Boosting
Seminar: Foundations of Data Science

Christian Peters, enrolment no.: 213996
Faculty of Statistics
TU Dortmund

January 15, 2021
winter term 2020/21

Abstract

Boosting is an algorithmic paradigm that aims to create an efficient strong learner by combining multiple weak learners. This paper introduces the concept of weak learnability and explains the AdaBoost algorithm, the first practical implementation of boosting.

1 Introduction

Training machine learning models in practice is not always as simple as it might seem from a theoretical standpoint. In [3, chapter 2], the empirical risk minimization (ERM) rule was introduced as the learning algorithm of choice. However, this theoretical principle of choosing a hypothesis $h \in \mathcal{H}$ such that $L_S(h) = \min_{h \in \mathcal{H}} L_S(h)$, where $L_S(h)$ is the error of h on the training sequence S, can be impossible to use in practical applications due to the sometimes enormous computational complexity of searching through interesting hypothesis classes $\mathcal{H}$.

This problem leads to the question if it is possible to arrive at a strong learning algorithm in a way that doesn't require the computational cost of searching through complex hypothesis classes. Is it perhaps possible to create a strong learner by finding a way to combine "weak" learners that are potentially easier to compute? The algorithmic paradigm of boosting deals with exactly this question, which was first raised by Kearns and Valiant in 1988 [2], resulting in the widely used AdaBoost algorithm that shows how

"weak" learners can be combined in order to obtain a strong learning algorithm.

The goal of this article is to first lay down the foundations of boosting by explaining the concept of weak learnability, which will be used to arrive at the AdaBoost algorithm, the first practical implementation of boosting.

2 Weak Learnability

Assuming that the realizable assumption [3, chapter 2] holds, the generalization error $L_{(\mathcal{D},f)}(h)$ of a PAC learner with respect to a distribution $\mathcal{D}$ and a labeling function f can, by the definition of PAC learning [3, chapter 2], be reduced to an arbitrarily small number (with confidence of $1-\delta$) by increasing the sample size of the training sequence S. This, however, can be computationally infeasible in practical applications.

The concept of weak learnability aims to relax the requirement that the generalization error of a hypothesis must become arbitrarily small the more the sample size is increased. For weak learning, it is sufficient that the learning algorithm yields a hypothesis h that performs only slightly better than a random guess. One can think of weak learning as applying a simple rule which isn't fully capable of modeling the data generating process, but can still learn a small fraction of the underlying problem so that it performs better than random guessing.

Formally, the definition of a weak learner differs only slightly from the definition of PAC learning. In the situation of a two-class classification problem, the definition of a weak learner can be given as follows:

Definition 1. *(γ-weak-learner) An Algorithm A is a γ-weak-learner for a hypothesis class $\mathcal{H}$ if for every $\delta \in (0,1)$ there exists a threshold $m_{\mathcal{H}}(\delta) \in \mathbb{N}$ such that for every distribution $\mathcal{D}$ over the instance space $\mathcal{X}$ and for every labeling function $f : \mathcal{X} \to \{\pm 1\}$ if running A on $m \geq m_{\mathcal{H}}(\delta)$ training examples, it will yield a hypothesis h such that with probability of at least $1-\delta$ the generalization error $L_{(\mathcal{D},f)}(h)$ is at most $\frac{1}{2} - \gamma$, provided that the realizable assumption holds.*

The parameter γ in Definition 1 tells us how well we can expect the weak learner to perform. For example, if a learning algorithm is a 1%-weak-learner for a class $\mathcal{H}$, then the generalization error can at most be 49% provided that first, the learner didn't fail (which can happen with probability δ of drawing a bad sample from $\mathcal{D}$) and second, the learner was run on at least $m_{\mathcal{H}}(\delta)$ training examples.

There still remains the question of how such a weak learning algorithm for a hypothesis class $\mathcal{H}$ can be obtained. We already saw that applying the ERM rule to complex hypothesis classes can be computationally hard. But what if we choose a simple hypothesis class B instead, where the ERM rule can be applied efficiently? It follows directly from Definition 1 that this can only work, if the new algorithm ERM_B has an error of at most $\frac{1}{2} - \gamma$ for every sample that was labeled by a hypothesis from $\mathcal{H}$. In this case, applying the ERM rule with respect to the simpler class B would yield a weak learner for $\mathcal{H}$.

2.1 An Efficient ERM Algorithm for Decision Stumps

One such hypothesis class, where the ERM algorithm can be implemented efficiently, is the class of decision stumps. On the instance space $\mathcal{X} = \mathbb{R}^d$, this class is given as follows:

$$\mathcal{H}_{DS} = \{\mathbf{x} \mapsto \text{sign}\,(\theta - x_i) \cdot b : \ \theta \in \mathbb{R}, i \in [d], b \in \{\pm 1\}\}$$

The idea behind decision stumps is to divide the instance space $\mathcal{X}$ along one of its dimensions $i \in [d]$ at a threshold θ into two partitions, such that all the instances in one partition are labeled $+1$ and all the instances in the other partition are labeled -1.

Fixing $b = 1$ without the loss of generality, an ERM algorithm for $\mathcal{H}_{DS}$ has to find the optimal splitting dimension $i \in [d]$, as well as the optimal splitting threshold θ, such that the training error $L_S(h)$ on a training sequence $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m))$ is minimized.

An extension of the empirical risk $L_S(h)$ that will be useful when introducing the AdaBoost algorithm later in section 3, is the weighted empirical risk, which is given as

$$L_{\mathbf{D}}(h) = \sum_{i=1}^{m} D_i \mathbb{1}_{[h(\mathbf{x}_i) \neq y_i]}.$$

The vector $\mathbf{D} \in \mathbb{R}^m$ is used to give a weight to each training example, where each weight D_i is nonnegative and $\sum_{i=1}^{m} D_i = 1$. For the special case that $D_i = \frac{1}{m}$, the weighted empirical risk is equal to the unweighted empirical risk.

In order to find a decision stump $h \in \mathcal{H}_{DS}$ that minimizes the weighted empirical risk $L_{\mathbf{D}}(h)$, the $\text{ERM}_{\mathcal{H}_{DS}}$ algorithm has to solve the following optimization problem that minimizes the weighted sum of misclassifications:

$$\min_{j \in [d]} \min_{\theta \in \mathbb{R}} \left(\sum_{i: y_i = 1}^{m} D_i \mathbb{1}_{[x_{i,j} > \theta]} + \sum_{i: y_i = -1}^{m} D_i \mathbb{1}_{[x_{i,j} \leq \theta]} \right)$$

Taking a closer look at this problem, it becomes clear that it is not necessary to try every $\theta \in \mathbb{R}$ during the optimization. In fact, if we first sort the examples for a fixed dimension $j \in [d]$ so that $x_{1,j} \leq x_{2,j} \leq ... \leq x_{m,j}$, we can see that we only have to consider a single splitting threshold in between two examples, which leaves us with $m + 1$ values for θ that the $\text{ERM}_{\mathcal{H}_{DS}}$ algorithm has to consider for a fixed dimension $j \in [d]$. Since the sum of misclassifications can be computed in $\mathcal{O}(m)$ by passing through the data once, the algorithm can find optimal values for j and θ in $\mathcal{O}(dm^2)$ by simple enumeration.

This time complexity can be reduced even more by avoiding to recalculate the sum of misclassifications for every new value of θ. It can be shown [3] that it is possible to update the sum for a new value of θ in constant time, requiring only a single pass through the data for a fixed dimension j. This leaves us with a time complexity of $\mathcal{O}(dm)$ of solving the optimization problem, after the sorting step is applied as preprocessing.

3 AdaBoost

In the previous section, it was demonstrated how an efficient weak learner can be constructed by applying the ERM rule to the class of decision stumps. This section presents the AdaBoost (Adaptive Boosting) algorithm, a procedure that shows how weak learners such as decision stumps can be used efficiently to find a hypothesis with an arbitrarily low empirical error $L_S(h)$ on a training sequence S. The AdaBoost algorithm was first proposed in 1995 by Yoav Freund and Robert Schapire [1] and became hugely popular due to its practicability.

The goal of AdaBoost is to invoke the weak learner multiple times on the training data and then to combine the resulting hypotheses similar to a weighted majority vote. Let T be the number of times that the weak learner is invoked by AdaBoost. Then the resulting output hypothesis of AdaBoost has the following form:

$$h(x) = \text{sign}\left(\sum_{t=1}^{T} w_t h_t(x)\right)$$

Here, $h_t(x)$ is the output hypothesis of the weak learner in iteration t and w_t is the corresponding weight that AdaBoost assigns to this hypothesis.

In the first iteration $t = 1$, the AdaBoost algorithm assigns an equal weight $D_i^{(1)} = \frac{1}{m}$ to each example in the training sequence S and then invokes the weak learner on the weighted training sequence. The error of the resulting

hypothesis is computed according to

$$\epsilon_t = \sum_{i=1}^{m} D_i^{(t)} \mathbb{1}_{[h_t(\mathbf{x}_i) \neq y_i]}$$

and is at most $\frac{1}{2} - \gamma$, provided that the weak learner didn't fail.

The weight w_t that is assigned to a hypothesis in boosting round t is computed as follows:

$$w_t = \frac{1}{2} \log \left(\frac{1}{\epsilon_t} - 1 \right)$$

As we can see, the smaller the error ϵ_t of the hypothesis h_t is, the bigger the weight w_t will be.

At the end of each iteration, the weights $D_i^{(t)}$ of each training example are updated according to

$$D_i^{(t+1)} = \frac{D_i^{(t)} \exp\left(-w_t y_i h_t(\mathbf{x}_i)\right)}{\sum_{j=1}^{m} D_j^{(t)} \exp\left(-w_t y_j h_t(\mathbf{x}_j)\right)}$$

This update assigns a higher weight to those examples, that weren't correctly classified by h_t.

In short, the Adaboost algorithm performs the following steps in each of the T iterations:

1. Invoke the weak learner on the training sequence S weighted by $\mathbf{D}^{(t)}$
2. Assign a weight w_t to the output hypothesis h_t of the weak learner. Hypotheses with a smaller training error ϵ_t will get a higher weight.
3. Compute a new weight vector $\mathbf{D}^{(t+1)}$ that gives a higher weight to incorrectly classified examples.

The computational complexity of AdaBoost essentially consists of invoking the weak learning algorithm T times on the training data. Thus, if the weak learner can be implemented efficiently (as it is the case for decision stumps), AdaBoost is also efficient.

On top of that, it can be shown [3], that the training error $L_S(h)$ of the AdaBoost hypothesis h decreases exponentially in the number of boosting rounds T:

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{[h(\mathbf{x}_i) \neq y_i]} \leq e^{-2\gamma^2 T}$$

Here, γ describes the γ-weak-learner as defined in Definition 1. This means that, by increasing the number of boosting rounds T, AdaBoost will achieve an arbitrarily low training error and is still an efficient algorithm.

In practical applications however, it is more important to also achieve a good out of sample error. The next section will show that the true risk $L_{\mathcal{D}}(h)$ of the AdaBoost hypothesis will also be small by taking a look at the hypothesis class that resembles all the possible output hypotheses of AdaBoost.

3.1 AdaBoost Out-of-Sample Performance

The hypothesis class of AdaBoost is parameterized by the hypothesis class B of the weak learner it uses, as well as by the number of boosting rounds T. Formally, it is given as follows:

$$L(B,T) = \left\{ x \mapsto \text{sign}\left(\sum_{t=1}^{T} w_t h_t(x) \right) : \ w \in \mathbb{R}^T, \ h_t \in B \right\}$$

The fundamental theorem of statistical learning [3, chapter 6] states that a hypothesis class is PAC-learnable if and only if its VC dimension is finite, i.e. there exists a threshold m, such that for every set with more than m elements there exists a labeling function that cannot be learned using that class. This means that if the VC dimension of $L(B,T)$ is finite and thus $L(B,T)$ is PAC-learnable, then, by the definition of PAC learning, there exists a threshold $m_{\mathcal{H}}(\epsilon,\delta) \in \mathbb{N}$ for every $\epsilon,\delta \in (0,1)$, such that the true error $L_{\mathcal{D}}(h)$ of the AdaBoost hypothesis $h \in L(B,T)$ is at most ϵ (with confidence $1-\delta$) for every distribution $\mathcal{D}$ when training AdaBoost on at least $m_{\mathcal{H}}(\epsilon,\delta)$ training examples. This tells us that if $L(B,T)$ is learnable, even the Out-of-Sample error of AdaBoost can be reduced arbitrarily (if the realizable assumption holds), by increasing the amount of training examples.

It can be shown [3] that an upper bound of the VC dimension of $L(B,T)$ is linear in the VC dimension of the hypothesis class B of the weak learner and also linear in T, i.e. $\text{VCdim}(L(B,T))$ is in $\mathcal{O}(\text{VCdim}(B)\cdot T)$. This means that if the VC dimension of B is finite, so is the VC dimension of $L(B,T)$.

In the case of B being the hypothesis class of all decision stumps, the VC dimension of B is 2, so it is clearly finite. It follows from the fundamental theorem of statistical learning, that when using decision stumps as the hypothesis class for the weak learner, AdaBoost is a PAC learner for the class $L(B,T)$. Furthermore, if the weak learner for B can be implemented efficiently (as we have shown for decision stumps), AdaBoost is also efficient.

3.2 The Expressive Power of L(B, T)

In section 3.1, we saw that the VC dimension of $L(B,T)$ grows linear in T and also linear in the VC dimension of B. If B is the class of decision stumps,

it follows that the VC dimension of $L(B,T)$ is in $\mathcal{O}(T)$, i.e. the expressive power of the class increases with the parameter T. To demonstrate what that means, we can take a look how $L(B,T)$ can be used to learn the class of piece-wise constant functions, with B being the class of decision stumps.

The class of piece-wise constant functions on the line ($\mathcal{X} = \mathbb{R}$) is given as follows:

$$\mathcal{G}_r = \left\{ x \mapsto \sum_{i=1}^{r} \alpha_i \mathbb{1}_{[x \in (\theta_{i-1}, \theta_i]]} : \ \alpha_i \in \{\pm 1\}, \ -\infty = \theta_0 < \theta_1 < ... < \theta_r = \infty \right\}$$

Here, $\theta_i \in \mathbb{R}$ are the thresholds and α_i determines the constant label $+1$ or -1 for each area between two thresholds. We will now show, that $L(B,T)$ can be used to learn the class of piece-wise constant functions with T pieces $\mathcal{G}_T$.

Theorem 1. *Let B be the class of decision stumps and $L(B,T)$ as defined above. Then $\mathcal{G}_T \subseteq L(B,T)$.*

Proof. Without loss of generality, let $g \in \mathcal{G}_T$ with $\alpha_t = (-1)^t$. We can assign the weight $w_1 = -0.5$ to the first decision stump in $L(B,T)$ and for $t > 1$ we can assign the weights $w_t = (-1)^t$ to the remaining decision stumps. If we choose θ_{t-1} as the θ parameter for decision stump t and $b = -1$, we get the following hypothesis:

$$h(x) = \text{sign}\left(\sum_{t=1}^{T} w_t \ \text{sign}(x - \theta_{t-1}) \right) \in L(B,T)$$

We can see that the first decision stump with threshold θ_0 and weight $w_1 = -0.5$ predicts the constant value -0.5 for every $x \in \mathbb{R}$. The other trees mirror the change of g for each threshold using the weights $w_t = (-1)^t$. It follows that if $g(x) = 1$, the weighted sum predicts 1.5 and for $g(x) = -1$, the weighted sum predicts -0.5. Thus, it follows that $h(x) = g(x) \ \forall x \in \mathbb{R}$, which concludes the proof. □

We have shown that $L(B,T)$ contains every function that is made up of T constant pieces. Since every labeled training sequence S that contains m examples can be perfectly classified by a function with m pieces, $L(B,T)$ contains every possible labeling of T data points in $\mathbb{R}$. This finding illustrates, why the VC dimension of $L(B,T)$ grows in T. In fact, if B is the class of decision stumps and $\mathcal{X} = \mathbb{R}$, we just showed that the VC dimension of $L(B,T)$ is at least T. From this, we can see that the parameter T directly influences the complexity of $L(B,T)$.

While an increasing model complexity of a hypothesis class will reduce the approximation error, the bias-complexity tradeoff [3] tells us that increasing complexity can also increase the model estimation error. Thus, a remarkable property of the class $L(B, T)$ and also of the AdaBoost algorithm is that the parameter T can be used to control the bias-complexity tradeoff. If, in a practical scenario, we wish to compare more complex models to simpler models, we could just see how the AdaBoost algorithm performs for different values of T and then make a decision how complex the model should be.

4 Conclusion

In this article, it was demonstrated how a weak learning algorithm can be boosted into a strong learner by applying the AdaBoost algorithm. The hypothesis class of decision stumps was identified to be efficiently learnable using an ERM algorithm. Thus, the AdaBoost algorithm is also efficient when using decision stumps as the base learner.

It was also shown that AdaBoost is a strong learner for the hypothesis class $L(B, T)$, which tells us that the true error of AdaBoost can be arbitrarily decreased by increasing the size of the training set (if the realizable assumption holds). Furthermore, it was demonstrated how the number of boosting rounds T can lead to arbitrarily complex models and and how it can be used to control the bias-complexity tradeoff.

All of these properties make AdaBoost a useful algorithm for practical applications of machine learning. It is a great example, of how a purely theoretical question has led to the creation of an algorithm that became hugely popular and is still widely used in practical applications.

References

[1] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997.

[2] M. Kearns and L. G. Valiant. Learning boolean formulae or finite automata is as hard as factoring. Technical Report TR 14-88, Harvard University Aiken Computation Laboratory, 1988.

[3] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014.