Foundations of Machine Learning

Lecturer: Yann Chevaleyre
Scribe: Ángel Luque Lázaro

Lecture nº9 #
24/11/2023

1 Decision Trees

A decision tree is a decision model where we start with a single node called "root" and at every step we make a division setting thresholds over a selected feature, resulting in new nodes. These new nodes are called "children" (therefore, the node from which these results, is called "father"). Nodes without children are called "leaves".

In a binary decision tree, every node can have either 2 or 0 children.

Semantically, a binary decision tree divides the input space into decision areas. These can be shown as follows for a binary tree where $\mathcal{X} = \mathbb{R}^2$:

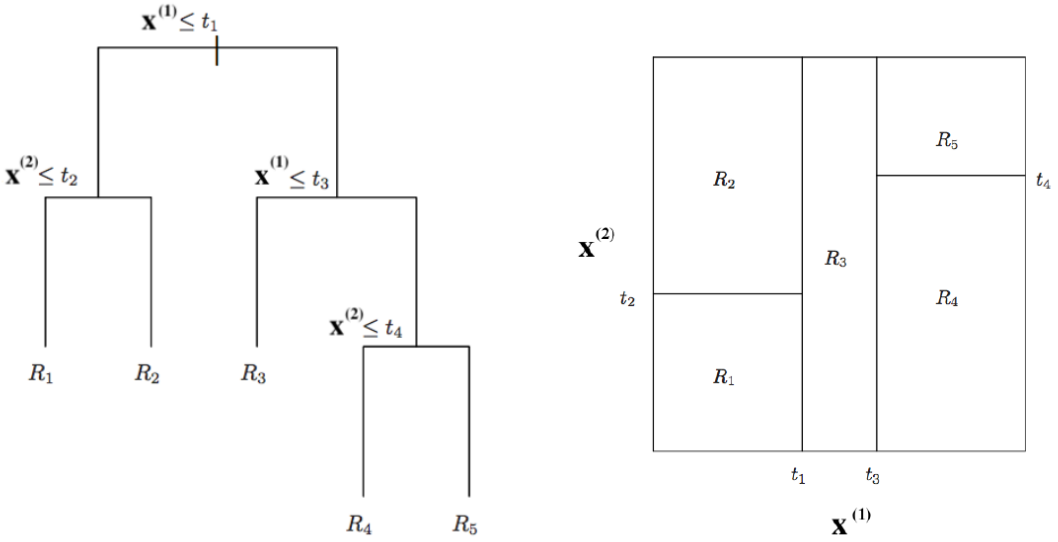

Figure 1: Division of input space in binary decision tree

In the root and in every inner node, a variable $x^{(j)}$ is called "test variable" or "decision variable" of the node, and t is called the "seed".

Apart from binary trees, there are other types of decision trees, like oblique decision trees or binary space partition trees (BSP trees), or sphere trees.

1.1 Regression Trees

A regression task can be modelled into a binary tree.

Coming back to its representation in the space, let it be a binary tree where $\mathcal{X} = \mathbb{R}^2$. One example could is $x = x^{(1)}, x^{(2)}$.

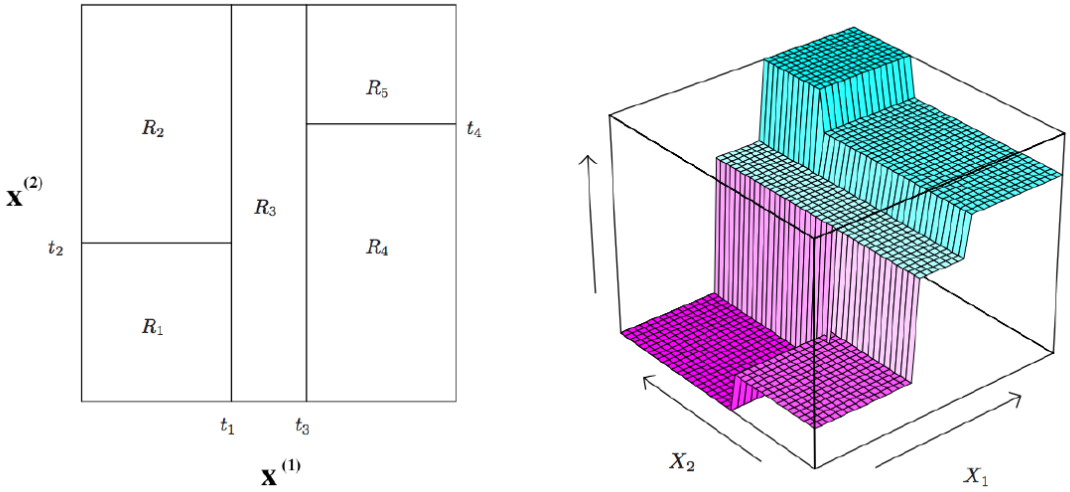

Figure 2: Division of input space in binary decision tree

A decision tree partitions the $\mathcal{X}$ in regions $R_1, ..., R_M$. These regions are disjoint, meaning that the follow both $\mathcal{X} = R_1 \cup R_2 \cup ... \cup R_M$ and $R_i \cap R_j = \emptyset \ \forall i \neq j$

With this partition, the final prediction follows:

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

where c_m is the value of the leaf, and $1(x \in R_m)$ is a function whose value is 1 if x is in the R_m region and 0 otherwise.

1.1.1 Values of leaves and loss

Let's suppose that we already have the partitions, then how can we chose $c_1, ..., c_M$ (values of the leafs)? Also, for the loss function $\ell(\hat{y}, y) = (\hat{y} - y)^2$, how can we apply the empirical risk minimization (ERM)?

$$\begin{aligned} c_1, ..., c_m &= argmin \sum_{i=1}^{M} (f(x_i) - y_i)^2 \\ &= \sum_{i=1}^{N} (\sum_{m=1}^{M} c_m 1(x \in R_m) - y_i)^2 \\ &= argmin \sum_{m=1}^{M} \sum_{i \in R_m} (c_m - y_i)^2 \end{aligned}$$

This problem is separable, given that each region is independent. So in each region we minimize the loss on the points that belong to it. Therefore,

$$\begin{aligned} \forall m, c_m &= argmin \sum_{i \in R_m} (c_m - y_i)^2 \\ &= average(y_i | y_i \in R_m) \end{aligned}$$

This tells us that, given the partitions, the right value for each leaf is the average of all samples belonging to that region.

In terms of the loss for each node:

$$\text{For } c_m, \sum_{i \in R_m} \ell(c_m, y_i) = \sum_{i \in R_m} (average(y_i | y_i \in R_m) - y_i)^2 = N_m * \hat{Var}(\{y_i \in R_m\})$$

1.1.2 Growing the regression tree

For the root node, using real variables, let it be $x = (x^{(1)}, ..., x^{(d)}) \in \mathbb{R}^d$ (d variables), a test variable $x^{(j)}$ and seed $s \in \mathbb{R}$, then a partition based on $x^{(j)}$ and s is:

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

For continuous variables, for each variable $x^{(j)}$ and seed s,

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

At each step, we need to find values j and s that minimize the sum of errors on both sides of the partition. This is minimizing:

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

How can we do this?

For finding the seed, the naive way would be as follows. Suppose we have chosen the test variable $x^{(j)}$. Suppose also that $x_1^j, ..., x_N^{(j)}$ are sorted in increasing order. Traditionally, all our candidates would be middle values between to consecutive values, meaning:

$$s_j = \{\tfrac{1}{2}(x_i^{(j)} + x_{i+1}^{(j)}) | i = 1, ..., N-1\}$$

Therefore, we test N-1 seeds, resulting on a complexity of $O(dN^2)$. Luckily, we can find an improved algorithm for this with complexity $O(dNlogN)$.

With this procedure, we would have divided the space in two regions R_1 and R_2, having then a tree with one root and two leaves. We can grow the tree recursively by repeating this algorithm on each of the resulting leaves. However, this begs the question: when to stop?.

Controlling the growth of the tree is important, as having a tree too large could result in overfitting (regions containing few or only one sample) and having a tree too small could be a sign of underfitting. One idea would be only divide a partition that has a minimum amount of samples (this minimum would be an hyperparameter), or another option would be to use a pruning algorithm a posteriori (as, for example, CART [1]).

1.2 Classification Trees

Decision trees can be used for classification tasks (probably their most used and straighforward setting).

1.2.1 Values of leaves and loss

Let it be $Y = \{1...K\}$. Same reasoning as before: we suppose we already have the regions set. Node m represents region R_m, with N_m examples. The proportion of examples of class $k \in Y$ in the region R_m is:

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

If we predict in the node m the class k, then the error rate of the examples in R_m will be:

$$1 - \hat{\eta}_{m,k}$$

Therefore, to minimize the error rate (0/1 loss), the predicted class for a node m will be:

$$\hat{y}(m) = argmin_k 1 - \hat{\eta}_{m,k} = argmax_k \hat{\eta}_{m,k}$$

This is to say, the predicted class is the majority class in the region.
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 3: Data for exercise

Test student

1 -> student = yes
$\hat{\eta}_{1,yes} = 2/3$
$\hat{\eta}_{1,no} = 1/3$
$\hat{c}_1 = yes$
$N_1 = 3$

2 -> student = no
$\hat{\eta}_{2,yes} = 2/5$
$\hat{\eta}_{2,no} = 3/5$
$\hat{c}_2 = no$
$N_2 = 5$

Error rate $= \frac{N_1}{N} * 1/3 + \frac{N_2}{N} * 2/5 = \frac{3}{N} = \frac{3}{8}$

Test rating

1 -> rating = fair
$\hat{\eta}_{1,yes} = 3/4$
$\hat{\eta}_{1,no} = 1/4$
$\hat{c}_1 = yes$
$N_1 = 4$

2 -> rating = excellent
$\hat{\eta}_{2,yes} = 1/4$
$\hat{\eta}_{2,no} = 3/4$
$\hat{c}_2 = no$
$N_2 = 4$

Error rate $= \frac{N_1}{N} * 1/4 + \frac{N_2}{N} * 1/4 = \frac{2}{N} = \frac{2}{8} = \frac{1}{4}$

We could adapt our classification tree for CP-loss, meaning that we learn a "soft"-classifier predicting the probabilities of the classes: $f : \mathcal{X} \to \Delta_K$, where Δ_K is the simplex of probabilities of dimension K. Therefore, adapting our last example to this, we would obtain for the test variable student something like:

Test student

1 -> student = yes
$\hat{c}_1$ = predict yes with probability 2/3 and no with probability 1/3

2 -> student = no
$\hat{c}_2$ = predict yes with probability 2/5 and no with probability 3/5

In the case of $Y = 0, 1$, and the node m representing the region R_m with N_m examples, then the proportion of examples of class 1 in R_m is

$$\hat{\eta}_m = \hat{P}(Y = 1|X \in R_m) = \frac{1}{N_m} \sum_{i:x_i \in R_m} 1(y_1 = k)$$

Then ERM porposes the following prediction for class 1:

$$\begin{aligned} \hat{c}_m &= argmin_{\hat{y}} \sum_{i:x_i \in R_m} \ell(\hat{y}, y_i) \\ &= argmin_{\hat{y}} (\hat{\eta}_m \ell(\hat{y}, 1) + (1 - \hat{\eta}_m)(\ell(\hat{y}, 0)) \end{aligned}$$

If the loss function is a CP-loss proper, then $\hat{c}_m = \hat{\eta}_m$.
Therefore, the loss value in R_m will be:

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

For cross-entropy $\ell(\hat{\eta}, y) = -ylog\hat{\eta} - (1 - y)log(1 - \hat{\eta})$, then it would be:

$$\begin{aligned} &\sum_{i:x_i \in R_m} y_i log\hat{\eta}_m + (1 - y_i)log(1 - \hat{\eta}_m) \\ = &\sum_{i:x_i \in R_m} y_i log\hat{\eta}_m + log(1 - \hat{\eta}_m) - y_i log(1 - \hat{\eta}_m) \\ = &\sum_{i:x_i \in R_m} y_i log\hat{\eta}_m + (1 - y_i)log(1 - \hat{\eta}_m) \end{aligned}$$

With $Y = \{0, 1\}$:

$$\begin{aligned} \sum_{i:x_i \in R_m} &= N_m * \hat{\eta}_m * \ell(\hat{\eta}_m, 1) + N_m * \hat{\eta}_m * (1 - \hat{\eta}_m) * \ell(\hat{\eta}_m, 0) \\ &= N_m * \hat{\eta}_m * (-log(\hat{\eta}_m)) + N_m * (1 - \hat{\eta}_m) * (-log(1 - \hat{\eta}_m)) \\ &= -N_m * (\hat{\eta}_m * log\hat{\eta}_m + (1 - \hat{\eta}_m) * log(1 - \hat{\eta}_m)) \\ &= \text{Shannon's entropy} \end{aligned}$$

If we're looking to predict a class more than a probability, then we will take $\hat{y}(R_m) = 1$ if $\hat{\eta}_m > \frac{1}{2}$, and 0 otherwise.

Remark: for all PC-loss ℓ, the value of these loss in the region R_m is the generalised entropy $H_l(\hat{\eta}_m)$.

1.2.2 Two-Class Node Impurity Measures

Consider leaf node m representing region R_m, with N_m observations. Three measures $Q_m(T)$ of node impurity for leaf node m:

Misclassification error:

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

Gini index:

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

Entropy or deviance (equivalent to using information gain):

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

As a graphical example:

- Consider binary classification
- Let p be the relative frequency of class 1.
- Here are three node impurity measures as a function of p

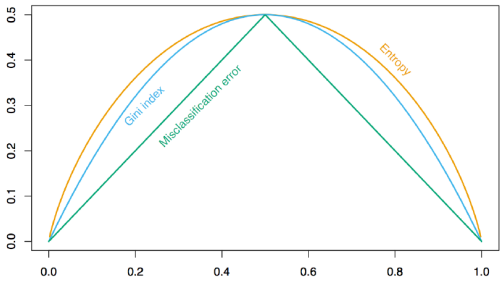

Figure 4: Graphical example for two-class node impurity measures

1.2.3 Splitting the tree

As an example, we could have this classes distribution before the split:

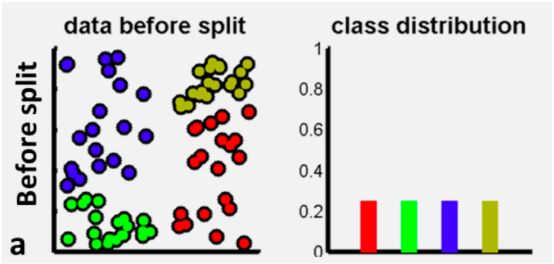

Figure 5: Example distribution pre-split

Two proposed splits (and their effects on the distribution) could be this:

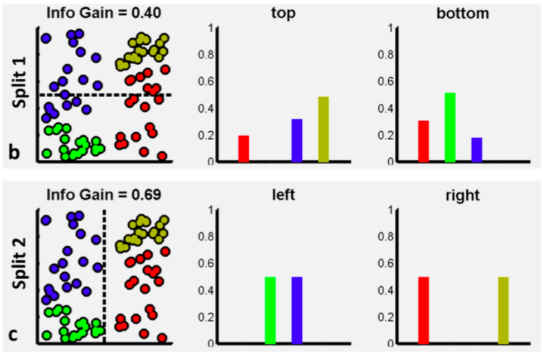

Figure 6: Proposed splits on distribution

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 $H_l(R_1)$ and $H_l(R_2)$ be generalized entropy (the node impurity measure). Then, to split a node, we have to find the split that **minimizes the weighted average of node impurities**, which is:

$$\text{avg entropy} = \frac{N_1}{N} H(R_1) + \frac{N_2}{N} H(R_2)$$

Then, coming back to the previous example.
Entropy for k classes:

$$H(\hat{\eta}_n ... \hat{\eta}_k) = -\sum_{k=1}^{K} \hat{\eta}_k log_2 \hat{\eta}_k$$

Initial class distribution = (1/4...1/4)

$$\text{entropy} = -log_2 \frac{1}{4} = 2$$

Entropy for split 2:

$$\text{entropy} = (-\frac{1}{2} log_2 \frac{1}{2}) * 2 + 0 * 2 = log_2 2 = 1$$

Average entropy of the tree (split 2):

$$\text{avg entropy} = \frac{1}{2} * 1 + \frac{1}{2} * 1 = 1$$

As a remark, for building the tree, Gini and Entropy seem to be the most effective. They push for more pure nodes, not just misclassification rate, given that a good split may not change misclassification rate at all.

2 Bagging and Random Forests

2.1 Ensemble methods

Ensemble methods are those that combine multiple models for their predictions. Two types:

- **Parallel ensembles**: each model is built independetly

- e.g. bagging and random forests.
- Main idea: combine many (high complexity, low bias) models to reduce variance.

- **Sequential ensembles**: models are generated sequentially.
  - Main idea: try to add new models that do well where previous models lack.

2.1.1 The benefits of averaging

Let $z, z_1, ..., z_n$ i.i.d., $\mathbb{E}z = \mu, and Var(z) = \sigma^2$.

If we use a single z_i to estimate μ, then:

$$\mathbb{E}z_i = \mu \text{ and } Var(z) = \sigma^2$$

However, if the consider the average of the z_i's, then:

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

Average has the same expected value but smaller standard error. Clearly, the average is preferred to a single z_i as estimator.

2.1.2 Averaging independent prediction functions

Suppose we have B independent training sets from the same distribution. The, learning algorithm gives B decision functions: $\hat{f}_1(x), \hat{f}_2(x), ..., \hat{f}_B(x)$. We define the average prediction function as:

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

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

For some fixed $x_0 \in \mathcal{X}$, the average prediction is:

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

Because we know that $\hat{f}_1(x), \hat{f}_2(x), ..., \hat{f}_B(x)$ are i.i.d., then we can prove that $\hat{(f)}_{avg}(x_0)$ has smaller variance than one just predictor: $Var(\hat{(f)}_{avg}(x_0)) = \frac{1}{B}Var(\hat{(f)}_1(x_0))$

Considering this and the fact that the expected value doesn't change, it seems like using this average predictor is a good idea.

However, in practice we don't have B independent training sets. Here's where bootstrapping comes in.

2.2 Bootstrap and bagging

Bootstrap samples are datasets with the same size as the original dataset, composed by sampling randomly with replacement from the original dataset.

For bagging, we draw B bootstrap samples $D^1, ..., D^B$ from original data S. We let $\hat{f}_1(x), \hat{f}_2(x), ..., \hat{f}_B(x)$ be the prediction functions from training on $D^1, ..., D^B$ respectively. Then, the **bagged prediction function** is a combination of these:

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

A visual representation of the process is as follows:

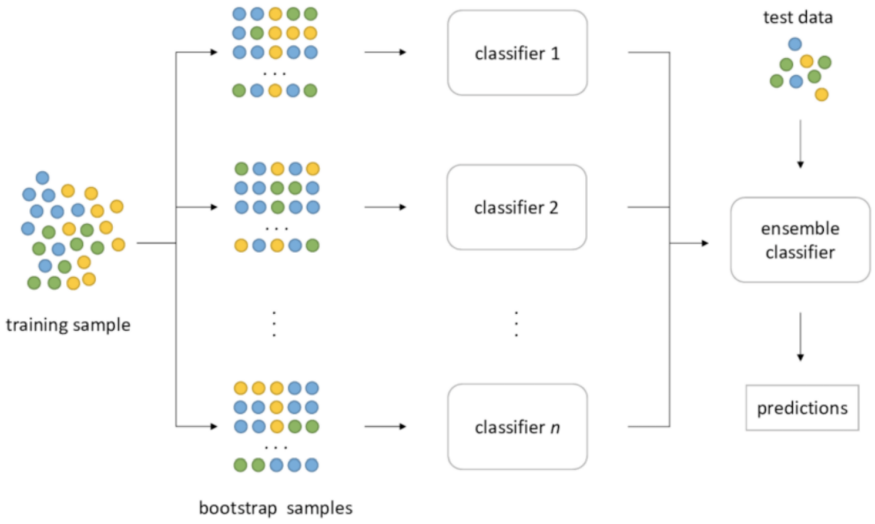

Figure 7: Visual representation of bagging

As an example, when using bagging for regression, the bagged prediction function would just be the average of all predictions.

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x)$$

Empirically, $\hat{f}_{bag}$ often performs similarly to what we'd get from training on B independent samples: $\hat{f}_{bag}(x)$ has the same expectation as $\hat{f}_1(x)$, but $\hat{f}_{bag}(x)$ has smaller variance than $\hat{f}_1(x)$.

2.2.1 Out-of-Bag error estimation

Each bagged predictor is trained on about 63% of the data. The remaining 37% are called out-of-bag (OOB) observations.

The OOB prediction on x_i is

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

where $S_i = \{b|D^b$ does not contain the ith point $\}$.

The OOB error is a good estimate of the test error.

2.2.2 Bagging classification tree (example)

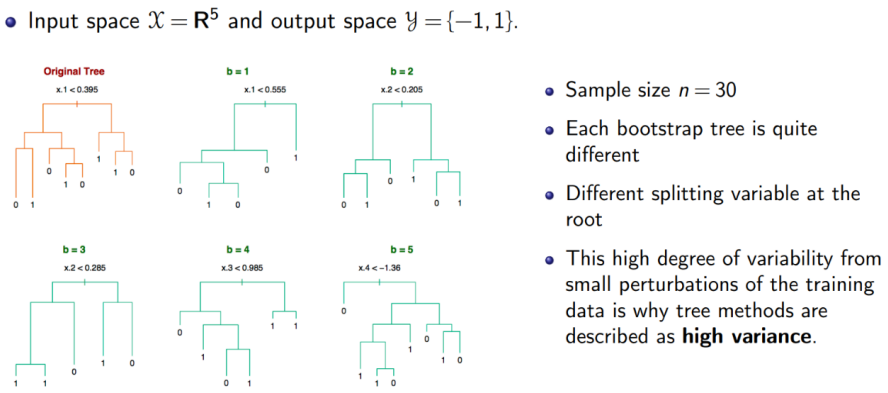

Figure 8: Example with bagging decision tree

2.2.3 Comparing classification combination methods

Two ways to combine classifications: consensus class or average probabilities.

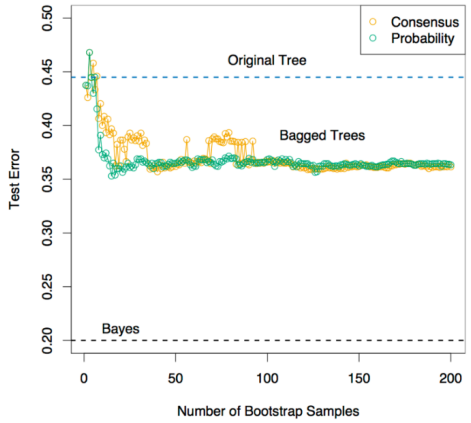

Figure 9: Effects of combination methods

2.3 Random Forest

The main idea of random forests is to use bagged decision trees, but modifying the tree-growing procedure to reduce the dependance between trees. The key step is to, when constructing each tree node, restrict the choice of splitting variable to a randomly chosen subset of features of size m. We typically choose $m \approx \sqrt{p}$ where p is the number of features, but it can also be chosen using cross validation. This m value has an effect on the classification error:

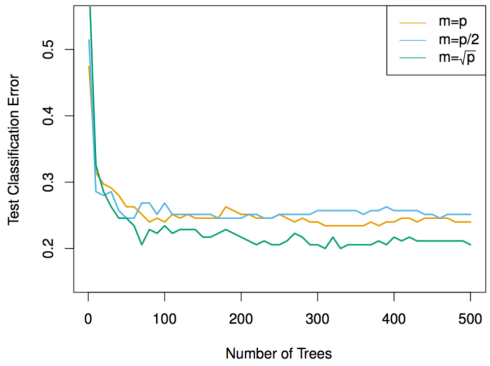

Figure 10: Effects of m value in random forest error

3 Boosting

In general, boosting is an ensemble learning technique where multiple weak learners (usually simple models) are combined to create a strong learner. Each weak learner is trained sequentially, and the focus is on correcting the errors made by the previous ones. Popular boosting algorithms include AdaBoost, Gradient Boosting, and XGBoost. It is useful for when gradient-based methods for adaptive basis function models don't apply.

3.1 Forward Stagewise Additive Modeling (FSAM)

FSAM is an iterative greedy optimization algorithm for fitting adaptive basis function models.

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

$$f_{m-1} = \sum_{i=0}^{m-1} \nu_i h_i$$

In the m'th round, we want to find step direction $h_m \in \mathcal{F}$ (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 as much as possible.

This is to say, in simple term, that in each step we add a new classifier that improves the objective function as much as possible.

3.1.1 Application of FSAM to regression: L^2 boosting

We apply FSAM to regression, using the mean squared error (MSE) as the loss function. For visualization, we will define the concept of regression stumps. A regression stump is a regression tree with a single split. It is a function of the form $h(x) = a1(x_i \leq c) + b1(x_i > c)$. Visually:

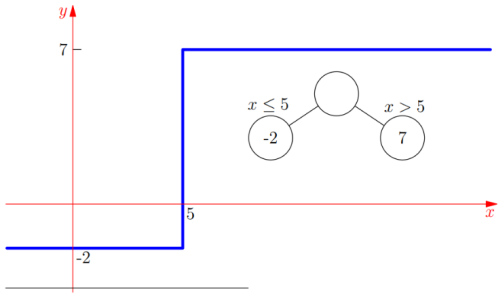

Figure 11: Visual representation of a regression stump

Using this concept, we can represent the working of FSAM in this case as follows:

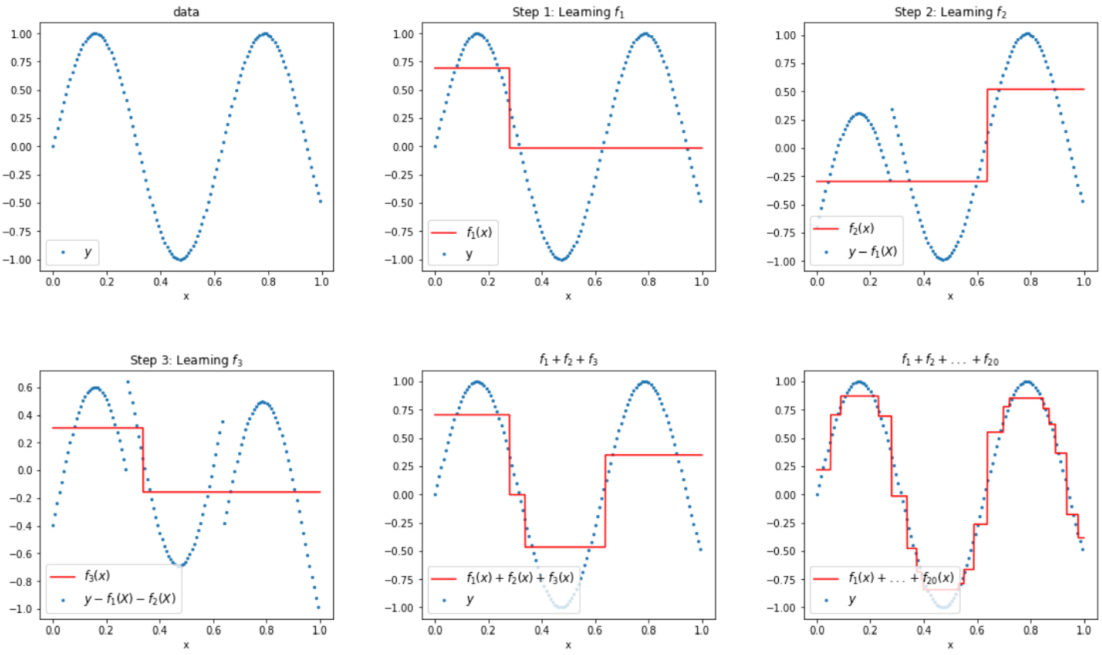

Figure 12: Visual representation of L2 boosting

3.1.2 Application of FSAM to classification: Adaboost algorithm

The Adaboost algorithm [2] introduces the idea of assigning weights to the training samples, and increasing it in the points that the model misclassifies, so that the next model may focus on them.

A rough sketch of the algorithm would be as follows:

- Training set $S = ((x_1, y_1), ..., (x_n, y_n))$.
- Start with equal weight on all training points $w_1 = ... = w_n = 1$.
- Repeat for $m = 1, ..., M$:

– Fing base classifier $h_m(x)$ that tries to fit weighted training data with 0/1 loss

$$\hat{R}^w(f) = \frac{1}{W}\sum_{i=1}^{N} w_i \ell(f(x_i), y_i), \text{ where } W = \sum_{i=1}^{n} w_i$$

– Increase weight w_i on the points $h_m(x)$ misclassifies.

- So far, we've generated M classifiers $h_1, ..., h_M : \mathcal{X} \to \{-1, 1\}$.
- Final scoring function is $f_M(x) = \sum_{m=1}^{M} \nu_m h_m(x)$ for some weight ν_m.

A general visualization of the algorithm would be:

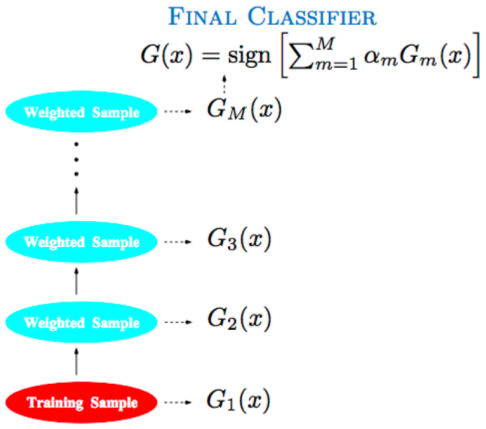

Figure 13: Visual representation of Adaboost algorithm

Visualizing the algorithm step by step in two examples:

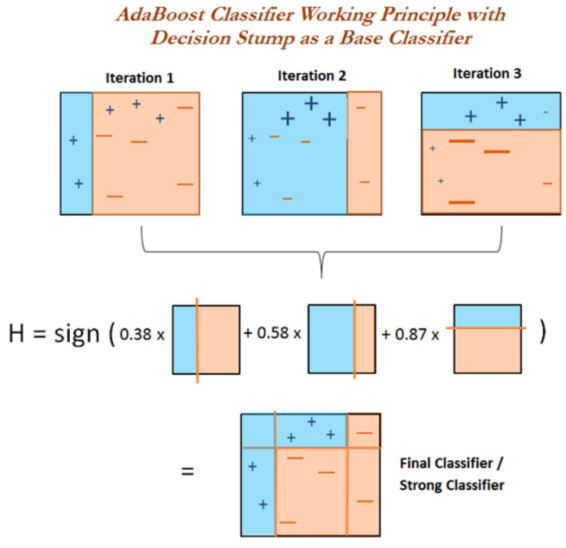

Figure 14: Step by step example of Adaboost algorithm

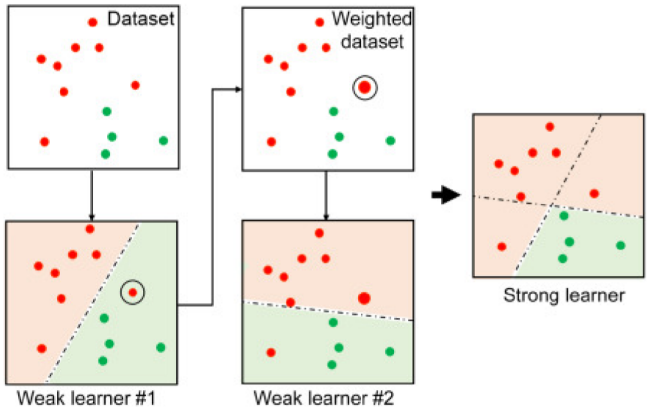

Figure 15: Step by step example of Adaboost algorithm

We can see Adaboost as the FSAM algorithm. Let it be $\ell(\hat{y}, y) = exp(-\hat{y}y)$, at each step of Adaboost, the algorithm calculates:

$$\begin{aligned}(\nu_m, h_m) &= argmin_{\nu\in\mathbf{R},h\in\mathbb{F}} \tfrac{1}{n}\textstyle\sum_{i=1}^{n}(f_{m-1}(x_i) + \nu h(x_i), y_i) \\ &= argmin_{\nu\in\mathbf{R},h\in\mathbb{F}} \tfrac{1}{n}\textstyle\sum_{i=1}^{n} exp(-y_i(f_{m-1}(x_i) + \nu h(x_i))) \\ &= argmin_{\nu\in\mathbf{R},h\in\mathbb{F}} \tfrac{1}{n}\textstyle\sum_{i=1}^{n} \alpha_i exp(-y_i \nu h(x_i)) \text{ with } \alpha_i = exp(-y_i f_{m-1}(x_i))\end{aligned}$$

3.1.3 Gradient Boosting / "Anyboost"

In gradient boosting [3] we perform a sort of "functional" gradient descent, in which in each step we add a function that makes the combined model move a step in the opposite direction of the gradient (gradient descent).

We want to minimize:

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

We define

$$\mathbf{f} = (f(x_i), ..., f(x_n))^T$$

which is to say, the vector of all predictions (size n). For notation simplification, we take $f_i = f(x_i)$.

Therefore, our objective function, and the one we will consider gradient descent on is:

$$J(\mathbf{f}) = \textstyle\sum_{i=1}^{n} \ell(y_i, \mathbf{f})$$

The negative gradient step direction at $\mathbf{f}$ is:

$$-\mathbf{g} = -\Delta_{\mathbf{f}} J(\mathbf{f}) = -(\delta_{\mathbf{f}_1} \ell(y_1, \mathbf{f}_1), ..., \delta_{\mathbf{f}_n} \ell(y_n, \mathbf{f}_n))$$

which we can easily calculate. This is the direction we want to change each of our n predictions on training data. This is also called the "pseudo-residuals".

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

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

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 out step direction.

Finally, we have to choose a step size. We can either do this with line search, or choosing a learning rate parameter (more common).

References

[1] L. Breiman et al. *Classification and Regression Trees*. Taylor & Francis, 1984. ISBN: 9780412048418. URL: `https://books.google.fr/books?id=JwQx-WOmSyQC`.

[2] Yoav Freund and Robert E. Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Proceedings of the Second European Conference on Computational Learning Theory (COLT '95)*. 1995.

[3] Jerome H Friedman. "Greedy function approximation: A gradient boosting machine". In: *Annals of statistics*. JSTOR, 2001, pp. 1189–1232.