Bandit-Aided Boosting

Tiancheng Zhao

May 2019

Abstract

In this paper, we presented two works that uses method in multi-armed bandit problems to accelerate the boosting algorithm. Adaboost constructs a strong classifier in a stepwise manner by selecting simple base classifiers and using their weighted vote to determine the final classification. The procedure can be modeled by the sequential decision problem, and the papers use algorithms of multi-armed bandit problems to balance exploration and exploitation. The experiment results on multiple datasets show that the performance of bandit-aided boosting on the test set is similar to that of full AdaBoost algorithm, while the convergence speed is improved close to an order of magnitude.

1 Boosting Method

In this section, we are going to introduce the basics of boosting methods, as well as its key characteristics.

Boosting is a family of ensemble methods which converts weak learner to strong learners. The basic idea is to train weak learners in a sequential manner, and each of the learner tries to correct all the previous predictions. It is shown in practice that boosting can increase the performance of any learner. The base learner is often decision stumps which is the one-decision two-leaf decision tree. However, other base learners such as decision trees are also frequently used.

Formally, a sufficient condition for an algorithm to be called boosting is that, given a base learner which always returns a classifier $h(t)$ with edge $\gamma(t) \geq \rho$ for given $\rho > 0$, it returns a strong classifier $f(T)$ with zero training error after a logarithmic number of iterations $T = O(logn)$.

One representative of boosting method is the AdaBoost, which is the abbreviate for Adaptive Boosting Method. This method combines multiple weak learners into a single strong learner iteratively in the following manner. At each iteration, the algorithm randomly include a weak learner, and train the weak learner according to the weighted loss function. Initially, the weights of all observations are set to be equal. To correct the previous error, the observations that were incorrectly classified now carry more weight in later iterations.

To formally describe the problem of AdaBoost, we need to introduce its basic notations:

- K: the number of possible classes
- $X = (x_1, \ldots, x_n)$: $n \times d$ observation matrix, and $x_i^{(j)}$ are the elements of the d-dimensional observation vectors $x_i \in \mathbb{R}^d$.
- $Y = (y_1, \ldots, y_n)$: $n \times K$ label matrix, where $y_i \in \{-1, +1\}^K$.

In multi-class classification there is one and only one element of y_i equal to $+1$, in this case we use $\ell(x_i)$ to denote the correct class corresponding to x_i. In multi-label or multi-task classification there is no such constraint, and the value of y_i is arbitrary.

One particular form of Adaboost is the AdaBoost.MH proposed in [4], which uses boosting to minimize Hamming loss, which is weighted empirical loss:

$$R_H(f^{(T)}, W^{(1)}) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell}^{(1)} \mathbb{I}\{\text{sign}(f_\ell^{(T)}(x_i)) \neq y_{i,\ell}\} \tag{1}$$

In the above equation, $f^{(T)}$ is the resulted strong learner obtained from the algorithm, $W^{(1)}$ are the corresponding weights to each observations, $\mathbb{I}\{x\}$ is the indicator function, which takes value 1 if x is true, and 0 otherwise, $\text{sign}(x)$ is the sign function, and $f_\ell^{(T)}(x_i)$ is the ℓth element of $f(x_i)$.

The pseudocode of the AdaBoost.MH algorithm is shown in Figure 1.

AdaBoost.MH$(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(1)}, \text{Base}(\cdot,\cdot,\cdot), T)$

1 **for** $t \leftarrow 1$ **to** T

2 $\quad \mathbf{h}^{(t)}(\cdot) \leftarrow \alpha^{(t)} \mathbf{v}^{(t)} \varphi^{(t)}(\cdot) \leftarrow \text{Base}\left(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)}\right)$

3 $\quad$ **for** $i \leftarrow 1$ **to** n **for** $\ell \leftarrow 1$ **to** K

4 $\quad\quad w_{i,\ell}^{(t+1)} \leftarrow w_{i,\ell}^{(t)} \dfrac{\exp\left(-h_\ell^{(t)}(\mathbf{x}_i) y_{i,\ell}\right)}{\sum_{i'=1}^{n} \sum_{\ell'=1}^{K} w_{i',\ell'}^{(t)} e^{-h_{\ell'}^{(t)}(\mathbf{x}_{i'}) y_{i',\ell'}}}$

5 **return** $\mathbf{f}^{(T)}(\cdot) = \sum_{t=1}^{T} \mathbf{h}^{(t)}(\cdot)$

Figure 1: Pseudocode of the AdaBoost.MH algorithm

From Figure 1 we can notice that, the run time of AdaBoost.MH is proportional to the number of data points n, the number of attributes d, the number of boosting iterations T. Although the running time is linear in each of these factors, the algorithm can be prohibitively slow if the data size n is large or the number of features d is large. In this paper, our primary focus is to accelerate the AdaBoost algorithm.

In order to minimize Hamming loss (1) in a tractable way, we can minimize its upper bound, the exponential margin loss or the surrogate loss (2):

$$R_e(f^{(T),W^{(1)}}) = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(1)} \exp\{-f_\ell^{(T)}(x_i)y_{i,\ell}\}, \tag{2}$$

In this paper we are only focusing on discrete based classifier $h(x)$, which can be represented as:

$$h(x) = \alpha v \phi(x), \tag{3}$$

where $\alpha \in \mathbb{R}^+$ is the base coefficient, $v \in \{+1, -1\}^k$ is the vote vector and $\phi(x) : \mathbb{R}^d \longrightarrow \{-1, +1\}$ is a scalar base classifier. We can also express the base objective for each iteration as:

$$E(h, W^{(t)}) = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(1)} \exp\{-h_\ell^{(T)}(x_i)y_{i,\ell}\}. \tag{4}$$

It can be shown that minimizing (2) is equivalent to minimizing (4) for each iteration t. This is equivalent to maximize the edge:

$$\gamma = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell} v_\ell \phi(x_i) y_{i,\ell} \tag{5}$$

And the base objective can be expressed as:

$$E(h, W) = \sqrt{1 - \gamma^2}. \tag{6}$$

For each base learner, the value of (6) can serve as an indicator for the reward of using that base learner, and since we are iteratively selecting the base learners, this falls into the framework of Multi-Armed Bandit (MAB) problems we are going to introduce next.

2 Multi-Armed Bandit

In this section, we are going to introduce the framework of multi-armed bandit, and its two variants: stochastic bandit and adversarial bandit.

A multi-armed bandit problem (or, simply, a bandit problem) is a sequential allocation problem defined by a set of actions. At each time step, a unit resource is allocated to an action and some observable payoff is obtained. The goal is to maximize the total payoff obtained in a sequence of allocations.

Bandit problems are basic instances of sequential decision making with limited information, and naturally address the fundamental tradeoff between *exploration* and *exploitation* in sequential experiments.

To formally describe the bandit problem, we need to introduce the basic notations. In this paper we follow the terminology of [1]

- K: Number of arms (possible actions)
- n: Number of time steps
- $X_{i,t}$: Reward of choosing arm i at time step t, initially unknown to the forecaster
- I_t: Arm actually chosen at time step t

Regret is the most frequently used performance measure of a bandit algorithm. It is the difference between the performance of a player and an optimal strategy that, for any horizon of n time steps, consistently plays the arm that is best in the first n steps. Generally there are two notions of averaged regret:

Expected regret:

$$\mathbb{E}R_n = \mathbb{E}[\max_{i=1,\dots,K} \sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}]$$

Pseudo-regret:

$$\overline{R_n} = \max_{i=1,\dots,K} \mathbb{E}[\sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}]$$

Note that pseudo-regret is a weaker notion of expected regret, since the latter is the regret with respect to the action which is optimal on the sequence of reward realizations. So $\overline{R_n} \leq \mathbb{E}R_n$.

In stochastic bandits, each arm corresponds to an unknown probability distribution v_i, and rewards $X_{i,t}$ are independent draws from the distribution v_i corresponding to the selected arm. Formally as in Figure 2

The stochastic bandit problem

Known parameters: number of arms K and (possibly) number of rounds $n \geq K$.
Unknown parameters: K probability distributions $\nu_1, \dots, \nu_K$ on $[0, 1]$.

For each round $t = 1, 2, \dots$

(1) the forecaster chooses $I_t \in \{1, \dots, K\}$;
(2) given I_t, the environment draws the reward $X_{I_t,t} \sim \nu_{I_t}$ independently from the past and reveals it to the forecaster.

Figure 2: Stochastic bandits

A very simple heuristic is to use the ε-greedy strategy which exploits the known best strategy with probability $1 - \varepsilon$, and explores with probability ε. Based on that, people have proposed more dedicated UCB algorithms.

The intuition of UCB strategy is that, different ε-greedy strategy is selecting arms randomly, it would be better if we select the arm with the largest potential to be actually optimal, this arm could either have high reward estimates or large uncertainties.

UCB algorithm chooses the arm with largest Upper Confidence Bound, which is the sum of average reward and a confidence interval term:

$$\max_j \frac{1}{T_j^{(t)}} \sum_{t'=1}^{t} \mathbb{I}\{\text{arm } j \text{ is selected}\} r_j^{(t')} + \sqrt{\frac{2 \ln t}{T_j^{(t)}}}. \tag{7}$$

In [1], the authors have proved that the UCB algorithm can achieve a pseudo regret bound which is linear in $\ln n$. We'll not going into details of that due to space limit.

Another important variation of bandit problem is the adversarial bandit. This problem assumes that there is an non-random adversarial which can choose the reward for each arm in each iteration, and the choice can be influenced by the decision maker's previous actions.

Formally, denote by $g_{i,t}$ the reward of arm i at time step t. The adversarial bandit is described as in Figure 3:

The adversarial bandit problem

Known parameters: number of arms $K \geq 2$ and (possibly) number of rounds $n \geq K$.

For each round $t = 1, 2, \ldots$

(1) the forecaster chooses $I_t \in \{1, \ldots, K\}$, possibly with the help of external randomization;
(2) simultaneously, the adversary selects a gain vector $g_t = (g_{1,t}, \ldots, g_{K,t}) \in [0, 1]^K$, possibly with the help of external randomization;
(3) the forecaster receives (and observes) the reward $g_{I_t,t}$, while the gains of the other arms are not observed.

Figure 3: Adversarial bandits

Obviously, for any deterministic forecaster, there is a sequence of $g_{i,t}$ such that the regret is large. So, the key idea is to add add randomization to the selection of the action I_t to play. One algorithm to solve this problem is the *Exp3.P* strategy.

The *Exponential weights for Exploration and Exploitation (Exp3.P)* strategy maintains a probability distribution $p(t)$ over the arms and draws a random arm from this distribution in each iteration. The probability value of an arm increases exponentially with the average of past rewards. The pseudo code of *Exp3* strategy is shown in Figure 4.

In [1], the authors have proved that the *Exp3.P* algorithm can achieve a pseudo regret bound which is of order $O(\sqrt{nK \ln K})$. We'll not going into details of that due to space limit.

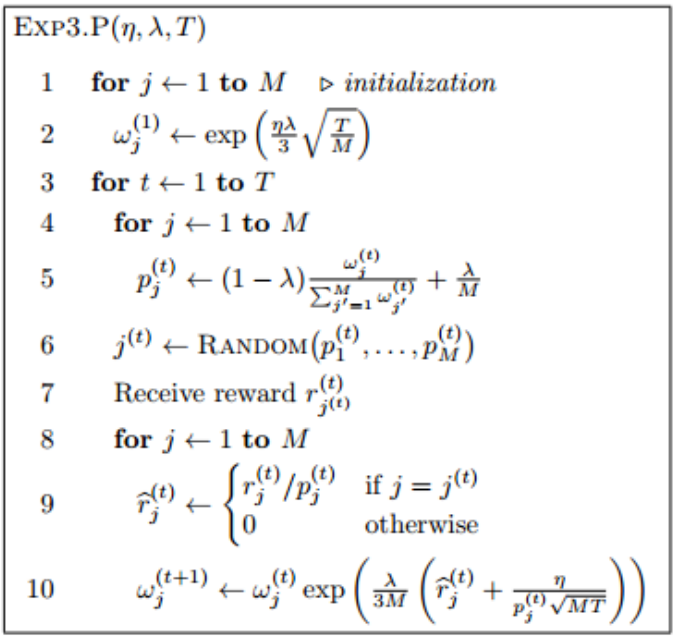

Figure 4: Pseudo code for Exp3.P algorithm

3 Bandit-Aided Boosting

In this section we are going to talk about how to incorporate bandit algorithms into boosting. Especially, two papers using UCB algorithm and *Exp3.P* algorithm to accelerate AdaBoost.MH respectively. Also, the authors of the papers performed numerical experiments to validate their result.

In [2], the authors incorporate the UCB algorithm as shown in (7) into AdaBoost.MH. From the derivation of AdaBoost.MH, (6) suggests that using reward of $r_j^{(t)} = \frac{1}{2}\log(1-\gamma^2)$ or $r_j^{(t)} = 1 - \sqrt{1-\gamma^2}$ as reward for choosing arm j at iteration t can fit into the framework of bandit problems. In fact, the later is a better reward function since the two are almost identical in the lower range of the $[0, 1]$ interval, and the latter has a value always in that interval, which is a requirement of MAB.

A natural partitioning of the base classifier set is the assign each feature to a subset, if the base classifier is decision stumps. The experiments were carried out using this setup. Probably there are other kind of partition methods, such as include more than one feature per subset. However, it makes no sense to split further, since the computational time of finding the best threshold on a feature is the same as that for evaluating a given stump on a data set.

In this work, the stochastic setup has an inherent mismatch between the adversarial nature of AdaBoost, in that the edges $\gamma_{j(t)}$ is deterministic. So the stochastic setup of UCB made it impossible to derive weak-to-strong-learning-

type performance guarantees on AdaBoost, and the connection between AdaBoost and bandits remained slightly heuristic. In light of this argument, a follow up work [3] applied an adversarial setup to the problem, and combine the *Exp3.P* algorithm with the AdaBoost. In this paper, they also derived a weak-to-strong-learning result for AdaBoost.MH.

Theorem 3.1. *Let $\mathcal{H}$ be the class of base classifiers and $\mathcal{G} = \{\mathcal{H}_1, ..., \mathcal{H}_M\}$ be an arbitrary partitioning of $\mathcal{H}$. Suppose that there exists a subset $\mathcal{H}_{j+}$ in $\mathcal{G}$ and a constant $0 < \rho \leq \gamma_{max}$ such that for any weighting over the training data set D, the base learner returns a base classifier from $\mathcal{H}_{j+}$ with an edge $\gamma_{\mathcal{H}_{j+}} \geq \rho$. Then, with probability at least $1 - \delta$, the training error $R(f^{(T)})$ of AdaBoost.MH.Exp3.P will become 0 after at most*

$$T = \max(\log^2 \frac{M}{\delta}, (\frac{4C}{\rho^2})^4, \frac{4 \log(n\sqrt{K-1})}{\rho^2})$$

iterations, with proper input parameters, where C is a constant.

The authors also conducted numerical experiments to test the performance of these algorithms using various data sets. They compared the performance of full AdaBoost.MH, AdaBoost.MH with random feature selection, stochastic-bandit-aided AdaBoost, and AdaBoost.MH.Exp3.P. Experiment result shows that:

- In terms of test error:
  - Full AdaBoost.MH wins most of the time although the differences are rather small
  - Exp3.P seems slightly better than UCB although the differences are even smaller
  - AdaBoost.MH with random feature selection has a much worse performance compared with other methods
- In terms of convergence speed
  - The improvement of bandit-aided boosting over full AdaBoost.MH is often close to an order of magnitude
  - Exp3.P also wins over UCB most of the time, and it is never significantly worse than the stochastic-bandit-based approach
  - The speed of AdaBoost.MH with random feature selection has a constant improvement than the full AdaBoost.MH, however, it is still much slower than either UCB or AdaBoost.MH.Exp3.P

References

[1] S. Bubeck, N. Cesa-Bianchi, et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012.

[2] R. Busa-Fekete and B. Kégl. Bandit-aided boosting. In *OPT 2009: 2nd NIPS Workshop on Optimization for Machine Learning*, 2009.

[3] R. Busa-Fekete and B. Kégl. Fast boosting using adversarial bandits. In *27th International Conference on Machine Learning (ICML 2010)*, pages 143–150, 2010.

[4] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336, 1999.