Boosting
Seminar: Foundations of Data Science

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

January 11, 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 [1, 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, 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 followed by a discussion of its implications as well as a practical example of how it can be used in the domain of image classification.

2 Weak Learnability

Assuming that the realizable assumption [1, 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 [1, 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 little bit about the underlying problem so that it performs better than random.

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 $f : \mathcal{X} \rightarrow \{\pm 1\}$ if running A on $m \geq m_{\mathcal{H}}$ 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.

It still remains the question of how to obtain a weak learning algorithm for a class $\mathcal{H}$. 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:

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

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 $\mathrm{ERM}_{\mathcal{H}_{DS}}$ 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 $\mathrm{ERM}_{\mathcal{H}_{DS}}$ hat 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 [1] 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 shown 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 procedcure 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 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 AdaBoost invokes the weak learner on the training data. Then the resulting output hypothesis of AdaBoost has the following form:

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

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 $\frac{1}{2} - \gamma$.

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 [1], 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 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:

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

The fundamental theorem of statistical learning [1, chapter 6] states that a hypothesis class is PAC-learnable if its VC dimension is finite. This means that if the VC dimension of $L(B,T)$ is finite, there exists a threshold $m_{\mathcal{H}}(\epsilon,\delta) \in \mathbb{N}$ for every $\epsilon, \delta \in (0,1)$, such that $L_{\mathcal{D}}(L(B,T)) \leq \epsilon$ (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 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 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. 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.

4 Conclusion

Even after centuries of research in the field of data science, there is nothing more versatile than the useful theorem of chapter 3. It is used everywhere and has led to the greatest and most intriguing results, cf. [2]. By the way, the book for the seminar [1] is a great reference and should be cited. Further literature can be found in the respective *Bibliographic Remarks* sections and of course you are welcome to search and add your own references.

Note: BibTEX entries can often be found in the DBLP collection. Google Scholar also offers BibTEX entries, which can be copied into the .bib file and may need some minor adjustments.

References

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

[2] J. Someone and J. Someoneelse. Useful theorems, 2003.