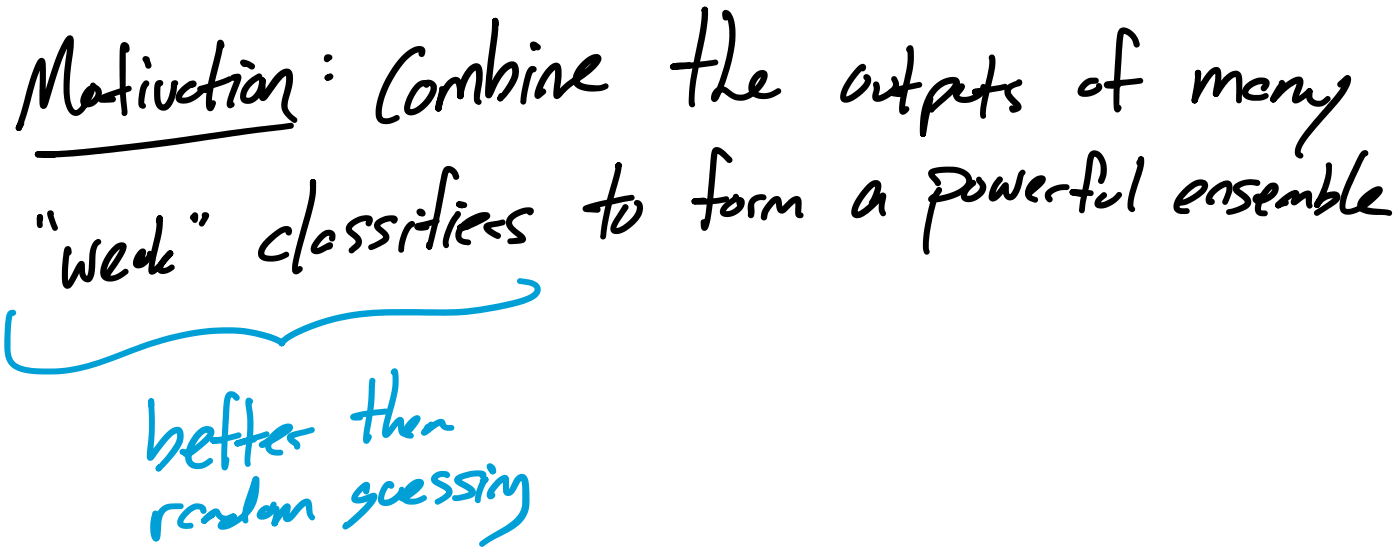
Lecture 16 - Mar 4

Boosting
AdaBoost.M1

Reading

- *Elements of Statistical Learning*
  - Ch 10: Boosting and Additive Trees

- QuantDare (2016). What is the difference between Bagging and Boosting? https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/



Upcoming Deadlines

Homework 4 (Mar 15)

Motivation: Combine the outputs of many "weak" classifiers to form a powerful "committee"

AdaBoost.M1 (Freund + Schapire 1997)

Let $Y \in \{-1, 1\}$ with predictors X + classifier $G(X) \in \{-1, 1\}$

Error rate on training sample:

$$\overline{err} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{\{y_i \neq G(x_i)\}}$$

and the error rate on future predictions is $E_{XY}\left[\mathbb{1}_{\{Y \neq G(X)\}}\right]$

A **weak classifier** has error rate slightly better than random guessing

Boosting: sequentially apply weak classification algorithm to repeatedly modified version of the data to produce weak classifiers $G_m(x)$, $m = 1, \ldots, M$

Combine predictions through a weighted majority vote

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

...boosting algorithm. They

$G(x) = \text{sign}\left[\sum_{m=1}^{} \right]$

with weights computed by the boosting algorithm. They give higher influence to more accurate classifiers

Data modifications at each boosting step apply weights $w_1, \ldots, w_N$ to training observations (x_i, y_i). Initially, all $w_i = \frac{1}{N}$. In successive iterations, observation weights are modified:

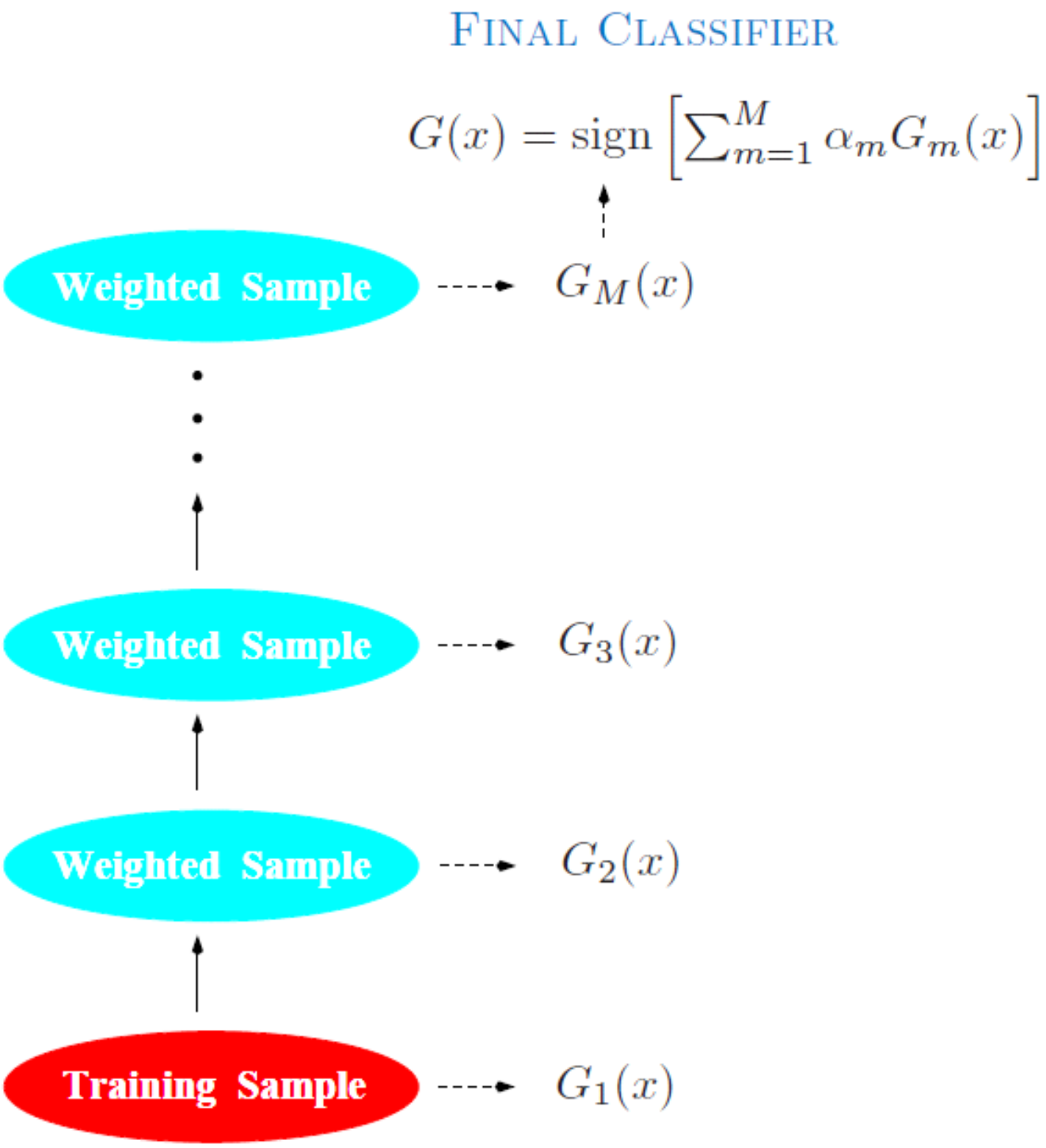
misclassified observations → w_i increases

correctly classified obs → w_i decreases

⇒ difficult points get more influence in new models fitted with weights

⇒ classifiers forced to focus on difficult points

Boosting Diagram



AdaBoost.M1 Algorithm

(1) Set $w_i = \frac{1}{N}$, $i = 1, \ldots, N$

(2) For $m = 1$ to M:

(a) fit classifier $G_m(x)$ to data with w_i's

(b) Compute $err_m = \frac{\sum_{i=1}^{N} w_i \mathbb{1}_{\{y_i \neq G(x_i)\}}}{\sum_{i=1}^{N} w_i}$ fraction of weight misclassified

(c) Compute $\alpha_m = \log\left(\frac{1 - err_m}{err_m}\right)$

(d) Set $w_i = w_i \exp\left(\alpha_m \mathbb{1}_{\{y_i \neq G(x_i)\}}\right)$

(3) Output $G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$

This is "discrete AdaBoost", "Real AdaBoost" works to predict $Y \in [-1, 1]$.

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

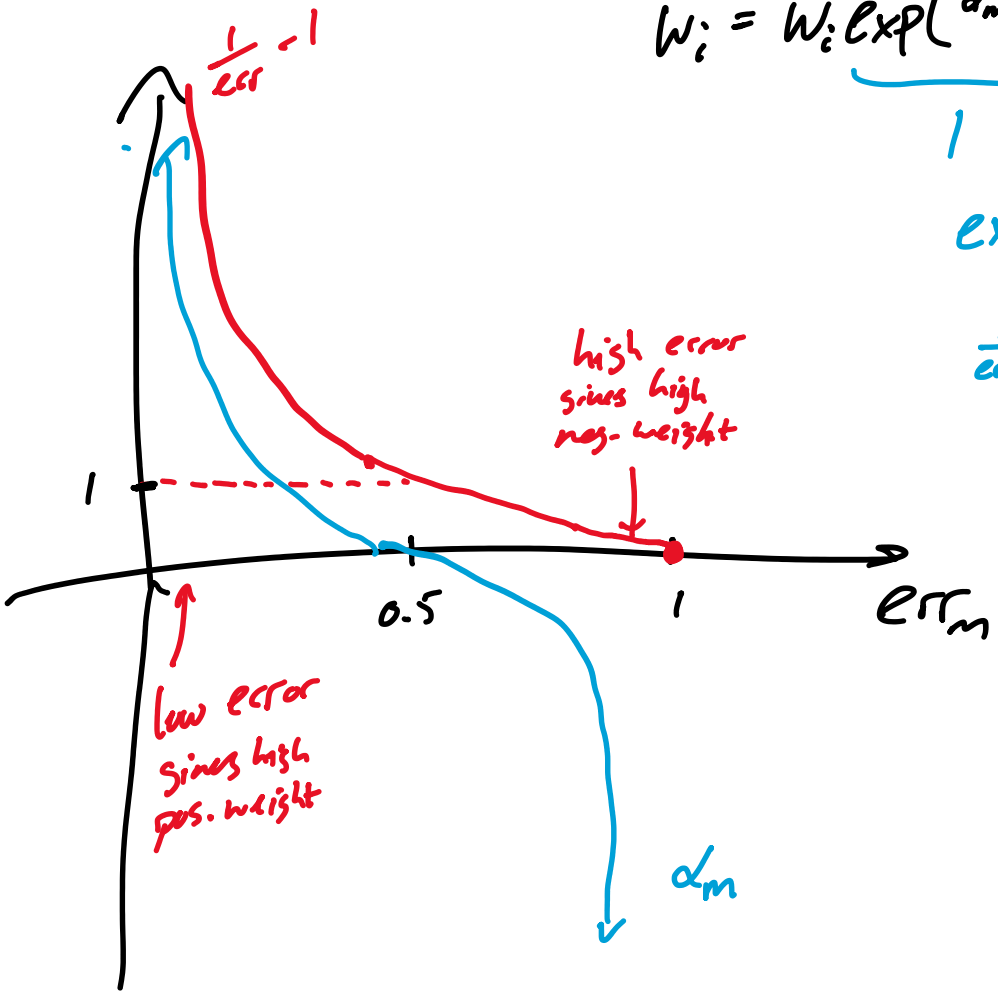
$$w_i = w_i \exp\left(\alpha_m \mathbb{1}_{\{y_i \neq G(x_i)\}}\right)$$

1 if correct (no effect)

$\exp(\alpha_m)$ if incorrect

$\frac{1}{err} - 1 > 1$ if $err < 0.5$

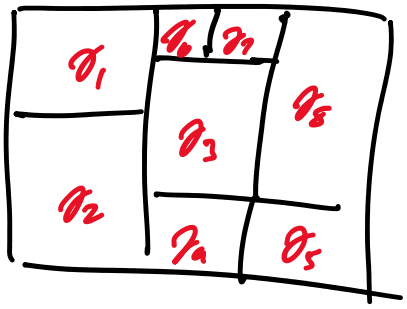
≤ 1 if $err \geq 0.5$



Boosted Trees

Monday, March 28, 2022 3:37 PM

Trees split the feature space into rectangles $R_1, R_2, \ldots, R_J$



if $x \in R_j$, then $f(x) = \gamma_j$

We can express the tree as

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j \mathbb{1}_{\{x \in R_j\}}$$

with parameters $\Theta = \{R_j, \gamma_j\}_{j=1,\ldots,J}$

We train T by trying to minimize

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \underbrace{\sum_{x_i \in R_j} L(y_i, \gamma_i)}_{\text{sum of losses for all } x_i \in R_j}$$

This is hard... so we rely on suboptimal solutions by

(1) Find γ_i given R_i ($\bar{y}_i$ or modal class)

(easy part)

(2) Find R_i's by greedy, recursive splitting

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{i=1}^{N} \tilde{L}(y_i, T(x_i;\Theta))$$

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{n} L(y_i, T(x_i;\theta))$$

Boosted tree model: $f_M(x) = \sum_{m=1}^{M} T(x;\theta_m)$

$$\hat{\theta}_m = \arg\min_{\theta_m} \sum_{i=1}^{n} L(y_i, f_{m-1}(x_i) + T(x_i;\theta_m))$$

frequently, we have to approx

$$\hat{\theta}_m = \{R_{jm}, \gamma_{jm}\}$$

Given R_{jm}, finding optimal γ_{jm} is easy.-

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$$

AdaBoost → two-class classification with exponential loss $e^{-y_i f_{m-1}(x_i)}$

$$\hat{\theta}_m = \arg\min_{\theta_m} \sum_{i=1}^{n} w_i^{(m)} \exp(-y_i T(x_i;\theta_m))$$

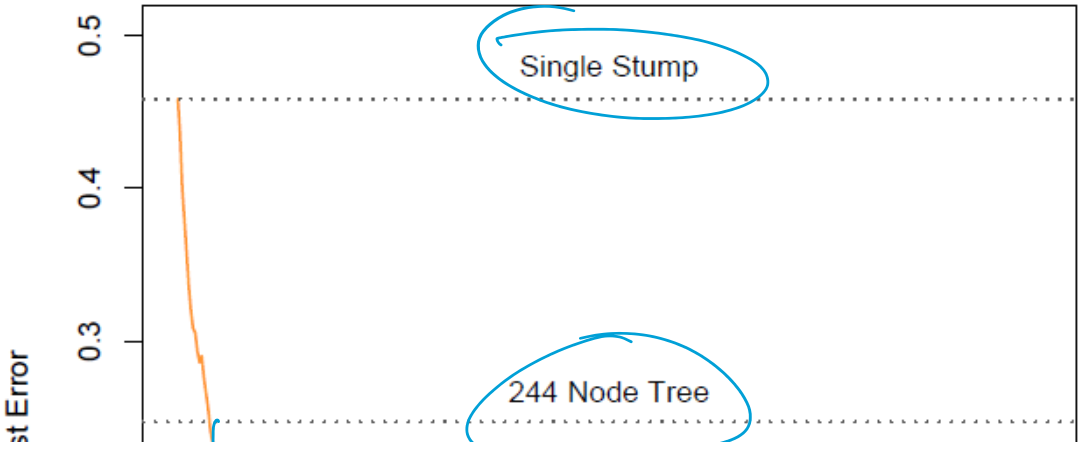
$$\hat{\gamma}_{jm} = \ln\left(\frac{\sum_{x_i \in R_{jm}} w_i^{(m)} 1\{y_i = 1\}}{\sum_{x_i \in R_{jm}} w_i^{(m)} 1\{y_i = -1\}}\right)$$

AdaBoost Performance

The power of AdaBoost to dramatically increase the performance of even a very weak classifier is illustrated in Figure 10.2. The features $X_1, \ldots, X_{10}$ are standard independent Gaussian, and the deterministic target Y is defined by

$$Y = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_j^2 > \chi^2_{10}(0.5), \\ -1 & \text{otherwise.} \end{cases} \tag{10.2}$$

Here $\chi^2_{10}(0.5) = 9.34$ is the median of a chi-squared random variable with 10 degrees of freedom (sum of squares of 10 standard Gaussians). There are 2000 training cases, with approximately 1000 cases in each class, and 10,000 test observations. Here the weak classifier is just a "stump": a two terminal-node classification tree. Applying this classifier alone to the training data set yields a very poor test set error rate of 45.8%, compared to 50% for random guessing. However, as boosting iterations proceed the error rate steadily decreases, reaching 5.8% after 400 iterations. Thus, boosting this simple very weak classifier reduces its prediction error rate by almost a factor of four. It also outperforms a single large classification tree (error rate 24.7%). Since its introduction, much has been written to explain the success of AdaBoost in producing accurate classifiers. Most of this work has centered on using classification trees as the "base learner" $G(x)$, where improvements are often most dramatic. In fact, Breiman (NIPS Workshop, 1996) referred to AdaBoost with trees as the "best off-the-shelf classifier in the world" (see also Breiman (1998)). This is especially the case for data-mining applications, as discussed more fully in Section 10.7 later in this chapter.



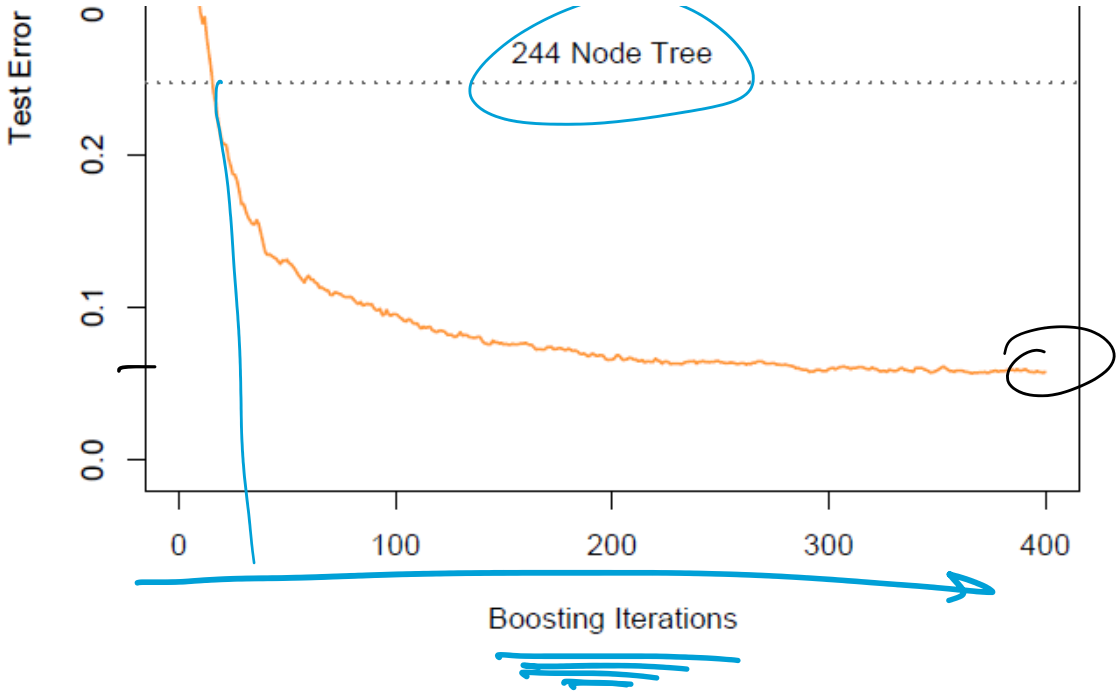


FIGURE 10.2. *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a* 244*-node classification tree.*

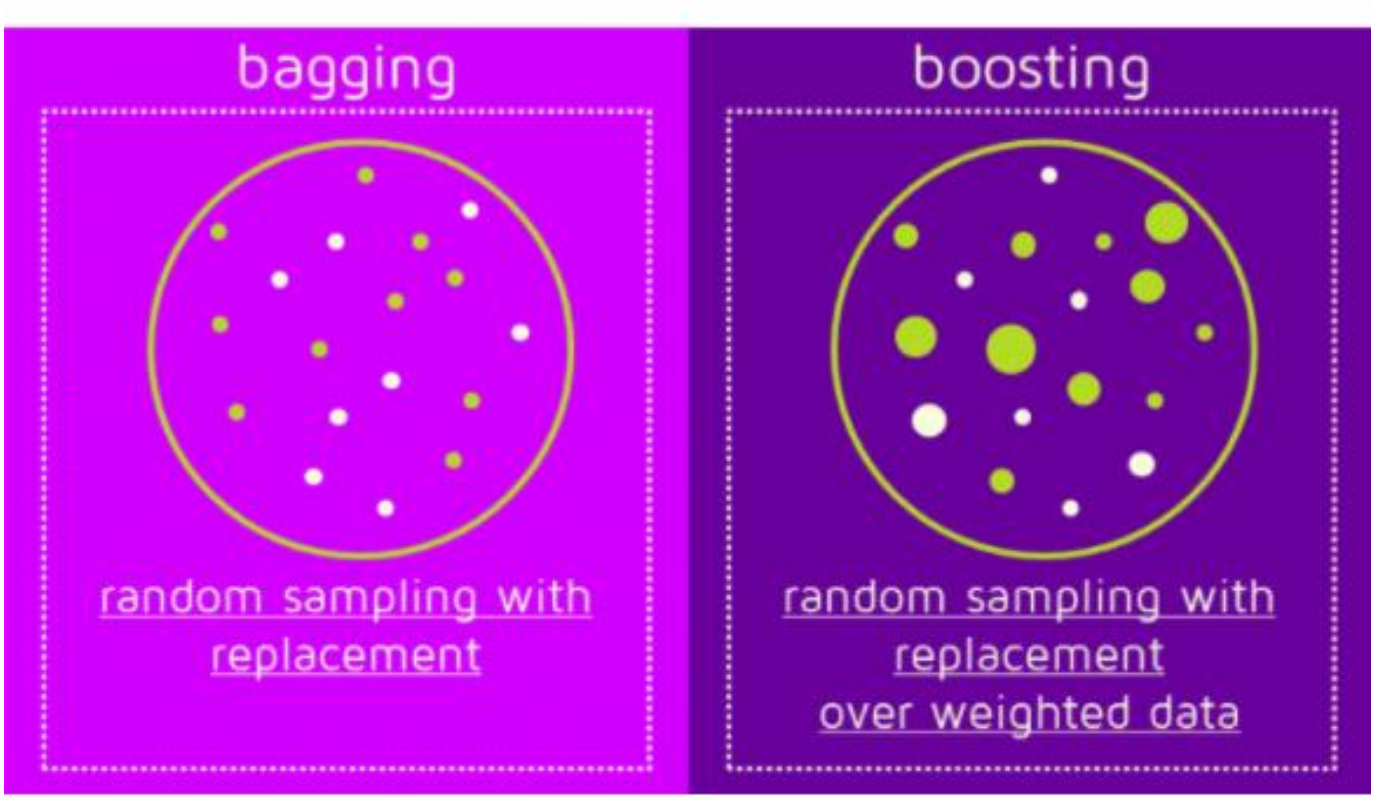
(from the QuantDare blog post referenced above)

The main causes of error in learning are due to **noise, bias, and variance**. Ensemble helps to minimize these factors.

These methods are designed to improve the stability and the accuracy of ML algorithms.

Combinations of classifiers **decrease variance**, especially in the case of unstable classifiers, and may produce a more reliable classification than a single classifier.

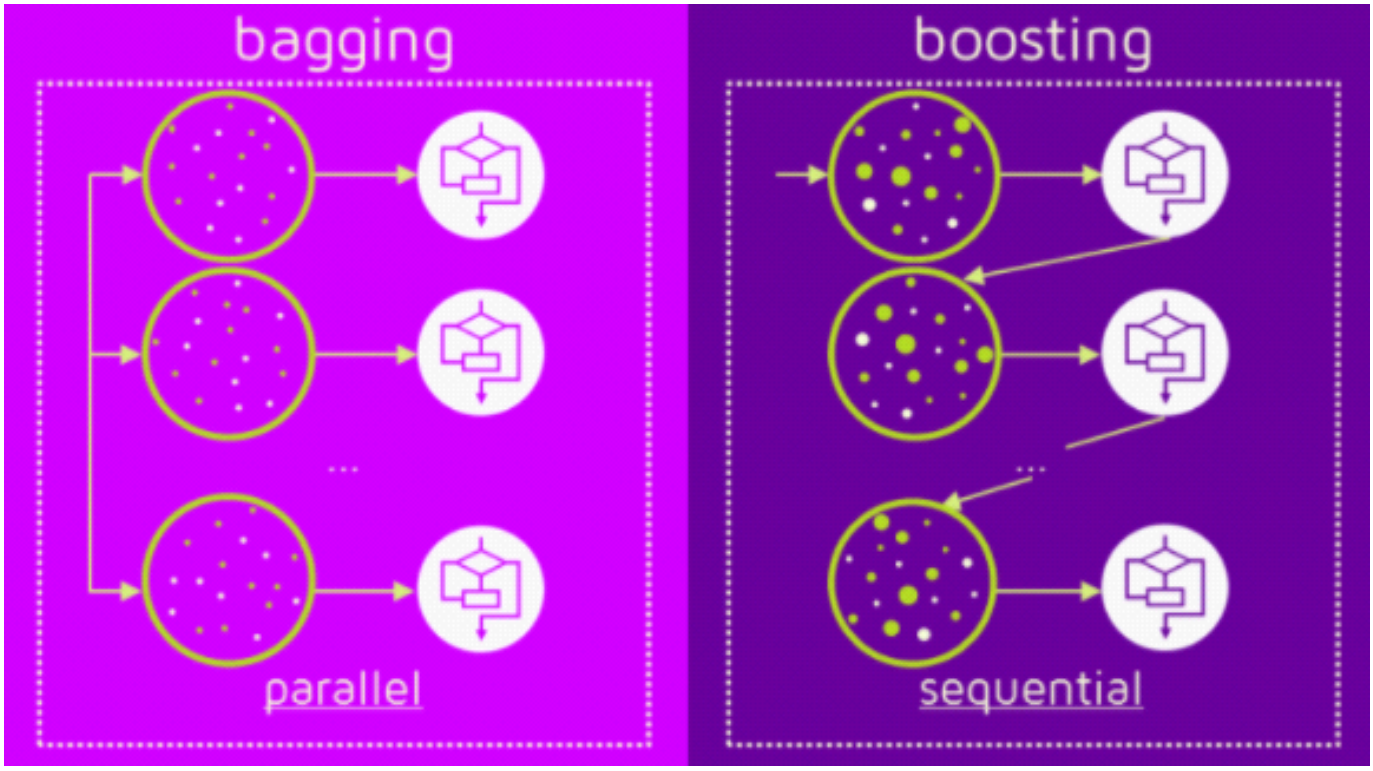
To use Bagging or Boosting you must select a base learner algorithm. For example, if we choose a classification tree, they use a pool of trees.



N new training data sets are produced by **bootstrap samples** from the original set.

Bagging: any element has the **same probability** to appear in a new dataset.

Boosting: observations are **weighted** so some are more likely to be in new datasets

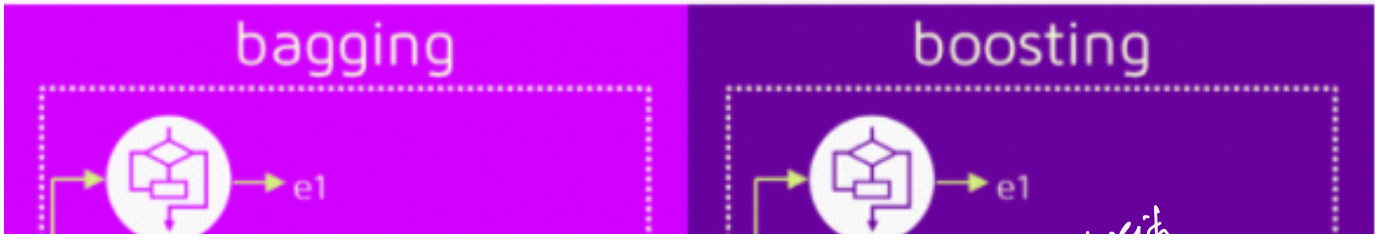


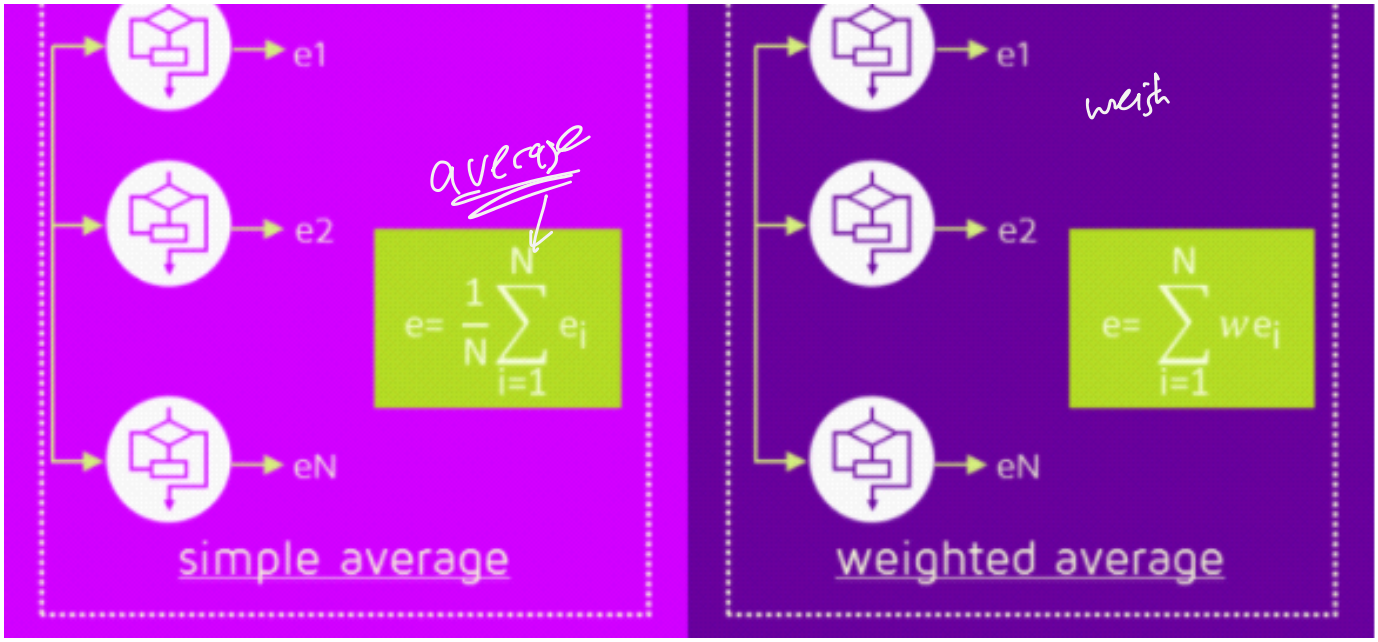
Boosting: each classifier is trained on data, taking into account the previous classifiers' success. After each training step, the weights are redistributed.

Misclassified data increases its weights to emphasize the most difficult cases. In this way, subsequent learners will focus on them during their training.

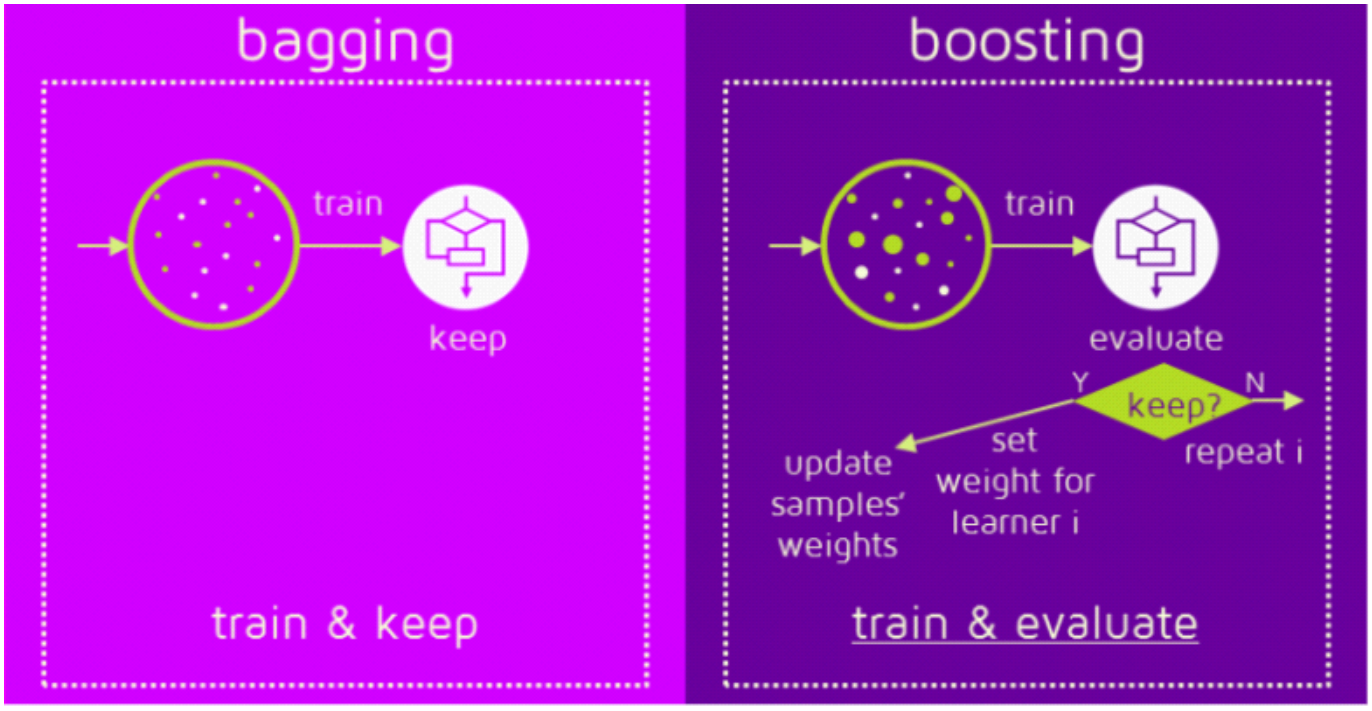
Bagging: result is average the responses of the N classifiers (majority vote).

Boosting: assigns a second set of weights for the N classifiers (weighted vote)





New versions of **AdaBoost**: a model is rejected if error is **worse than random guessing**. Otherwise, repeated until achieving a learner better than a random guess.



In boosting, several alternatives exist to determine the weights to use in the next

training step and in the classification stage.

Most popular: **AdaBoost**, LPBoost, **XGBoost**, **GradientBoost**, BrownBoost

Should I used Bagging or Boosting?

They both **decrease variance**, the resulting in a model with higher stability.

Boosting can generate a combined model with lower errors as it optimizes the advantages and reduces pitfalls of the single model to **reduce underfitting**.
(Bagging doesn't help.)

Bagging is good at **reducing overfitting**.
(Boosting doesn't help ~~a~~ much)

Similarities	Differences
Both are ensemble methods to get N learners from 1 learner...	... but, while they are built independently for Bagging, Boosting tries to add new models that do well where previous models fail.
Both generate several training data sets by random sampling...	... but only Boosting determines weights for the data to tip the scales in favor of the most difficult cases.
Both make the final decision by averaging the N learners (or taking the majority of them)...	... but it is an equally weighted average for Bagging and a weighted average for Boosting, more weight to those with better performance on training data.
Both are good at reducing variance and provide higher stability...	... but only Boosting tries to reduce bias. On the other hand, Bagging may solve the over-fitting problem, while Boosting can increase it.

Boosting Fits an Additive Model

The success of boosting is really not very mysterious. The key lies in expression (10.1). Boosting is a way of fitting an additive expansion in a set of elementary "basis" functions. Here the basis functions are the individual classifiers $G_m(x) \in \{-1, 1\}$. More generally, basis function expansions take the form

Basis expansion models

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$$

β_m: expansion coefficients; $b(x;\gamma_m)$: $G_m(x)$'s

$$\cdots$$

where $\beta_m, m = 1, 2, \ldots, M$ are the expansion coefficients, and $b(x; \gamma) \in \mathbb{R}$ are usually simple functions of the multivariate argument x, characterized by a set of parameters γ. We discuss basis expansions in some detail in Chapter 5.

Typically fit by minimizing loss

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)\right)$$

This can be expensive depending on L + b's, but a feasible alternative when it is feasible to rapidly solve the subproblem of fitting a single basis function

$$\min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, \beta b(x_i; \gamma))$$

$$\min_{\beta,\gamma} \sum_{i=1} L(y_i, \beta b(x_i;\gamma))$$

Idea: Sequentially add new basis functions without adjusting parameters + coefficients of those already added

Algorithm 10.2

(1) Init $f_0(x) = 0$

(2) For $m = 1$ to M:

(a) Compute $(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma))$

(b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma_m)$

one more tree

Adds one basis function greedily

AdaBoost.M1 = fwd-stagewise additive modeling with loss $L(y, f(x)) = \exp(-y f(x))$

(simple proof in *ESL* 10.4)

For AdaBoost the basis functions are the individual classifiers $G_m(x) \in \{-1, 1\}$. Using the exponential loss function, one must solve

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} \exp[-y_i(f_{m-1}(x_i) + \beta\, G(x_i))]$$

for the classifier G_m and corresponding coefficient β_m to be added at each step. This can be expressed as

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta\, y_i\, G(x_i)) \qquad (10.9)$$

with $w_i^{(m)} = \exp(-y_i\, f_{m-1}(x_i))$. Since each $w_i^{(m)}$ depends neither on β nor $G(x)$, it can be regarded as a weight that is applied to each observation. This weight depends on $f_{m-1}(x_i)$, and so the individual weight values change with each iteration m.

The solution to (10.9) can be obtained in two steps. First, for any value of $\beta > 0$, the solution to (10.9) for $G_m(x)$ is

$$G_m = \arg\min_{G} \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i)), \qquad (10.10)$$

which is the classifier that minimizes the weighted error rate in predicting y. This can be easily seen by expressing the criterion in (10.9) as

$$e^{-\beta} \cdot \sum_{y_i = G(x_i)} w_i^{(m)} + e^{\beta} \cdot \sum_{y_i \neq G(x_i)} w_i^{(m)},$$

which in turn can be written as

$$\left(e^{\beta} - e^{-\beta}\right) \cdot \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i)) + e^{-\beta} \cdot \sum_{i=1}^{N} w_i^{(m)}. \tag{10.11}$$

Plugging this G_m into (10.9) and solving for β one obtains

$$\beta_m = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m}, \tag{10.12}$$

where err_m is the minimized weighted error rate

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i^{(m)}}. \tag{10.13}$$

The approximation is then updated

$$f_m(x) = f_{m-1}(x) + \beta_m G_m(x),$$

which causes the weights for the next iteration to be

$$w_i^{(m+1)} = w_i^{(m)} \cdot e^{-\beta_m y_i G_m(x_i)}. \tag{10.14}$$

Using the fact that $-y_i G_m(x_i) = 2 \cdot I(y_i \neq G_m(x_i)) - 1$, (10.14) becomes

$$w_i^{(m+1)} = w_i^{(m)} \cdot e^{\alpha_m I(y_i \neq G_m(x_i))} \cdot e^{-\beta_m}, \tag{10.15}$$

where $\alpha_m = 2\beta_m$ is the quantity defined at line 2c of AdaBoost.M1 (Algorithm 10.1). The factor $e^{-\beta_m}$ in (10.15) multiplies all weights by the same value, so it has no effect. Thus (10.15) is equivalent to line 2(d) of Algorithm 10.1.

One can view line 2(a) of the Adaboost.M1 algorithm as a method for approximately solving the minimization in (10.11) and hence (10.10). Hence we conclude that AdaBoost.M1 minimizes the exponential loss criterion (10.8) via a forward-stagewise additive modeling approach.