Machine Learning — Statistical Methods for Machine Learning

Boosting and ensemble methods

Instructor: *Nicolò Cesa-Bianchi* version of June 1, 2022

Ensemble methods are used to form combinations of predictors that achieve a bias-variance trade-off better than the one achieved by the algorithm generating the predictors in the combination. Many stochastic gradient descent algorithms, like Pegasos, can be viewed as ensemble methods because they output a combination of all the predictors generated during the sequential run over the training set. We now look at ensemble methods that are not based on online algorithms. As usual, our focus is on binary classification.

Fix a training set $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)$ for a binary classification problem with zero-one loss, and assume an ensemble of classifiers $h_1, \ldots, h_T$ is available (later, we describe how to obtain the ensemble). Consider the majority classifier f defined by

$$f(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{i=1}^{T} h_i(\boldsymbol{x})\right) \tag{1}$$

Clearly, f is wrong on $\boldsymbol{x}$ if and only if at least half of the classifiers $h_1, \ldots, h_T$ are wrong on $\boldsymbol{x}$ (assume T is odd to avoid ties). We now study the conditions under which the majority classifier achieves a small training error. Assume

$$\mathbb{P}\big(h_1(\boldsymbol{x}_Z) \neq y_Z \wedge \cdots \wedge h_T(\boldsymbol{x}_Z) \neq y_Z\big) = \prod_{i=1}^{T} \mathbb{P}\big(h_i(\boldsymbol{x}_Z) \neq y_Z\big) \tag{2}$$

where Z is a random variable uniformly distributed on the set $\{1, \ldots, m\}$ of indices of training examples. In other words, each classifier is wrong independently of the others with respect to the uniform distribution over the training set. The indicator functions of these events define the training error $\ell_S(h_i)$ of each classifier h_i. Indeed,

$$\ell_S(h_i) = \frac{1}{m}\sum_{t=1}^{m} \mathbb{I}\{h_i(\boldsymbol{x}_t) \neq y_t\} = \mathbb{P}\big(h_i(\boldsymbol{x}_Z) \neq y_Z\big)$$

The majority classifier (1) is a simple example of an *ensemble method*, combining the predictions of an ensemble of classifiers in order to boost the accuracy. We now bound the training error of f. Introduce

$$\ell_{\text{ave}} = \frac{1}{T}\sum_{i=1}^{T} \ell_S(h_i)$$

Without loss of generality, we can assume that $\ell_S(h_i) \leq \frac{1}{2}$ for each $i = 1, \ldots, T$. This implies

$\ell_{\text{ave}} \leq \frac{1}{2}$ and we can write

$$\begin{aligned}
\ell_S(f) &= \mathbb{P}\big(f(\boldsymbol{x}_Z) \neq y_Z\big) \\
&= \mathbb{P}\left(\sum_{i=1}^{T} \mathbb{I}\{h_i(\boldsymbol{x}_Z) \neq y_Z\} > \frac{T}{2}\right) \\
&= \mathbb{P}\left(\frac{1}{T}\sum_{i=1}^{T} \mathbb{I}\{h_i(\boldsymbol{x}_Z) \neq y_Z\} > \ell_{\text{ave}} + \left(\frac{1}{2} - \ell_{\text{ave}}\right)\right)
\end{aligned}$$

Now introduce the Bernoulli random variables $B_1, \ldots, B_m$ defined by $B_i = \mathbb{I}\{h_i(\boldsymbol{x}_Z) \neq y_Z\}$. Note that these random variables are independent, due to our assumption (2). Also, $\mathbb{E}[B_i] = \ell_S(h_i)$ and

$$\frac{1}{T}\sum_{i=1}^{T} \mathbb{E}[B_i] = \ell_{\text{ave}}$$

Let $\varepsilon = \frac{1}{2} - \ell_{\text{ave}} \geq 0$. A slight generalization of Chernoff-Hoeffding bounds to independent random variables with unequal expectations gives

$$\mathbb{P}\left(\frac{1}{T}\sum_{i=1}^{T} B_i > \ell_{\text{ave}} + \varepsilon\right) \leq e^{-2\varepsilon^2 T}$$

By setting $\gamma_i = \frac{1}{2} - \ell_S(h_i) > 0$ we get

$$\ell_S(f) \leq \exp\left(-2T\left(\frac{1}{2} - \ell_{\text{ave}}\right)^2\right) = \exp\left(-2T\left(\frac{1}{T}\sum_{i=1}^{T}\gamma_i\right)^2\right) \leq e^{-2T\gamma^2} \tag{3}$$

where for the last inequality we assumed $\gamma_i \geq \gamma > 0$ for all $i = 1, \ldots, T$.

This result tells us that if we manage to obtain classifiers with independent training errors in the sense of (2), then the training error of the majority vote classifier decreases exponentially with respect to $T\gamma^2$, where γ measures how better than random guessing is each classifier h_i on the training set. Note that being able to reduce the training error on arbitrary datasets implies a decrease of the bias error.

Bagging. How can we obtain classifiers with independent training errors? A popular heuristic, known as Bagging, applies to any learning algorithm A for binary classification and to any training set S. Let m be the size of S. Bagging builds $h_1, \ldots, h_T$ by drawing m examples uniformly at random with replacement from S. This process is repeated T times so to obtain the resampled training sets $S_1, \ldots, S_T$. Then A is run on each S_i setting $h_i = A(S_i)$. The idea is that the resampling procedure helps enforce condition (2).

One may wonder how different from S any S_i can be. To find that out, we take a little detour and compute the fraction of unique data points in S_i. As you see in a moment, more than one third of the points of S are missing from S_i in expectation! Let N be the number of unique points drawn, and let X_t be the indicator function of the event that $(\boldsymbol{x}_t, y_t)$ is drawn. Then the probability that $(\boldsymbol{x}_t, y_t)$ is not drawn is

$$\mathbb{P}(X_t = 0) = \left(1 - \frac{1}{m}\right)^m$$

So we have

$$\mathbb{E}[N] = \sum_{t=1}^{m} \mathbb{E}[X_t] = \sum_{t=1}^{m} \mathbb{P}(X_t = 1) = \sum_{t=1}^{m} \left(1 - \left(1 - \frac{1}{m}\right)^m\right) = m - m\left(1 - \frac{1}{m}\right)^m$$

Therefore, the fraction of unique points in S_i is

$$1 - \left(1 - \frac{1}{m}\right)^m \approx 1 - \frac{1}{e} = 0.632\ldots$$

where the approximation becomes exact for $m \to \infty$.

We saw that independence of errors helps reduce the bias by driving the training error to zero. On the other hand, subsampling of the training set helps reduce the variance. If we think of the m training points arranged in a $m \times d$ matrix (called the data matrix), then what bagging does is subsampling the rows of this matrix. We now briefly describe another ensemble method that increases the protection against overfitting by also subsampling the columns of the data matrix.

Random Forest. This ensemble method works by taking a majority vote over an ensemble $h_1, \ldots, h_T$ of tree predictors. Similarly to bagging, each tree predictor h_i is obtained by running a learning algorithm over a dataset S_i obtained by subsampling the rows of the full data matrix. However, the algorithm for learning tree predictors does not have direct access to S_i. Indeed, when considering a leaf ℓ for splitting, instead of being given $S_{i,\ell}$ (the set of training examples in S_i that are routed to ℓ), the algorithm has access to a version of $S_{i,\ell}$ containing a random subset of the original features (typically, $\sqrt{d}$ features are sampled from the original d features). This additional sampling provides a better control on the variance at the expense of the bias. Because of its good performance on many learning tasks, Random Forest is often used as a baseline when testing a new learning algorithm.

Boosting. We now introduce boosting, a principled ensemble method that achieves the exponential bound (3) on the training error without requiring the demanding condition (2). Boosting is an incremental method to build classifiers of the form $\operatorname{sgn}(f)$, where

$$f = \sum_{i=1}^{T} w_i h_i$$

and $\boldsymbol{w} = (w_1, \ldots, w_T)$ is a vector of real coefficients. We assume $h_1, \ldots, h_T$ are generated by some learning algorithm A and belong to some family $\mathcal{H}$ of **base classifiers**.

In practice, also in order to save computational costs, base classifiers are very simple. A typical choice for $\mathcal{H}$ is that of **decision stumps**. These are all classifiers of the form $h_{i,\tau} : \mathbb{R}^d \to \{-1, 1\}$ defined by $h_{i,\tau}(\boldsymbol{x}) = \pm \operatorname{sgn}(x_i - \tau)$, where $i = 1, \ldots, d$ and $\tau \in \mathbb{R}$.

The specific boosting algorithm we introduce is known as **AdaBoost** (adaptive boosting). Fix a training set S with m examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)$, and a sequence $h_1, \ldots, h_T$ of base classifiers. We now show how to choose the coefficients $\boldsymbol{w}$ so that the training error is bounded as in (3).

AdaBoost uses the convex upper bound $\mathbb{I}\{z \leq 0\} \leq e^{-z}$ on the zero-one loss function $\mathbb{I}\{f(\boldsymbol{x}_t)\, y_t \leq 0\}$.

This gives

$$\ell_S(f) = \frac{1}{m}\sum_{t=1}^{m} \mathbb{I}\{f(\boldsymbol{x}_t)y_t \leq 0\} \leq \frac{1}{m}\sum_{t=1}^{m} e^{-f(\boldsymbol{x}_t)y_t} = \frac{1}{m}\sum_{t=1}^{m} e^{-\sum_{i=1}^{T} w_i h_i(\boldsymbol{x}_t)y_t}$$

Note that other algorithms use different convex upper bounds on the zero-one loss. For example, SVM uses the hinge loss.

Introduce now the functions $L_1, \ldots, L_T$ defined by $L_i(t) = h_i(\boldsymbol{x}_t)y_t$. Note that $L_i(t) \in \{-1, 1\}$ and $L_i(t) = 1$ if and only if $h_i(\boldsymbol{x}_t) = y_t$. Recalling that Z is a random variable uniformly distributed in $\{1, \ldots, m\}$, we can view each $L_i(Z)$ as a random variable and write

$$\ell_S(f) \leq \frac{1}{m}\sum_{t=1}^{m} e^{-\sum_{i=i}^{T} w_i L_i(t)} = \mathbb{E}\left[e^{-\sum_{i=i}^{T} w_i L_i(Z)}\right] = \mathbb{E}\left[\prod_{i=1}^{T} e^{-w_i L_i(Z)}\right]$$

If condition (2) were true, we could write the expectation of the product as a product of expectations. In order to sidestep the condition, we change the probability space and write

$$\mathbb{E}\left[\prod_{i=1}^{T} e^{-w_i L_i(Z)}\right] = \prod_{i=1}^{T} \mathbb{E}_i\left[e^{-w_i L_i(Z_i)}\right] \tag{4}$$

where each $Z_i \in \{1, \ldots, m\}$ is distributed according to a law $\mathbb{P}_i$ yet to be specified.

Assuming (4) holds, which we verify later, we can proceed as follows

$$\begin{aligned}
\ell_S(f) &\leq \prod_{i=1}^{T} \mathbb{E}_i\left[e^{-w_i L_i(Z_i)}\right] \\
&= \prod_{i=1}^{T} \left(e^{-w_i}\,\mathbb{P}_i(L_i(Z_i) = 1) + e^{w_i}\,\mathbb{P}_i(L_i(Z_i) = -1)\right) \\
&= \prod_{i=1}^{T} \left(e^{-w_i}(1 - \varepsilon_i) + e^{w_i}\varepsilon_i\right) \qquad (5)
\end{aligned}$$

where we set

$$\varepsilon_i \stackrel{\text{def}}{=} \mathbb{P}_i(L_i(Z_i) = -1) = \sum_{t=1}^{m} \mathbb{I}\{L_i(t) = -1\}\mathbb{P}_i(t)$$

Note that ε_i is the error of h_i with respect to the probability $\mathbb{P}_i$. Namely, ε_i is the weighted training error of h_i where the weights are determined by $\mathbb{P}_i$.

Before computing the $\mathbb{P}_i$, we show how to pick $w_1, \ldots, w_T$ in order to minimize (5). By computing the zeros of the derivative of $e^{-w}(1 - \varepsilon_i) + e^{w}\varepsilon_i$ with respect to w, we find a single zero at

$$w_i = \frac{1}{2}\ln\frac{1 - \varepsilon_i}{\varepsilon_i} \ .$$

Note that the above expression is only defined for $0 < \varepsilon_i < 1$. As we will see, $\mathbb{P}(t) > 0$ for all $t \in \{1, \ldots, m\}$. Hence $\varepsilon_i \in \{0, 1\}$ implies that either h_i or $-h_i$ has zero training error on S. If this

happens, than we can throw away all h_j for $j \neq i$ and avoid using boosting altogether. Therefore, without loss of generality we may assume $0 < \varepsilon_i < 1$ for all $i = 1, \ldots, m$.

Substituting in (5) and simplifying, we get

$$\ell_S(f) \leq \prod_{i=1}^{T} \sqrt{4\varepsilon_i(1-\varepsilon_i)} \; .$$

Note that $w_i = 0$ if and only if $\varepsilon_i = \frac{1}{2}$, meaning that the weight (according to $\mathbb{P}_i$) of the training points where h_i errs is exactly $\frac{1}{2}$. Because such a h_i does not affect the value of f (since $\varepsilon_i = \frac{1}{2}$ implies $w_i = 0$) without loss of generality we may also assume that $\varepsilon_i \neq \frac{1}{2}$.

Set $\gamma_i \stackrel{\text{def}}{=} \frac{1}{2} - \varepsilon_i$ and note that $\varepsilon_i \neq \frac{1}{2}$ implies $\gamma_i \neq 0$. Using the inequality $1 + x \leq e^x$, which holds for all $x \in \mathbb{R}$, we get

$$\ell_S(f) \leq \prod_{i=1}^{T} \sqrt{4\varepsilon_i(1-\varepsilon_i)} = \prod_{i=1}^{T} \sqrt{1-4\gamma_i^2} \leq \prod_{i=1}^{T} e^{-2\gamma_i^2} = e^{-2\sum_{i=1}^{T}\gamma_i^2} = e^{-2T\gamma^2}$$

where in the last step we assumed $\left|\gamma_i\right| > \gamma > 0$. This is the same bound as the one we proved in (3) under the condition (2). Note, however, that the definition of $\gamma_i = \frac{1}{2} - \varepsilon_i$ changes because ε_i now is the weighted training error of h_i.

Just like (3), this bound provides a pretty strong control on the bias. Using the observation that $\ell_S(f) = 0$ if and only if $\ell_S(f) < 1/m$, we conclude that a number

$$T > \frac{\ln m}{2\gamma^2}$$

of boosting rounds is sufficient to bring the training error of f down to zero.

We now move on to derive the $\mathbb{P}_1, \ldots, \mathbb{P}_T$ satisfying condition (4). Setting $\mathbb{P}_1 = \mathbb{P}$, $\mathbb{E}_1 = \mathbb{E}$, and

$$\mathbb{P}_{i+1}(t) = \frac{\mathbb{P}_i(t) e^{-w_i L_i(t)}}{\mathbb{E}_i\left[e^{-w_i L_i(Z_i)}\right]} \qquad \text{for } t = 1, \ldots, m \text{ and } i = 1, \ldots, T-1 \tag{6}$$

where

$$\mathbb{E}_i\left[e^{-w_i L_i(Z_i)}\right] = \sum_{s=1}^{m} e^{-w_i L_i(s)} \mathbb{P}_i(s)$$

It is easy to check that $\mathbb{P}_1, \ldots, \mathbb{P}_T$ are indeed probability distributions on $\{1, \ldots, m\}$. In particular, $\mathbb{P}_i(t) > 0$ and $\mathbb{P}_i(1) + \cdots + \mathbb{P}_i(m) = 1$.

For this choice of $\mathbb{P}_i$ we can prove (4) as follows. First, we solve (6) for $e^{-w_i L_i(t)}$ obtaining

$$e^{-w_i L_i(t)} = \mathbb{E}_i\left[e^{-w_i L_i(Z_i)}\right] \frac{\mathbb{P}_{i+1}(t)}{\mathbb{P}_i(t)}$$

Then, we write

$$\begin{aligned}
\mathbb{E}\left[\prod_{i=1}^{T} e^{-w_i L_i(Z_i)}\right] &= \frac{1}{m}\sum_{t=1}^{m}\left(\prod_{i=1}^{T}\mathbb{E}_i\left[e^{-w_i L_i(Z_i)}\right]\frac{\mathbb{P}_{i+1}(t)}{\mathbb{P}_i(t)}\right) \\
&= \frac{1}{m}\sum_{t=1}^{m}\left(\frac{\mathbb{P}_{T+1}(t)}{\mathbb{P}_1(t)}\right)\left(\prod_{i=1}^{T}\mathbb{E}_i\left[e^{-w_i L_i(Z_i)}\right]\right) \\
&= \left(\sum_{t=1}^{m}\frac{\mathbb{P}_{T+1}(t)}{\mathbb{P}_1(t)}\mathbb{P}_1(t)\right)\left(\prod_{i=1}^{T}\mathbb{E}_i\left[e^{-w_i L_i(Z_i)}\right]\right) \\
&= \prod_{i=1}^{T}\mathbb{E}_i\left[e^{-w_i L_i(Z_i)}\right]
\end{aligned}$$

concluding the proof.

These probability distributions have a simple interpretation when one studies how $\mathbb{P}_{i+1}$ depends on $\mathbb{P}_i$. Fix $\mathbb{P}_i$ and suppose $\varepsilon_i < \frac{1}{2}$. Then $w_i > 0$ and each $\mathbb{P}_{i+1}(t)$ is obtained multiplying $\mathbb{P}_i(t)$ for the quantity $e^{-w_i L_i(t)}$, which is bigger than 1 if and only if $h_i(\boldsymbol{x}_t) \neq y_t$. In other words, the weight of each training example $(\boldsymbol{x}_t, y_t)$ is increased when $\mathbb{P}_i$ is updated to $\mathbb{P}_{i+1}$ if and only if h_i errs on $(\boldsymbol{x}_t, y_t)$. Intuitively, the boosting process concentrates the weight on the training examples that are misclassified by the previous classifiers. A similar argument applies to the case when $\varepsilon_i > \frac{1}{2}$.

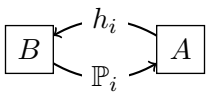
Input:	Training set S of examples $(\boldsymbol{x}, y) \in \mathbb{R}^d \times \{-1, 1\}$.
	Learning algorithm A.
	Maximum number T of boosting rounds.
Initialize	$\mathbb{P}_1(t) \leftarrow 1/m$ for $t = 1, \ldots, m$.

For $i = 1, \ldots, T$

1. Feed A with S weighted by $\mathbb{P}_i$ and get h_i in response
2. Compute ε_i for h_i
3. If $\varepsilon_i \in \left\{0, \frac{1}{2}, 1\right\}$ then BREAK
4. Let $w_i \leftarrow \frac{1}{2}\ln\frac{1-\varepsilon_i}{\varepsilon_i}$.
5. Compute $\mathbb{P}_{i+1}$ using (6).

If for loop exited on BREAK, then deal with the special case
Else output $f = \operatorname{sgn}(w_1 h_1 + \cdots + w_T h_T)$.

We are now ready to introduce the pseudo-code for AdaBoost. It is convenient to view the boosting process as sequence of rounds between the boosting algorithm and the learning algorithm A.



In each round i, the booster B gives $\mathbb{P}_i$ to A and gets h_i in response. If A return h_i such that $\varepsilon_i = \frac{1}{2}$, then the boosting process stops and the booster outputs $f = \operatorname{sgn}(w_1 h_1 + \cdots + w_{i-1} h_{i-1})$. If A return h_i such that $\varepsilon_i \in \{0, 1\}$, then the boosting process also stops, but in this case booster outputs $f = h_i$ or $f = -h_i$.