Fondamentaux de l'Apprentissage Automatique

Lecturer: Yann Chevaleyre
Scribe: Julien Orsini

Lecture n°9 #
24/11/2023

1 Introduction

In this course we study how to build decision trees for regression and classification tasks. We will then see ensemble methods that aim to create more resilient estimators based on multiple decision trees. Bagging methods first approximate the result by combining weak models built in parallel to reduce their variance. Boosting methods, on the other hand, try to build a model sequentially by focusing on the errors made by each estimator to correct them as the forest grows.

2 Decision Trees

2.1 Introduction

Definition 1. ***Binary Tree***
A binary tree is a hierarchical data structure in which each node has at most two children, referred to as the left child and the right child. The topmost node in a binary tree is called the root. Nodes that have no children are called leaves.

Let's consider a binary tree on $X = \mathbb{R}^2$. At the root and at each internal node, a variable $x^{(j)}$ is called the **test variable** or **decision variable** of the node, and t is called the **threshold**. Decisions at each node are based on a single variable, with decisions on continuous variables of the form $x_j \leqslant t$, and decisions on discrete variables separating the possible values in two groups. As a result, each leaf corresponds to a region of the partition that is associated with a label.
Note : Other types of trees include oblique decision trees sometimes called Binary Space Partition trees (BSP trees), sphere trees, etc.

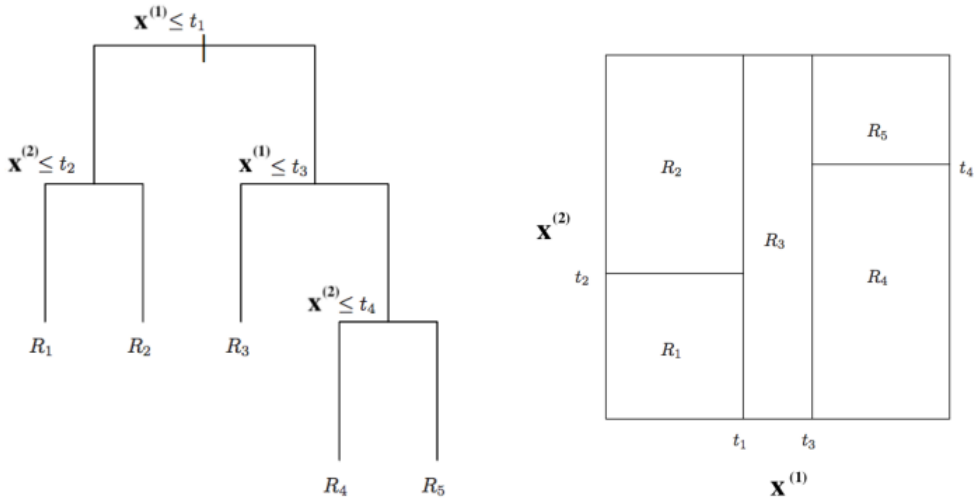

FIGURE 1 – Binary Tree on $\mathbb{R}^2$ with the resulting associated decision partition

2.2 Decision Trees for regression

A decision tree divides X into regions $\{R_1, \ldots, R_M\}$ such that :

$$X = R_1 \cup R_2 \cup \ldots \cup R_M$$

$$R_i \cap R_j = \emptyset \quad \text{for all} \quad i \neq j$$

Prediction
Let $N_m = |\{i : x_i \in R_m\}|$.
With the partition $\{R_1, \ldots, R_M\}$ associated to the labels $c_1, \ldots, c_M$, the final prediction is :

$$f(x) = \sum_{m=1}^{M} c_m 1(x \in R_m).$$

Suppose we already have the partition. To choose $c_1, \ldots, c_M$, we can apply the Empirical Risk Minimization (ERM) for the loss function $\ell(\hat{y}, y) = (\hat{y} - y)^2$ such that :

$$\begin{aligned} c_1, \ldots, c_M &= \arg\min \sum_{i=1}^{N} \left(f(x_i) - y_i\right)^2 \\ &= \arg\min \sum_{i=1}^{N} \left(\sum_{m=1}^{M} c_m 1(x_i \in R_m) - y_i \right)^2 \\ &= \arg\min \sum_{m=1}^{M} \sum_{i \in R_m} \left(c_m - y_i\right)^2 \end{aligned}$$

Thus,

$$c_m = \frac{1}{N_m} \sum_{i \in R_m} y_i = \text{average}(y_i | x_i \in R_m)$$

The resulting loss associated with the node c_m is then :

$$\sum_{i : x_i \in R_m} (\hat{c}_m - y_i)^2 = \sum_{i : x_i \in R_m} (\text{average}(y_j \,|\, x_j \in R_m) - y_i)^2 = N_m \cdot \hat{\text{Var}}(\{y_i \,|\, x_i \in R_m\})$$

Choice of the splitting
Let's denote $R_1(j, s), R_2(j, s)$ the partition based on $x^{(j)}$ and s :

$$\begin{aligned} R_1(j, s) &= \{x \,|\, x^{(j)} \leq s\} \\ R_2(j, s) &= \{x \,|\, x^{(j)} > s\} \end{aligned}$$

The objective is to find j and s that minimize the empirical loss :

$$\begin{aligned} L(j, s) &= \sum_{i : x_i \in R_1(j,s)} (y_i - \hat{c}_1(j, s))^2 + \sum_{i : x_i \in R_2(j,s)} (y_i - \hat{c}_2(j, s))^2 \\ &= N_1 \hat{\text{Var}}(\{y_i : x_i \in R_1(j, s)\}) + N_2 \hat{\text{Var}}(\{y_i : x_i \in R_2(j, s)\}) \end{aligned}$$

Suppose we choose the test variable $x^{(j)}$. Assume that $x_1^{(j)}, \ldots, x_N^{(j)}$ are sorted in ascending order. We only need to consider the threshold between two consecutive values :

$$s_j \in \left\{ \frac{x_i^{(j)} + x_{i+1}^{(j)}}{2} \,|\, i = 1, \ldots, N-1 \right\}$$

Thus, we have a total of ND pairs (j, s) to consider. If done by brute force, the value of the least-squares error for each can be computed in time $O(N)$, for a total time complexity of $O(N^2 D)$. However, it is in fact possible to compute the function values for all the thresholds in a particular dimension in time $O(N \log N)$ with a particular sorting, so to reduce the overall complexity to $O(ND \log N)$. We then build the tree recursively until a stopping rule is reached.

The stopping criteria should be decided so to control the complexity of the tree. Indeed, if the tree is too large, each example x_i will have its own partition, leading to overfitting. Conversely, if it's too small, it may result in underfitting. To address this, one can limit the depth of the tree or only continue splitting nodes that contain a minimum number of examples. Another approach is **post-pruning**, as outlined in **CART** by Breiman et al. (1984). This involves initially constructing a large and deep tree (e.g., until each region has at least 5 points) and then greedily pruning the tree from the bottom up as long as the estimated performance on a test set does not decrease.

2.3 Decision Trees for classification

Prediction

Let $Y = \{1...K\}$. By using the same reasoning as before, we assume that we already have the regions R_i.

The node m represents the region R_m, with N_m examples.

The proportion of examples of class $k \in Y$ in R_m is

$$\hat{\eta}_{m,k} = P(\{Y = k\} | \{X \in R_m\}) = \frac{1}{N_m} \sum_{i : x_i \in R_m} 1(y_i = k).$$

If we predict class k at node m, then the error rate on the training examples from R_m will be $1 - \hat{\eta}_{m,k}$.

So, to minimize the error rate with $0/1$ loss, the predicted class for node m will be

$$\hat{y}(m) = \arg\min_k (1 - \hat{\eta}_{m,k}) = \arg\max_k \hat{\eta}_{m,k}.$$

Example

Student	Credit Rating	Class: Buy PDA
No	Fair	No
No	Excellent	No
No	Fair	Yes
No	Fair	Yes
Yes	Fair	Yes
Yes	Excellent	No
Yes	Excellent	Yes
No	Excellent	No

FIGURE 2 – Example

We try to predict the Buy PDA class with a single node tree.

-Test on the student feature : Let R1 be the leaf corresponding to students and R2 the leaf corresponding to the others.

$$\hat{\eta}_{1,yes} = \frac{2}{3} \qquad \hat{\eta}_{2,yes} = \frac{2}{5}$$
$$\hat{\eta}_{1,no} = \frac{1}{3} \qquad \hat{\eta}_{2,no} = \frac{3}{5}$$

The tree predicts *yes* for R1 and *no* for R2. The error rate is therefore :

$$\text{Error Rate} = \frac{N1}{N} \cdot \frac{1}{3} + \frac{N2}{N} \cdot \frac{2}{5} = \frac{3}{8}$$

-Test on the credit rating feature : Let R1 be the leaf corresponding to fair credit ratings and R2 the leaf corresponding excellent credit ratings.

$$\hat{\eta}_{1,yes} = \frac{3}{4} \qquad \hat{\eta}_{2,yes} = \frac{1}{4}$$
$$\hat{\eta}_{1,no} = \frac{1}{4} \qquad \hat{\eta}_{2,no} = \frac{3}{4}$$

The tree predicts *yes* for R1 and *no* for R2. The error rate is therefore :

$$\text{Error Rate} = \frac{N1}{N} \cdot \frac{1}{4} + \frac{N2}{N} \cdot \frac{1}{4} = \frac{1}{4}$$

Therefore the second splitting is the best. □

Let's now go back to the case $Y = \{0, 1\}$. The proportion of examples of class 1 in R_m is given by $\hat{\eta}_m$. The ERM suggests the following prediction for class 1 :

$$\hat{c}_m = \arg\min_{\hat{y}} \sum_{i:x_i \in R_m} \mathcal{L}(\hat{y}, y_i) = \arg\min \hat{\eta}_m \mathcal{L}(\hat{y}, 1) + (1 - \hat{\eta}_m)\mathcal{L}(\hat{y}, 0)$$

If the loss function is a proper CP-loss (conditional probability estimation) such as cross-entropy, then :

$$\hat{c}_m = \hat{\eta}_m.$$

The value of the loss in R_m will then be for the cross-entropy ($\mathcal{L}(\hat{\eta}, y) = -y \log(\hat{\eta}) - (1 - y) \log(1 - \hat{\eta})$) :

$$\begin{aligned}\sum_{i:x_i \in R_m} \mathcal{L}(\hat{\eta}_m, y_i) &= N_m \hat{\eta}_m \mathcal{L}(\hat{\eta}_m, 1) + N_m (1 - \hat{\eta}_m) \mathcal{L}(\hat{\eta}_m, 0) \\ &= -N_m (\hat{\eta}_m \log(\hat{\eta}_m) + (1 - \hat{\eta}_m) \log(1 - \hat{\eta}_m)) \\ &= N_m H_{\mathcal{L}} \qquad \text{(Shannon Entropy)}\end{aligned}$$

If we aim to predict a class rather than a probability, we will take $\hat{y}(R_m) = 1$ if $\hat{\eta}_m > \frac{1}{2}$, and 0 otherwise.
Note : For any proper CP-loss $\mathcal{L}$, the value of this loss in region R_m is the generalized entropy $H_{\mathcal{L}}(\hat{\eta}_m)$.

Impurity measures
More generally, three measures $Q_m(T)$ may be used for the node impurity of leaf node m :

1. The misclassification error :

$$H_{0/1}(\hat{\eta}) = \min_k (1 - \hat{\eta}_{m,k}).$$

2. The Gini index :

$$H_{\text{Gini}}(\hat{\eta}) = \sum_{k=1}^{K} \hat{\eta}_{m,k} (1 - \hat{\eta}_{m,k}).$$

3. The Entropy or deviance (equivalent to using information gain) :

$$H_{\text{CE}}(\hat{\eta}) = -\sum_{k=1}^{K} \hat{\eta}_{m,k} \log(\hat{\eta}_{m,k}).$$

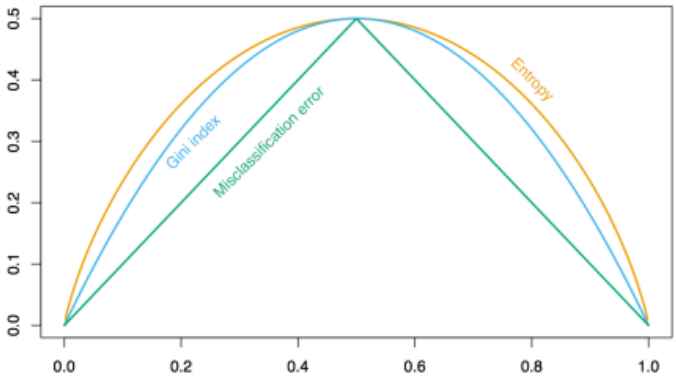

FIGURE 3 – Comparison of the impurity measures for binary classification

Example

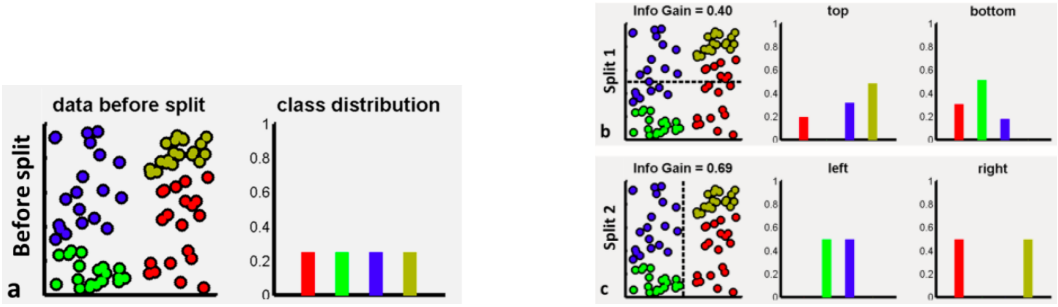

FIGURE 4 – Example of splitting

For k classes, $H(\hat{\eta_1}, \ldots, \hat{\eta_k}) = -\sum_{k=1}^{K} \hat{\eta}_k \log_2 \hat{\eta}_k$
The initial class distribution is (1/4, 1/4, 1/4, 1/4) with an entropy equals to $H = 4 * -\frac{1}{4}\log(\frac{1}{4}) = 2$. In the second split the distribution is (0, 1/2, 1/2, 0) in the left part and (1/2, 0, 0, 1/2) in the right part. In each parts the entropy is therefore : $H_l = H_r = (-(\frac{1}{2}\log(\frac{1}{2})) * 2 + 0 * 2 = \log_2(2) = 1$ so that the average total entropy is equals to $H = \frac{1}{2} * 1 + \frac{1}{2} * 1 = 1$ Hence the entropy has reduced since the splitting. □

Choice of the splitting
Let R_1 and R_2 be regions corresponding to a potential node split. Suppose we have N_1 points in R_1 and N_2 points in R_2. Let $H(R_1)$ and $H(R_2)$ be generalized entropy (the node impurity measures). The goal is to find the split that minimizes the weighted average of node impurities :

$$\frac{N_1}{N} H(R_1) + \frac{N_2}{N} H(R_2).$$

When building the tree, Gini and entropy losses appear to be more effective as they advocate for more pure nodes rather than just considering the misclassification rate. A good split may not change the misclassification rate at all.
Consider for instance a two-class problem with 4 observations in each class. For two splits :
1. Split 1 : (3,1) and (1,3) (each region has 3 of one class and 1 of the other).
2. Split 2 : (2,4) and (2,0) (one region has 2 of one class and 4 of the other, while the other region is pure).
The misclassification rate for both splits is the same. However, Gini and entropy splits prefer Split 2 as it leads to more pure nodes.

3 Bagging and Random Forests

3.1 Ensemble Methods

Ensemble methods combine multiple models to make a prediction. On one hand, **parallel ensembles** involve building each model independently, as seen in methods like bagging and random forests. The main idea is to combine many models, typically with high complexity and low bias, to reduce variance. On the other, **sequential ensembles** generate models sequentially, attempting to add new models that perform well in areas where previous models may lack.

3.2 On the benefits of averaging

Let $z, z_1, \ldots, z_n$ be i.i.d. random variables with $E[z] = \mu$ and $\text{Var}(z) = \sigma^2$. Let's consider the average of the z_i's.
The average has the same expected value but a smaller standard error :

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} z_i\right] = \mu \quad \text{Var}\left[\frac{1}{n}\sum_{i=1}^{n} z_i\right] = \frac{\sigma^2}{n}.$$

Clearly, the average is preferred to a single z_i as an estimator.
Can we apply this to reduce the variance of general prediction functions ?

Suppose we have B independent training sets from the same distribution. Suppose $\hat{Y} = \mathbb{R}$, the learning algorithm gives B decision functions : $\hat{f}_1(x), \hat{f}_1(x), \ldots, \hat{f}_B(x)$.
We define the average prediction function as :

$$\hat{f}_{\text{avg}} = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b$$

The randomness here comes from the B independent training sets, which give rise to variation among the $\hat{f}_b$.
Let's fix $x_0 \in X$. The average prediction on x_0 is then :

$$\hat{f}_{\text{avg}}(x_0) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x_0).$$

Since the training sets are random, we can consider $\hat{f}_{\text{avg}}(x_0)$ and $\hat{f}_1(x_0), \ldots, \hat{f}_B(x_0)$ as random variables. For the same reason, we don't know the distributions of $\hat{f}_1(x_0), \ldots, \hat{f}_B(x_0)$, however, we do know that $\hat{f}_1(x_0), \ldots, \hat{f}_B(x_0)$ are i.i.d.. Therefore $\hat{f}_{\text{avg}}(x_0)$ and $\hat{f}_b(x_0)$ have the same expected value, but $\hat{f}_{\text{avg}}(x_0)$ has a smaller variance :

$$\text{Var}(\hat{f}_{\text{avg}}(x_0)) = \frac{1}{B^2}\sum_{b=1}^{B} \text{Var}\left(\hat{f}_b(x_0)\right) = \frac{1}{B}\text{Var}\left(\hat{f}_1(x_0)\right).$$

We will use this idea to obtain better estimators. The decision trees that are poor estimators can hence be combined to obtain a more reliable one by reducing their variance.

3.3 Boostrap and Bagging

In practice, we don't have B independent training sets. Instead, we can use **bootstrap**. Bagging as proposed by Leo Breiman (1996) consists in drawing B bootstrap samples $D_1, \ldots, D_B$ of same size and with replacement, from the original data S. Let $\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_B$ be the prediction functions from training on $D_1, \ldots, D_B$, respectively. The bagged prediction function is then computed as a combination of these :

$$\hat{f}_{\text{bag}}(x) = \text{Combine}\left(\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_B(x)\right).$$

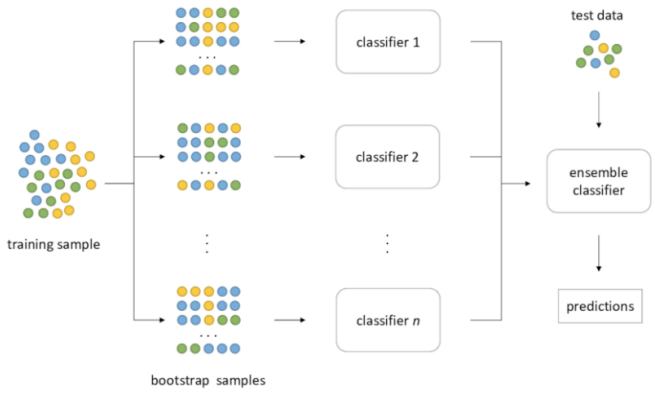

FIGURE 5 – Bagging method illustration

For regression we take $\hat{f}_{\text{bag}}$ equals to the mean of the predictions on each bag. For classification tasks, either the most predicted class, or the average of the probabilities can be used for the prediction. Small numbers of bootstrapp samples tend to give betters results for the probability average method, however, the two methods give the same results as this number grows.

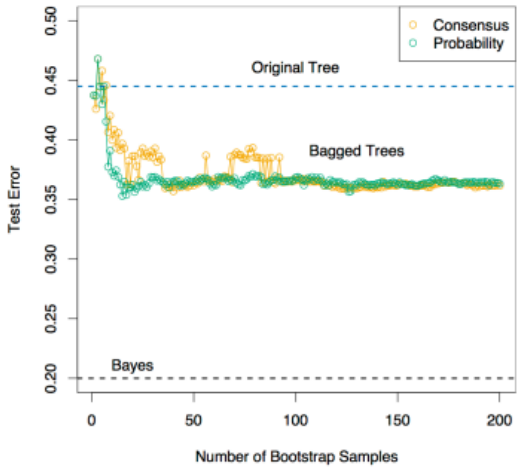

FIGURE 6 – Comparison of the bagging methods for classification

Remark
Each bagged predictor is trained on about 63% of the data. The remaining 37% are called out-of-bag (OOB) observations. For the i-th training point, let $S_i = \{b \,|\, D_b \text{ does not contain the } i\text{-th point}\}$. The OOB prediction on x_i is

$$\hat{f}_{\text{OOB}}(x_i) = \frac{1}{|S_i|} \sum_{b \in S_i} \hat{f}_b(x_i).$$

The OOB error is a good estimate of the test error. OOB error is similar to cross-validation error – both are computed on the training set.

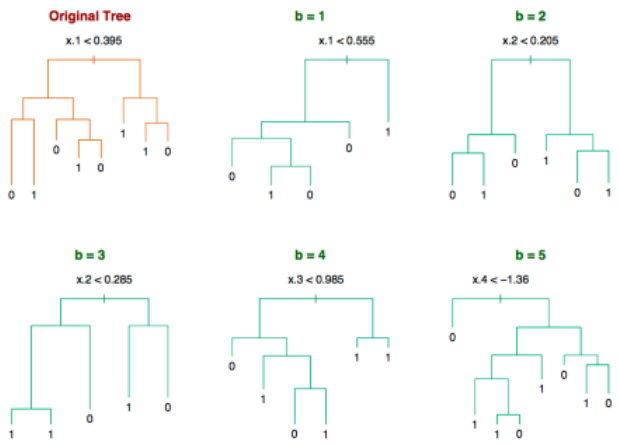

FIGURE 7 – Bagging method

Each bootstrap tree is quite different. Different splitting variables at the root contribute to this high degree of variability from small perturbations of the training data. This is why tree methods are described as high variance.

The hope is that bagging reduces variance without making bias worse. Bagging seems to help mostly when the approximation error (bias) is low and when there is high variance and low stability, i.e., small changes in the training set can cause large changes in predictions. It's challenging to find clear and convincing theoretical results on this, but following this intuition leads to improved machine learning methods, for example, Random Forests.

3.4 Random Forests

Bootstrap samples are independent samples from the training set, but are not independent from other samples in other bags. The main idea of random forests is to use bagged decision trees, but modify the tree-growing procedure to reduce the dependence between trees. A key step in random forests is that when constructing each tree node, the choice of the splitting variable is restricted to a randomly chosen subset of features of size m. Typically, m is chosen to be approximately $\sqrt{p}$, where p is the number of features. The value of m can also be chosen using cross-validation.

Algorithme 1 : Random Forest

Data : Training dataset $D = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\}$; Number of variables to use when splitting m $\in \{1, ..., d\}$.

Result : Ensemble of decision trees $F = \{T_1, T_2, \ldots, T_k\}$

1 **foreach** *Tree* $T_i \in F$ **do**

2 Sample a bootstrap dataset D_i from D;

3 Learn the classification tree T_i on D_i ; each time when splitting a node, search the optimal variable among m variables randomly chosen out of all d variables.

4 **return** *Ensemble of decision trees* F;

The usual approach is to build very deep trees (low bias). Diversity in individual tree prediction functions comes from bootstrap samples (somewhat different training data) and randomized tree building. Bagging seems to work better when we are combining a diverse set of prediction functions.

4 Boosting methods

With boosting methods, the idea is to build each classifiers sequentially such that each compensates the errors of the previous ones.

4.1 Introduction

The base hypothesis space is $\mathcal{F}$ of $\hat{Y}$-valued functions. Let's define the **combined hypothesis space** $\mathcal{F}_M$ such as :

$$\mathcal{F}_M = \left\{ \sum_{m=1}^{M} v_m h_m(x) \,|\, v_m \in \mathbb{R}, h_m \in \mathcal{F}, m = 1, \ldots, M \right\}$$

Suppose we're given some data $S = ((x_1, y_1), \ldots, (x_n, y_n))$. The learning procedure here comes down to choosing $v_1, \ldots, v_M \in \mathbb{R}$ and $h_1, \ldots, h_M \in \mathcal{F}$ to fit S.
Note : In bagging, we learn h_i, but v_i is set to $\frac{1}{M}$ for all classifiers. In boosting we will learn both coefficients.

We'll consider learning by empirical risk minimization : $\hat{h} = \arg\min_{f \in \mathcal{F}_M} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(\hat{h}(x_i), y_i\right)$, for some loss function $\mathcal{L}(y, \hat{y})$.
The ERM objective function writes as :

$$J(v_1, \ldots, v_M, h_1, \ldots, h_M) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, \sum_{m=1}^{M} v_m h_m(x_i)).$$

Suppose our base hypothesis space is parameterized by $\Theta = \mathbb{R}^d$:

$$J(v_1, \ldots, v_M, \theta_1, \ldots, \theta_M) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(\sum_{m=1}^{M} v_m h_{\theta_m}(x_i), y_i\right).$$

We can differentiate J with respect to v_m's and θ_m's and optimize with SGD for some hypothesis spaces and typical loss functions. Neural networks for instance, fall into this category with $h_1, \ldots, h_M$ being the neurons of the last hidden layer.

What if the base hypothesis space $\mathcal{F}$ consists of decision trees ? Can we even parameterize trees with $\Theta = \mathbb{R}^b$? Even if we could for some set of trees, predictions would not change continuously with respect to $\Theta \in \mathbb{R}^b$, and are not differentiable. Boosting applies whenever we can compute a particular form of the above ERM (FSAM algorithms). The loss function is [sub]differentiable with respect to training predictions $f(x_i)$, and we can do regression with the base hypothesis space $\mathcal{F}$ (gradient-boost).

Algorithme 2 : FSAM for ERM

Data : Training data (x_i, y_i) for $i = 1, \ldots, n$

Result : Final function f_M

Initialize $f_0(x) = 0$;

for $m = 1$ **to** M **do**

  Compute $(\nu_m, h_m) = \arg\min_{\nu \in \mathbb{R}, h \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} L\left(f_{m-1}(x_i) + \nu h(x_i), y_i\right)$;

  Set $f_m = f_{m-1} + \nu_m h_m$;

return f_M;

4.2 Forward Stagewise Additive Modeling (FSAM)

FSAM is an iterative optimization algorithm for fitting adaptive basis function models. In the m-th round, we want to find the step direction $h_m \in F$ (i.e., a basis function) and step size $\nu_i > 0$ such that $f_m = f_{m-1} + \nu_i h_m$ improves the objective function value by as much as possible.

4.3 Application to regression (L2 boosting)

Let's use the mean square error.

$$L(v, h) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \left(f_{m-1}(x_i) + v h(x_i)\right)\right)^2$$

If $\mathcal{F}$ is closed w.r.t. change of scale, then we only need to learn h such that it minimizes :

$$L(h) = \frac{1}{n} \sum_{i=1}^{n} \left([y_i - f_{m-1}(x_i)] - h(x_i)\right)^2$$

This is equivalent to perform least squares regression on the residuals $(y_i - f_{m-1}(x_i))$. The algorithm is sometimes called **matching pursuit**.

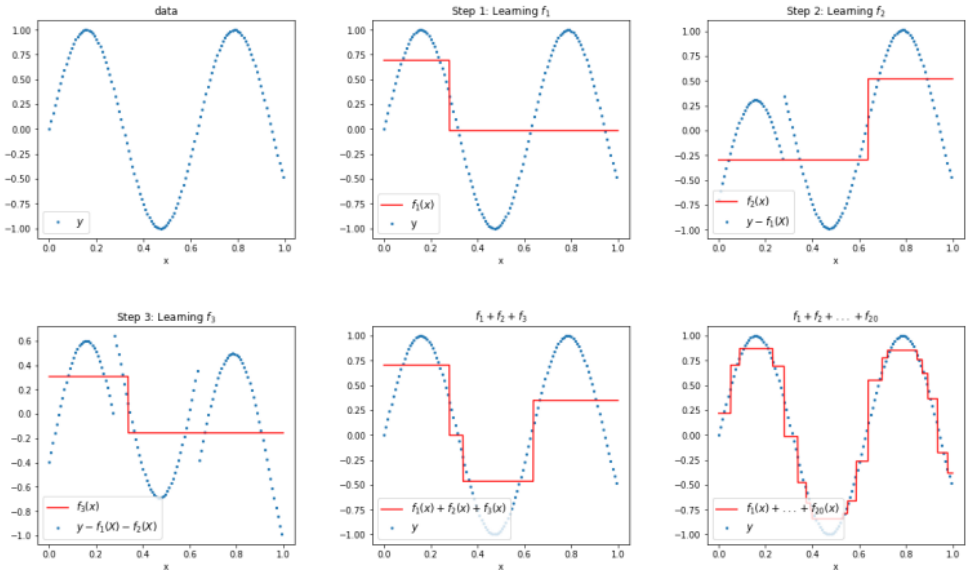

FIGURE 8 – L2 boosting with decision stumps

4.4 Application to classification (AdaBoost)

Let's set $Y = \{-1, 1\}$ the outcome space, and $F \subset X \to \{-1, 1\}$ the set of base classifiers (e.g., decision stumps). We want to learn a scoring function $f_M \in \mathcal{F}_{\mathcal{M}}$. As usual, this scoring function induces a "hard"-classifier $\text{sign}(f_M(x))$. We seek to optimize a scoring loss $f_M = \arg\min_{f \in F_M} \sum_{i=1}^{N} \mathcal{L}(f(x_i), y_i)$.

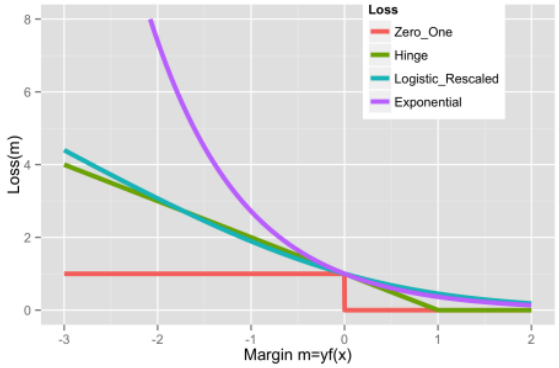

FIGURE 9 – Scoring losses for classification

All these losses are well calibrated. For these functions, a direct computation of FSAM is not easy. For the exponential loss : $\mathcal{L}(f(x), y) = \exp(-yf(x))$, there is an indirect algorithm possible named Adaboost.

Algorithme 3 : AdaBoost

Data : Training set $S = \{(x_i, y_i)\}$

Result : Final function $f_M(x)$

Initialize observation weights $w_i = 1$ for $i = 1, 2, \ldots, n$;

for $m = 1$ **to** M **do**

  Learner fits weighted training data and returns $h_m(x)$;

  Compute weighted empirical 0-1 risk :

$$\text{err}_m = \frac{1}{W} \sum_{i=1}^{n} w_i \cdot \mathbb{I}(y_i \neq h_m(x_i)),$$

  where $W = \sum_{i=1}^{n} w_i$;

  Compute $v_m = \ln\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$ [classifier weight];

  Set $w_i = w_i \cdot \exp(v_m \cdot \mathbb{I}(y_i \neq h_m(x_i)))$ for $i = 1, 2, \ldots, n$ [example weight adjustment];

return $f_M(x) = \sum_{m=1}^{M} v_m h_m(x)$

This way, we increase the weights on the points $h_m(x)$ misclassifies, so that the next learner corrects the error.

Note : One can show that each step of the AdaBoost algorithm is equivalent to a step of the FSAM to find (ν_m, h_m).

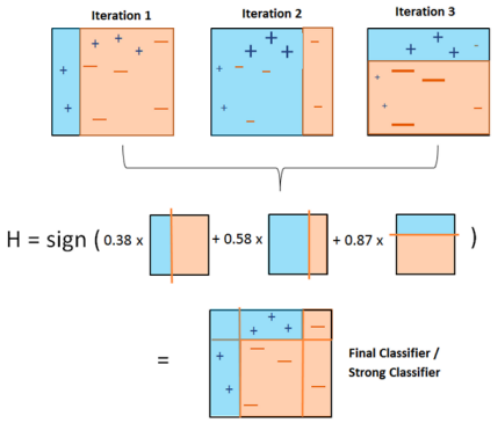

FIGURE 10 – AdaBoost with a decision stump as weak classifier

4.5 Gradient Boosting

In the FSAM step $(\nu_m, h_m) = \arg\min_{\nu\in\mathbb{R}, h\in F} \sum_{i=1}^{n} \mathcal{L}\left(y_i, f_{m-1}(x_i) + \nu h(x_i)\right)$, the difficult part is finding the best step direction h. In gradient boosting, the idea is to look for the locally best step direction like in gradient descent.

We want to minimize $J(f) = \sum_{i=1}^{n} \mathcal{L}(y_i, f(x_i))$. We will compute the gradient with respect to f. $J(f)$ only depends on f at the n training points. Therefore, we define $\mathbf{f} = (f(x_1), \ldots, f(x_n))^T$ and write the objective function as $J(\mathbf{f}) = \sum_{i=1}^{n} \mathcal{L}(y_i, f_i)$.
We consider gradient descent on

$$J(\mathbf{f}) = \sum_{i=1}^{n} \mathcal{L}(y_i, f_i)$$

The negative gradient step direction at f, also called the vector of **pseudo-residuals**, is

$$-\mathbf{g} = -\nabla_{\mathbf{f}} J(\mathbf{f}) = -\left(\frac{\partial \mathcal{L}(y_1, f_1)}{\partial f_1}, \ldots, \frac{\partial \mathcal{L}(y_n, f_n)}{\partial f_n}\right)$$

This can be easily calculated and represent the direction we want to change each of our n predictions on the training data.
Eventually, we need more than just $\mathbf{f}$ that is not precise enough. Therefore we will find f by finding the closest base hypothesis $h \in F$ (in the ℓ_2 sense) as such :

$$\min_{h\in F} \sum_{i=1}^{n} (-g_i - h(x_i))^2.$$

This is a least squares regression problem over the hypothesis space $\mathcal{F}$, that take the $h \in \mathcal{F}$ that best approximates $-\mathbf{g}$ as our step direction.

Finally we choose a stepsize :
-with a line search :

$$\nu_m = \arg\min_{\nu>0} \sum_{i=1}^{n} \mathcal{L}\left(y_i, f_{m-1}(x_i) + \nu h_m(x_i)\right).$$

-or as a fixed learning rate parameter. We consider $\nu = 1$ to be the full gradient step. We choose a fixed $\nu \in (0, 1)$ – called a **learning rate** or **shrinkage parameter**. This value can be optimized as a hyperparameter ($\nu = 0.1$ is typical).