Trees and Aggregating Methods

Ensemble Methods

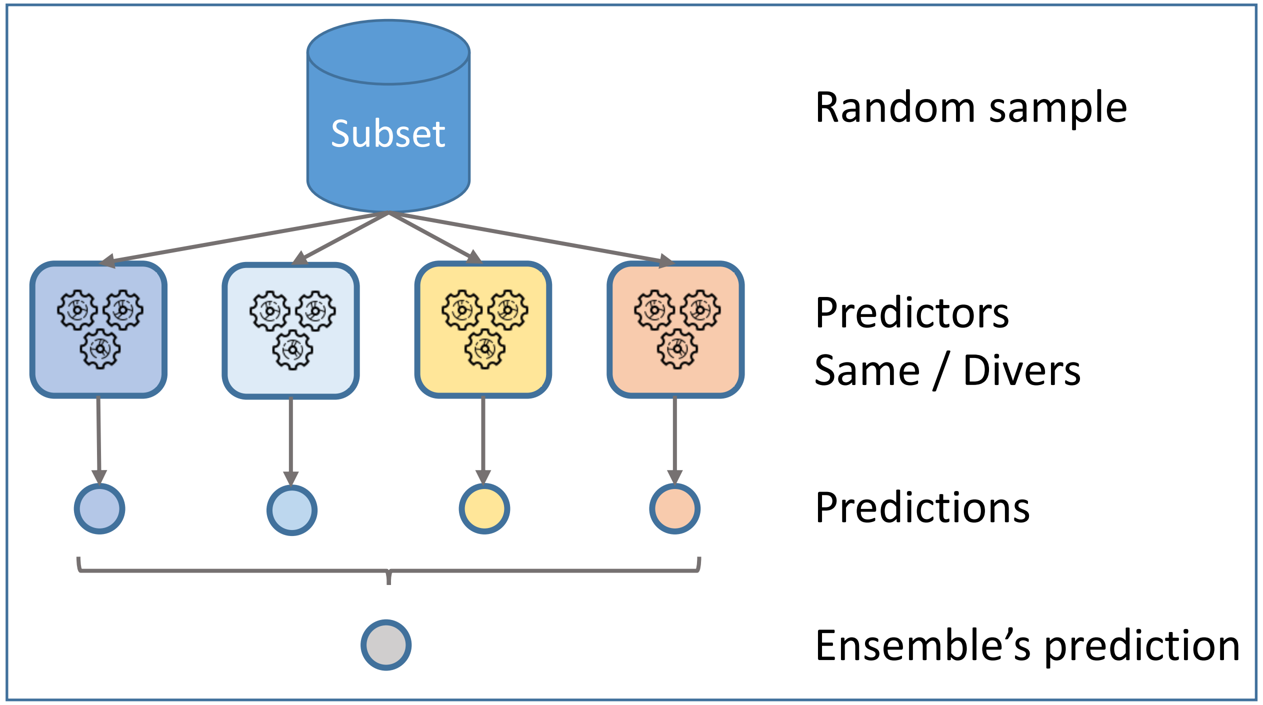
Pr Roch Giorgi

roch.giorgi@univ-amu.fr

SESSTIM, Faculty of Medical and Paramedical Sciences, Aix-Marseille University, Marseille, France
https://sesstim.univ-amu.fr/

Ensemble Methods (1)

- Machine learning technique combining several base models in order to obtain better predictive performance
  - Predictions from multiple models are combined



Simple ensemble technique

Ensemble Methods (2)

- Different independent estimates / decisions are combined into a final outcome
  - Models trained on random sample(s)
- Objective
  - To be more accurate than any single contribution
- Complexity of an ensemble learning model is higher than using a single model
  - More sophisticated technique for preparing the model, computational resources to train the model
  - Less reflection to understand why a specific prediction was made

Ensemble Methods (3)

- Potential advantages
  - (can give) Better robustness ↔ More stable predictions than a single model
    - Variance in the predictions when using a machine learning algorithm, even though it is trained on the same data or even slightly different data
  - (can give) Better predictions ↔ Importance of predictive performances
- Main families of ensemble learning algorithms
  - Bagging
    - Random forest
    - Bagged decision trees
    - …
  - Boosting
    - Gradient boost
    - AdaBoost
    - XGBoost
    - …
  - Staking
    - Voting
    - Weighted average
    - Super learner
    - …

Table of Contents

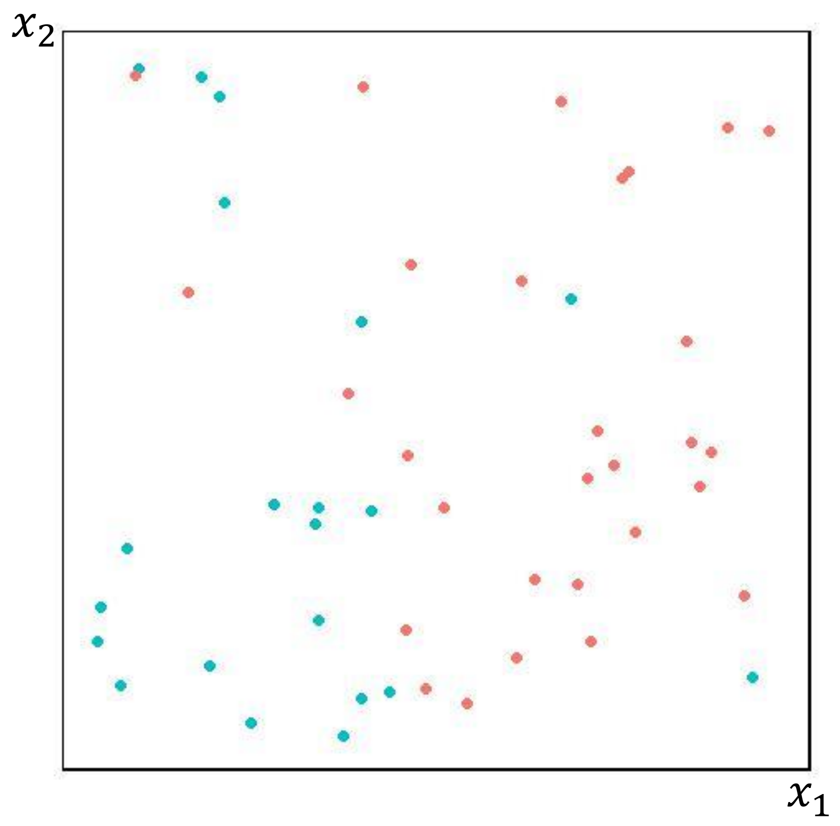
Decision Trees – Introduction

- Class of nonparametric predictive algorithms that work in regression (continuous outcome) and classification (categorical outcome)
- Can produce simple rules that are easy to interpret and visualize using tree diagrams
- General principle
  - Trees construction: partition the feature space into a number of smaller (non-overlapping) regions with similar response values using a set of splitting rules
  - Predictions: by fitting a simpler model in each region
- Several algorithms
  - CART: Classification And Regression Trees
  - CHAID (Chi-squared Automatic Interaction Detection), ID3 (Iterative Dichotomiser 3)

Decision Trees – Bases

- Y (continuous, categorical) feature to be explained by p (continuous, categorical) explanatories features $X_1, \dots, X_p$
- For the following
  - Y: binary feature (0, 1)
  - X_1, X_2: continuous features
  - n observations $(X_{1,1}, X_{2,1}, Y_1), \dots, (X_{1,n}, X_{2,n}, Y_n)$

  $\rightarrow$ Find a partition of the observation that best separates the red points ($y_i = 0$) from the blue points ($y_i = 1$), $i = 1, \dots, n$

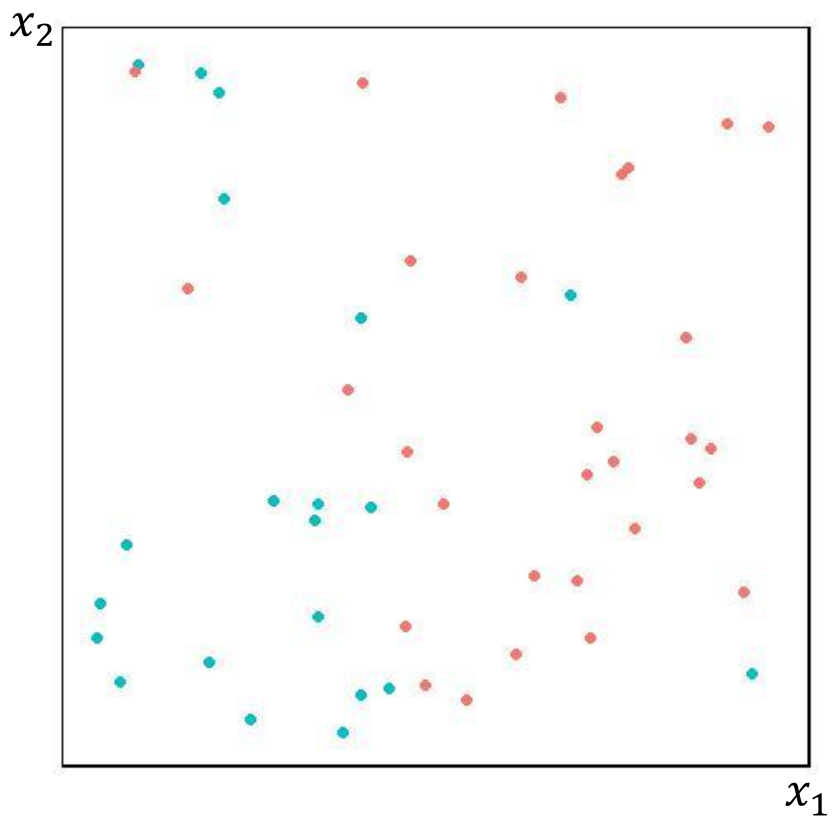


CART Binary Tree

- Partitions the training data into homogeneous subgroups
  - Groups with similar response values
- Then fits a simple *constant* in each subgroup
  - Regression problem: mean of the within group response values
  - Classification problem: proportion of the within group response values
- The subgroups (also called nodes) are formed recursively using binary partitions formed by asking simple yes-or-no questions about each feature (e.g., is $X < s$)
  - Partitions are created by successive divisions by means of hyperplanes orthogonal to the axes of $\mathbb{R}^p$ depending on the data (X_i, Y_i)
- Done until a suitable stopping criteria is satisfied

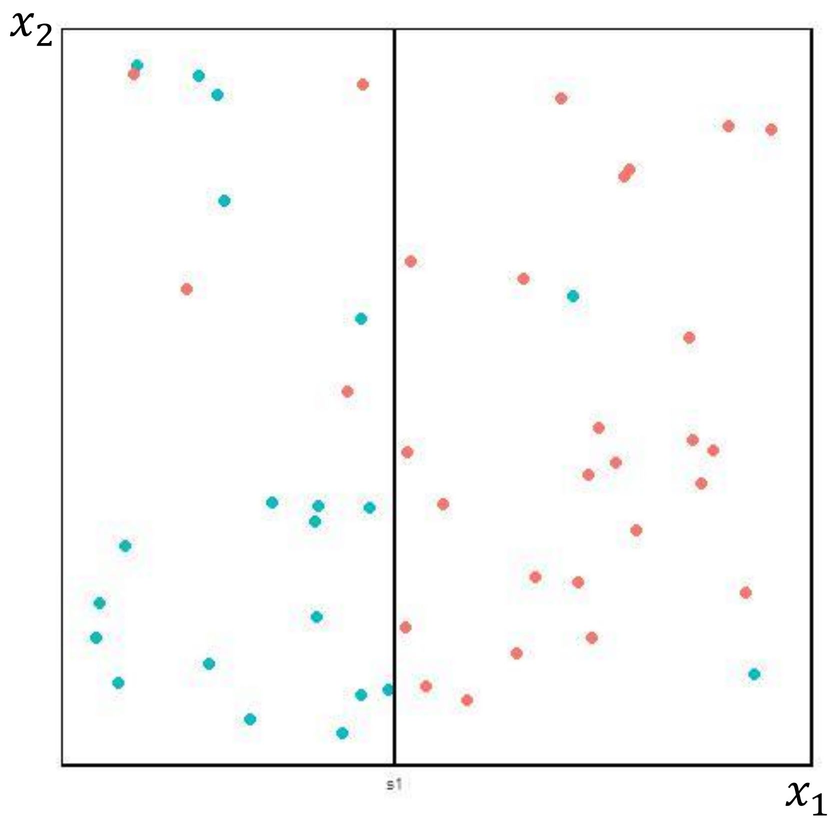
CART Binary Tree – Bases (1)

- At each step, CART method proposes a new partition
  - A feature and a cut-off threshold/rule



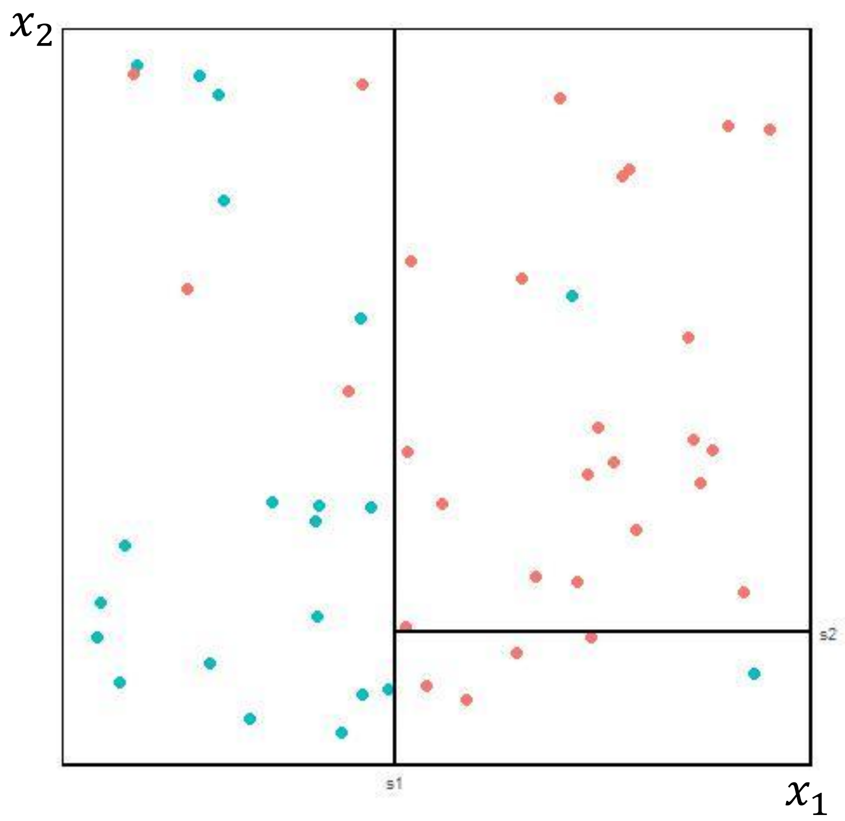
CART Binary Tree – Bases (2)

- At each step, CART method proposes a new partition
  - A feature and a cut-off threshold/rule



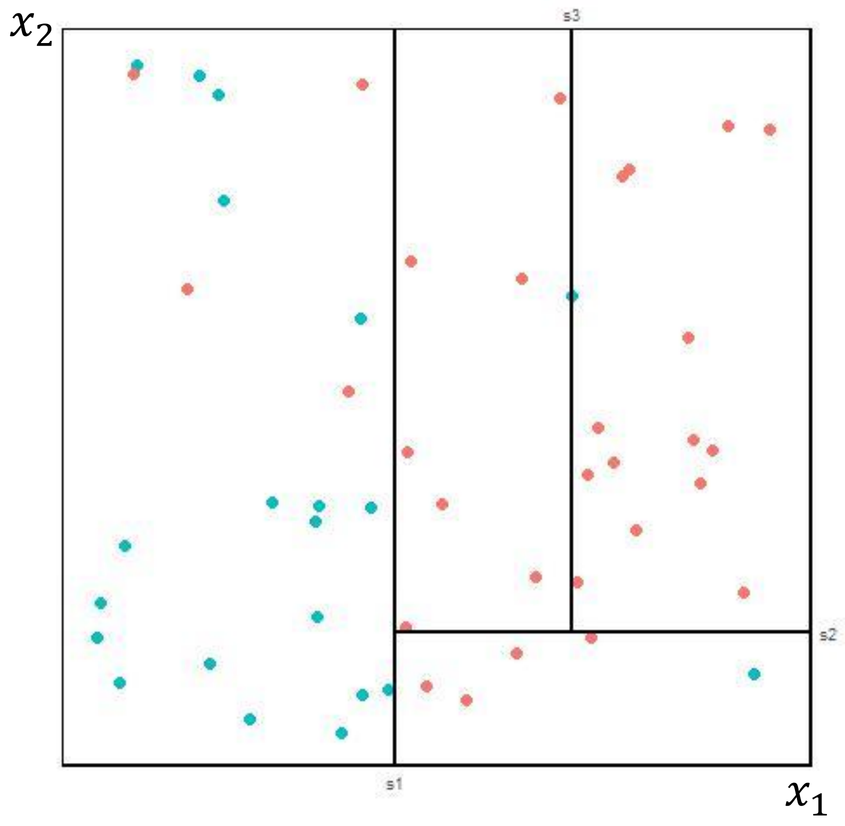
CART Binary Tree – Bases (3)

- At each step, CART method proposes a new partition
  - A feature and a cut-off threshold/rule



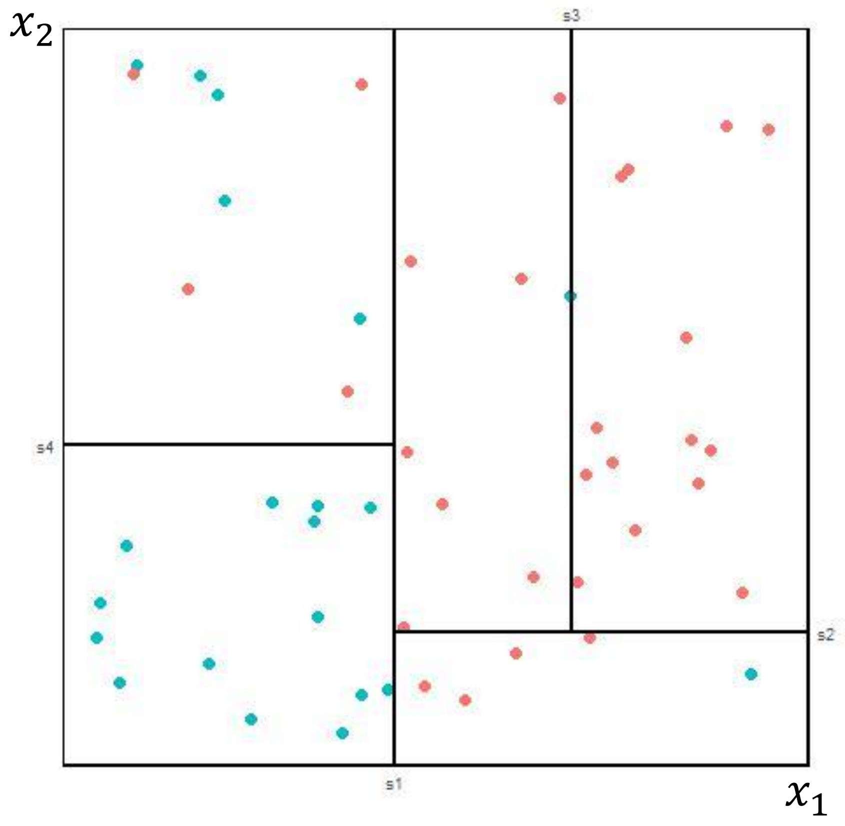
CART Binary Tree – Bases (4)

- At each step, CART method proposes a new partition
  - A feature and a cut-off threshold/rule



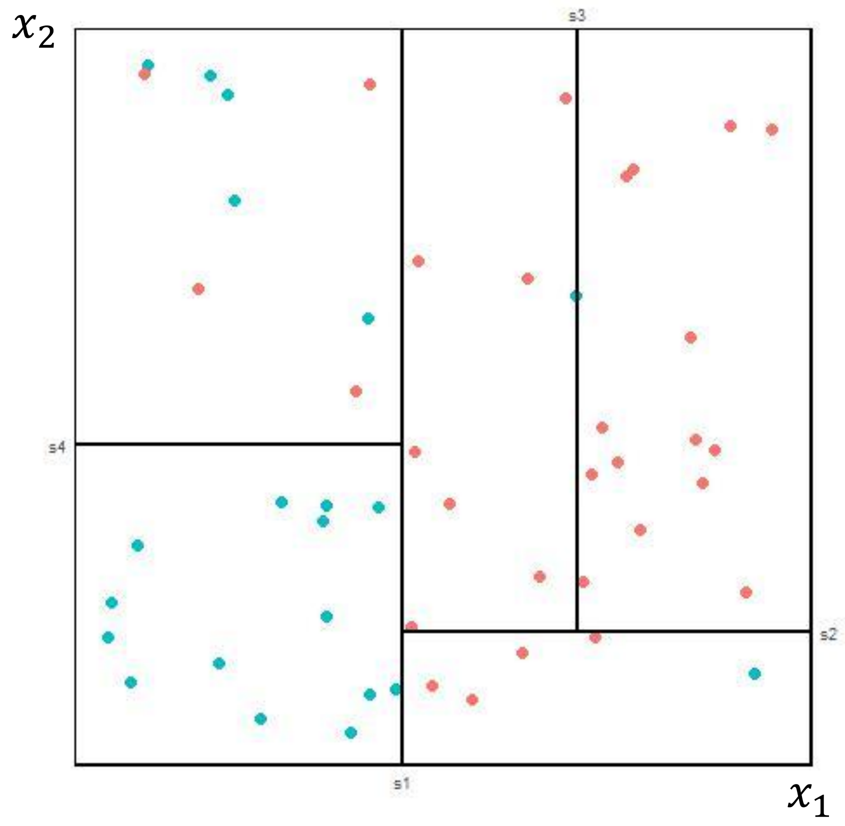
CART Binary Tree – Bases (5)

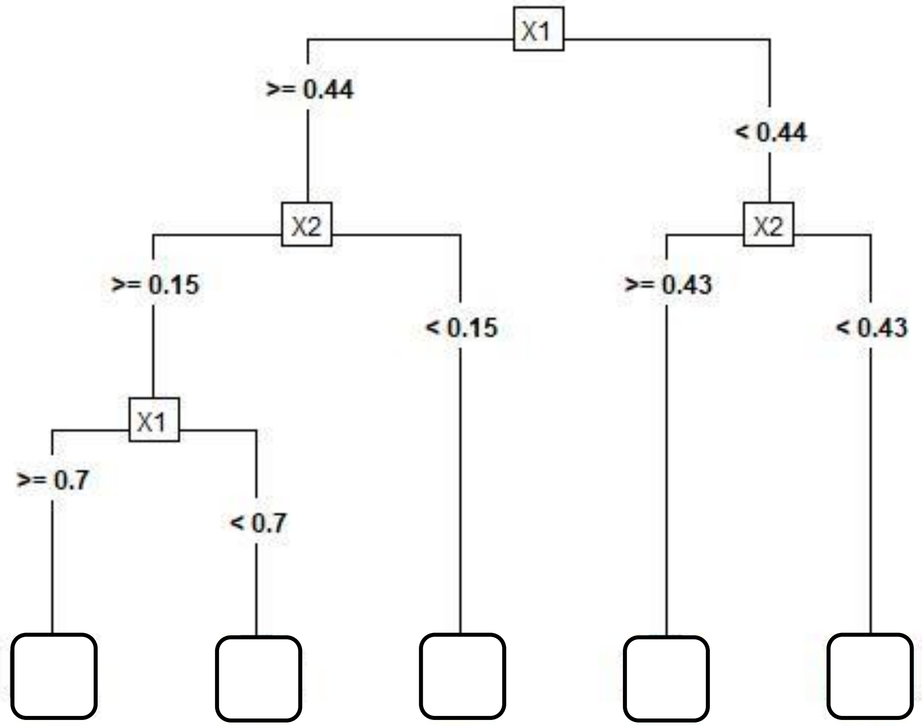
- At each step, CART method proposes a new partition
  - A feature and a cut-off threshold/rule



CART Binary Tree – Bases (6)

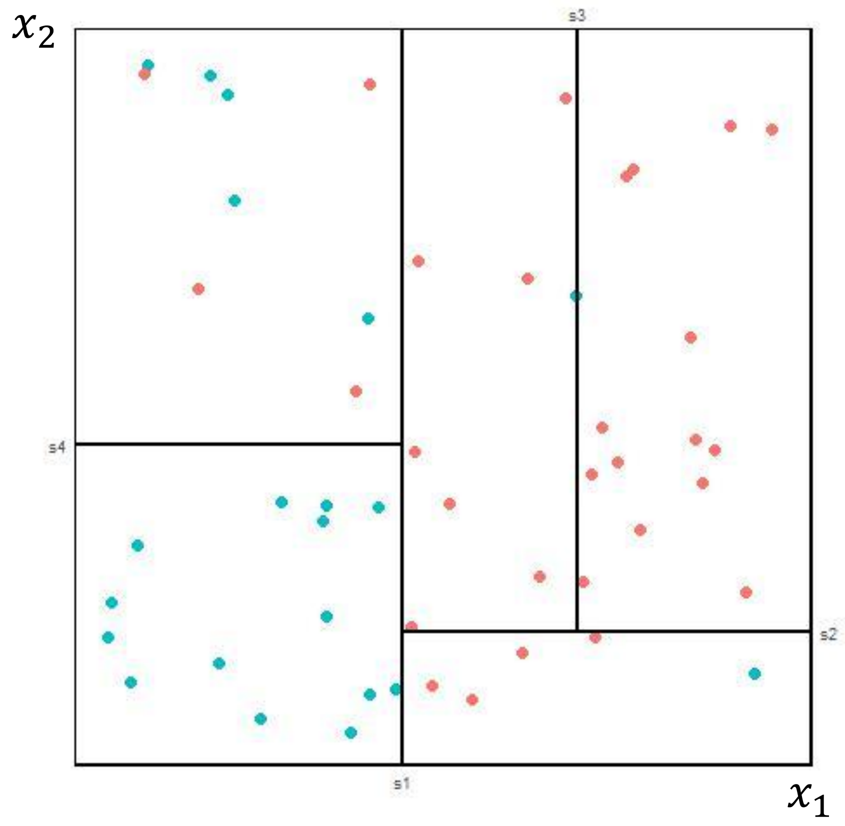
- At each step, CART method proposes a new partition
  - A feature and a cut-off threshold/rule

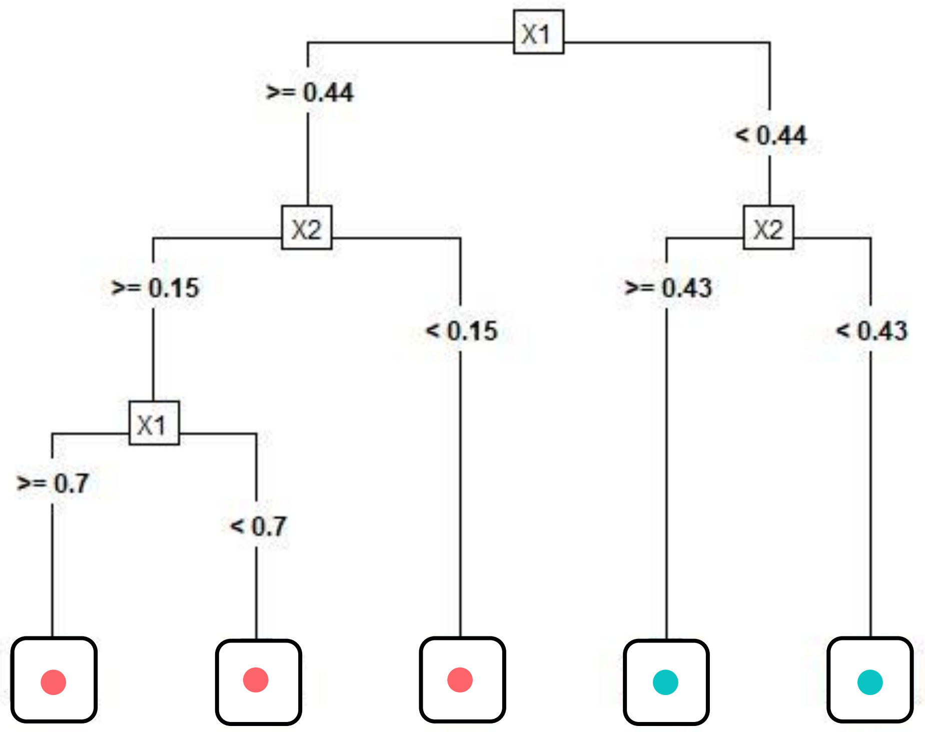




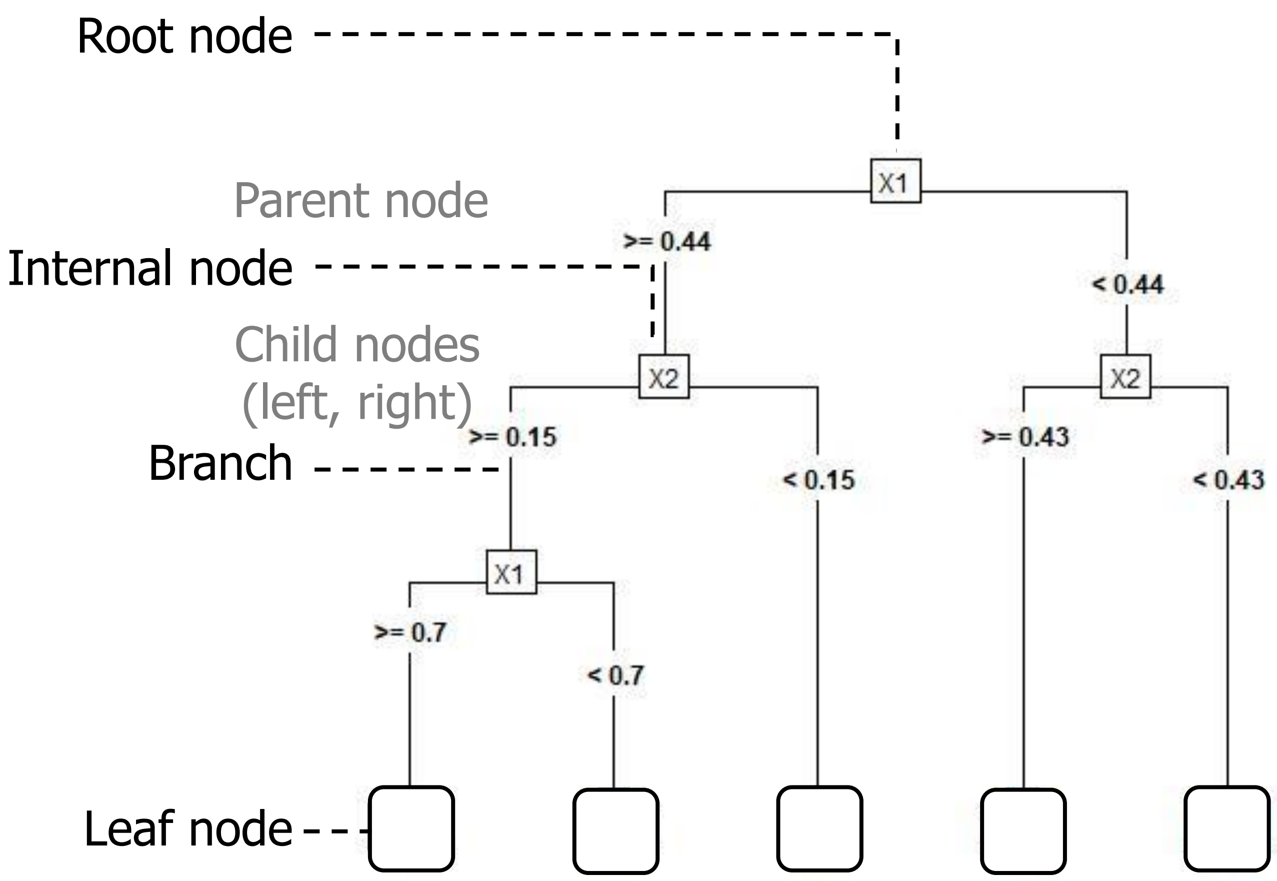
CART Binary Tree – Bases (7)

- Classification rule
  - A majority vote is taken in the terminal nodes of the tree





CART Binary Tree – Structure



CART Binary Tree – Partitioning (1)

- Partitioning the observations into 2 according to a cut-off threshold/rule parallel to the axes and then iterating this binary separation process on the two groups
  - Objective: obtain the best cut-off threshold/rule for a fixed data set $(X_1, Y_1), \dots, (X_n, Y_n)$
- At each step, choosing a feature j among the p explanatory features and a threshold s in $\mathbb{R}$ which split a node $\mathcal{N}$ into two child nodes

  $\mathcal{N}_1(j,s) = \{X \in \mathcal{N} | X_j \leq s\}$ and $\mathcal{N}_2(j,s) = \{X \in \mathcal{N} | X_j > s\}$
- Data driven

CART Binary Tree – Partitioning (2)

- Selection done by maximising an impurity function I
  - Measure the degree of heterogeneity of a node $\mathcal{N}$
  - Takes high values for heterogeneous nodes: Y values are dispersed within the node
  - Takes small values for homogeneous nodes: Y values are close within the node
- Once I is defined, we will choose the couple (j, s) that maximizes the impurity gain

  $$\Delta(\mathrm{I}) = P(\mathcal{N})\mathrm{I}(\mathcal{N}) - \left(P(\mathcal{N}_1)\mathrm{I}\big(\mathcal{N}_1(j,s)\big) + P(\mathcal{N}_2)\mathrm{I}\big(\mathcal{N}_2(j,s)\big)\right)$$

  where $P(\mathcal{N})$ is the proportion of observation in node $\mathcal{N}$

CART Binary Tree – Partitioning (3)

- Regression problem
  - Measure of impurity: variance node $\mathcal{N}$

$$\mathrm{I}(\mathcal{N}) = \frac{1}{|\mathcal{N}|} \sum_{i:X_i \in \mathcal{N}} (Y_i - \bar{Y}_{\mathcal{N}})^2$$

where $\bar{Y}_{\mathcal{N}}$ is the mean of Y_i in $\mathcal{N}$

  - Partitioning: at each step, choosing the couple (j, s) that minimizes

$$\sum_{X_i \in \mathcal{N}_1(j,s)} (Y_i - \bar{Y}_1)^2 + \sum_{X_i \in \mathcal{N}_2(j,s)} (Y_i - \bar{Y}_2)^2$$

where $\bar{Y}_k = \frac{1}{|\mathcal{N}_k(j,s)|} \sum_{i:X_i \in \mathcal{N}_k} Y_i$, $k = 1,2$

CART Binary Tree – Partitioning (4)

- Classification problem (Y: binary feature (0, 1))
  - Measure of impurity: Gini impurity (Gini index)

    $$\mathrm{I}(\mathcal{N}) = 2p(\mathcal{N})\big(1 - p(\mathcal{N})\big)$$

    where $p(\mathcal{N})$ is the proportion of 1 in $\mathcal{N}$
  - Partitioning: at each step, choosing the couple (j, s) that minimizes $\mathrm{I}_k(\mathcal{N}_k)$, $k = 1,2$
  - A node is "pure" if
    - It contains many 0 and few 1 (or the reverse)
    - The proportion of 1 is closed to 1 (or to 0)

CART Binary Tree – Depth of a Tree (1)

- How deep (i.e., complex) should we make the tree?
  - The objective being to produce homogeneous groups it would seem natural to choose the tree with the maximum number of pure leaves
  - But, if we grow an overly complex tree, we tend to overfit to our training data resulting in poor generalization performance to new individuals

CART Binary Tree – Depth of a Tree (1)

- How deep (i.e., complex) should we make the tree?
  - The objective being to produce homogeneous groups it would seem natural to choose the tree with the maximum number of pure leaves
  - But, if we grow an overly complex tree, we tend to overfit to our training data resulting in poor generalization performance to new individuals
- How to choose the right size tree?
  - There is a balance to be achieved in the depth and complexity of the tree to optimize predictive performance on future unseen data
    - Complexity of a tree determined by its size or depth, which determines the bias/variance trade-off
  - Two primary approaches: early stopping and pruning
  - Classical approach: grow a very large, complex tree and then prune it back to find an optimal subtree

CART Binary Tree – Early Stopping

- Tree depth
  - Limiting tree depth to a fixed number of levels
  - The shallower the tree (low number of levels) the less variance we have in our predictions
  - However, there is a risk of introducing too much bias as shallow trees do not capture the complex interactions and patterns in our data

CART Binary Tree – Early Stopping

- Tree depth
  - Limiting tree depth to a fixed number of levels
  - The shallower the tree (low number of levels) the less variance we have in our predictions
  - However, there is a risk of introducing too much bias as shallow trees do not capture the complex interactions and patterns in our data
- Number of observations in any terminal node
  - Restricting it to an a priori value #, no split after this depth
  - Implies that each leaf nodes must contain at least # observations for predictions
  - Value #
    - Small: high variance, poor generalizability. In the extreme, a single observation is captured in the leaf node and used as a prediction, resulting in an interpolation of the training data
    - Large: restrict further splits therefore reducing variance

CART Binary Tree – Pruning

- Grow a tree
- Prune it back to find an “optimal subtree”
  - Compare the performance of the two trees by estimating their probability of misclassification/score on a test sample
  - Use a cost/complexity parameter

CART Binary Tree – Depth of a Tree (2)

1. Grow a very large, complex tree $\mathcal{T}_{max}$
2. Select a sequence of nested trees: $\mathcal{T}_{max} = \mathcal{T}_0 \supset \mathcal{T}_1 \supset \cdots \supset \mathcal{T}_k$ by optimising a cost/complexity criterion that allows the trade-off between fit and complexity of the tree to be regulated
3. Select a tree $\mathcal{T}$ in this sub-sequence which optimises a performance criterion (minimizes the estimated risk)

CART Binary Tree – Depth of a Tree (3)

- In `rpart` R package, values of the function `print`
  - `CP`: complexity parameter of the tree (CP ↘ ⇒ complexity ↗)
  - `nsplit`: number of splits in the tree
  - `rel.error`: classification error from training data (adjustment error)
  - `xerror`: classification error from 10-fold cross-validation (prevision error)
  - `xstd`: standard deviation associated with the cross-validation error
- Choose the tree whose the classification error `xerror` is minimal

CART Binary Tree – Prediction for New Individuals

- The final tree $\mathcal{T}$ yields a partition $\mathbb{R}^p$ into $|\mathcal{T}|$ terminal nodes $\mathcal{N}_1, \dots, \mathcal{N}_{|\mathcal{T}|}$
- Used for prediction for new individuals from a new test data
- Classification rule
  - $\hat{g}(x) = \begin{cases} 1, \text{ if } \sum_{i:X_i \in \mathcal{N}(x)} 1_{Y_i=1} \geq \sum_{i:X_i \in \mathcal{N}(x)} 1_{Y_i=0} \\ 0, \text{ otherwise} \end{cases}$
- Score
  - $\hat{S}(x) = \hat{P}(Y = 1|X = x) = \frac{1}{n} \sum_{i:X_i \in \mathcal{N}(x)} 1_{Y_i=1}$
- In `rpart` R package, using the function `predict`

CART Binary Tree – Tree Performances

- For the final tree, to compare trees between them
- Indicators obtained on a new test data
  - Misclassification rates
  - ROC curves and AUC

CART Binary Tree – Feature Interpretation

- The plot of the final tree allows easy interpretation of the model
- Feature importance measure allows to measure the importance of feature
  - It is possible that features that do not appear in the tree construction are important in explaining the feature of interest (a feature could be used multiple times in a tree)
  - The total reduction in the loss function across all splits by a feature are summed up and used as the total feature importance
  - After standardization, the most important feature has a value of 100 and the remaining features are scored based on their relative reduction in the loss function

CART Binary Tree – Conclusion

- Quite simple method, relatively easy to use (no particular pre-processing requirements, such as monotonic transformation, dealing with outliers,…)
- For regression and classification problems
- Interpretable results (difficulty increase with the depth of the tree)
- Drawback
  - Known to be unstable, sensitive to slight perturbations of the sample
  - Do not often achieve state-of-the-art predictive accuracy

Bagging – Introduction

- **B**ootstrap **Agg**regat**ing**
  - One of the first ensemble algorithms machine learning practitioners learn
  - Designed to improve the stability and accuracy of regression and classification algorithms
  - Instead of constructing a single estimator, constructing, on bootstrap samples, a large number B, $g_1, \dots, g_B$, then aggregating them, $\hat{g} = \frac{1}{B}\sum_{k=1}^{B} g_k(x)$
  - Helps to reduce variance and minimize overfitting by model averaging
  - Usually applied to decision tree methods, but can be used with any type of method

Bagging – Context

- As in previous part, Y (continuous, categorical) feature to be explained by p (continuous, categorical) explanatories features $X_1, \dots, X_p$
- Regression or classification problem
- For further simplification, we will consider the regression problem
  - Notations
    - (X, Y) random pair of values in $\mathbb{R}^d \times \mathbb{R}$
    - $\mathcal{D}_n = (X_1, Y_1), \dots, (X_n, Y_n)$ a n –sample $i.i.d.$ with same probability distribution as (X, Y)

Bagging – Interest

- Regression model
  - $Y = m(X) + \varepsilon$
- One notes
  - $\hat{m}_B(x) = \frac{1}{B}\sum_{k=1}^{B} m_k(x)$
  - an estimator of m obtained by aggregating B estimators $m_1, \dots, m_B$
- Reminder
  - $\hat{m}_B(x) = \hat{m}_B\big(x; (X_1, Y_1), \dots, (X_n, Y_n)\big)$, and $m_k(x) = m_k\big(x; (X_1, Y_1), \dots, (X_n, Y_n)\big)$ are random variables
- Therefore
  - Interest in aggregation measured by comparing the performance of $\hat{m}_B(x)$ with that of $m_k(x)$, $k = 1, \dots, B$ (e.g. bias and variance of these estimators)

Aggregating – Bias and Variance (1)

Hypothesis: the random variables $m_1, \dots, m_B$ are $i.i.d.$

- Bias
  - $\mathrm{E}[\widehat{m}_B(x)] = \mathrm{E}[m_k(x)]$

  $\Rightarrow$ Aggregating does not change the bias

Aggregating – Bias and Variance (1)

Hypothesis: the random variables $m_1, \ldots, m_B$ are $i.i.d.$

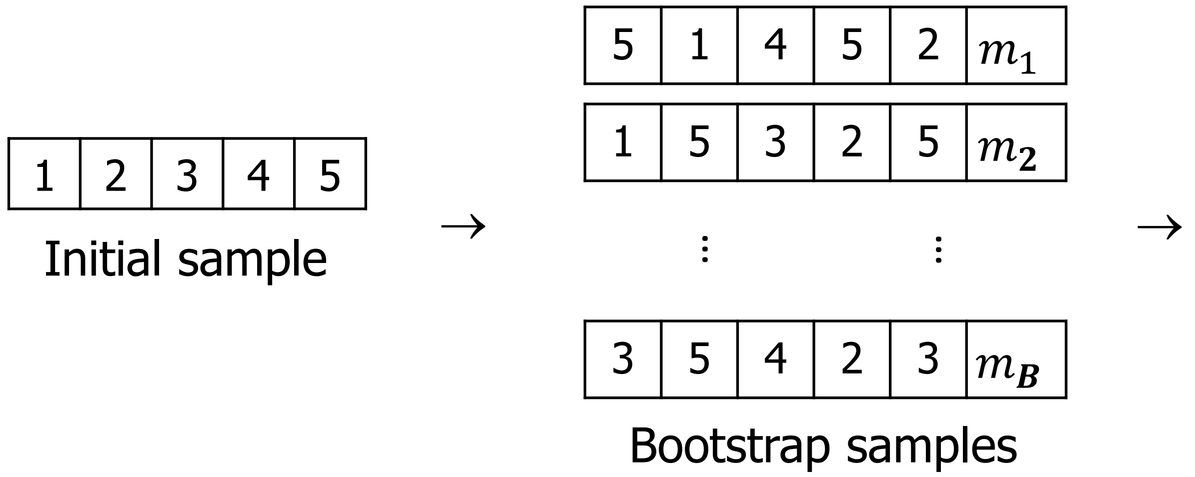
- Bias
  - $\mathrm{E}[\widehat{m}_B(x)] = \mathrm{E}[m_k(x)]$

  $\Rightarrow$ Aggregating does not change the bias
- Variance
  - $\mathrm{V}[\widehat{m}_B(x)] = \frac{1}{B}\mathrm{V}[m_k(x)]$

  $\Rightarrow$ Aggregating make the variance tend toward 0

Aggregating – Bias and Variance (2)

- Warning
  - The estimators $m_1, \dots, m_B$ being obtained on the same sample, the hypothesis independence is not suitable
- Reduction of the dependency between estimators m_k, $k = 1, \dots, B$ introducing new sources of randomness
  - Bootstrap samples



Aggregating $\hat{m}_B(x) = \frac{1}{B}\sum_{k=1}^{B} m_k(x)$

Bagging – Principle (1)

- m_k will not be built on the $\mathcal{D}_n = (X_1, Y_1), \dots, (X_n, Y_n)$ sample, but on bootstrap samples of $\mathcal{D}_n$
- Inputs
  - $x \in \mathbb{R}^d$ the observation to be predicted, $\mathcal{D}_n$ the sample
  - A regressor (CART, 1-nearest neighbours (and other small values),…)
  - B number of estimators that are aggregated
- For $k = 1, \dots, B$
  - Draw a bootstrap sample in $\mathcal{D}_n$
  - Fit the regressor on this bootstrap sample $m_k(x)$
- Output
  - Estimator $\widehat{m}_B(x) = \frac{1}{B} \sum_{k=1}^{B} m_k(x)$

Bagging – Drawing of Bootstrap Sample

- Bootstrap draw are
  - Represented by B random variables $\theta_k = 1, \dots, B$
  - Carried out according to the same law and independently
    - $\theta_1, \dots, \theta_B$ are $i.i.d.$ of the same law of θ
- 2 technics
  - Drawn of n observations with replacement
  - Drawn of $l < n$ observations without replacement
- Consequence
  - Aggregated estimators includes 2 sources of randomness (sample and bootstrap draw): $m_k(x) = m(x, \theta_k, \mathcal{D}_n)$

Bagging – Principle (2)

- Choice of the number of iterations

- Choice of the regressor

Bagging – Choice of the Number of Iterations

- According to the law of large numbers

$$\lim_{B\to\infty} \widehat{m}_B(x) = \lim_{B\to\infty} \frac{1}{B}\sum_{k=1}^{B} m_k(x) = \lim_{B\to\infty} \frac{1}{B}\sum_{k=1}^{B} m_k(x,\theta_k,\mathcal{D}_n)$$

$$= \mathrm{E}_\theta[m(x,\theta,\mathcal{D})] = \overline{m}(x,\mathcal{D}_n)\ \ p.s|\mathcal{D}_n$$

Bagging – Choice of the Number of Iterations

- According to the law of large numbers

$$\lim_{B\to\infty} \widehat{m}_B(x) = \lim_{B\to\infty} \frac{1}{B}\sum_{k=1}^{B} m_k(x) = \lim_{B\to\infty} \frac{1}{B}\sum_{k=1}^{B} m_k(x, \theta_k, \mathcal{D}_n)$$
$$= \mathrm{E}_\theta[m(x,\theta,\mathcal{D})] = \overline{m}(x, \mathcal{D}_n) \ \ p.s|\mathcal{D}_n$$

$\Rightarrow$ When B is large, $\widehat{m}_B$ stabilise towards the bagging estimator $\overline{m}(x, \mathcal{D}_n)$

$\Rightarrow$ The number of iterations B is not a parameter to be calibrated. Take it as large as possible according to the computation time

Bagging – Choice of the Regressor

- We have

$$\mathrm{E}[\widehat{m}_B(x)] = \mathrm{E}[m_k(x, \theta_k, \mathcal{D}_n)]$$

$$\mathrm{V}[\widehat{m}_B(x)] = \rho(x)\mathrm{V}[m(x, \theta_k, \mathcal{D}_n)] - \frac{1-\rho(x)}{B}\mathrm{V}[m(x, \theta_k, \mathcal{D}_n)]$$

where $\rho(x)=\mathrm{corr}\left(m(x, \theta_k, \mathcal{D}_n), m(x, \theta_{k'}, \mathcal{D}_n)\right)$ for$k \neq k'$

Bagging – Choice of the Regressor

- We have

$$\mathrm{E}[\widehat{m}_B(x)] = \mathrm{E}[m_k(x, \theta_k, \mathcal{D}_n)]$$

$$\mathrm{V}[\widehat{m}_B(x)] = \rho(x)\mathrm{V}[m(x, \theta_k, \mathcal{D}_n)] - \frac{1-\rho(x)}{B}\mathrm{V}[m(x, \theta_k, \mathcal{D}_n)]$$

where $\rho(x)=\mathrm{corr}\Big(m(x, \theta_k, \mathcal{D}_n), m(x, \theta_{k'}, \mathcal{D}_n)\Big)$ for$k \neq k'$

$\Rightarrow$ Bagging does not change the bias

$\Rightarrow$ B large $\rightarrow \mathrm{V}[\widehat{m}_B(x)] \approx \rho(x)\mathrm{V}[m(x, \theta_k, \mathcal{D}_n)] \rightarrow$ the variance decreases as the correlation between the predictors decreases

$\Rightarrow$ Need to aggregate estimators that are sensitive to small perturbations in the sample, e.g. trees

Bagging – Remarks (1)

- For algorithms that are stable or have high bias, bagging offers less improvement on predicted outputs since there is less variability (e.g., bagging a linear regression model will effectively just return the original predictions for large enough B)

Bagging – Remarks (1)

- For algorithms that are stable or have high bias, bagging offers less improvement on predicted outputs since there is less variability (e.g., bagging a linear regression model will effectively just return the original predictions for large enough B)
- A benefit to creating ensembles via bagging, based on resampling with replacement, is that it can provide its own internal estimate of predictive performance with the out-of-bag (OOB) sample (*see latter*)
  - The OOB sample can be used to test predictive performance
  - The results are generally comparable to those of the k-fold cross-validation, provided that the data set is large enough ($n > 1000$)

  $\Rightarrow$ As the data sets become larger and the bagging iterations increase, it is common to use the OOB error estimate as a proxy for predictive performance

Bagging – Remarks (2)

- Computation
  - Bagging can become computationally intense as the number of iterations increases
  - The process of bagging involves fitting models to each of the bootstrap samples which are completely independent of one another
  - As a solution, each model can be trained in parallel and the results aggregated in the end for the final model

Bagging – Remarks (2)

- Computation
  - Bagging can become computationally intense as the number of iterations increases
  - The process of bagging involves fitting models to each of the bootstrap samples which are completely independent of one another
  - As a solution, each model can be trained in parallel and the results aggregated in the end for the final model
- Feature interpretation
  - Models that are normally perceived as interpretable are no longer so
  - A solution, use of measure of feature importance

Bagging – Remarks (3)

- Bagging trees
  - Trees in bagging are not completely independent of each other since all the original features are considered at every split of every tree
  - Trees from different bootstrap samples typically have similar structure to each other (especially at the top of the tree) due to any underlying strong relationships
  - Known as "tree correlation" which prevents bagging from further reducing the variance of the base learner
  - A solution, random forests which extend and improve upon bagged decision trees by reducing this correlation and thereby improving the accuracy of the overall ensemble

Random Forests – Introduction

- A random forest is defined by a collection of de-correlated trees
- Aggregates trees built on bootstrap samples
- Reduce tree correlation by injecting more randomness into the tree-growing process
- Léo Breiman's algorithm has largely become the authoritative procedure
    - To reduce the correlation between the trees being aggregated, proposes to select the best "feature" from a set composed of only m features randomly chosen from the initial d features

Random Forests – Algorithm

- Inputs
  - $x \in \mathbb{R}^d$ the observation to be predicted, $\mathcal{D}_n$ the sample
  - B number of trees; n_{max} maximum number of observations per node
  - $m \in \{1, \dots, d\}$ the number of candidate features to split a node
- For $k = 1, \dots, B$
  - Draw a bootstrap sample in $\mathcal{D}_n$
  - Grow a CART regression/classification tree on the bootstrap sample, each split being selected by minimizing the CART cost function on a set of m features randomly chosen among the d. We denotes $\mathcal{T}(., \theta_k, \mathcal{D}_n)$ the resulting tree
- Output
  - Estimator $\hat{\mathcal{T}}_B(x) = \frac{1}{B}\sum_{k=1}^{B} \mathcal{T}_k(x)$

Random Forests – Remarks

- Aggregation step consist of
  - A majority vote (classification problem)
  - A mean (regression problem)
- 2 sources of randomness in θ_k
  - Bootstrapping
  - m features randomly selected at each step of the tree grow
- Estimator known to provide accurate estimates on complex data (many variables, missing data,...)
- Estimator not very sensitive to the choice of its parameters (B, n_{max}, m)
- Implemented on most statistical software (`RandomForest` function from the `randomForest` R package)

Random Forests – Choice of B

- Interest in bagging: to reduce the variance of the estimators being aggregated
- Bias is not improved by bagging $\Rightarrow$ aggregate estimators with a low bias (unlike boosting)
- Deep trees, with few observations in terminal nodes
  - By default in `RandomForest`: $n_{max} = 5$ in regression and in 1 classification
  - Tree is complete (but do not prune)

Random Forests – Choice of m

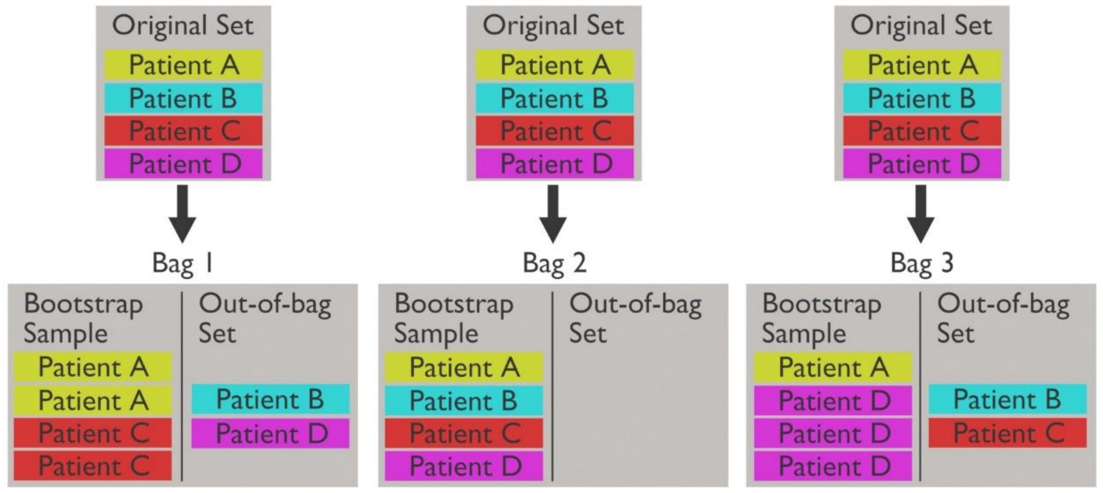
- Related to the correlation between the trees $\rho(x)$
- m has an influence on the bias/variance trade-off of the forest
  - $m \searrow$
    - The tendency is to move towards a "random" choice of tree-cutting features $\Rightarrow$ trees are more and more different $\Rightarrow \rho(x) \searrow \Rightarrow$ the variance of the forest decreases
  - Inversely when $m \nearrow$
- Recommendation
  - m should be considered a tuning parameter
  - Compare the performance of the forest for several values of m
- Comment
  - By default, $m = d/3$ in regression, and $\sqrt{d}$ in classification

Random Forests – Performance Measurement

- As with other classifiers and regressors, it is necessary to define criteria to measure the performance of random forests
- Examples
  - Prediction error $\mathrm{E}\left[\left(Y - \hat{T}_B(X)\right)^2\right]$ in regression
  - Probability error $P\left[\left(Y \neq \hat{T}_B(X)\right)\right]$ in classification
- As with other methods, these criteria can be assessed by learning/validation or cross-validation
- The bootstrap phase of the bagging algorithms makes it possible to define a method for estimating these criteria: Out Of Bag (OOB) method

Performance Measurement – Out Of Bag (1)

- OOB method consists in using the observations that are not in the bootstrap samples as a validation sample to estimate the performance of the forest
- Avoids the need for cross-validation to estimate the prediction performance



https://en.wikipedia.org/wiki/Out-of-bag_error

Performance Measurement – Out Of Bag (2)

- For each observation (X_i, Y_i) of $\mathcal{D}_n$, $\mathcal{O}_B$ is the set of trees in the forest that do not contain this observation in their bootstrap sample
- The prediction of Y at point X_i is done according to
  - $\hat{Y}_i(x) = \frac{1}{|\mathcal{O}_B|} \sum_{k \in \mathcal{O}_B} \mathcal{T}(X_i, \theta_k, \mathcal{D}_n)$
- OOB estimators
  - Prediction error is estimated by $\frac{1}{n} \sum_{i=1}^{n} \left(\hat{Y}_i - Y_i\right)^2$
  - Probability error is estimated by $\frac{1}{n} \sum_{i=1}^{n} 1_{\hat{Y}_i \neq Y_i}$

Performance Measurement – Out Of Bag (3)

- Example

5	1	4	5	2	m_1
4	5	3	2	5	m_2
3	5	3	4	3	m_3
4	3	3	1	5	m_4

  - Samples 2 and 3 do not contain the first observation, so $\hat{Y}_1 = \frac{1}{2}\left(m_2(X_1) + m_3(X_1)\right)$
  - The same applies to all observations $\Rightarrow \hat{Y}_2, \dots, \hat{Y}_4$
  - The error is estimated by $\frac{1}{4}\sum_{i=1}^{4}\left(\hat{Y}_i - Y_i\right)^2$

Random Forests – Importance Variable (1)

- Random forests often seen as a black box lacking interpretability compared to parametric models, such as the logistic model
- Importance variable
  - Allows to measure the importance of the variables in the model
  - Like the OOB error, it is based on the fact that not all observations are used to construct the forest trees

Random Forests – Importance Variable (2)

- Let OOB_k the OOB sample associated with the k^{th} tree, containing observations that are not in the k^{th} bootstrap sample

Random Forests – Importance Variable (2)

- Let OOB_k the OOB sample associated with the k^{th} tree, containing observations that are not in the k^{th} bootstrap sample
- Let E_{OOB_k} be the prediction error of tree k^{th} measured on this sample
  - $E_{OOB_k} = \frac{1}{|OOB_k|} \sum_{i \in OOB_k} (T(X_i, \theta_k, \mathcal{D}_n) - Y_i)^2$

Random Forests – Importance Variable (2)

- Let OOB_k the OOB sample associated with the k^{th} tree, containing observations that are not in the k^{th} bootstrap sample
- Let E_{OOB_k} be the prediction error of tree k^{th} measured on this sample
  - $E_{OOB_k} = \frac{1}{|OOB_k|} \sum_{i \in OOB_k} (T(X_i, \theta_k, \mathcal{D}_n) - Y_i)^2$
- Let OOB_k^j the sample OOB_k in which the values of feature j have been randomly perturbed and $E_{OOB_k}^j$ the prediction error of tree k measured on this sample
  - $E_{OOB_k}^j = \frac{1}{|OOB_k^j|} \sum_{i \in OOB_k^j} \left(T\left(X_i^j, \theta_k, \mathcal{D}_n\right) - Y_i\right)^2$

Random Forests – Importance Variable (2)

- Let OOB_k the OOB sample associated with the k^{th} tree, containing observations that are not in the k^{th} bootstrap sample
- Let E_{OOB_k} be the prediction error of tree k^{th} measured on this sample
  - $E_{OOB_k} = \frac{1}{|OOB_k|} \sum_{i \in OOB_k} (T(X_i, \theta_k, \mathcal{D}_n) - Y_i)^2$
- Let OOB_k^j the sample OOB_k in which the values of feature j have been randomly perturbed and $E_{OOB_k}^j$ the prediction error of tree k measured on this sample
  - $E_{OOB_k}^j = \frac{1}{|OOB_k^j|} \sum_{i \in OOB_k^j} \left(T\left(X_i^j, \theta_k, \mathcal{D}_n\right) - Y_i\right)^2$
- The importance of feature j is
  - $Imp(X_j) = \frac{1}{B} \sum_{k=1}^{B} \left(E_{OOB_k}^j - E_{OOB_k}\right)$ $\Rightarrow$ Features with the largest average decrease in accuracy are considered most important

Random Forests – Remarks

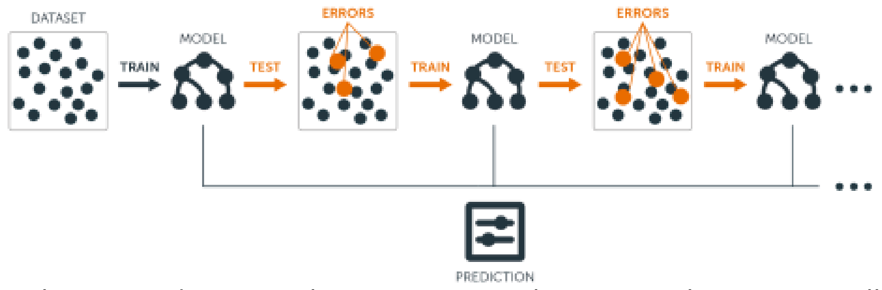
- They are a very powerful ready-made algorithm that often offers high predictive accuracy
- They have all the advantages of decision trees and bagging, and greatly reduce instability and correlation between trees
- Due to the added split feature selection attribute, they are also faster than bagging as they have a smaller feature search space at each tree split
- They will still suffer from slow computational speed as the data sets get larger, but modern implementations allow for parallelization to improve training time

Boosting – Introduction (1)

- Algorithms of gradient boosting provide answer to regression and classification problems
- Whereas random forests build an ensemble of deep independent trees, gradient boosting build an ensemble of shallow trees in sequence with each tree learning and improving on the previous one
- Although shallow trees by themselves are rather weak predictive models, they can be “boosted” to produce a powerful “committee”
- Bagging and random forests work by combining multiple models together into an overall ensemble

Boosting – Introduction (2)

- Main idea: to add new models to the ensemble sequentially
- Boosting attacks the bias-variance trade-off by starting with a weak model
  - e.g., a decision tree with only a few splits
- Then, sequentially boosts its performance by continuing to build new
  - e.g., new trees
- Where each new model in the sequence tries to fix up where the previous one made the biggest mistakes
  - i.e., in each new tree in the sequence, it will focus on the training rows where the previous tree had the largest prediction errors



Boehmke B, Greenwel BM. Hands-On Machine Learning with R. 2019. Chapman & Hall/CRC The R Series.

Boosting – Principle (1)

- (X, Y) random pair of values in $\mathbb{R}^d \times \mathcal{Y}$
- Given $\mathcal{G}$ a family of rules, we want to find the best rule in $\mathcal{G}$
  - Choose the rule that minimise a loss function, for example $\mathcal{R}(g) = \mathrm{E}\left[\ell\left(Y, g(x)\right)\right]$
- Problem
  - The loss function is not calculable
- Idea
  - Choose the rule that minimise the empirical version of the loss function

    $\mathcal{R}(g) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(Y_i, g(X_i)\right)$
  - where $\ell\left(y, g(x)\right)$ measure the error between the prediction $g(x)$ and the observation y

Boosting – Principle (2)

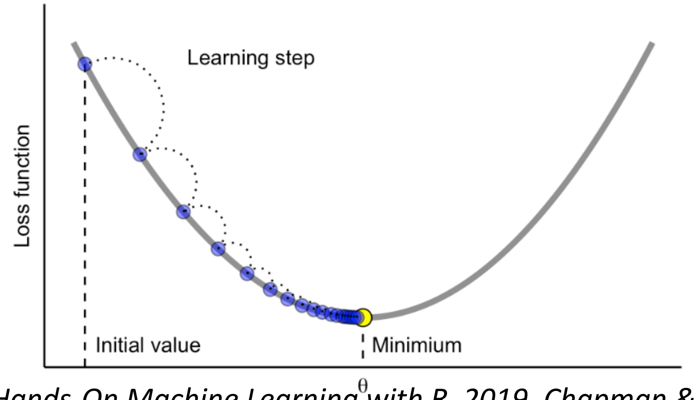
- Problem
  - No explicit solution $\Rightarrow$ need to find an algorithm to reach the solution

Boosting – Principle (2)

- Problem
  - No explicit solution $\Rightarrow$ need to find an algorithm to reach the solution
- Idea
  - To use a gradient descent algorithm (iterative algorithm)
    - Very generic optimization algorithm capable of finding optimal solutions to a wide range of problems
    - Adjusts the parameter(s) iteratively in order to minimize a loss function
    - Measures the local gradient of the loss function for a given set of parameters and takes steps in the direction of the descending gradient
    - The minimum is reach once the gradient is zero

Boosting – Principle (2)

- Problem
  - No explicit solution ⇒ need to find an algorithm to reach the solution
- Idea
  - To use a gradient descent algorithm (iterative algorithm)
    - Very generic optimization algorithm capable of finding optimal solutions to a wide range of problems
    - Adjusts the parameter(s) iteratively in order to minimize a loss function
    - Measures the local gradient of the loss function for a given set of parameters and takes steps in the direction of the descending gradient
    - The minimum is reach once the gradient is zero



Boehmke B, Greenwel BM. Hands-On Machine Learning with R. 2019. Chapman & Hall/CRC The R Series.

Boosting – Classical Algorithms

- adaboost
  - For binary classification
  - Loss function $\ell(y, g(x)) = exp(-yg(x))$ with $y \in \{-1,1\}$
- logiboost
  - For binary classification
  - Loss function $\ell(y, g(x)) = log\left(1 + exp(-yg(x))\right)$ with $y \in \{-1,1\}$
- L2-boosting
  - For regression
  - Loss function $\ell(y, g(x)) = (y - g(x)^2)$ with $y \in \mathbb{R}$

Boosting – Algorithms

- Returns a recursive sequence of estimators $(g_m)_m$ such as

$$g_m(x) = g_{m-1}(x) + \lambda g_m(x)$$

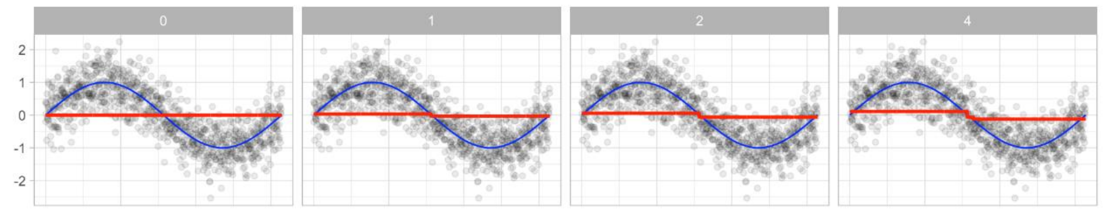
  - Where the learning rate, regularization parameter, $\lambda \in]0,1[$ and g_m is "weak" rule
  - Most often, these "weak" rules are trees with very few splits
- Choose an estimator (or estimation rule) in the sequence $(g_m)_m$
- Estimating performance of each g_m by cross-validation methods
- Iteratively from $m = 1$ to a choose value M
- Remark
  - Although boosting, like bagging, can be applied to any type of model, it is often most effectively applied to decision trees

Boosting – Boosted Trees

1. Fit a decision tree to the data: $\mathcal{T}_1(x) = y$
2. Fit the next decision tree to the residuals of the previous: $U_1(x) = y - \mathcal{T}_1(x)$
3. Add this new tree to the algorithm (updating): $\mathcal{T}_2(x) = \mathcal{T}_1(x) + U_1(x)$
4. Fit the next decision tree to the residuals of $\mathcal{T}_2$: $U_2(x) = y - \mathcal{T}_2(x)$
5. Add this new tree to the algorithm (updating): $\mathcal{T}_3(x) = \mathcal{T}_2(x) + U_2(x)$
6. Continue this process until cross-validation (or other) indicate to stop

Boosting – Boosted Trees

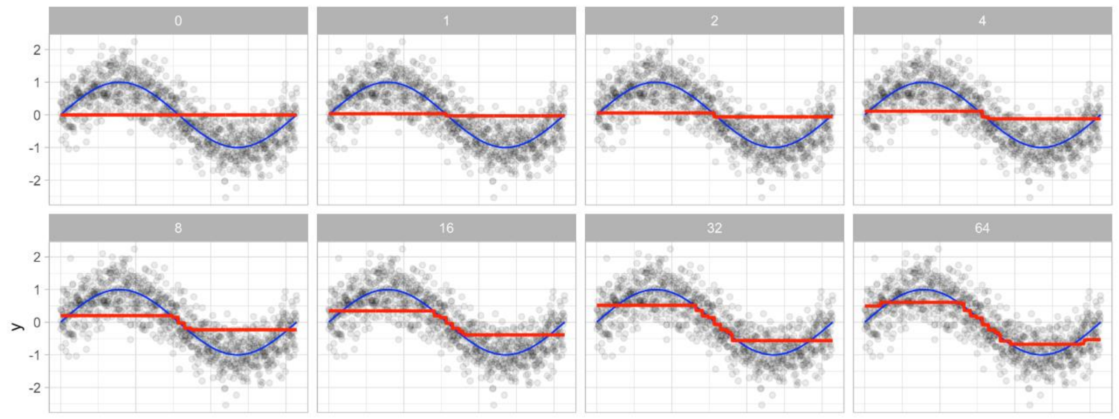
- Simple example
  - A true predictor (x) has a true underlying sine wave relationship (blue line) with y along with some irreducible error
  - The first tree fit in the series is a single decision stump
  - Each successive decision stump thereafter is fit to the previous one's residuals



Boehmke B, Greenwel BM. Hands-On Machine Learning with R. 2019. Chapman & Hall/CRC The R Series.

Boosting – Boosted Trees

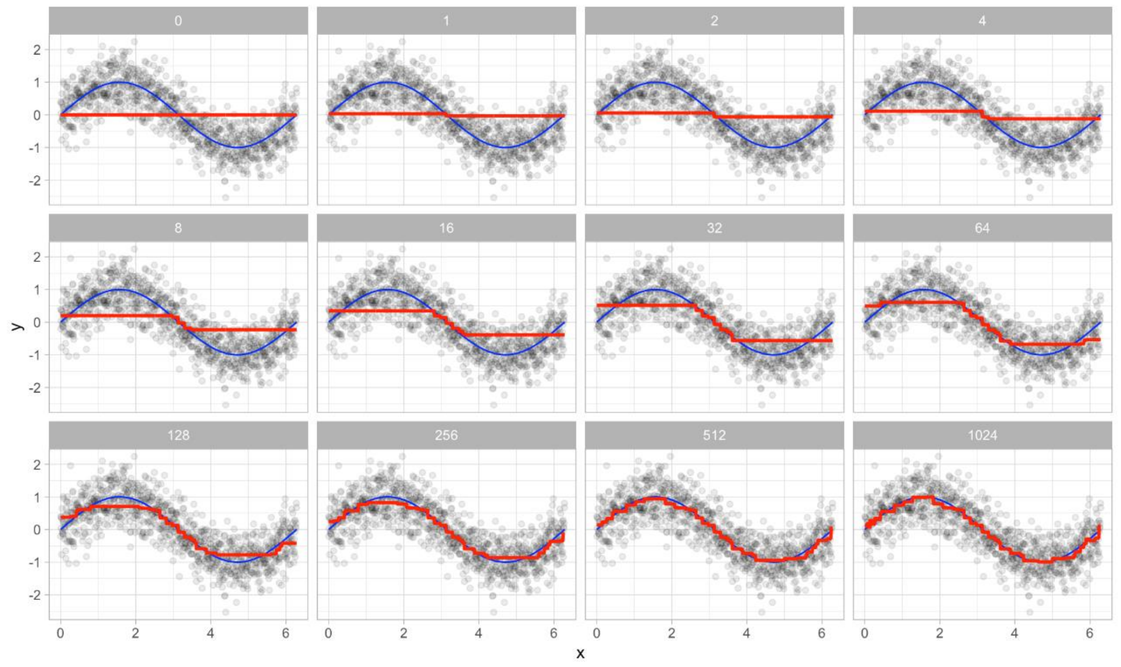
- Simple example
  - A true predictor (x) has a true underlying sine wave relationship (blue line) with y along with some irreducible error
  - The first tree fit in the series is a single decision stump
  - Each successive decision stump thereafter is fit to the previous one's residuals
  - Initially there are large errors, but each additional decision stump in the sequence makes a small improvement in different areas across the feature space where errors still remain



Boehmke B, Greenwel BM. Hands-On Machine Learning with R. 2019. Chapman & Hall/CRC The R Series.

Boosting – Boosted Trees

- Simple example
  - A true predictor (x) has a true underlying sine wave relationship (blue line) with y along with some irreducible error
  - The first tree fit in the series is a single decision stump
  - Each successive decision stump thereafter is fit to the previous one's residuals
  - Initially there are large errors, but each additional decision stump in the sequence makes a small improvement in different areas across the feature space where errors still remain
  - 0 to 1024 successive trees are added



Boehmke B, Greenwel BM. Hands-On Machine Learning with R. 2019. Chapman & Hall/CRC The R Series.

Boosting – Remark

- At each iteration
  - The bias decreases
  - The variance increases
  - ⇒ Importance to use weak rules (weak learner): large bias and small variance (shallow trees)
  - ⇒ The algorithm overfits if the number of iterations is (too) large

Boosting – Choice of m and λ

- Choice of m is crucial for boosting estimators
  - Overfitting if m is too large (estimators with low bias but large variance); conversely if m is small
- The regularization parameter λ represents the step size of the decrease of the gradient (learning rate)
  - Is linked to m: large value of λ will require few iterations; conversely if λ is small
- In practice
  - 2 or 3 (small) values are considered for λ (0.1, 0.01)
  - For each λ, the best m is chosen using technics such as cross-validation and the same loss function

Conclusion

- Random forests and boosting algorithms aggregate trees
- To be effective, trees must be weak learner, therefore poorly performing tress
  - Random forest: grow trees with large variance and small bias
  - Boosting: shallow trees with small variance and large bias
- Aggregating
  - Random forest: variance reduction
  - Boosting: bias reduction

Sources

- Laurent Rouvière: https://lrouviere.github.io/machine_learning/ and https://lrouviere.github.io/machine_learning/cours.pdf (access: November, 2022)
- Bradley Boehmke & Brandon Greenwell. Hands-On Machine Learning with R. *2019. Chapman & Hall/CRC The R Series.* *https://bradleyboehmke.github.io/HOML/* (access: November, 2022)
- François Husson. R pour la science des données. 2018. *Presses universitaires de Rennes*.