Boosting

Zhi-Hua Zhou
National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing, China
http://cs.nju.edu.cn/zhouzh/

SYNONYMS
None.

DEFINITION
Boosting is a kind of ensemble methods which produce a strong learner that is capable of making very accurate predictions by combining rough and moderately inaccurate learners (which are called as base learners or weak learners). In particular, Boosting sequentially trains a series of base learners by using a base learning algorithm, where the training examples wrongly predicted by a base learner will receive more attention from the successive base learner; after that, it generates a final strong learner through a weighted combination of these base learners.

HISTORICAL BACKGROUND
In 1988, Kearns and Valiant posed an interesting question for the research of computational learning theory, i.e., whether a weak learning algorithm that performs just slightly better than random guess can be “boosted” into an arbitrarily accurate strong learning algorithm. In other words, whether two complexity classes, weakly learnable and strongly learnable problems, are equal. In 1989, Schapire [9] proved that the answer to the question is “yes”, and the proof he gave is a construction, which is the first Boosting algorithm. One year later, Freund developed a more efficient algorithm. However, both algorithms suffered from some practical drawbacks. Later, in 1995, Freund and Schapire [4] developed the AdaBoost algorithm which is effective and efficient in practice, and then a hot wave of research on Boosting arose.

SCIENTIFIC FUNDAMENTALS
AdaBoost is the most influential Boosting algorithm. Let \( \mathcal{X} \) and \( \mathcal{Y} \) denote the instance space and the set of class labels, respectively, and assume \( \mathcal{Y} = \{-1, +1\} \). Given a training set \( \mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\} \) where \( x_i \in \mathcal{X} \) and \( y_i \in \mathcal{Y} \) \( (i = 1, \ldots, m) \), and a base learning algorithm which can be decision tree, neural networks or any other learning algorithms, the AdaBoost algorithm works as follows.

First, it assigns equal weights to all the training examples \( (x_i, y_i) \) \( (i \in \{1, \ldots, m\}) \). Denote the distribution of the weights at the \( t \)-th learning round as \( D_t \). From the training set and \( D_t \) the algorithm generates a base learner \( h_t : \mathcal{X} \to \mathcal{Y} \) by calling the base learning algorithm. Then, it uses the training examples to test \( h_t \), and the weights of the incorrectly classified examples will be increased. Thus, an updated weight distribution \( D_{t+1} \) is obtained. From the training set and \( D_{t+1} \) AdaBoost generates another base learner by calling the
base learning algorithm again. Such a process is repeated for \( T \) times, each of which is called a *round*, and the final learner is derived by weighted majority voting of the \( T \) base learners, where the weights of the learners are determined during the training process. In practice, the base learning algorithm may be a learning algorithm which can use weighted training examples directly; otherwise the weights can be exploited by sampling the training examples according to the weight distribution \( D_t \). The pseudo-code of AdaBoost is shown in Figure 1.

\[
\text{Input: Data set } \mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}; \\
\text{Base learning algorithm } \mathcal{L}; \\
\text{Number of learning rounds } T.
\]

**Process:**

\[
D_1(i) = 1/m. \quad \% \text{Initialize the weight distribution for } t = 1, \ldots, T:
\]

\[
h_t = \mathcal{L}(D, D_t); \quad \% \text{Train a base learner } h_t \text{ from } \mathcal{D} \text{ using distribution } D_t
\]

\[
\epsilon_t = \Pr_{i \sim D} [h_t(x_i) \neq y_i]; \quad \% \text{Measure the error of } h_t
\]

\[
\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right); \quad \% \text{Determine the weight of } h_t
\]

\[
D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(x_i) = y_i \\ \exp(\alpha_t) & \text{if } h_t(x_i) \neq y_i \end{cases} \quad \% \text{Update the distribution, where } Z_t \text{ is a normalization factor which enables } D_{t+1} \text{ to be a distribution}
\]

\% end.

**Output:** \( H(x) = \text{sign}(f(x)) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right) \)

Figure 1: The AdaBoost algorithm

Freund and Schapire [4] proved that the training error of the final learner \( H \) is upper-bounded by

\[
\epsilon_D = \Pr_{i \sim \mathcal{D}} [H(x_i) \neq y_i] \leq 2T \prod_{t=1}^{T} \epsilon_t (1 - \epsilon_t),
\]

which can be written as

\[
\epsilon_D \leq \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq \exp \left( -2 \sum_{t=1}^{T} \gamma_t^2 \right),
\]

where \( \gamma_t = 1/2 - \epsilon_t \). Thus, if each base learner is slightly better than random so that \( \gamma_t \geq \gamma \) for some \( \gamma > 0 \), the training error will drop exponentially fast in \( T \) since the upper bound is at most \( e^{-2T\gamma^2} \).

Freund and Schapire [4] also gave a generalization error bound of \( H \) in terms of its training error \( \epsilon_D \), the size \( m \) of the training set, the VC-dimension \( d \) of the base learner space, and the number of rounds \( T \), by

\[
\epsilon \leq \epsilon_D + \tilde{O} \left( \sqrt{\frac{Td}{m}} \right)
\]

with high probability, where \( \tilde{O}(\cdot) \) is used to hide all logarithmic and constant factors [10] instead of using \( O(\cdot) \) which hides only constant factors.

The above generalization error bound suggests that AdaBoost will overfit if it runs for many rounds since \( T \) is in the numerator. However, empirical observations show that AdaBoost often does not overfit even after a large number of rounds, and sometimes it is even able to reduce the generalization error after the training error has already reached zero. Thus, later, Schapire et al. [11] presented another generalization error bound,

\[
\epsilon \leq \Pr_{i \sim \mathcal{D}} [\text{margin}_f (x_i, y_i) \leq \theta] + \tilde{O} \left( \sqrt{\frac{d}{m\theta^2}} \right)
\]

2
for any $\theta > 0$ with high probability, where the margin of $f$ on $(x_i, y_i)$ was defined as

$$\text{margin}_f(x_i, y_i) = \frac{y_if(x_i)}{\sum_{t=1}^{T} |\alpha_t|} = \frac{y_i \sum_{t=1}^{T} \alpha_t h_t(x)}{\sum_{t=1}^{T} |\alpha_t|},$$

whose value is in $[-1, +1]$ and is positive only if $H$ classifies $(x_i, y_i)$ correctly. In fact, the magnitude of margin can be explained as a measure of confidence in prediction. The larger the magnitude of margin, the higher confidence of prediction. Note that when $H(x_i) = y_i$, margin$_f(x_i, y_i)$ can still be increased as $t$ increases. Thus, the above margin-based generalization bound gives an answer to the question that why AdaBoost is able to reduce the generalization error even after the training error reaches zero, that is, the confidence in prediction can be increased further. However, Breiman [2] showed that improving the margin does not necessarily lead to the improvement of generalization, which doubted the above margin-based explanation to AdaBoost.

In addition to the aforementioned studies, the behavior of AdaBoost has been explained from the views of game theory [3, 2], additive model [5], etc. Many variants or extensions of AdaBoost have been developed [10, 6], which makes Boosting become a big family of ensemble methods.

In contrast to another famous ensemble method, Bagging, which reduces variance significantly but has little effect on bias, Boosting can significantly reduce bias in addition to reducing variance. So, on weak learners such as decision stumps, which are one-level decision trees, Boosting is usually more effective.

**KEY APPLICATIONS**

The first application of Boosting was on Optical Character Recognition by Drucker et al. Later, Boosting was applied to diverse tasks such as text categorization, speech recognition, image retrieval, medical diagnosis, etc. [10, 6]. It is worth mentioning that AdaBoost has been combined with a cascade process for face detection [12], and the resulting face detector was 15 times faster than state-of-the-art face detectors at that time but with comparable accuracy, which was recognized as one of the major breakthroughs in computer vision (in particular, face detection) during the past decade.

**FUTURE DIRECTIONS**

The margin-based explanation to why Boosting often does not overfit was seriously challenged by Breiman’s indication that larger margin does not necessarily mean better generalization [2]. Recently, Reyzin and Schapire [8] found that Breiman considered minimum margin instead of average or median margin. If the margin-based explanation can survive, it may be possible to establish a unified theoretical framework for the two powerful learning approaches, i.e., Boosting and support vector machine, since it is well-known that support vector machine works by maximizing the margin in a feature space.

It has been observed that Boosting performs poorly when abundant noise exists. Making Boosting more robust to noise is an important task. Moreover, Boosting suffers from some general deficiencies of ensemble methods, such as the lack of comprehensibility, i.e., the knowledge learned by Boosting is not understandable to the user. Trying to overcome those deficiencies is an important future direction.

**EXPERIMENTAL RESULTS**

Empirical studies on Boosting have been reported in many papers, such as [1, 7].
DATA SETS
A large collection of datasets commonly used for experiments can be found at http://www.ics.uci.edu/~mlearn/MLRepository.html

URL TO CODE
The code of an extended AdaBoost algorithm, BoosTexter, which was designed for multi-class text categorization, can be found at http://www.cs.princeton.edu/~schapire/boostexter.html

CROSS REFERENCE
ENSEMBLE
BAGGING
SUPPORT VECTOR MACHINE
DECISION TREE
NEURAL NETWORKS

RECOMMENDED READING
Between 3 and 15 citations to important literature, e.g., in journals, conference proceedings, and websites.