A quick look at AdaBoost

Derek Huang

March 15, 2020

1 Introduction

In this short article, we discuss the two-class AdaBoost algorithm as a specific case of forward stagewise boosting, where we minimize an empirical risk function where per-example loss is given by an appropriately weighted exponential loss function. We give a brief overview of forward stagewise additive modeling, consider the two-class AdaBoost problem and derive expressions for the training example and base learner weight updates, and give an explicit algorithm for the two-class AdaBoost algorithm. Included is a brief appendix on the limiting behavior of AdaBoost as number of iterations $M \to \infty$.

2 AdaBoost additive modeling

2.1 Fitting additive models

Informally, a general additive model $F : X \to Y$ for classification or regression takes the form

$$F(\mathbf{x}) = \sum_{m=1}^{M} \phi_m(\mathbf{x}) \tag{2.1}$$

Commonly Y is $\mathbb{R}$, $\{-1, 1\}$, or some finite set with each element representing a class. Each of the functions $\phi_m : X \to Y$ is a simpler estimator, typically all of the same class. For example, if $\mathcal{H}_\theta$ was some space of decision trees with hyperparameter vector θ, then $\forall \phi_m$, $\phi_m \in \mathcal{H}_\theta$. The general method of training an additive model F is a process called *forward stagewise additive modeling*, where each ϕ_m is trained sequentially in a greedy manner, so given F_{m-1}, a partially fit F at iteration m, ϕ_m is given by

$$\phi_m = \arg\min_{\phi} \mathbb{E}_{\mathbf{x},\, y}[L(y,\, F_{m-1}(\mathbf{x}) + \phi(\mathbf{x}))] \tag{2.2}$$

Here we assumed no regularization term, with $L : Y \times \mathbb{R} \to [0, \infty)$ a per-example loss function. In general, this is not a simple computation problem, but we can show how AdaBoost solves this analytically for the binary classification problem by its particular choice of loss function L and simple learners ϕ_m.

2.2 The AdaBoost problem

Consider the binary classification problem, with positive label 1 and negative label -1, with a definition of L as the exponential loss function $L(y,\, \hat{y}) \triangleq e^{-y\hat{y}}$, where we define each ϕ_m as $\phi_m(\mathbf{x}) = \alpha_m f_m(\mathbf{x})$, a base learner $f_m : X \to \{-1,\, 1\}$, $f_m \in \mathcal{H}_\theta$, weighted with $\alpha_m \in \mathbb{R}$. Intuitively, if f_m is more accurate, than its weight should be higher in the final classifier $G : X \to \{-1,\, 1\}$, where we define

$$G(\mathbf{x}) = \operatorname{sgn}\left(\sum_{m=1}^{M} \alpha_m f_m(\mathbf{x})\right) = \operatorname{sgn}(F(\mathbf{x})) \tag{2.3}$$

Here $F : X \to \mathbb{R}$ is the discriminant function, where we predict the positive class if $F(\mathbf{x}) > 0$, or the negative class when $F(\mathbf{x}) < 0$. Our optimization problem to solve during training is the stagewise training of F, as

we want to include the effect of prediction confidence in our loss function, which cannot be reflected if we are outputting only -1 or 1. From our definition of L and ϕ_m, we can rewrite (2.2) more specifically as

$$\min_{\alpha,\, f} \mathbb{E}_{\mathbf{x},\, y}\left[e^{-y(F_{m-1}(\mathbf{x})+\alpha f(\mathbf{x}))}\right] \tag{2.4}$$

Since it is clear that $e^{-yF_{m-1}(\mathbf{x})}$ is not affected by our choice of α or f, we can define an mth iteration per-example weight $w_m \in \mathbb{R}_+$, $w_m \triangleq w_m(\mathbf{x},\, y) = e^{-yF_{m-1}(\mathbf{x})}$. We can then write (2.4) as the simplified

$$\min_{\alpha,\, f} \mathbb{E}_{\mathbf{x},\, y|w_m}\left[w_m e^{-y\alpha f(\mathbf{x})}\right] = \min_{\alpha,\, f} \mathbb{E}_{\mathbf{x},\, y}\left[L_{w_m}(y,\, \alpha f(\mathbf{x}))\right] \tag{2.5}$$

Note that this expectation has now become conditionally weighted on the mth iteration weights. Therefore, given a training set $\mathcal{D} \subset X \times Y$ of training examples $(\mathbf{x}_k,\, y_k)$, each with a corresponding mth-iteration weight $w_m^{(k)}$, we can write the mth-iteration empirical risk $J_m : \mathcal{H}_\theta \times \mathbb{R} \times 2^{X\times Y} \to [0,\, \infty)$ as

$$J_m(f,\, \alpha,\, \mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{k=1}^{|\mathcal{D}|} w_m^{(k)} e^{-y_k \alpha f(\mathbf{x}_k)} \tag{2.6}$$

We now define $I(f) \triangleq \{k \in \mathbb{N} : y_k = f(\mathbf{x}_k),\, (\mathbf{x}_k,\, y_k) \in \mathcal{D}\}$, $I^{\mathrm{c}}(f) \triangleq \{k \in \mathbb{N} : y_k \neq f(\mathbf{x}_k),\, (\mathbf{x}_k,\, y_k) \in \mathcal{D}\}$, i.e. index sets of correctly and incorrectly classified training examples, respectively. Since for any $(\mathbf{x}_k,\, y_k) \in \mathcal{D}$, $y_k \alpha f(\mathbf{x}_k) = \alpha\left(\mathbb{I}_{\{y_k = f(\mathbf{x}_k)\}} - \mathbb{I}_{\{y_k \neq f(\mathbf{x}_k)\}}\right)$, and $|\mathcal{D}| = |I(f)| + |I^{\mathrm{c}}(f)|$ necessarily, we can rewrite (2.6) as

$$J_m(f,\, \alpha,\, \mathcal{D}) = \frac{1}{|\mathcal{D}|}\left[\sum_{k\in I(f)} w_m^{(k)} e^{-\alpha} + \sum_{k\in I^{\mathrm{c}}(f)} w_m^{(k)} e^{\alpha}\right] = \frac{1}{|\mathcal{D}|}\left[\sum_{k=1}^{|\mathcal{D}|} w_m^{(k)} e^{-\alpha} + \sum_{k\in I^{\mathrm{c}}(f)} w_m^{(k)}(e^{\alpha} - e^{-\alpha})\right]$$

It is now clear that the solution f_m to $\min_f J_m(f,\, \alpha,\, \mathcal{D})$ is the f_m that solves $\min_f \sum_{k\in I^{\mathrm{c}}(f)} w_m^{(k)}$, which, if properly normalized, is the weighted misclassification rate of the estimator f on training set $\mathcal{D}$, given sample weights $w_m^{(1)}, \ldots w_m^{(|\mathcal{D}|)}$. Now that we have determined our optimal f_m, we still need to find the α_m that solves $\min_\alpha J(f_m,\, \alpha,\, \mathcal{D})$. Noting that (2.6) is convex in α, it is then clear that the solution α_m to $\min_\alpha J_m(f_m,\, \alpha,\, \mathcal{D})$ is unique, with $\partial J_m/\partial\alpha = 0$. We can compute $\partial J_m/\partial\alpha$ analytically at α_m, obtaining

$$\frac{\partial J_m}{\partial \alpha}(f_m,\, \alpha_m,\, \mathcal{D}) = \frac{1}{|\mathcal{D}|}\left[-\sum_{k\in I(f_m)} w_m^{(k)} e^{-\alpha_m} + \sum_{k\in I^{\mathrm{c}}(f_m)} w_m^{(k)} e^{\alpha_m}\right] = 0$$

Multiplying out $|\mathcal{D}|$, rearranging, taking the natural logarithm and using properties of logarithms, we have

$$\alpha_m + \log\left(\sum_{k\in I^{\mathrm{c}}(f_m)} w_m^{(k)}\right) = -\alpha_m + \log\left(\sum_{k\in I(f_m)} w_m^{(k)}\right)$$

Therefore, our expression for α_m in terms of sample weights $w_m^{(1)}, \ldots w^{(|\mathcal{D}|)}$ is simply

$$\alpha_m = \frac{1}{2}\log\left(\frac{\sum_{k\in I(f_m)} w_m^{(k)}}{\sum_{k\in I^{\mathrm{c}}(f_m)} w_m^{(k)}}\right) \tag{2.7}$$

Defining the misclassification rate ε_m of f_m as $\varepsilon_m \triangleq \sum_{k\in I^{\mathrm{c}}(f_m)} w_m^{(k)} / \sum_{k=1}^{|\mathcal{D}|} w_m^{(k)}$, we can write (2.7) as

$$\alpha_m = \frac{1}{2}\log\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right)$$

Having solved the mth-iteration problem $\min_{\alpha,\, f} J_m(f,\, \alpha,\, \mathcal{D})$, with J_m as defined in (2.6), we can now write down the binary classification AdaBoost algorithm, which we do in the next subsection.

2.3 The AdaBoost algorithm

In the previous subsection, we proceeded inductively in our derivation, assuming that for our training set $\mathcal{D}$, we already had a partial model F_{m-1}, $1 \leq m-1 < M$, and focused on solving the problem, for every iteration m, by choosing $\alpha_m \in \mathbb{R}$, $f_m \in \mathcal{H}_\theta$, such that $\alpha_m, f_m = \arg\min_{\alpha, f} J(f, \alpha, \mathcal{D})$. Having found an expression for α_m and a criterion for choosing f_m, we still need to put all our steps together.

The first step is to determine how to initialize our model and original weights $w_1^{(1)}, \ldots w_1^{(|\mathcal{D}|)}$. Consider (2.4) again. At $m = 1$, the first iteration, we have not yet trained a single f, so it follows that a natural choice for F_0 is $F_0 \triangleq 0$. Setting F_0 as 0 also has the effect of defining, $\forall k \in \{1, \ldots |\mathcal{D}|\}$, $w_1^{(k)} = 1$. In other words, we equally weight each of the training examples $(\mathbf{x}_k, y_k)$, as in typical unweighted estimator fitting.

After determining initial values for the model and example weights, the training and selection of f_m and α_m for any iteration m is straightforward. But how are the weights updated after each iteration's selection of f_m and α_m? Consider iteration m, where the per-example weight is $w_m^{(k)} \triangleq e^{-y_k F_{m-1}(\mathbf{x}_k)}$. After selection of f_m and α_m, we have a model F_m, which implies that our per-example weight $w_{m+1}^{(k)}$ for iteration $m+1$ is

$$\begin{aligned} w_{m+1}^{(k)} &= e^{-y_k F_m(\mathbf{x}_k)} = e^{-y_k(F_{m-1}(\mathbf{x}_k) + \alpha_m f_m(\mathbf{x}_k))} = w_m^{(k)} e^{-y_k \alpha_m f_m(\mathbf{x}_k)} \\ &= w_m^{(k)} e^{-\alpha_m\left(\mathbb{I}_{\{y_k = f_m(\mathbf{x}_k)\}} - \mathbb{I}_{\{y_k \neq f_m(\mathbf{x}_k)\}}\right)} = w_m^{(k)} e^{\alpha_m\left(2\mathbb{I}_{\{y_k \neq f_m(\mathbf{x}_k)\}} - 1\right)} \end{aligned}$$

For numerical reasons, in practice the weights $w_m^{(k)}$ are chosen so that $\sum_{k=1}^{|\mathcal{D}|} w_m^{(k)} = 1$. Denoting unnormalized weights by $\widetilde{w}_m^{(k)}$, re-normalizing the weights to equal 1 every iteration can be seen to modify (2.5) to be

$$\min_{\alpha, f} \frac{1}{Z_m} \mathbb{E}_{\mathbf{x}, y | \widetilde{w}_m} \left[\widetilde{w}_m e^{-y\alpha f(\mathbf{x})} \right]$$

Here Z_m is the mth-iteration constant normalizing the weights $\widetilde{w}_m$ such that they sum up to 1. Note that being a constant factor for each iteration, Z_m does not affect the dynamics of our minimization problem.

We may now write the algorithm for two-class AdaBoost as follows.

Algorithm: Two-class AdaBoost

1. Initialize weights $w_1^{(1)}, \ldots w_1^{(|\mathcal{D}|)}$ to $1 \,/\, |\mathcal{D}|$ and define the initial score function $F_0 \triangleq 0$.
2. For each boosting stage $m = 1, \ldots M$,
   (a) Train a classifier $f_m : X \to \{-1, 1\}$, $f_m \in \mathcal{H}_\theta$, such that
   $$f_m = \arg\min_f \sum_{k \in I^c(f)} w_m^{(k)}$$
   Here $I^c(f) \triangleq \{k \in \mathbb{N} : y_k \neq f(\mathbf{x}_k), (\mathbf{x}_k, y_k) \in \mathcal{D}\}$ is an index set.
   (b) Compute $\varepsilon_m \triangleq \sum_{k \in I^c(f_m)} w_m^{(k)}$, and choose $\alpha_m \in \mathbb{R}$ such that
   $$\alpha_m = \frac{1}{2} \log\left(\frac{1 - \varepsilon_m}{\varepsilon_m}\right)$$
   Define the new score function F_m, where $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \alpha_m f_m(\mathbf{x})$.
   (c) If $m < M$, define new weights $\widetilde{w}_m^{(k)}$ by the relation
   $$\widetilde{w}_m^{(k)} = w_m^{(k)} e^{\alpha_m\left(2\mathbb{I}_{\{y_k \neq f_m(\mathbf{x}_k)\}} - 1\right)}$$
   Define $Z_m \triangleq \sum_{k=1}^{|\mathcal{D}|} \widetilde{w}_m^{(k)}$ and compute normalized weights $w_{m+1}^{(k)}$ for iteration $m+1$, where
   $$w_{m+1}^{(k)} = \frac{\widetilde{w}_m^{(k)}}{Z_m}$$
3. Output our new "boosted" classifier $G : X \to \{-1, 1\}$, where
   $$G(\mathbf{x}) = \operatorname{sgn}(F(\mathbf{x})) = \operatorname{sgn}\left(\sum_{m=1}^{M} \alpha_m f_m(\mathbf{x})\right)$$

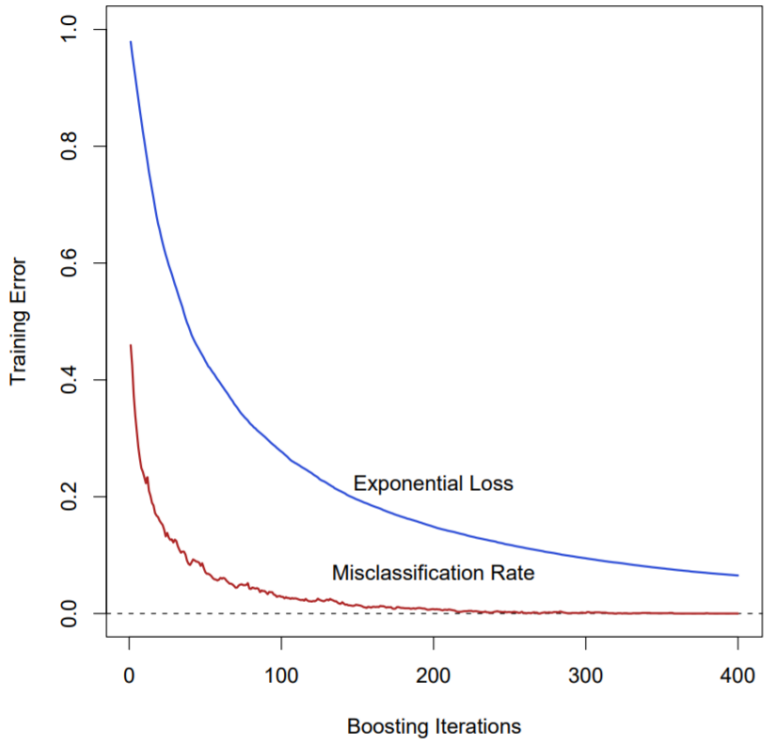
3 Appendix

3.1 Behavior as $M \to \infty$

Assuming that for any iteration m, the weights $w_m^{(k)}$ for examples $(\mathbf{x}_k, y_k) \in \mathcal{D}$ are properly normalized, we can define at each iteration a discrete distribution of weights over the training set $\mathcal{D}$, with an associated probability mass function $p_m : \{1, \dots |\mathcal{D}|\} \times \mathcal{D} \to [0, 1]$. Given a training data set $\mathcal{D}$ and an index k in that set, p_m returns the associated mth iteration weight $w_m^{(k)}$ of the kth training example. To achieve some clarity on how the final learner G behaves as M increases by considering limiting behavior as $M \to \infty$, we consider the following simple scenario. Define a final "terminal" weight distribution with an associated probability mass function p_∞. If we suppose that $p_m \to p_\infty$ uniformly, or more precisely, that $\exists M' \in \mathbb{N}$ such that $\forall m \geq M'$, $\forall k, \mathcal{D} \in \{1, \dots |\mathcal{D}|\} \times 2^{X \times Y}$, $|p_m(k, \mathcal{D}) - p_\infty(k, \mathcal{D})| < \varepsilon$, $\forall \varepsilon > 0$, then it follows that $\forall m \geq M'$, $\phi_m = \alpha_{M'} f_{M'}$. In other words, from iteration M' onwards, each function f_m added is the same as the function $f_{M'}$, and given the weight $\alpha_{M'}$. Then, given some $M > M'$, we can write F as

$$F(\mathbf{x}) \triangleq \sum_{m=1}^{M} \alpha_m f_m(\mathbf{x}) = \sum_{m=1}^{M'-1} \alpha_m f_m(\mathbf{x}) + \sum_{n=M'}^{M} \alpha_{M'} f_{M'}(\mathbf{x}) = \sum_{m=1}^{M'-1} \alpha_m f_m(\mathbf{x}) + (M - M' + 1)\alpha_{M'} f_{M'}(\mathbf{x})$$

Assuming $\alpha_{M'} > 0$, it becomes clear that as $M \to \infty$, $G \to f_{M'}$ uniformly. Intuitively, it is then clear that given the training data set $\mathcal{D}$ and a very large number of iterations M, that AdaBoost will eventually find a learner $f_{M'}$ at an iteration $M' < M$ that achieves the minimum misclassification rate possible on $\mathcal{D}$. An equivalent statement is that as the number of iterations increases, the AdaBoost training error will decrease to the minimum possible misclassification rate on the training set $\mathcal{D}$. However, this property can be explained by the fact that the AdaBoost algorithm minimizes exponential loss, an upper bound to misclassification error. Figure 10.3 from page 345 of the second edition of *The Elements of Statistical Learning*, reproduced below, shows this clearly in the case of boosting with decision tree stumps as base estimators.[1] Although misclassification rate reaches zero, average exponential loss is still decreasing.



[1] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning* (2nd ed.). Springer.