

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.

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

26 2 Weak Learnability

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

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

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

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

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

57 It still remains the question of how to obtain a weak learning algorithm
 58 for a class \mathcal{H} . We already saw that applying the ERM rule to complex hy-
 59 pothesis 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:

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

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

92 examples for a fixed dimension $j \in [d]$ so that $x_{1,j} \leq x_{2,j} \leq \dots \leq x_{m,j}$, we
 93 can see that we only have to consider a single splitting threshold in between
 94 two examples, which leaves us with $m + 1$ values for θ that the $\text{ERM}_{\mathcal{H}_{DS}}$ has
 95 to consider for a fixed dimension $j \in [d]$. Since the sum of misclassifications
 96 can be computed in $\mathcal{O}(m)$ by passing through the data once, the algorithm
 97 can find optimal values for j and θ in $\mathcal{O}(dm^2)$ by simple enumeration.

98 This time complexity can be reduced even more by avoiding to recalculate
 99 the sum of misclassifications for every new value of θ . It can be shown [1] that
 100 it is possible to update the sum for a new value of θ in constant time, requiring
 101 only a single pass through the data for a fixed dimension j . This leaves us
 102 with a time complexity of $\mathcal{O}(dm)$ of solving the optimization problem after
 103 the sorting step is applied as preprocessing.

104 3 AdaBoost

105 In the previous section it was shown how an efficient weak learner can be
 106 constructed by applying the ERM rule to the class of decision stumps. This
 107 section presents the AdaBoost (Adaptive Boosting) algorithm, a procedure
 108 that shows how weak learners such as decision stumps can be used efficiently
 109 to find a hypothesis with an arbitrarily low empirical error $L_S(h)$ on a training
 110 sequence S .

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

$$116 \quad h(x) = \text{sign} \left(\sum_{t=1}^T w_t h_t(x) \right)$$

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

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

$$123 \quad \epsilon_t = \sum_{i=1}^m D_i^{(t)} \mathbb{1}_{[h_t(\mathbf{x}_i) \neq y_i]}$$

124 and is at most $\frac{1}{2} - \gamma$.

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

$$127 \quad w_t = \frac{1}{2} \log \left(\frac{1}{\epsilon_t} - 1 \right)$$

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

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

$$132 \quad D_i^{(t+1)} = \frac{D_i^{(t)} \exp(-w_t y_i h_t(\mathbf{x}_i))}{\sum_{j=1}^m D_j^{(t)} \exp(-w_t y_j h_t(\mathbf{x}_j))}$$

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

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

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

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

146 On top of that, it can be shown [1], that the training error $L_S(h)$ of the
 147 AdaBoost hypothesis h decreases exponentially in the number of boosting
 148 rounds T :

$$149 \quad L_S(h) = \frac{1}{m} \sum_{i=1}^m \mathbb{1}_{[h(\mathbf{x}_i) \neq y_i]} \leq e^{-2\gamma^2 T}$$

150 Here, γ describes the γ -weak-learner as defined in Definition 1. This means,
 151 that AdaBoost will achieve an arbitrarily low training error and is still an
 152 efficient algorithm.

153 In practical applications however, it is more important to also achieve
 154 a good out of sample error. The next section will show that the true risk
 155 $L_{\mathcal{D}}(h)$ of the AdaBoost hypothesis will also be small by taking a look at
 156 the hypothesis class that resembles all the possible output hypotheses of
 157 AdaBoost.

158 3.1 AdaBoost Out-of-Sample Performance

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

$$162 \quad L(B, T) = \left\{ x \mapsto \text{sign} \left(\sum_{t=1}^T w_t h_t(x) \right) : w \in \mathbb{R}^T, h_t \in B \right\}$$

163 The fundamental theorem of statistical learning [1, chapter 6] states
164 that a hypothesis class is PAC-learnable if its VC dimension is finite. This
165 means that if the VC dimension of $L(B, T)$ is finite, there exists a threshold
166 $m_{\mathcal{H}}(\epsilon, \delta) \in \mathbb{N}$ for every $\epsilon, \delta \in (0, 1)$, such that the true error $L_{\mathcal{D}}(h)$ of the
167 AdaBoost hypothesis $h \in L(B, T)$ is at most ϵ (with confidence $1 - \delta$) for
168 every distribution \mathcal{D} when training AdaBoost on at least $m_{\mathcal{H}}(\epsilon, \delta)$ training
169 examples. This tells us that even the Out-of-Sample error of AdaBoost can
170 be reduced arbitrarily (if the realizable assumption holds), by increasing the
171 amount of training examples.

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

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

182 4 Conclusion

183 In this article it was shown how a weak learning algorithm can be boosted
184 into a strong PAC learner using the AdaBoost algorithm. The hypothesis
185 class of decision stumps was identified to be efficiently learnable using an
186 ERM algorithm, so it followed that the AdaBoost algorithm is also efficient
187 using decision stumps as the base learner.

References

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