11 Boosting

Goal

Understand the ensemble method for combining classifiers. Bagging, Random Forest, and the celebrated Adaboost.

Alert 11.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.
This note is likely to be updated again soon.

Remark 11.2: Together and stronger?

Often it is possible to train a variety of different classifiers for a particular problem at hand, and a lot of time, energy and discussion are spent on debating and choosing the most appropriate classifier. This makes sense when the classifiers are “expensive” to obtain (be it computationally or financially or resourcefully).

Putting operational costs aside, however, is it possible to combine a bunch of classifiers and get better performance, for instance when compared against the best classifier in the gang? Of course, one usually does not know which classifier in the gang is “best” (unless when we try all of them out).

Remark 11.3: The power of aggregation

To motivate our development, let us consider an ideal scenario where we have a collection of classifiers $\{h_t : \mathcal{X} \to \{0,1\}, t = 1, 2, \ldots, 2T+1\}$ for a binary classification problem (where we encode the labels as $\{0,1\}$). Conditional on a given test sample (X, Y), we assume the classifiers h_t ~~independently~~ achieve accuracy

$$p := \Pr(h_t(X) = Y | X, Y) > \tfrac{1}{2}.$$

(For instance, if classifier h_t is trained on an independent dataset $\mathcal{D}_t$.) We predict according to the majority:

$$h(X) = \begin{cases} 1, & \text{if } \#\{t : h_t(X) = 1\} \geq T+1 \\ 0, & \text{if } \#\{t : h_t(X) = 0\} \geq T+1 \end{cases}.$$

What is the accuracy of this “meta-classifier” h? Simple calculation reveals:

$$\begin{aligned}
\Pr(h(X) = Y | X, Y) &= \Pr\left(\sum_{t=1}^{2T+1} [\![h_t(X) = Y]\!] \geq T+1 \,\middle|\, X, Y \right) = \sum_{k=T+1}^{2T+1} \binom{2T+1}{k} p^k (1-p)^{2T+1-k} \\
&= \Pr\left(\frac{1}{\sqrt{(2T+1)p(1-p)}} \sum_{t=1}^{2T+1} [[\![h_t(X) = Y]\!] - p] \geq \frac{T+1-p(2T+1)}{\sqrt{(2T+1)p(1-p)}} \,\middle|\, X, Y \right) \\
&\xrightarrow{T\to\infty} 1 - \Phi\left(\frac{T+1-p(2T+1)}{\sqrt{(2T+1)p(1-p)}} \right) \xrightarrow[2p>1]{T\to\infty} 1,
\end{aligned}$$

where $\Phi : \mathbb{R} \to [0,1]$ is the cdf of a standard normal distribution, and the convergence follows from the central limit theorem.

Therefore, provided that we can combine a large number of conditionally independent classifiers, each of which is slightly better than random guessing, we can approach perfect accuracy! The caveat of course is the difficulty (or even impossibility) in obtaining *decent* independent classifiers in the first place.

Exercise 11.4: Odd vs. even

Perform a similar analysis as in Remark 11.3 for $2T$ classifiers. Is there any advantage in using the odd $2T+1$

over the even $2T$?

Algorithm 11.5: Bootstrap Aggregating (Bagging, Breiman 1996)

One way to achieve (approximately) independent classifiers is to simply train them on independent datasets, which in most (if not all) applications is simply a luxury. However, we can use the same bootstrapping idea as in cross-validation (cf. Algorithm 2.29):

```
Algorithm: Bagging predictors.
  Input: training set $\mathcal{D}$, number of reptitions $T$
  Output: meta-predictor $h$
1 for $t = 1, 2, \ldots, T$ do                                    // in parallel
2     sample a new training set $\mathcal{D}_t \subseteq \mathcal{D}$        // with or without replacement
3     train predictor $h_t$ on $\mathcal{D}_t$
4 $h \leftarrow \mathsf{aggregate}(h_1, \ldots, h_T)$   // majority vote for classification; average for regression
```

There are two ways to sample a new training set:

- Sample with replacement: copy a (uniformly) random element from $\mathcal{D}$ to $\mathcal{D}_t$ and repeat $|\mathcal{D}|$ times. Usually there will be repetitions of the same element in $\mathcal{D}_t$ (which has no effect on most machine learning algorithms). *This is the common choice.*

- Sample without replacement: "cut" a (uniformly) random element from (a copy of) $\mathcal{D}$ to $\mathcal{D}_t$ and repeat say $70\% \times |\mathcal{D}|$ times. There will be no repetitions in each $\mathcal{D}_t$. Note that had we repeated $|\mathcal{D}|$ times (just as in sample with replacement) we would have $\mathcal{D}_t \equiv \mathcal{D}$, which is not very useful.

Of course the training sets $\{\mathcal{D}_t : t = 1, \ldots, T\}$ are not really independent of each other, but aggregating predictors trained on them usually (but not always) improves performance.

Breiman, Leo (1996). "Bagging Predictors". *Machine Learning*, vol. 24, no. 2, pp. 123–140.

Algorithm 11.6: Randomizing output (e.g. Breiman 2000))

Bagging perturbs the training set by taking a random subset. We can also perturb the training set by simply adding noise:

- for regression tasks: replace line 2 in the bagging algorithm by "adding small Gaussian noise to each response y_i"

- for classification tasks: replace line 2 in the bagging algorithm by "randomly flip a small portion of the labels y_i"

We will come back to this perturbation later. Intuitively, adding noise can prevent overfitting, and aggregating reduces variance.

Breiman, Leo (2000). "Randomizing outputs to increase prediction accuracy". *Machine Learning*, vol. 40, no. 3, pp. 229–242.

Algorithm 11.7: Random forest (Breiman 2001))

One of the most popular machine learning algorithms is random forest, where we combine bagging and decision trees. Instead of training a usual decision tree, we introduce yet another layer of randomization:

- during training, at each internal node where we need to split the training samples, we select a random subset of features and perform the splitting. A different subset of features is used at a different internal node.

Then we apply bagging and aggregate a bunch of the above "randomized" decision trees.

Breiman, Leo (2001). "Random Forest". *Machine Learning*, vol. 45, no. 1, pp. 5–32.

Algorithm 11.8: Hedging (Freund and Schapire 1997)

```
Algorithm: Hedging.
  Input: initial weight vector $\mathbf{w}_1 \in \mathbb{R}^{\mathsf{n}}_{++}$, discount factor $\beta \in [0,1]$
  Output: last weight vector $\mathbf{w}_{T+1}$
1 for $t = 1, 2, \ldots, T$ do
2     learner chooses probability vector $\mathbf{p}_t = \frac{\mathbf{w}_t}{\mathbf{1}^\top \mathbf{w}_t}$        // normalization
3     environment chooses loss vector $\boldsymbol{\ell}_t \in [0,1]^{\mathsf{n}}$        // $\boldsymbol{\ell}_t$ may depend on $\mathbf{p}_t$!
4     learner suffers (expected) loss $\langle \mathbf{p}_t, \boldsymbol{\ell}_t \rangle$
5     learner updates weights $\mathbf{w}_{t+1} = \mathbf{w}_t \odot \beta^{\boldsymbol{\ell}_t}$        // element-wise product $\odot$ and power
6     optional scaling: $\mathbf{w}_{t+1} \leftarrow c_{t+1}\mathbf{w}_{t+1}$        // $c_{t+1} > 0$ can be arbitrary
```

Imagine the following horse racing game[a]: There are n horses in a race, which we repeat for T rounds. On each round we bet a fixed amount of money, with p_{it} being the proportion we spent on horse i at the t-th round. At the end of round t we receive a loss $\ell_{it} \in [0,1]$ that we suffer on horse i. Note that the losses can be completely arbitrary (e.g. no i.i.d. assumption), although for simplicity we assume they are bounded. How should we place our bets (i.e. p_{it}) to hedge our risk? Needless to say, we must decide the proportions $\mathbf{p}_t$ *before* we see the losses $\boldsymbol{\ell}_t$. On the other hand, the losses $\boldsymbol{\ell}_t$ can be completely adversarial (i.e. depend on $\mathbf{p}_t$).

The Hedging algorithm gives a reasonable (in fact in some sense optimal) strategy: basically if a horse i is doing well (badly) on round t, then we spend a larger (smaller) proportion $p_{i,t+1}$ on it in the next round $t+1$. On a high level, this is similar to the idea behind perceptron (see Section 1).

Note that line 5 continues decreasing each entry in the weight vector (because both $\beta \in [0,1]$ and $\ell \in [0,1]$). To prevent the weights from underflowing, we can re-scale them by a positive number $c > 0$, as shown in the optional line 6. Clearly, this scaling does not change $\mathbf{p}$ hence the algorithm (in any essential way).

Freund, Yoav and Robert E. Schapire (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139.

[a]Viewer discretion is advised; please do not try at home!

Alert 11.9: Multiplicative Updates

The Hedging algorithm belongs to the family of multiplicative updates, where we repeatedly multiplying, instead of adding (cf. Perceptron in Section 1), our weights by some correction terms. For multiplicative updates, it is important to start with strictly positive initializations, for a zero weight will remain as zero in multiplicative updates.

Theorem 11.10: Hedging Guarantee

For any nonempty $S \subseteq \{1, \ldots, \mathsf{n}\}$, we have

$$L \le \frac{-\ln(\sum_{i \in S} p_{i1}) - (\ln \beta) \max_{i \in S} L_i}{1-\beta},$$

where $L := \sum_{t=1}^{T} \langle \mathbf{p}_t, \boldsymbol{\ell}_t \rangle$ is our total (expected) loss and $L_i := \sum_{t=1}^{T} \ell_{it}$ is the total loss on the i-th horse.

Proof. We first lower bound the sum of weights at the end:

$$\mathbf{1}^\top \mathbf{w}_{T+1} = \sum_{i=1}^{\mathsf{n}} w_{i,T+1} \ge \sum_{i \in S} w_{i,T+1} \qquad // \text{ weights remain nonnegative}$$

$$= \sum_{i \in S} w_{i,1} \beta^{L_i} \qquad // \text{ line 5 of Algorithm 11.8} \tag{11.1}$$

$$\geq \beta^{\max_{i\in S} L_i} \sum_{i\in S} w_{i,1} \qquad // \ \beta \in [0,1]. \tag{11.2}$$

Then, we upper bound the sum of weights:

$$\begin{aligned}
\mathbf{1}^\top \mathbf{w}_{t+1} &= \sum_{i=1}^{\mathsf{n}} w_{i,t+1} = \sum_{i=1}^{\mathsf{n}} w_{i,t}\beta^{\ell_{i,t}} && // \text{ line 5 of Algorithm 11.8}\\
&\leq \sum_{i=1}^{\mathsf{n}} w_{i,t}(1-(1-\beta)\ell_{i,t}) && // \ x^\ell \leq 1^\ell + (x-1)\ell\text{: } x^\ell \text{ concave when } \ell\in[0,1] \text{ and } x \geq 0 \quad (11.3)\\
&= (\mathbf{1}^\top\mathbf{w}_t)[1-(1-\beta)\mathbf{p}_t^\top \boldsymbol{\ell}_t] && // \text{ line 2 of Algorithm 11.8.}
\end{aligned}$$

Thus, by telescoping:

$$\frac{\mathbf{1}^\top \mathbf{w}_{T+1}}{\mathbf{1}^\top \mathbf{w}_1} \leq \prod_{t=1}^{T}[1-(1-\beta)\mathbf{p}_t^\top\boldsymbol{\ell}_t] \leq \exp\left[-(1-\beta)\sum_{t=1}^{T}\mathbf{p}_t^\top\boldsymbol{\ell}_t\right] = \exp[-(1-\beta)L], \tag{11.4}$$

where we have used the elementary inequality $1-x \leq \exp(-x)$ for any x.

Finally, combining the inequalities (11.2) and (11.4) completes the proof. □

By inspecting the proof closely, we realize that the same conclusion still holds if we change the update to

$$\mathbf{w}_{t+1} = \mathbf{w}_t \odot U_\beta(\boldsymbol{\ell}_t)$$

as long as the function U_β satisfies

$$\beta^\ell \leq U_\beta(\ell) \leq 1-(1-\beta)\ell$$

(so that inequalities (11.1) and (11.3) still hold).

Exercise 11.11: Optional re-scaling has no effect

Show that the same conclusion in Theorem 11.10 still holds even if we include the optional line 6 in Algorithm 11.8.

Corollary 11.12: Comparable to best "horse" in hindsight

If we choose $\mathbf{w}_1 = \mathbf{1}$ *and* $|S| = 1$, *then*

$$\sum_{t=1}^{T}\langle \mathbf{p}_t, \boldsymbol{\ell}_t\rangle \leq \frac{\min_i L_i \ln\frac{1}{\beta} + \ln \mathsf{n}}{1-\beta}. \tag{11.5}$$

In addition, if we choose $\beta = \frac{1}{1+\sqrt{(2m)/U}}$, *where* $U \geq L_{\min} := \min_i L_i$ *and* $m \geq \ln \mathsf{n}$, *then*

$$\frac{1}{T}\sum_{t=1}^{T}\langle \mathbf{p}_t, \boldsymbol{\ell}_t\rangle \leq \frac{L_{\min}}{T} + \frac{\sqrt{2mU}}{T} + \frac{\ln\mathsf{n}}{T}. \tag{11.6}$$

If we also choose $m = \ln \mathsf{n}$ *and* $U = T$ *(in our choice of* β*), then*

$$\frac{1}{T}\sum_{t=1}^{T}\langle \mathbf{p}_t, \boldsymbol{\ell}_t\rangle \leq \frac{L_{\min}}{T} + \sqrt{\frac{2\ln\mathsf{n}}{T}} + \frac{\ln\mathsf{n}}{T}. \tag{11.7}$$

Proof. Indeed, for $\beta \in [0,1]$ we have

$$\begin{aligned} 2 \le 2/\beta &\implies 2\beta \ge 2\ln\beta + 2 && // \ 2\beta - 2\ln\beta - 2 \text{ is decreasing and nonnegative at } \beta = 1 \\ &\implies \beta^2 \le 1 + 2\beta\ln\beta && // \ \beta^2 - 2\beta\ln\beta - 1 \text{ is increasing and nonpositive at } \beta = 1 \\ &\iff \ln\tfrac{1}{\beta} \le \tfrac{1-\beta^2}{2\beta}. \end{aligned}$$

Plugging our choice of β into (11.5) and apply the above inequality we obtain (11.6). The bound (11.7) holds because we can choose $U = T$ (recall that we assume $\ell_{i,t} \in [0,1]$ for all t and i). $\square$

Based on a result due to Vovk (1998), Freund and Schapire (1997) proved that the coefficients in (11.5) (i.e. $\frac{-\ln\beta}{1-\beta}$ and $\frac{\ln \mathsf{n}}{1-\beta}$) cannot be simultaneously improved for any β, using any algorithm (not necessarily Hedging).

In our last setting, $\beta = \frac{1}{1+\sqrt{(2\ln \mathsf{n})/T}}$, indicating for a longer game (larger T) we should use a larger β (to discount less aggressively) while for more horses (larger n) we should do the opposite (although this effect is much smaller due to the log).

Vovk, Valadimir (1998). "A Game of Prediction with Expert Advice". *Journal of Computer and System Sciences*, vol. 56, no. 2, pp. 153–173.
Freund, Yoav and Robert E. Schapire (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139.

Remark 11.13: Appreciating the significance of Hedging

Corollary 11.12 implies that the *average* loss of Hedging on the left-hand side of (11.7) is no larger than the average loss of the "best horse", plus a constant term proportional to $\sqrt{\frac{2\ln \mathsf{n}}{T}}$ (where we omit the higher order term $\frac{\ln \mathsf{n}}{T}$). As the number of rounds T goes to infinity, Hedging can compete against the best horse in hindsight! However, we emphasize three important points:

- It does not mean Hedging will achieve small loss, simply because the best horse may itself achieve a *large* average loss, in which case being competitive against the best horse does not really mean much.
- The loss vectors $\boldsymbol{\ell}_t$ can be adversarial against the Hedging algorithm! However, if the environment always tries to "screw up" the Hedging algorithm, then the guarantee in Corollary 11.12 implies that the environment inevitably also "screws up" the best horse.
- If the best horse can achieve very small loss, i.e. $L_{\min} \approx 0$, then by setting $U \approx 0$ we know from (11.6) that Hedging is off by at most a term proportional to $\frac{\ln \mathsf{n}}{T}$, which is much smaller than the dominating term $\sqrt{\frac{2\ln \mathsf{n}}{T}}$ in (11.7).

Remark 11.14: History of Boosting

Schapire (1990) first formally proved the possibility to combine a few mediocre classifiers into a very accurate one, which was subsequently improved in (Freund 1995) and eventually in (Freund and Schapire 1997), which has since become the main source of the celebrated Adaboost algorithm. Freund and Shapire received the Gödel prize in 2003 for this seminal contribution.

Schapire, Robert E. (1990). "The strength of weak learnability". *Machine Learning*, vol. 5, no. 2, pp. 197–227.
Freund, Y. (1995). "Boosting a Weak Learning Algorithm by Majority". *Information and Computation*, vol. 121, no. 2, pp. 256–285.
Freund, Yoav and Robert E. Schapire (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139.

Algorithm 11.15: Adaboost (Freund and Schapire 1997)

Algorithm: Adaptive Boosting.

Input: initial weight vector $\mathbf{w}_1 \in \mathbb{R}^{\mathsf{n}}_{++}$, training set $\mathcal{D}_{\mathsf{n}} = \wr(\mathbf{x}_i, y_i)\wr_{i=1}^{\mathsf{n}} \subseteq \mathbb{R}^d \times \{0,1\}$

Output: meta-classifier $\bar{h} : \mathbb{R}^d \to \{0,1\}$, $\mathbf{x} \mapsto [\![\sum_{t=1}^{T}(\ln\frac{1}{\beta_t})(h_t(\mathbf{x}) - \frac{1}{2}) \geq 0]\!]$

```
1 for $t = 1, 2, \ldots, T$ do
2     $\mathbf{p}_t = \mathbf{w}_t / \mathbf{1}^\top \mathbf{w}_t$                                   // normalization
3     $h_t \leftarrow \mathsf{WeakLearn}(\mathcal{D}_{\mathsf{n}}, \mathbf{p}_t)$                    // $t$-th weak classifier $h_t : \mathbb{R}^d \to [0,1]$
4     $\forall i,\ \ell_{it} = 1 - |h_t(\mathbf{x}_i) - y_i|$          // loss is higher if prediction is more accurate!
5     $\epsilon_t = 1 - \langle \mathbf{p}_t, \boldsymbol{\ell}_t \rangle = \sum_{i=1}^{\mathsf{n}} p_{it} |h_t(\mathbf{x}_i) - y_i|$     // weighted (expected) error $\epsilon_t \in [0,1]$ of $h_t$
6     $\beta_t = \epsilon_t / (1 - \epsilon_t)$                 // adaptive discounting parameter $\beta_t \leq 1 \iff \epsilon_t \leq \frac{1}{2}$
7     $\mathbf{w}_{t+1} = \mathbf{w}_t \odot \beta_t^{\boldsymbol{\ell}_t}$                              // element-wise product $\odot$ and power
8     optional scaling: $\mathbf{w}_{t+1} \leftarrow c_{t+1}\mathbf{w}_{t+1}$                          // $c_{t+1} > 0$ can be arbitrary
```

Provided that $\epsilon_t \leq \frac{1}{2}$ hence $\beta_t \in [0,1]$ (so the classifier h_t is better than random guessing), if h_t predicts *correctly* on a training example $\mathbf{x}_i$, then we suffer a *larger* loss ℓ_{ti} so that in the next iteration we assign *less* weight to $\mathbf{x}_i$. In other words, each classifier is focused on hard examples that are *misclassified* by the previous classifier. On the other hand, if $\epsilon_t > \frac{1}{2}$ then $\beta_t > 1$ and we do the opposite.

For the meta-classifier $\bar{h}$, we first perform a weighted aggregation of the confidences of individual (weak) classifiers and then threshold. Alternatively, we could threshold each weak classifier first and then perform (weighted) majority voting. The former approach, which we adopt here, is found to work better in practice. Note that, a classifier h_t with lower (training) error ϵ_t will be assigned a higher weight $\ln\frac{1}{\beta_t} = \ln(\frac{1}{\epsilon_t} - 1)$ in the final aggregation, making intuitive sense.

Freund, Yoav and Robert E. Schapire (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139.

Exercise 11.16: Implicitly fixing "bad" classifiers

If $\epsilon_t \geq \frac{1}{2}$ (hence $\beta_t \geq 1$), the t-th (weak) classifier is in fact worse than random guessing! In our binary setting here, a natural idea is to discard h_t but use instead $\tilde{h}_t := 1 - h_t$. Prove the following (all tilde quantities are w.r.t. the flipped classifier $\tilde{h}_t$):

- $\tilde{\boldsymbol{\ell}}_t = 1 - \boldsymbol{\ell}_t$
- $\tilde{\epsilon}_t = 1 - \epsilon_t$
- $\tilde{\beta}_t = 1/\beta_t$
- The Adaboost algorithm is not changed in any essential way even if we flip all "bad" classifiers. In particular, we arrive at the same meta-classifier.

In other words, Adaboost implicitly fixes all "bad" classifiers!

Theorem 11.17: Adaboost Guarantee

The following bound on the training error holds for the Adaboost Algorithm 11.15:

$$\sum_{i=1}^{\mathsf{n}} p_{i1} [\![\bar{h}(\mathbf{x}_i) \neq y_i]\!] \leq \prod_{t=1}^{T} \sqrt{4\epsilon_t(1-\epsilon_t)}.$$

Proof. The proof closely follows what we have seen in the proof of Theorem 11.10. As before, we upper bound the sum of weights in the same manner:

$$\frac{\|\mathbf{w}_{T+1}\|_1}{\|\mathbf{w}_1\|_1} \leq \prod_{t=1}^{T}[1-(1-\beta_t)\mathbf{p}_t\boldsymbol{\ell}_t] = \prod_{t=1}^{T}[1-(1-\beta_t)(1-\epsilon_t)].$$

To lower bound the sum of weights, let $S = \{i : \bar{h}(\mathbf{x}_i) \neq y_i\}$, where recall that $\bar{h}$ is the meta-classifier constructed in Algorithm 11.15. Proceed as before:

$$\frac{\|\mathbf{w}_{T+1}\|_1}{\|\mathbf{w}_1\|_1} \geq \sum_{i\in S}\frac{w_{i,T+1}}{\|\mathbf{w}_1\|_1} = \sum_{i\in S}p_{i,1}\prod_{t=1}^{T}\beta_t^{\ell_{i,t}} \geq \sum_{i\in S}p_{i,1}\left[\prod_{t=1}^{T}\beta_t\right]^{1/2},$$

where the last inequality follows from the definition of S and the meta-classifier $\bar{h}$:

$$i\in S \implies 0 \geq \operatorname{sign}(2y_i-1)\sum_t(\ln\tfrac{1}{\beta_t})(h_t(\mathbf{x}_i)-\tfrac{1}{2}) = \sum_t(\ln\tfrac{1}{\beta_t})(\tfrac{1}{2}-|h_t(\mathbf{x}_i)-y_i|) = \sum_t(\ln\tfrac{1}{\beta_t})(\ell_{i,t}-\tfrac{1}{2}).$$

Combine the upper bound and the lower bound:

$$\sum_{i\in S}p_{i,1}\prod_{t=1}^{T}\sqrt{\beta_t} \leq \prod_{t=1}^{T}[1-(1-\beta_t)(1-\epsilon_t)], \quad i.e., \quad \sum_{i\in S}p_{i,1} \leq \prod_{t=1}^{T}\frac{1-(1-\beta_t)(1-\epsilon_t)}{\sqrt{\beta_t}}.$$

Optimizing w.r.t. β_t we obtain $\beta_t = \epsilon_t/(1-\epsilon_t)$. Plugging it back in we complete the proof. □

Importantly, we observe that the training error of the meta-classifier $\bar{h}$ is upper bounded by the errors of all classifiers h_t: improving any individual classifier leads to a better bound. Moreover, the symmetry on the right-hand side confirms again that in the binary setting a very "inaccurate" classifier (i.e. large ϵ_t) is as good as a very accurate classifier (i.e. small ϵ_t).

Corollary 11.18: Exponential decay of training error

Assume $|\epsilon_t - \frac{1}{2}| > \gamma_t$, *then*

$$\sum_{i=1}^{n}p_{i1}[\![\bar{h}(\mathbf{x}_i)\neq y_i]\!] \leq \prod_{t=1}^{T}\sqrt{1-4\gamma_t^2} \leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right).$$

In particular, if $\gamma_t \geq \gamma$ *for all* t, *then*

$$\sum_{i=1}^{n}p_{i1}[\![\bar{h}(\mathbf{x}_i)\neq y_i]\!] \leq \exp(-2T\gamma^2).$$

□

Thus, to achieve ϵ (weighted) training error, we need to combine at most

$$T = \lceil \frac{1}{2\gamma^2}\ln\frac{1}{\epsilon}\rceil$$

weak classifiers, each of which is slightly better than random guessing (by a margin of γ).

Remark 11.19: Generalization Error of Adaboost

It is possible to bound the generalization error of Adaboost as well. For instance, Freund and Schapire (1997) bounded the VC dimension of the (family of) meta-classifiers constructed by Adaboost. A standard application of the VC theory then relates the generalization error with the training error. More refined analysis can be found in (Schapire et al. 1998; Koltchinskii and Panchenko 2002; Koltchinskii et al. 2003; Koltchinskii and Panchenko 2005; Freund et al. 2004; Rudin et al. 2007) (just to give a few pointers).

Freund, Yoav and Robert E. Schapire (1997). “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”. *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139.
Schapire, Robert E., Yoav Freund, Peter Bartlett, and Wee Sun Lee (1998). “Boosting the margin: a new explanation for the effectiveness of voting methods”. *The Annals of Statistics*, vol. 26, no. 5, pp. 1651–1686.
Koltchinskii, V. and D. Panchenko (2002). “Empirical Margin Distributions and Bounding the Generalization Error of Combined Classifiers”. *The Annals of Statistics*, vol. 30, no. 1, pp. 1–50.
Koltchinskii, Vladimir, Dmitriy Panchenko, and Fernando Lozano (2003). “Bounding the generalization error of convex combinations of classifiers: balancing the dimensionality and the margins”. *The Annals of Applied Probability*, vol. 13, no. 1, pp. 213–252.
Koltchinskii, Vladimir and Dmitry Panchenko (2005). “Complexities of convex combinations and bounding the generalization error in classification”. *The Annals of Statistics*, vol. 33, no. 4, pp. 1455–1496.
Freund, Yoav, Yishay Mansour, and Robert E. Schapire (2004). “Generalization bounds for averaged classifiers”. *The Annals of Statistics*, vol. 32, no. 4, pp. 1698–1722.
Rudin, Cynthia, Robert E. Schapire, and Ingrid Daubechies (2007). “Analysis of boosting algorithms using the smooth margin function”. *The Annals of Statistics*, vol. 35, no. 6, pp. 2723–2768.

Alert 11.20: Does Ababoost Overfit? (Breiman 1999; Grove and Schuurmans 1998)

It has long been observed that Adaboost, even after decreasing the training error to zero, continues to improve test error. In other words, Adaboost does not seem to overfit even when we combine many many (weak) classifiers.

A popular explanation, due to Schapire et al. (1998), attributes Adaboost’s resistance against overfitting to margin maximization: after decreasing the training error to 0, Adaboost continues to improve the margin (i.e. $y\bar{h}(\mathbf{x})$), which leads to better generalization. However, Breiman (1999) and Grove and Schuurmans (1998) later designed the LPboost that explicitly maximizes the margin but observed inferior generalization.

Schapire, Robert E., Yoav Freund, Peter Bartlett, and Wee Sun Lee (1998). “Boosting the margin: a new explanation for the effectiveness of voting methods”. *The Annals of Statistics*, vol. 26, no. 5, pp. 1651–1686.
Breiman, Leo (1999). “Prediction Games and Arcing Algorithms”. *Neural Computation*, vol. 11, no. 7, pp. 1493–1517.
Grove, Adam J. and Dale Schuurmans (1998). “Boosting in the Limit: Maximizing the Margin of Learned Ensembles”. In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 692–699.

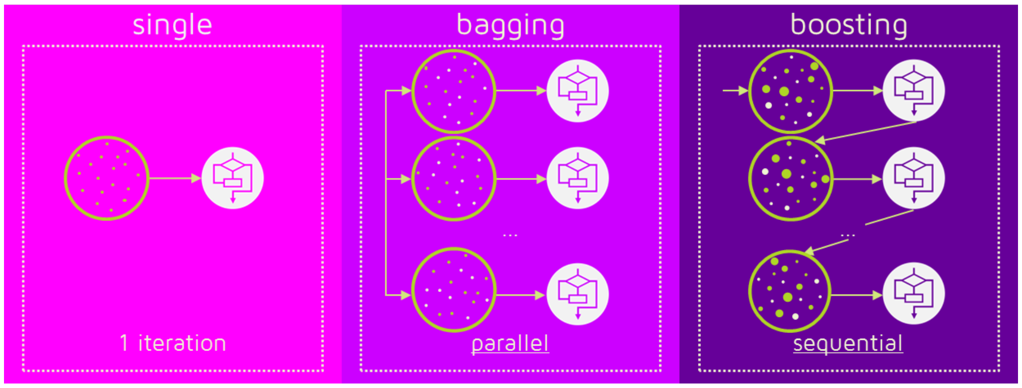
Algorithm 11.21: Face Detection (Viola and Jones 2004)

Viola and Jones (2004) applied the Adaboost algorithm to real-time face detection and are among the first few people who demonstrated the power of Adaboost in real challenging applications.

Viola, Paul and Michael J. Jones (2004). “Robust Real-Time Face Detection”. *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154.

Remark 11.22: Comparison

The following figure illustrates the similarity and difference between bagging and boosting:



To summarize:

- Both bagging and boosting train an ensemble of (weak) classifiers, which are combined in the end to produce a "meta-classifier";
- Bagging is amenable to parallelization while boosting is strictly sequential;
- Bagging resamples training examples while boosting reweighs training examples;
- As suggested in Breiman (2004), we can think of bagging as averaging *independent* classifiers (in order to reduce variance), while boosting averages dependent classifiers which may be analyzed through dynamic system and ergodic theory;
- We can of course combine bagging with boosting. For instance, each classifier in boosting can be obtained through bagging (called bag-boosting by Bühlmann and Yu, see the discussion of (Friedman et al. 2000)).

Breiman, Leo (2004). "Population theory for boosting ensembles". *The Annals of Statistics*, vol. 32, no. 1, pp. 1–11.
Friedman, Jerome, Trevor Hastie, and Robert Tibshirani (2000). "Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors)". *The Annals of Statistics*, vol. 28, no. 2, pp. 337–407.

Exercise 11.23: Diversity

Recall that $\epsilon_t(h) := \sum_{i=1}^{\mathsf{n}} p_{it}|h(\mathbf{x}_i) - y_i|$ is the weighted error of classifier h at iteration t. Assume the (weak) classifiers are always binary-valued (and $\beta_t \in (0,1)$ for all t). Prove:

$$\epsilon_{t+1}(h_t) \equiv \tfrac{1}{2}.$$

In other words, Adaboost would never choose the same weak classifier twice in a row.