses input-dependent weighting factors for weak learners.

For the multi-class case we also have several AdaBoost variants, as depicted in figure 6.

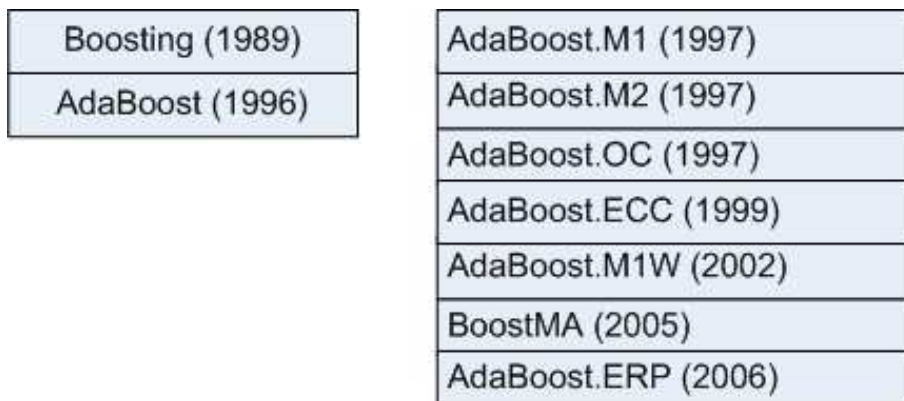


Figure 6: AdaBoost and its variants for supervised learning (multi-class case).

For each variant, we point out its main features.

- AdaBoost.M1 and AdaBoost.M2 [9] - are multi-class extensions of Discrete AdaBoost. They differ between them in the way they treat each class. In AdaBoost.M1 the weight of a base classifier is a function of the error rate. In AdaBoost.M2, the sampling weights are increased for instances for which the pseudo-loss exceeds 0.5.
- AdaBoost.OC [24] - OC stands for output codes.
- AdaBoost.ECC [13] - ECC stands for error-correcting codes.
- AdaBoost.M1W [5] - In AdaBoost.M1 the weight of a base classifier is a function of the error rate. This variant consist on the modification of this function so that it gets positive, if the error rate is less than the error rate of random guessing.

- BoostMA [6] - a simple modification of AdaBoost.M1 in order to make it work for weak base classifiers;
- AdaBoost.ERP [17] - a multi-class classification problem can be reduced to a collection of binary problems with the aid of a coding matrix. The quality of the final solution, which is an ensemble of base classifiers learned on the binary problems, is affected by both the performance of the base learner and the error-correcting ability of the coding matrix. A coding matrix with strong error-correcting ability may not be overall optimal if the binary problems are too hard for the base learner. Thus a trade-off between error-correcting and base learning should be sought a multi-class boosting algorithm that modifies the coding matrix according to the learning ability of the base learner.

### 4.3 Variants for semi-supervised learning

Semi-Supervised Learning (SSL) has attracted many researchers, in the last few years. SSL can be stated in one of two ways:

- supervised learning + additional unlabeled data;
- unsupervised learning + additional labeled data.

Figure 7 shows the AdaBoost algorithm and its variants for semi-supervised learning.

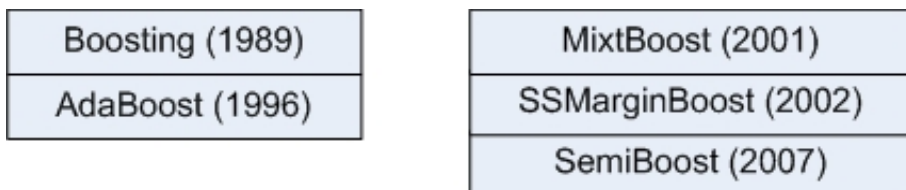


Figure 7: AdaBoost and its variants for semi-supervised learning.

#### 4.3.1 MixtBoost

The MixtBoost algorithm [12] was proposed in 2001. It is the first variant of AdaBoost to address the problem of SSL. The main goal of this research work was to assess the fitness of boosting to SSL, that is, the authors intend to answer the question: “Can boosting be adapted for SSL learning?”. The base classifiers are mixture models and this way MixtBoost can be seen as boosting of mixture models. A loss definition is proposed for unlabeled data, from which margins are defined.

The main components of AdaBoost are the *loss* and the *margin*, at steps 2b) and 2d), respectively. The simplest way to generalize to semi-supervised problems consists on defining a loss/margin for unlabeled data. This generalization to unlabeled data should not affect the loss for labeled examples, and should penalize inconsistencies between the classifier output and available information.

The missing labels are due to the absence of class information: “the pattern belongs to a class, but the class is unknown”.

#### 4.3.2 SSMarginBoost

The MarginBoost algorithm was proposed in 2000. It explores the clustering assumption of SSL and the large margin criterion. In 2002, the extension of this algorithm for semi-supervised learning was proposed, named SSMarginBoost.

#### 4.3.3 SemiBoost

The SemiBoost algorithm was proposed recently.

## 5 Experimental results and discussion

### 5.1 Successful applications of boosting algorithms

This section outlines some applications of boosting algorithms.

For face detection, boosting algorithms have been the most effective of all those developed so far, achieving the best results. They produce classifiers with about the same error rate than neural networks, but they have faster training [18]. Table 2 summarizes the use of boosting algorithms for face detection. A *Stub* is a decision tree with only one decision node.

Detector	AdaBoost Variant	Weak Learner
Viola-Jones [27]	Discrete AdaBoost	Stubs
Float Boost	Float Boost [18, 19]	1D Histograms
KLBoost	KLBoost [20]	1D Histograms
Schneiderman	Real AdaBoost [25]	One group of n-D Histograms

Table 2: Use of boosting algorithms for face detection (adapted from [18]).

## 5.2 Discussion on the performance of boosting

The impressive generalization performance of boosting can be attributed to the classifier having large margins on the training data.

AdaBoost had been perceived to be resistant to overfitting despite the fact that it can produce combinations involving very large numbers of classifiers. However, recent studies have shown that this is not the case, even for base classifiers as simple as decision stumps. Grove and Schuurmans (1998) demonstrated that running AdaBoost for hundreds of thousands of rounds can lead to significant overfitting, while a number of authors (Dietterich, 1998; Ratsch et al., 1998; Bauer and Kohavi, 1997; Maclin and Opitz, 1997) showed that, by adding label noise, overfitting can be induced in AdaBoost even with relatively few classifiers in the combination.

## References

- [1] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–139, 1999.
- [2] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- [3] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. John Wiley & Sons, 2nd edition, 2001.
- [4] B. Efron. The jackknife, the bootstrap and other resampling plans. *Society for Industrial and Applied Mathematics (SIAM)*, 1982.
- [5] G. Eibl and K. Pfeiffer. How to make adaboost.M1 work for weak classifiers by changing only one line of the code. In *Machine Learning: Thirteenth European Conference*, volume 1, pages 109–120, 2002.
- [6] G. Eibl and K. Pfeiffer. Multiclass boosting for weak classifiers. *Journal of Machine Learning Research*, 6:189–210, 2005.
- [7] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Thirteenth International Conference on Machine Learning*, pages 148–156, Bari, Italy, 1996.
- [8] Y. Freund and R. Schapire. Game theory on-line prediction and boosting. In *Ninth Annual Conference on Computer Learning Theory*, pages 325–332, Desenzano del Garda, Italy, 1996.
- [9] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.
- [10] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 28(2):337–374, April 2000.
- [11] V. Gómez-Verdejo, M. Ortega-Moral, J. Arenas-García, and A. Figueiras-Vidal. Boosting of weighting critical and erroneous samples. *Neurocomputing*, 69(7-9):679–685, March 2006.
- [12] Y. Grandvalet, F. A. Buc, and C. Ambroise. Boosting mixture models for semi-supervised learning. In *ICANN International Conference on Artificial Neural Networks*, volume 1, pages 41–48, Vienna, Austria, 2001.

- [13] V. Guruswami and A. Sahai. Multiclass learning, boosting, and error-correcting codes. In *12th Annual Conference on Computational Learning Theory (COLT-99)*, Santa Cruz, USA, 1999.
- [14] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2nd edition, 2001.
- [15] X. Huang, S. Li, and Y. Wang. Jensen-shannon boosting learning for object recognition. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 144–149, 2005.
- [16] D.-D. Le and S. Satoh. Ent-boost: Boosting using entropy measures for robust object detection. *Pattern Recognition Letters*, 2007.
- [17] L. Li. Multiclass boosting with repartitioning. In *23rd International Conference on Machine Learning (ICML-2006)*, Pennsylvania, USA, 2006.
- [18] S. Li and A. Jain. *Handbook of Face Recognition*. Springer, 2005.
- [19] S. Li and Z. Zhang. Floatboost learning and statistical face detection. *Transactions on Pattern Analysis and Machine Intelligence*, 26(9):23–38, 2004.
- [20] C. Liu and H. Shum. Kullback-Leibler boosting. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 587–594, Madison, Wisconsin, USA, 2003.
- [21] S. Lyu. Infomax boosting. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 533–538, 2005.
- [22] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Functional gradient techniques for combining hypotheses. *Advances in Large Margin Classifiers*, 1:109–120, 2000.
- [23] J. Rodriguez and J. Maudes. Boosting recombined weak classifiers. *Pattern Recognition Letters*, 2007.
- [24] R. Schapire. Using output codes to boost multiclass learning problems. In *14th International Conference on Machine Learning (ICML-97)*, pages 313–321, Tennessee, USA, 1997.
- [25] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.
- [26] A. Vezhnevets and V. Vezhnevets. Modest adaboost - teaching adaboost to generalize better. *Graphics*, 12(5):987–997, September 2005.
- [27] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceeding International Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 511–518, Hawaii, 2001.
- [28] L.-M. Wang, S.-M. Yuan, L. Li, and H.-J. Li. Boosting naive bayes by active learning. In *Third International Conference on Machine Learning and Cybernetics*, volume 1, pages 41–48, Shanghai, China, 2004.