

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

22 "weak" learners can be combined in order to obtain a strong learning algo-
 23 rithm.

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

27 2 Weak Learnability

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

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

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

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

52 The parameter γ in Definition 1 tells us how well we can expect the weak
 53 learner to perform. For example, if a learning algorithm is a 1%-weak-learner
 54 for a class \mathcal{H} , then the generalization error can at most be 49% provided that
 55 first, the learner didn't fail (which can happen with probability δ of drawing
 56 a bad sample from \mathcal{D}) and second, the learner was run on at least $m_{\mathcal{H}}(\delta)$
 57 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 $S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m))$ 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:

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

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

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

107 3 AdaBoost

108 In the previous section, it was demonstrated how an efficient weak learner
 109 can be constructed by applying the ERM rule to the class of decision stumps.
 110 This section presents the AdaBoost (Adaptive Boosting) algorithm, a pro-
 111 cedure that shows how weak learners such as decision stumps can be used
 112 efficiently to find a hypothesis with an arbitrarily low empirical error $L_S(h)$
 113 on a training sequence S . The AdaBoost algorithm was first proposed in
 114 1995 by Yoav Freund and Robert Schapire [1] and became hugely popular
 115 due to its practicability.

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

$$121 \quad h(x) = \text{sign} \left(\sum_{t=1}^T w_t h_t(x) \right)$$

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

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

127 hypothesis is computed according to

$$128 \quad \epsilon_t = \sum_{i=1}^m D_i^{(t)} \mathbb{1}_{[h_t(\mathbf{x}_i) \neq y_i]}$$

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

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

$$132 \quad w_t = \frac{1}{2} \log \left(\frac{1}{\epsilon_t} - 1 \right)$$

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

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

$$137 \quad D_i^{(t+1)} = \frac{D_i^{(t)} \exp(-w_t y_i h_t(\mathbf{x}_i))}{\sum_{j=1}^m D_j^{(t)} \exp(-w_t y_j h_t(\mathbf{x}_j))}$$

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

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

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

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

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

$$154 \quad L_S(h) = \frac{1}{m} \sum_{i=1}^m \mathbb{1}_{[h(\mathbf{x}_i) \neq y_i]} \leq e^{-2\gamma^2 T}$$

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

158 In practical applications however, it is more important to also achieve
 159 a good out of sample error. The next section will show that the true risk
 160 $L_{\mathcal{D}}(h)$ of the AdaBoost hypothesis will also be small by taking a look at
 161 the hypothesis class that resembles all the possible output hypotheses of
 162 AdaBoost.

163 3.1 AdaBoost Out-of-Sample Performance

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

$$167 \quad L(B, T) = \left\{ x \mapsto \text{sign} \left(\sum_{t=1}^T w_t h_t(x) \right) : w \in \mathbb{R}^T, h_t \in B \right\}$$

168 The fundamental theorem of statistical learning [3, chapter 6] states that
 169 a hypothesis class is PAC-learnable if and only if its VC dimension is finite,
 170 i.e. there exists a threshold m , such that for every set with more than m
 171 elements there exists a labeling function that cannot be learned using that
 172 class. This means that if the VC dimension of $L(B, T)$ is finite and thus
 173 $L(B, T)$ is PAC-learnable, then, by the definition of PAC learning, there
 174 exists a threshold $m_{\mathcal{H}}(\epsilon, \delta) \in \mathbb{N}$ for every $\epsilon, \delta \in (0, 1)$, such that the true
 175 error $L_{\mathcal{D}}(h)$ of the AdaBoost hypothesis $h \in L(B, T)$ is at most ϵ (with
 176 confidence $1 - \delta$) for every distribution \mathcal{D} when training AdaBoost on at
 177 least $m_{\mathcal{H}}(\epsilon, \delta)$ training examples. This tells us that if $L(B, T)$ is learnable,
 178 even the Out-of-Sample error of AdaBoost can be reduced arbitrarily (if the
 179 realizable assumption holds), by increasing the amount of training examples.

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

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

190 3.2 The Expressive Power of $L(B, T)$

191 In section 3.1, we saw that the VC dimension of $L(B, T)$ grows linear in T
 192 and 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:

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

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 tree 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. \square

4 Conclusion

In this article, it was demonstrated how a weak learning algorithm can be boosted into a strong PAC learner by using 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.

218 Furthermore, it was also shown that AdaBoost is a PAC learner for the
219 hypothesis class $L(B, T)$, which tells us that the true error of AdaBoost
220 can be arbitrarily decreased by increasing the size of the training set (if the
221 realizable assumption holds).

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

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.