Fondamentaux de l'Apprentissage Automatique

Lecturer: Yann Chevaleyre
Scribe: Liora TAIEB

Lecture n°9 #
24/11/2023

1 Introduction

This lecture is about decision trees and ensemble methods. We reviewed the definitions, constructions and motivations to use trees, bagging and boosting.

Summary : Decision trees use Empirical Risk Minimization (ERM) to solve classification and regression problems. But they are high variance models ; ensemble methods have been introduced to aggregate the decisions of several trees. The bagging method averages the results of the trees. The boosting method iteratively corrects weak learners.

Goals :
- Show how trees and ensemble methods heavily rely on Empirical Risk Minimization (ERM).
- Understand methods construction and use.

2 Decision Trees

2.1 Definitions and motivations

Definition 1. *A **decision tree** is a supervised machine learning algorithm structured into nodes (root and internal nodes) and leaves, joined by branches.*

Ex :
- Binary trees : each node has 2 or 0 children.
- Oblique decision trees or Binary Space Partition trees (BSP trees).
- Sphere trees.

Each leaf is a prediction, and each node a condition to separate data.

Definition 2. *The **test variable of a node** is the training variable used in the node to partition data. The **threshold of a node** is the dividing value of the test variable.*

If the test variable is continuous, the threshold can be a single value, a range, an inequality, or something else. For a discrete test variable, there is no threshold per say, the node will output as many branches as there are discrete values.

In this class, for simplicity, we will focus on binary trees, with univariate nodes, inequality threshold for continuous variable (2 output branches, one for the samples below the condition threshold, one for those above), and also nodes with 2 output branches for discrete values.

Remark : In the context of a binary tree in $\mathbb{R}^2$ with continuous variables, a classifier is a partition of the input space, i.e. a way of dividing the input space into different regions. This *is the idea behind every decision tree.*

Motivations

Here are 2 motivations for decision trees :

- Interpretability : Decision trees are easy to understand, and explicit features importance.
- Visualization : Decision trees can be graphically visualized, which makes them convincing.

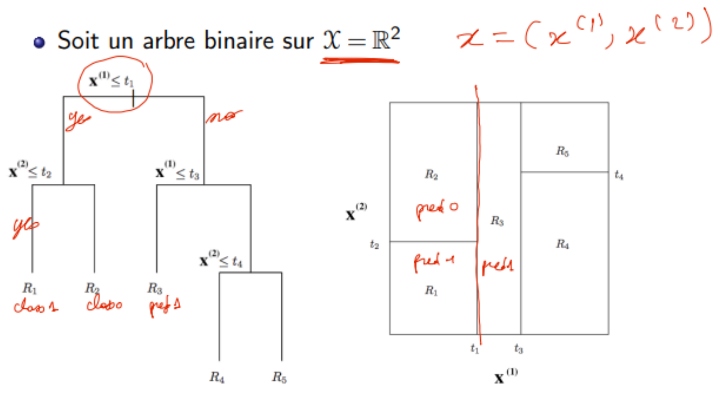

Soit un arbre binaire sur $\mathcal{X} = \mathbb{R}^2$
$x = (x^{(1)}, x^{(2)})$
$\mathbf{x}^{(1)} \leq t_1$
yes
no
$\mathbf{x}^{(2)} \leq t_2$
$\mathbf{x}^{(1)} \leq t_3$
$\mathbf{x}^{(2)} \leq t_4$
R_1
R_2
R_3
R_4
R_5
class 1
class 0
pred 1
pred 0
pred 1
t_1
t_2
t_3
t_4
$\mathbf{x}^{(1)}$
$\mathbf{x}^{(2)}$

FIGURE 1 – Visualization of a binary tree in $\mathbb{R}^2$.

2.2 Construction and limits

2.2.1 Regression

Let $R_1, ..., R_M$ be the partition into M regions of the input space ($\mathbb{R}^2$) by our binary decision tree. We remind our reader that a partition is a division of the space into non-overlapping and exhaustive subsets. The final prediction of the decision tree is

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

with c_m the predicted value of leaf m.

This formula relies on 2 unknown values : c_m and R_m. This separated our study in 2 parts, one to understand how to compute R_m and another to compute c_m.

How to choose a value for each c_m ?

Let's suppose we have a partition $R_1, ..., R_M$. Naturally,

$$\hat{c}_m = \text{average}\,(y_i \mid x_i \in R_m) = \frac{1}{|R_m|} \sum_{i \mid x_i \in R_m} y_i$$

This value is an application of Empirical Risk Minimization (ERM) with minimization of the square loss. Indeed,

$$\arg\min_{\hat{y}} \sum_{i|x_i \in R_m} \ell(\hat{y}, y_i) = \arg\min_{\hat{y}} \sum_{i|x_i \in R_m} (\hat{y} - y_i)^2 = \text{average}(y_i \mid x_i \in R_m)$$

Therefore, the ERM loss here is

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

Summary : The leaf values are calculated by ERM, with square loss minimization. Those values are not what matters, rather than the loss we use to calculate them.

How to decide on a partition $R_1, ..., R_M$?

This question is similar to asking : what variables and thresholds should the tree use, and in what order ?

Let's focus on the root of the tree. The tree repeats this step to compute the rest of the nodes. Let call $x^{(j)}$ a test variable, $s \in \mathbb{R}$ the threshold, $R_1(j,s)$, $R_2(j,s)$ the regions partitioned by x^j such that

$$R_1(j,s) = \{x^{(j)} \leq s\} \text{ and } R_2(j,s) = \{x^{(j)} \geq s\}.$$

Let's define $\hat{c}_1(j,s)$ and $\hat{c}_2(j,s)$ as the leaf values of $R_1(j,s)$ and $R_2(j,s)$, such that

$$\hat{c}_1(j,s) = average(y_i | x_i \in R_1(j,s)) \text{ and } \hat{c}_2(j,s) = average(y_i | x_i \in R_2(j,s)).$$

As before, a regression decision tree uses ERM with square loss to try to find j, s that minimize

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

To do that, we use the following algorithm.

Algorithm 1 Partition of the input space

```
1: Input : $(x_i, y_i)$ the dataset.
2: Output : $(j, s)$.
3: Choose a test variable $x^{(j)}$.
4: Sort $x_1^{(j)}, \ldots, x_N^{(j)}$ in ascending order.
5: for $i = 1$ to $N - 1$ do
6:     Compute $s_j = \frac{1}{2}\left(x_i^{(j)} + x_{i+1}^{(j)}\right)$.
7:     Store $s_j$ that minimizes $L(j, s)$.
8: end for
```

Complexity : $\mathcal{O}(dN \log N)$ with d the number of variables (sorting data is in $\mathcal{O}(N \log N)$). If we hadn't sorted the data points, the complexity would have been $\mathcal{O}(dN^2)$ (for each of the $N-1$ thresholds s_j, the algorithm goes through all N samples).

We can pinpoint information on the efficiency of decision trees. A too large tree will lead to overfitting (every x_i could have its own partition), and a too small tree will lead to underfitting. One can implement actions to prevent this :

- Limit the depth of the tree.
- Stop the splitting of a node when its regions contain too few examples.
- Use a posteriori pruning algorithm such as CART, Breiman et al 1984.

Summary : Splits are created following an ERM with square loss. ERM is therefore the base of regression decision trees construction. Moreover, one can implement techniques to prune the tree and improve its results.

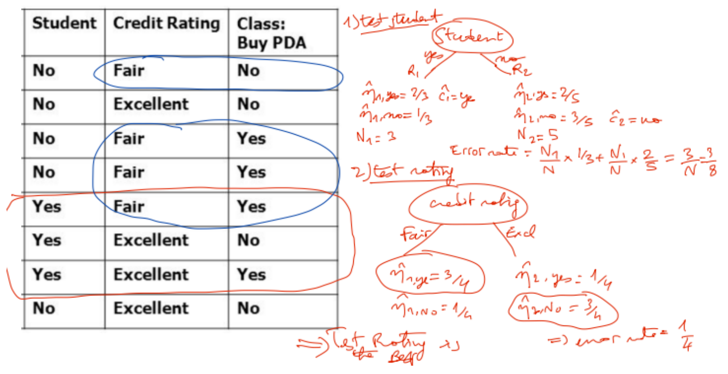
2.2.2 Classification

We will see that the construction of a classification decision tree is similar than the one of a regression tree. This time, let's denote $\mathscr{Y} = \{1 \dots K\}$ the set of classes, and $\hat{\eta}_{m,k}$ the proportion of examples of class K in R_m, such that

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

The classification prediction will naturally be the k maximizing $\hat{\eta}_{m,k}$, for each R_m. We apply ERM with 0/1 loss, minimizing the prediction error

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



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

1) test student
Student
yes
no
R_1
R_2
$\hat{\eta}_{1,yes} = 2/3$ $\hat{c}_1 = yes$
$\hat{\eta}_{1,no} = 1/3$
$N_1 = 3$
$\hat{\eta}_{2,yes} = 2/5$
$\hat{\eta}_{2,no} = 3/5$ $\hat{c}_2 = no$
$N_2 = 5$
Error rate $= \frac{N_1}{N} \times 1/3 + \frac{N_1}{N} \times \frac{2}{5} = \frac{3}{N} \frac{3}{8}$
2) test rating
credit rating
Fair
Excl
$\hat{\eta}_{1,yes} = 3/4$
$\hat{\eta}_{1,No} = 1/4$
$\hat{\eta}_{2,yes} = 1/4$
$\hat{\eta}_{2,No} = 3/4$
$\Rightarrow$ error rate $= \frac{1}{4}$
$\Rightarrow$ Test Rating ~~the~~ Best

Figure 2 – Example of root node construction in a classification decision tree.

Classification tree for class density prediction (CP loss)

One can apply the exact same method to predict class density instead of a single class. CP losses are functions that measures the quality of probabilistic predictions, i.e. the uncertainty of probabilistic predictions. So in that case, $\hat{c_m} = \hat{\eta}_{m,k}$ (we naturally defined the proportion of examples of class k in a region as the probability of occurrences of examples of class k in that same region), that will be given to the loss the tree wants to minimize. The error will therefore be

$$\sum_{i|x_i \in R_m} \ell\left(\hat{\eta}_m, y_i\right)$$

For the cross-entropy loss $\ell(\hat{\eta}, y) = -y \log \hat{\eta}(1-y)\log(1-\hat{\eta})$, the error is

$$N_m \cdot H_\ell = -N_m \cdot \left(\hat{\eta}_m \log \hat{\eta}_m + (1-\hat{\eta}_m)\log(1-\hat{\eta}_m)\right) = \text{Shannon entropy.}$$

Remarks :

— *We can go back to regular classification by defining $\hat{y}\left(R_m\right) = 1$ if $\hat{\eta}_m > \frac{1}{2}, 0$ otherwise.*

— *The value of the classification loss function in the region R_m is equivalent to the generalized entropy of the empirical probability $\hat{\eta}_m$ in that region. This highlights the connection between classification loss measures and the uncertainty associated with predictions in a specific region of feature space.*

— *We could also have used the information gain measure, equivalent to the entropy measure. It measures the reduction in entropy.*

Node impurity

We can further develop the previous idea of computing uncertainty with computing node impurity. Node impurity is a measure of how mixed or impure the classes are within a specific node. Impurity measures therefore provide a quantification of unpredictability within a node.

We mention 3 impurity measures :

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

— Gini index : $H_{Gini}(\hat{\eta}) = \sum_{k=1}^{K} \hat{\eta}_{m,k}\left(1-\hat{\eta}_{m,k}\right)$.

— Entropy (equivalent to using information gain) : $H_{CE}(\hat{\eta}) = -\sum_{k=1}^{K} \hat{\eta}_{m,k} \log \hat{\eta}_{m,k}$.

To construct such a tree, we find splits that minimize the weighted average of impurities

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

with N_1, N_2 the number of points in R_1 and R_2 respectively, and $H_\ell\left(R_1\right)$ and $H_\ell\left(R_2\right)$ generalized entropy (the node impurity measures).

We give results on the construction of classification decision trees by minimization of node impurity :

— For building the tree, Gini and Entropy seem to be more effective, as they push for more pure nodes, not just misclassification rate (see Figure 3).

— A good split may not change misclassification rate at all.

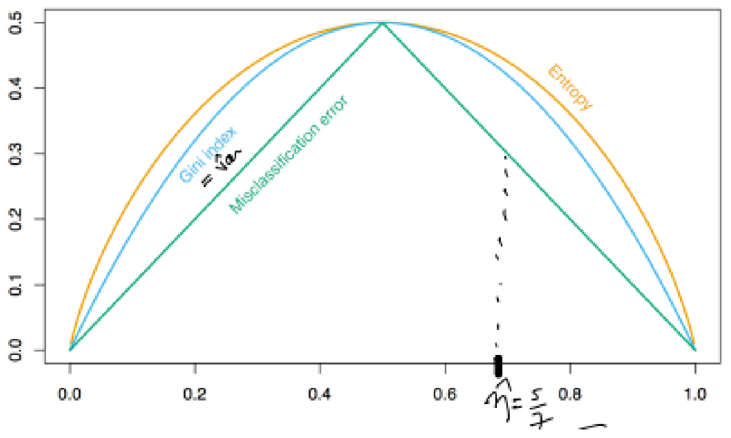


FIGURE 3 – Node impurity measures as a function of p, the relative frequency of class 1.

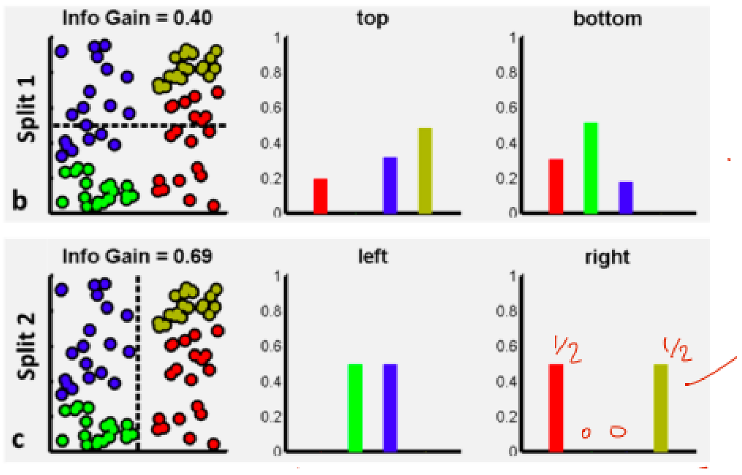


FIGURE 4 – 2 different splits of 4 equally distributed classes.

— In the following Figure 4, the misclassification rate for the two splits are the same, but Gini and entropy splits prefer Split 2.

Summary : ERM is also the base of classification decision trees construction. Using the 0/1 loss minimizes the prediction error. Using CP losses minimizes uncertainty of predictions. The tree can be constructed by minimizing node impurity. All of those losses are equivalent.

2.2.3 Limits

Decision trees, while powerful and interpretable, have several limitations that can impact their performance. These limitations have led to the development of ensemble methods, such as Random Forests and Gradient Boosting, which combine multiple decision trees to mitigate these issues.

To name just a few, overfitting (poor generalization skills) and instability (sensitivity to data changes) are problems leading to unsatisfaying prediction results.

3 Ensemble methods

There are 2 types of ensemble methods :

— Parallel ensembles : each model is built independently (ex : random forests). The idea is to combine many (high complexity, low bias) models to reduce variance.
— Sequential ensembles : models are generated sequentially (ex : gradient-boosting). The idea is to add new models that do well where previous models lack.

Ensemble methods improve robustness of decision trees. Indeed, let $z, z_1, \ldots, z_n$ be i.i.d. with $\mathbb{E}z = \mu$ and $\mathrm{Var}(z) = \sigma^2$. Let's consider the average of the z_i's. We end up with the same expected value for the average of the z_i's compared to the one of z_1, but with a smaller standard error :

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

3.1 Bagging

Bagging is a parallel ensemble methods aggregating the results of several models. For a regression problem, bagging models compute the mean of the aggregated models. For classification, this is typically done through a majority voting mechanism.

We saw previously that ensemble methods like bagging lower the variance of a model. We would like to see its effect on the bias/variance decomposition, to assess of the better generalization and robustness of a bagging method. Suppose we have B independent training sets from the same distribution. Suppose $\hat{y} = \mathbb{R}$. The learning algorithm gives B decision functions : $\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_B(x)$. Let's define the average prediction function as :

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

Remark : The B independent training sets are random, which gives rise to variation among the $\hat{f}_b$'s.

Let's fix some particular $x_0 \in X$. Consider $\hat{f}_{\text{avg}}(x_0)$ and $\hat{f}_1(x_0), \ldots, \hat{f}_B(x_0)$ as random variables. This lead to $\hat{f}_1(x_0), \ldots, \hat{f}_B(x_0)$ being i.i.d., even though we have no idea about their distributions.

The average prediction on x_0 is $\hat{f}_{\text{avg}}(x_0) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x_0)$. $\hat{f}_{avg}(x_0)$ has smaller variance : $Var\left(\hat{f}_{avg}(x_0)\right) = \frac{1}{B} Var\left(\hat{f}_1(x_0)\right)$. But $\hat{f}_{\text{avg}}(x_0)$ and $\hat{f}_b(x_0)$ have the same expected value, so the bias does not change :

$$\underbrace{\mathbb{E}\left[\hat{f}_{avg}(x_0) - \mathbb{E}[Y \mid x_0]\right]}_{bias(\hat{f}_{avg})} = \underbrace{\mathbb{E}\left[\hat{f}_1(x_0) - \mathbb{E}[Y \mid x_0]\right]}_{bias(\hat{f}_1)}.$$

This gives us the bias/variance decomposition of the mean square error at x_0 :

$$\begin{aligned}
\mathbb{E}\left[\left(\hat{f}_{vg}\left(x_0\right)-Y\right)^2 \mid X=x_0\right] &= bias^2\left(\hat{f}_{avg}\right)+Var\left(\hat{f}_{avg}\left(x_0\right)\right)+Var\left(Y \mid x_0\right) \\
&= bias^2\left(\hat{f}_1\right)+\frac{1}{B}Var\left(\hat{f}_1\left(x_0\right)\right)+Var\left(Y \mid x_0\right)
\end{aligned}$$

The bias/variance trade-off shows how the ensemble effectively reduces variance while maintaining bias, leading to an overall improvement in predictive performance.

But in practice, we don't have B independent training sets : we introduce the bootstrap method.

3.1.1 The bootstrap method

Bootstrap creates B new and randomly selected training sets to recreate an environment of i.i.d. samples. The sets are the same size as the original dataset, and the samples are selected with replacement. The bagging prediction function is then a combination of the prediction functions yielded by individual models on those training sets.

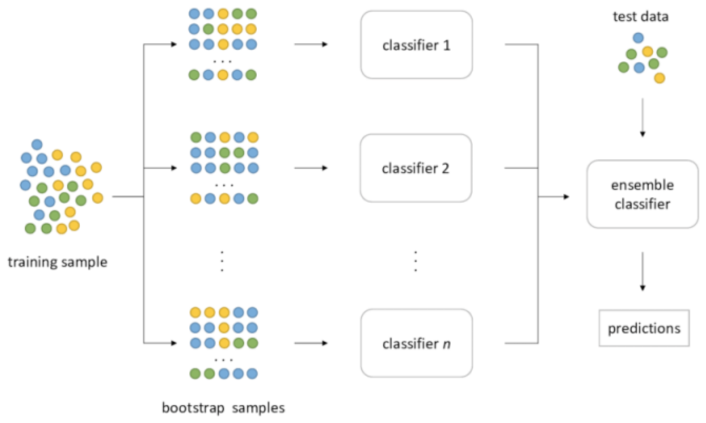


FIGURE 5 – Complete bagging process with bootstrap method.

For regression, the prediction is the mean of the predicted values of the set of models. We find the same results on variance and bias as previously. There is two way to combine classifications : either consensus class (majority vote) or average probabilities, as we saw in the first part of the class.

To dig deeper into the process of selecting samples, it is possible that some data points from the original dataset are never selected. Those are called "out-of-bag" (OOB) observations, and represent 37% of the original training set.

Among N samples, each one has a $\frac{1}{N}$ probability of being selected, and a $1-\frac{1}{N}$ probability of not being selected. The probability that a specific data point is not selected in a single draw is therefore $(1 - \frac{1}{N})^N$. As the number of draw approaches infinity, this probability converges to $e^{-1} \approx 0.368$, so there is a 36.8% chance that a particular data point will not be selected in a single draw. All of this mean that each bagged predictor is trained on about 63% of the data.

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

To conclude, general sentiment is that bagging helps most when the approximation error (bias) is low or the model is high variance. But it is hard to find clear and convincing theoretical results on this.

3.1.2 Random forests

Bootstrap samples are independent samples from the training set, but are not independent from other samples in other bags. This dependence limits the amount of variance reduction we can get.

Random forests are a bagged decision trees model, but modify the tree-growing procedure to reduce the dependence between trees. When constructing each tree node, the algorithm restrict choice of splitting variable to a randomly chosen subset of features of size m.

Typically, the chosen m if approximately $\sqrt{p}$, where p is the number of features. It is a heuristic that strikes a balance between introducing randomness and ensuring that a sufficient number of features are considered. It prevents the subset size from being too small (which might lead to under-fitting) or too large (which might reduce the decorrelation among trees). The m can be chosen by cross validation if needed.

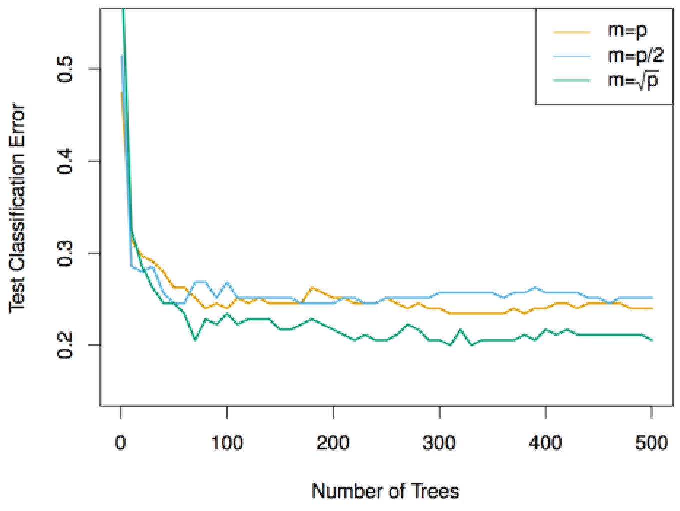


FIGURE 6 – Ablation study on the impact of the value of m on the classification error.

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. Moreover, random forest algorithms work well with tabular data, as we select features and do not take all points into account (unlike for example image or text processing algorithms, which need all the data.)

3.1.3 Limits

Suppose $Z, Z_1, \dots, Z_n$ i.i.d. with $\mathbb{E}Z = \mu$ and $VarZ = \sigma^2$ are correlated, and $\forall i \neq j, Corr\left(Z_i, Z_j\right) = \rho$. Then

$$Var\left[\frac{1}{n}\sum_{i=1}^{n} Z_i\right] = \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2.$$

For large n, the $\rho\sigma^2$ term dominates. This limits benefit of averaging.

Summary : Bagging combines results of individual models to lower the variance of predicted values, without increasing the bias. Bagging methods use a resampling method called bootstrap to simulate an i.i.d. environment. Random forests are the application of bagging on decision trees.

3.2 Boosting

Boosting combines the strengths of multiple weak models. The final prediction of the boosted model is a weighted combination (or voting mechanism) applied to these weak models. Each model creates its own hypothesis, which results in the following combined hypothesis space

$$\mathcal{F}_M = \left\{\sum_{m=1}^{M} v_m h_m(x) \mid v_m \in \mathbf{R}, h_m \in \mathcal{F}, m = 1, \dots, M\right\}$$

with $\mathcal{F}$ the base hypothesis space.

Learning is choosing $v_1, \dots, v_M \in \mathbf{R}$ and $h_1, \dots, h_M \in \mathcal{F}$ to fit S. In bagging, we learn h_i, but $v_i = \frac{1}{M}$ for all classifiers. Boosting will learn both.

Just as for individual decision trees and bagging, we'll consider learning the individual hypothesis by empirical risk minimization :

$$\hat{h} = f \in \mathcal{F}_M \arg\min \frac{1}{n}\sum_{i=1}^{n} \ell\left(h\left(x_i\right), y_i\right)$$

for some loss function $\ell(y, \hat{y})$. The ERM objective function becomes

$$J(v_1, \dots, v_M, h_1, \dots, h_M) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(y_i, \sum_{m=1}^{M} v_m h_m(x)\right).$$

We suppose our base hypothesis space is parameterized by $\Theta = \mathbf{R}^d$:

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

For some hypothesis spaces and typical loss functions, we can differentiate J w.r.t. v_m's and θ_m's, and optimize with Stochastic Gradient Descent (SGD).

Boosting applies whenever we can compute a particular form of the above ERM, and our loss function is [sub]differentiable w.r.t. training predictions $f(x_i)$, so that we can do regression with the base hypothesis space $\mathcal{F}$ (gradient-boost). We will study the case when the base hypothesis space $\mathcal{F}$ consists of decision trees.

3.2.1 Forward Stagewise Additive Modeling (FSAM algorithms)

FSAM is a boosting algorithm ; it focuses on the remaining error (residuals) of the combined model. At each iteration, the weights of the goodly classified examples are unchanged, and the rest is boosted.

This is a greedy algorithm. Indeed, the idea behind greedy algorithms is to make a serie of choices that leads to a solution that is as close as possible to the optimal. FSAM, at each stage of its iterative process, adds a new weak learner information to the final score function that best improves the current ensemble's fit to the data.

We start with $f_0 \equiv 0$. After $m-1$ stages, we have

$$f_{m-1} = \sum_{i=1}^{m-1} v_i h_i.$$

In the m'th round, we want to find the step direction $h_m \in \mathcal{F}$ (i.e. a hypothesis) and a step size $v_i > 0$ such that

$$f_m = f_{m-1} + v_i h_m$$

improves the objective function value by as much as possible.

Here is the complete FSAM algorithm.

Algorithm 2 FSAM algorithm

```
1: Input : $(x_i, y_i)$ the dataset.
2: Output : a classifier $f_M$.
3: Initialize $f_0(x) = 0$.
4: for $m = 1$ to $M$ do
5:     Compute $(v_m, h_m) = v \in \mathbf{R}, h \in \mathcal{F} \arg\min \frac{1}{n}\sum_{i=1}^{n} \ell(f_{m-1}(x_i), y_i)$.
6:     Set $f_m = f_{m-1} + v_m h$.
7: end for
```

Application of FSAM on regression

For regression, we naturally choose the mean square error loss

$$L(v,h) = \frac{1}{n}\sum_{i=1}^{N} (y_i - [f_{m-1}(x_i) + vh(x_i)])^2.$$

If $\mathcal{F}$ is "closed by change of scale", then v becomes negligible in comparison to $f_{m-1}(x_i)$. We can forgot it and focus on learning h. We try to minimize

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

This algorithm, called "matching pursuit", is equivalent to using a least squares method on the residuals, hence learning on weak learners.

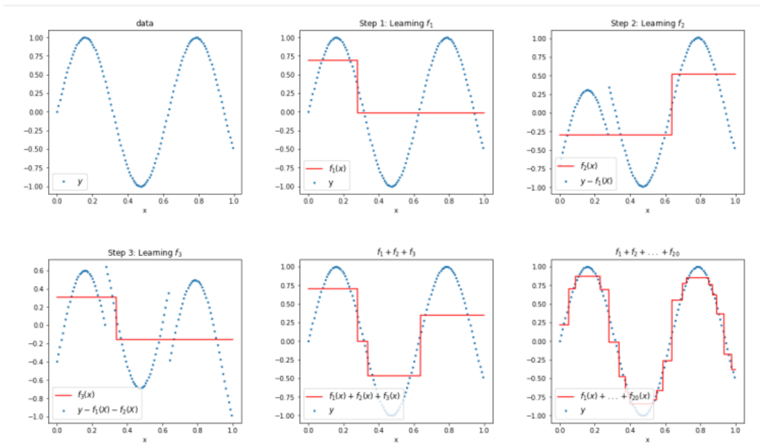


FIGURE 7 – Application of simple boosting with decision stomps for regression.

A regression stump is a regression tree with a single split. It is a function of the form $h(x) = a 1\left(x_i \leqslant c\right) + b 1\left(x_i > c\right)$. We observe on the previous Figure (Figure 7) the creation at each iteration of a regression stump split. The dataset of residuals is provided to the following tree, on which it will place another regression stump. At the end, we sum the intermediate classifiers to obtain a more complex and fitted prediction function.

Application of FSAM on classification

For classification, we come back to our goal of producing a "hard"-classifier (i.e. a binary scoring function). Computing each new classifier is not easy for losses different than the exponential loss. The choice of the exponential loss is connected to the convex nature of its optimization landscape, which facilitates efficient optimization during boosting iterations. Also, it places higher emphasis on instances that are misclassified by the current ensemble.

Boosting algorithms, including AdaBoost, rely on a specific update rule that minimizes the loss function during each iteration. The update rule in AdaBoost is derived from minimizing the exponential loss. If you were to use a different loss function, the mathematical derivation of these weights would likely change, and it might not be straightforward to find a direct substitute. However, the exponential loss is not robust to outliers.

Let's dive deeper into Adaboost. Adaboost is a boosting algorithm, mixing an FSAM algorithm with an exponential loss. In the model, each example in the training set is assigned an initial weight, initially equal. In each iteration, the algorithm focuses on the examples that were misclassified in the previous iteration, and adjusts the weights of these misclassified examples to give them more importance in the next iteration. This construction is what makes Adaboost an FSAM algorithm. Here is the complete algorithm.

Algorithm 3 Adaboost algorithm

```
1: **Input :** training set $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$.
2: **Output :** a classifier $f_M$.
3: Initialize observation weights $w_i = 1, i = 1, 2, \ldots, N$.
4: **for** $m = 1$ to $M$ **do**
5:     Learner fits weighted training data and returns $h_m(x)$.
6:     Compute weighted empirical 0-1 risk :
           $$err_m = \frac{1}{W} \sum_{i=1}^{n} w_i 1\left(y_i \neq h_m\left(x_i\right)\right) \quad whereW = \sum_{i=1}^{n} w_i.$$
7:     Compute classifier weight $v_m = \ln\left(\frac{1 - \mathrm{err}_m}{\mathrm{err}_m}\right)$.
8:     Set $w_i \leftarrow w_i \cdot \exp\left[v_m 1\left(y_i \neq h_m\left(x_i\right)\right)\right], \quad i = 1, 2, \ldots, n$ [example weight adjustment].
9: **end for**
```

Here is an example of the implementation of Adaboost.

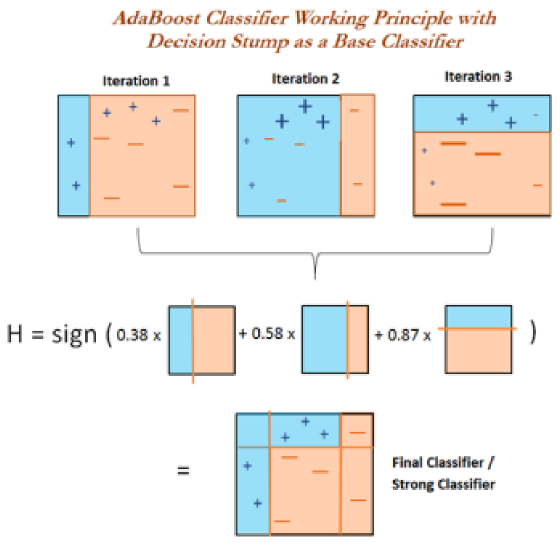

AdaBoost Classifier Working Principle with Decision Stump as a Base Classifier
Iteration 1
Iteration 2
Iteration 3
H = sign (0.38 x
+ 0.58 x
+ 0.87 x
)
=
Final Classifier / Strong Classifier

FIGURE 8 – Iterations of Adaboost.

Link with online machine learning

Adaboost resembles the hedge online algorithm we saw in a previous course, as it learns from one sample at a time to predict the best sequence of classifiers - the one that minimizes the cumulated loss. Online learning we saw can be reversed to build the worst example for a classifier, which is nearly the same as Adaboost.

Through the reversed online learning algorithm, we were able to formulate the following theorem.

Theorem 1. *Under the hypothesis of weak learning, after* $T = \frac{2}{\gamma^2}\ln(N)$ *timesteps, the majority classifier has a zero classification error rate on the sample* S.

Generalization - "Anyboost"

The hardest part in the FSAM step is finding the best h hypothesis, i.e. direction to minimize the loss. We could try and apply a local method such as gradient descent to optimize the step direction.

We want to minimize

$$J(f) = \sum_{i=1}^{n} \ell\left(y_i, f\left(x_i\right)\right).$$

In some sense, we want to take the gradient w.r.t. "f", but we don't have f. So we create a vector of predictions of f and rewrite the objective function as

$$J(\mathbf{f}) = \sum_{i=1}^{n} \ell\left(y_i, \mathbf{f}_i\right).$$

We then consider gradient descent on the objective function. The negative gradient step direction at f is $g \in \mathbb{R}^N$

$$-\mathbf{g} = -\nabla_f J(\mathbf{f}) = -\left(\partial_{\mathbf{f}_1}\ell\left(y_1, \mathbf{f}_1\right), \ldots, \partial_{\mathbf{f}_n}\ell\left(y_n, \mathbf{f}_n\right)\right)$$

which we can easily calculate. Therefore, $-\mathrm{g} \in \mathrm{R}^n$ is the direction we want to change each of our n predictions on training data. It is also called the "pseudo-residuals" (for square loss, they are exactly the residuals).

We need to find the closest base hypothesis $h \in \mathcal{F}$ (in the ℓ^2 sense) :

$$\min_{h \in \mathcal{F}} \sum_{i=1}^{n} \left(-\mathbf{g}_i - h\left(x_i\right)\right)^2.$$

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

Finally, we need to choose a stepsize. We can use line search :

$$v_m = v > 0 \arg\min \sum_{i=1}^{n} \ell\left\{y_i, f_{m-1}\left(x_i\right) + v h_m\left(x_i\right)\right\}.$$

The second option, and the more commom one, is to add a learning rate parameter to control the behavior of the gradient. We consider $v = 1$ to be the full gradient step. We choose a fixed $v \in (0, 1)$ called a learning rate or shrinkage parameter. A typical value of v is 0.1, and it should be optimize as a hyperparameter.

To conclude, here is a recap of gradient descent in boosting algorithm.

— Take any loss function [sub]differentiable w.r.t. the prediction.
— Choose a base hypothesis space for regression.
— Choose number of steps (or a stopping criterion).
— Choose step size methodology.

Summary : Boosting learns individuals classifiers and assigns each one a weight, in contrast with bagging that doesn't discriminate. It boosts the weights of wrongly classified samples and uses ERM to minimize the prediction error. This method is applicable to regression with the "matching pursuit" algorithm, and to classification with Adaboost. We can implement gradient descent to find the best individuals classifiers.

4 Conclusion

This course provides a comprehensive understanding of decision trees and their role in machine learning, as well as the benefits of using ensemble methods to improve results. Decision trees, bagging, and boosting all revolve around minimizing empirical risk, i.e., reducing training error. By addressing overfitting (high variance), bagging and boosting enhance the model's ability to generalize to unseen data. We saw a variety of optimization to apply to give the models more robustness and efficiency.