# Accelerating AdaBoost algorithm using Multi-armed bandits

Hieu Huynh - hthuynh2

May 10, 2019

## 1 Introduction

This report study the connection between the AdaBoost algorithm and the multi-armed bandit problem in the way such that multi-armed bandits can help to reduce the search space for finding the weak classifier at each iteration, which will eventually leads to faster performance of AdaBoost algorithm. In the first section, we will discuss the AdaBoost algorithm, and how to speed up the process of finding weak learner at each iteration. We also show how this problem can be connected to the Multi-armed bandit problem. In the section 3, we will discuss the reward function, and the framework of using multi-arm bandit to speed up AdaBoost algorithm. Then, we will analyze the 2 different Multi-arm bandits algorithms, UCB and EXP3.P.

## 2 AdaBoost

### 2.1 The algorithm

Let $\mathcal{G}$ be a class of classifiers $g : \mathbb{R}^d \mapsto \{-1, +1\}$. Given the training data $Z^n = (Z_1, .., Z_n)$, where each $Z_i = (X_i, Y_i)$ with $X_i \in \mathbb{R}^d$ and $Y_i \in \{-1, +1\}$, the AdaBoost algorithm works as described in Figure 1.

Let define some notations that we will use later:
$e_t := e_t(g_t)$: The weighted emperical error of the weak learner chosen at iteration $t$.
$\gamma_t := 1 - 2e_t$: The edge, which represents how much the performance of the weak learner chosen at iteration $t$ is better-than-chance.
$L_n := \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{\{sgn(f_T(X_i)) \neq Y_i\}}$: The emperical zero-one loss of the output classifier

**Algorithm 1** AdaBoost algorithm

1: Initialize $w^{(1)} = (w_1^{(1)}, .., w_n^{(1)})$ with $w_i^{(1)} = 1/n$ for all $i$
2: **for** $t = 1$ **to** $T$ **do**
3: **begin**
4: Find $g_t \in \mathcal{G}$ that minimize the weighted emperical error
5: $e_t(g) := \sum_{i=1}^{n} w_i^{(t)} \mathbb{1}_{\{Y_i \neq g(X_i)\}}$
6: $w_i^{(t+1)} := \frac{w_i^{(t)} exp(-\alpha_t Y_i g_t(X_i))}{Z_t}$
7: where $Z_t := \sum_{i=1}^{n} w_i^{(k)} exp(-\alpha_k Y_i g_t(X_i))$
8:
9: and $\alpha_t := \frac{1}{2} log(\frac{1-e_t}{e_t})$
10: **end**
11: Output classifier: $f_T(x) := \sum_{t=1}^{T} \alpha_t g_t(x)$

$L_e := \frac{1}{n} \sum_{i=1}^{n} \exp(-f_T(X_i)Y_i)$: The emperical exponential loss of the output classifier

## 2.2 Reducing the search space when finding weak learner

Noticing that in step 4 of the algorithm, we have to search over all elements $g \in \mathcal{G}$ to find the weak learner that minimize the weighted emperical error. A way to reduce the search space is that we can search over only a subset of $\mathcal{G}$ to find the weak learner that minimize the weighted emperical error. Specifically, let $\mathcal{H} = \{\mathcal{H}_1, \mathcal{H}_2, ..., \mathcal{H}_M\}$ with $\mathcal{H}_j \subseteq \mathcal{G}$ for all $j$; at each iteration, we pick an element $\mathcal{H}_j \in \mathcal{H}$ and search over that $\mathcal{H}_j$ to find the best weak learner. Although the weak learner that we find using the new method might not be as good as the one that is found using the original method, the process of finding weak learner is faster. And it can be shown that appropriately choosing the subspaces to search over and with some extra assumptions, we can still get a strong classifier after $O(\log n)$ iterations with high probability.

To show that choosing only a subspace to search over can help to speed up the AdaBoost algorithm, let's consider the following example. Assuming there are $d$ features, we use decision stump as weak learner and partition the whole space $\mathcal{G}$ by assigning a subset to each feature (i.e. $\mathcal{H} = \{\mathcal{H}_1, ..., \mathcal{H}_d\}$). Assuming the running time to find the weak learner from each of the $\mathcal{H}_j$ is the same and the time to make decision on which subset to use is $O(1)$, for each iteration, the time to find a weak learner by searching only a subspace

will be $O(d)$ times faster than by searching over the whole space $\mathcal{G}$. Therefore, if there are lots of features and if the total number of iterations for the modified algorithm and the original algorithm is about the same, then the total running time of the modified algorithm is much smaller than the original algorithm.

# 3 Adaboost combined with Multi-armed bandit problem

Papers [2] and [3] discussed a framework in which the process of chosing which subset of $\mathcal{G}$ to use is treated as a multi-armed bandit problem. Specifically, for each iteration, choosing an element $\mathcal{H}_j$ from $\mathcal{H}$ is equivalent to choosing an arm to pull. And the reward for that choice is calculated based on the accuracy of the weak learner selected from that subset $\mathcal{H}_j$ so that maximizing the total reward is equivalent to maximizing the total accuracy (i.e. minimizing the total error).
Papers [2] and [3] suggested to use the reward function $r_{\mathcal{H}_j}^{(t)} = \min(1, -\frac{1}{2}\log(1-\gamma_{\mathcal{H}_j}^2))$ for the action choosing $\mathcal{H}_j$ at time $t$ with $\gamma_{\mathcal{H}_j}$ be the edge of the weak learner chosen from $\mathcal{H}_j$. This suggestion coming from the fact that we want to minimize the $L_e$ (i.e. the emperical exponential loss of the output classifier), and according to Lemma 8.3 in the course note, $L_e$ can be written as $\prod_{t=1}^{T} 2\sqrt{e_t(1-e_t)}$ with $e_t$ be the weighted emperical error of the weak classifier chosen at time $t$. Replacing $e_t$ with the edge $\gamma_t$ (defined in section 2), we have $L_e = \prod_{t=1}^{T} \sqrt{1-\gamma_t^2}$. Taking the log on both sides, we have $\log(L_e) = \sum_{t=1}^{T} -\frac{1}{2}\log(1-\gamma_t^2)$. Now, it's intuitive to choose $r_{\mathcal{H}_j}^{(t)} = \min(1, -\frac{1}{2}\log(1-\gamma_{\mathcal{H}_j}^2))$ to be the reward for choosing $\mathcal{H}_j$ at time $t$. The reason of taking the min of 1 and $-\frac{1}{2}\log(1-\gamma_{\mathcal{H}_j}^2)$ is to bound the reward in the interval $[0, 1]$.

## 3.1 Accelerating AdaBoost with UCB

With the framework above, paper [2] suggested to use the Upper Confidence Bound (UCB) algorithm to solve the multi-armed bandit problem. The UCB algorithm uses an upper confidence bound, $\hat{U}_t(a)$, of the reward value to measure the potential of an action having optimal value, so that the true expected reward receiving by an action, $Q_t(a)$, is bounded by the sum of the sample mean, $\hat{Q}_t(a)$, and the upper confidience bound with high

probability (i.e. $Q_t(a) \leq \hat{Q}_t(a) + \hat{U}_t(a)$). The action time $t$ is selected greedily to maximize the upper confidence bound (i.e. $\hat{Q}_t(a) + \hat{U}_t(a)$). The paper conducted some experiments and arrived at the conclusion that using this framework and UCB algorithm, the computational time of the AdaBoost.MH (a more general version of AdaBoost that can handle multi-class and multi-task classification problem) can often be improved by an order the magnitude without sacrificing much accuracy.
Although UCB is a good choice for solving multi-armed bandit problem in many cases, in this case, there's a mismatch between the stochastic setup of UCB and the choice of reward function. Specifically, on one hand, for each iteration, given an action (i.e. the choice of $\mathcal{H}_j$) the value of $\gamma_t$ is deterministic, which means that the reward function $r_{\mathcal{H}_j}^{(t)}$ is also deterministic. On the other hand, the UCB assumes that the reward is randomly drawn from a distribution. Although the framework is still applicable here, but this mismatch makes it hard to give insight of how UCB can significantly speed up the algorithm while not sacrificing much accuracy.

## 3.2 Accelerating AdaBoost with EXP3.P

With the framework above, paper [3] suggested to use the EXP3.P algorithm [1] to solve the multi-armed bandit problem. EXP3.P is an Exponentially Weighted Average Forecaster method, in which a probability distribution over all arms are used to randomly drawn an arm, and the probability value of an arm increases exponentially with the average of past rewards. The detail of EXP3.P algorithm is describe in the subsection below.

### 3.2.1 EXP3.P Algorithm

Let's discuss the EXP3.P algorithm and some important properties that we will use later to prove that applying this algorithm into the framework above, we can get a strong classifier after $O(\log n)$ iterations with high probability and under some assumptions.
Let's first discuss some notations. Let $G_i^{(t+1)}$ be the total reward received up to time $t$ by following the EXP3.P algorithm (i.e. $G_i^{(t+1)} := \sum_{s=1}^{t} r_{i^{(t)}}^{(t)}$). Let $\hat{G}_i^{(t+1)}$ be the total estimated reward received up to time $t$ by following the EXP3.P algorithm (i.e. $\hat{G}_i^{(t+1)} := \sum_{s=1}^{t} \hat{r}_{i^{(t)}}^{(t)}$). Let $G_{\text{EXP3.P}}$ be the total reward received from time 1 to $T$ by following the EXP3.P algorithm (i.e. $G_{\text{EXP3.P}} := \sum_{t=1}^{T} r_{i^{(t)}}^{(t)}$). Let $G_{Max}$ be the total reward received from time

**Algorithm 2** EXP3.P algorithm

Parameters: $\alpha \in \mathbb{R}^+, \gamma \in (0,1]$, $T$

Initialize $w^{(1)} = (w_1^{(1)}, .., w_M^{(1)})$ with $w_i^{(1)} = \exp(\frac{\alpha\gamma}{3}\sqrt{\frac{T}{M}})$ for all $i$
**for** $t = 1$ **to** $T$ **do**
  **begin**
  **for** $i = 1$ **to** $M$ **do**
    $p_i^{(t)} = (1-\gamma)\frac{w_i^{(t)}}{\sum_{j=1}^{M} w_j^{(t)}} + \frac{\gamma}{M}$
  Choose $i^{(t)}$ randomly according to $p_1^{(t)}, ..., p_M^{(t)}$
  Receive reward $r_{i^{(t)}}^{(t)} \in [0,1]$
  **for** $j = 1$ **to** $M$ **do**
    **begin**
    $\hat{r}_j^{(t)} = r_j^{(t)}/p_j^{(t)}$ if $j = i^{(t)}$, $\hat{r}_j^{(t)} = 0$ otherwise
    $w_j^{(t+1)} = w_j^{(t)}\exp(\frac{\gamma}{3K}(\hat{r}_j^{(t)} + \frac{\alpha}{p_j^{(t)}MT}))$
    **end**
  **end**

1 to $T$ by following the best fixed strategy (i.e. $G_{Max} := max_j \sum_{t=1}^{T} r_j^{(t)}$). Let $\hat{\sigma}_i^{(t+1)} := \sqrt{MT} + \sum_{s=1}^{t} \frac{1}{p_i^{(s)}\sqrt{MT}}$.

Following is some explaination about the EXP3.P algorithm and an important property. The distribution $p_1^{(t)}, ..., p_M^{(t)}$ is a mixture of uniform distirbution and a distribution which assign probability value to action based on its weight. Therefore, the value of $\gamma$ controls how much the exploration factor. We estimated reward $\hat{r}_j^{(t)}$ is set to $r_j^{(t)}/p_j^{(t)}$ in order to guarantee that this is an unbiased estimator, and to compensate the reward of unlikely chosen action. With the definition of $\hat{\sigma}_i^{(t+1)}$ above, the weight of action $j$ at time $t+1$ (i.e. $w_j^{(t+1)}$) can be written in term of upper confidence bound $\hat{G}_j(t) + \alpha\hat{\sigma}_j^{(t+1)}$. And with the appropriate choice of $\alpha$ and $\gamma$, it can be shown that the regret of using the EXP3.P algorithm instead of the best fixed strategy is $O(\sqrt{MT\ln(MT/\delta)})$, more detail is described in Theorem 1.

**Theorem 1**: For any fixed $T > 0$, for all $M \geq 2$, and for all $\delta > 0$, if $\gamma = min\{\frac{3}{5}, 2\sqrt{\frac{3}{5}\frac{M\ln M}{T}}\}$ and $\alpha = 2\sqrt{\ln(MT/\delta)}$, then for any assignment of rewards with probability at least $1-\delta$, we have:

$$G_{Max} - G_{\text{EXP3.P}} \leq 4\sqrt{MT\ln(\frac{MT}{\delta})} + 4\sqrt{\frac{5}{3}MT\ln M} + 8\ln(\frac{MT}{\delta})$$

Note that a downside of this algorithm is that the time horizon $T$ is needed as an input parameter. However, the value of time horizon $T$ might not be available in some cases.

### 3.2.2 Accelerating AdaBoost with EXP3.P Algorithm

Using the framework above and the EXP3.P algorithm, we can show that the modified version of AdaBoost still have the weak-to-strong-learning-type performance guarantee under some extra assumption.

**Theorem 2**:

Let $\mathcal{G}$ be the clss of the base classifiers and $\mathcal{H} = \{\mathcal{H}_1, ..\mathcal{H}_M\}$ be an arbitrary partitioning of $\mathcal{G}$. Suppose that there exists a subset $\mathcal{H}_{j^+} \in \mathcal{H}$ and a constant $0 < \rho \leq \gamma_{max}$ such that for any weighting over the training data set $\mathcal{D}$, we can always find a base classifier in $\mathcal{H}_{j^+}$ with an edge $\gamma_{\mathcal{H}_{j^+}} \geq \rho$. Then, with probability at least $1 - \delta$, the emperical zero-one loss $L_n$ of the output classifier will become 0 after at most

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

iterations, where $C = \sqrt{32M} + \sqrt{27M\log(M)} + 16$, $K$ is the number of classes, and the input parameters for the EXP3.P algorithm are set to

$$\gamma = \min\left\{\frac{3}{5}, 2\sqrt{\frac{3}{5}\frac{M\log M}{T}}\right\}$$

$$\alpha = 2\sqrt{\log\frac{MT}{\delta}}$$

Before discussing the proof, let's first talk about the theorem. First of all, the assumption in the theorem 2 is a stronger assumption than the assumption in the original AdaBoost algorithm. In the original Adaboost algorithm, we only assume that over the whole space $\mathcal{G}$, there exist a weak classifier with the weighted emperical error $e_t \leq 1/2 - \rho/2$ (i.e. the edge $\gamma_t >= \rho$) for any weighting over the training data set. However, in the

theorem 2, we assume that there exist a subset of $\mathcal{H}_j^+$ such that there exist a weak classifier with the edge $\gamma \geq \rho$ for any weighting over the training data set. This stronger assumption might be hard to satisfy in some cases. Another note on this theorem is that it shows an interplay between the number of iterations $T$, the size of the subsets, number of subsets $M$, and the quality of the subset in terms of $\rho$. Specifically, when the quality of subset increases the total number of iteration needed to reach 0 training error will decreases. When the total number of subsets increases, the total number of iterations will also increase. This suggests that when we partition the whole space $\mathcal{G}$ into more subsets, we will need to run more iterations, but the running time of each iteration will be smaller. So, it is important to find an optimal way to partition the whole space $\mathcal{G}$ in order to minimize the total running time of the algorithm to achieve zero training error. A limitation of this theorem is that there are restrictions on the parameters $\gamma$ and $\alpha$, which require knowing the value of time horizon $T$ beforehand.

**Proof of theorem 2:**

Let denote the reward of choosing subset $\mathcal{H}_j$ at time t by $r_j^{(t)}$
Let denote $j^* = \text{argmax}_j \sum_{t=1}^{T} r_j^{(t)}$
Let denote the average reward of the retrospectively optimal arm by:

$$r^* = \frac{1}{T} \sum_{t=1}^{T} r_{j^*}^{(t)}$$

From section 3, we have $\log(L_e) = \sum_{t=1}^{T} -\frac{1}{2}\log(1-\gamma_t^2)$ with $\gamma_{\mathcal{H}_j(t)}^{(t)}$ is the edge of the weak learner that is chosen from $\mathcal{H}_j(t)$
Let assume $T > \max\{(\frac{4C}{\rho^2})^4, \log^2(\frac{M}{\delta})\}$, then we have:

$$\sum_{t=1}^{T} \log \sqrt{1 - {\gamma^{(t)}_{\mathcal{H}_{j^{(t)}}}}^2} \leq \sum_{t=1}^{T} -r^{(t)}_{j^{(t)}} \tag{17}$$

$$\leq -Tr^* + 4\sqrt{MT \log \frac{MT}{\delta}} + 4\sqrt{\frac{5}{3} MT \log M} + 8 \log \frac{MT}{\delta} \tag{18}$$

$$\leq -\sum_{t=1}^{T} r^{(t)}_{j^\dagger} + 4\sqrt{MT \log \frac{MT}{\delta}} + 4\sqrt{\frac{5}{3} MT \log M} + 8 \log \frac{MT}{\delta} \tag{19}$$

$$= \sum_{t=1}^{T} \log \sqrt{1 - \min(\gamma_{\max}, \gamma^{(t)}_{\mathcal{H}_{j^\dagger}})^2} + 4\sqrt{MT \log \frac{MT}{\delta}} + 4\sqrt{\frac{5}{3} MT \log M} + 8 \log \frac{MT}{\delta} \tag{20}$$

$$\leq -\frac{1}{2} \sum_{t=1}^{T} \min(\gamma_{\max}, \gamma^{(t)}_{\mathcal{H}_{j^\dagger}})^2 + 4\sqrt{MT \log \frac{MT}{\delta}} + 4\sqrt{\frac{5}{3} MT \log M} + 8 \log \frac{MT}{\delta} \tag{21}$$

$$\leq -\frac{1}{2} T\rho^2 + 4\sqrt{MT \log \frac{MT}{\delta}} + 4\sqrt{\frac{5}{3} MT \log M} + 8 \log \frac{MT}{\delta} \tag{22}$$

$$\leq -\frac{1}{2} T\rho^2 + 4\sqrt{MT \left( \log \frac{M}{\delta} + \sqrt{T} \right)} + 4\sqrt{\frac{5}{3} MT \log M} + 8 \log \frac{M}{\delta} + 8\sqrt{T} \tag{23}$$

$$\leq -\frac{1}{2} T\rho^2 + 4\sqrt{MT \left( \sqrt{T} + \sqrt{T} \right)} + 4\sqrt{\frac{5}{3} MT \log M} + 8\sqrt{T} + 8\sqrt{T} \tag{24}$$

$$= -\frac{1}{2} T\rho^2 + T^{3/4} \sqrt{32M} + T^{1/2} \left( \sqrt{\frac{80}{3} M \log M} + 16 \right) \tag{25}$$

$$\leq -\frac{1}{2} T\rho^2 + T^{3/4} \left( \sqrt{32M} + \sqrt{27M \log M} + 16 \right) \tag{26}$$

$$= -\frac{1}{2} T\rho^2 + T^{3/4} C \tag{27}$$

$$= -T \left( \frac{1}{2} \rho^2 + T^{-1/4} C \right) \tag{28}$$

$$\leq -\frac{1}{4} T\rho^2 \tag{29}$$

Explanation of the proof:
(17), (20): Follow from the definition of reward function
(18): Follow from theorem 1
(19): Follow from the fact that $r^*$ is average reward of the retrospectively optimal arm
(21): Follow from the fact that $ln(\sqrt{1-a^2}) \leq -\frac{1}{2}a^2$
(22): Follow from the assumption that $0 < \rho < \gamma_{max}$ and $\gamma_{j^*} \geq \rho$
(23): Follow from the fact that $\sqrt{T} > \log(T)$
(24): Follow from the assumption that $T > \log^2(\frac{M}{\delta})$
(26): Follow from the fact that $T > 1$
(27): Follow from the definition of $C = \sqrt{32M} + \sqrt{27M\log(M)} + 16$
(29): Follow from the assumption that $T > (\frac{4C}{\rho^2})^4$

Therefore, we have: $\log(L_e) \leq -\frac{1}{4}T\rho^2 \Rightarrow L_e \leq exp(-\frac{1}{4}T\rho^2)$.
For the binary classification problem discussed in section 2, we have the fact that the emperical zero-one loss is upper bound by the emperical exponential loss, therefore we have: the emperical zero-one loss $L \leq exp(-\frac{1}{4}T\rho^2)$. With the condition that $T < \frac{4log(n)}{\rho^2}$, we will have $L < \frac{1}{n}$. On the other hand, we have $L$ is multiple of $\frac{1}{n}$. Therefore $L$ must be 0.
Note that this result can be generalized to multi-classes and multi-task problem by using the fact that $L \leq \sqrt{K-1}L_e$ with appropriate initialization of the initial weights. Therefore, combining this result with the assumptions about $T$, Theorem 2 is proven.

# 4 Conclusion

This report studies the application of multi-armed bandit problem to accelerate the AdaBoost algorithm. First of all, we made the connection between the AdaBoost algorithm and the multi-armed bandit problem, then 2 different algorithms were considered for deciding which the subset of the whole space $\mathcal{G}$ to search for the weak learner at each iteration. It's shown that the algorithm EXP3.P is more suitable for this purpose compared to the UCB algorithm. This is because of the setting of EXP3.P is more suitable for the adversarial setting of the AdaBoost algorithm, while the stochastic setting of UCB algorithm prevents us from achieving the proof for the weak-to-strong-learning-type performance guarantee. As suggested in [3], since the process of training weak classifiers is sequential and not stateless, a Markov Decision Process might be a more natural choice.

## References


[1] Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R.E. *The Nonstochastic Multiarmed Bandit Problem.* SIAM J. on Computing, 32(1):48-77, 2002b.

[2] Busa-Fekete, R. and Kegl, B. *Bandid-Aided Boosting.* PT 2009: 2nd NIPS Workshop on Optimization for Machine Learning

[3] Busa-Fekete, R. and Kegl, B. *Fast Boosting Using Adversarial Bandits.* ICML'10 Proceedings of the 27th International Conference on International Conference on Machine Learning, 143-150