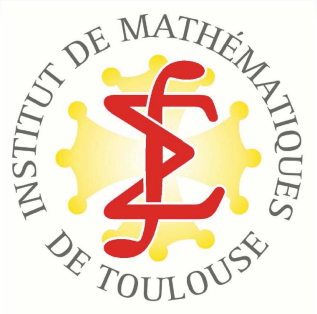


INSA Toulouse, Institut de Mathématiques de Toulouse

Classification and Regression Trees
Random Forests
Boosting

ML Training - CERFACS
October, 2022

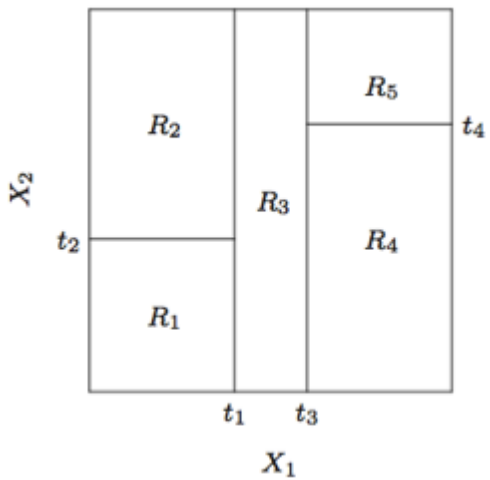
Philippe Besse - Béatrice Laurent - Olivier Roustant

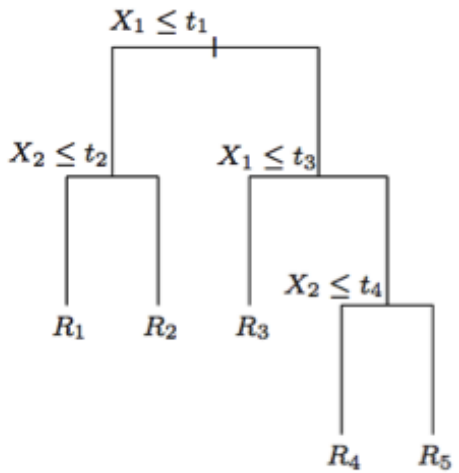
Outline

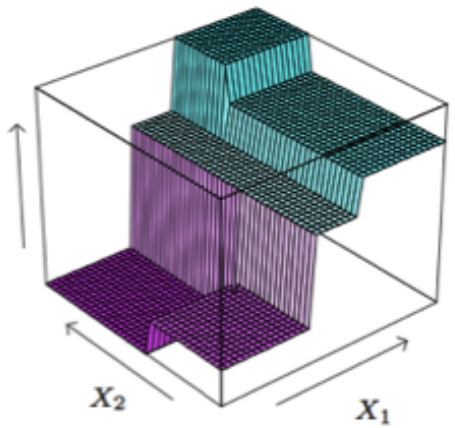
Introduction

- The recursive partitioning or segmentation methods were first introduced in the 1960's.
- The method studied in this course was presented in a paper by Breiman et al. in 1984 under the acronym of CART for Classification and Regression Trees. This method can be used either for regression or for classification.
- The CART algorithm is a non parametric method to build estimators in a multidimensional framework.
- The method, based on trees, relies on a partition of the space of input variables. We then infer a simple model (constant piecewise functions in regression and a single class in classification) on each element of the partition.
- The obtained solutions can be represented in a graphic with a tree that is very easy to interpret. The trees are based on a recursive sequence of division rules or splits, each of them based on a single explanatory variable.

Example of binary regression tree







Source: Hastie, Tibshirani, Friedman (2019), "The elements of statistical learning"

Introduction

- A first very simple and natural non parametric procedure in supervised regression or classification is the k-Nearest Neighbors (k-NN) method.
- Given a leaning sample $\{(\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_n, Y_n)\}$ in $\mathcal{X} \times \mathcal{Y}$, we want to predict the output Y associated to a new entry $\boldsymbol{x}$.
- For this, it seems natural to built the predictor from the observations in the training sample that are "close" to $\boldsymbol{x}$.
- We consider a distance d on $\mathcal{X}$. We fix an integer k and we retain the k nearest to $\boldsymbol{x}$ observations $\{\boldsymbol{X}_{(1)}, \ldots, \boldsymbol{X}_{(k)}\}$ and the associated outputs $(Y_{(1)}, \ldots, Y_{(k)})$.
- In a regression context, the prediction at point $\boldsymbol{x}$ is obtained from the mean of the observations $(Y_{(1)}, \ldots, Y_{(k)})$ while in classification we consider a majority vote.
- The choice of k is of course crucial. A too small value leads to overfitting (small bias but high variance) while a large value of k may lead to underfitting (small variance but probably high bias).

Introduction

- CART uses the same idea of local mean or majority vote, but the cell in $\mathcal{X}$ that is used to predict at point $\boldsymbol{x}$ is obtained from a more sophisticated way than simply considering the k-Nearest Neighbors of $\boldsymbol{x}$ in the learning sample.
- It will also take into account the values of the Y_i's.
- When partitioning ends, each terminal node of the complete tree becomes a leaf to which is assigned a value if Y is quantitative and a class if Y is qualitative.
- The last step consists in pruning the complete tree, which corresponds to a model selection procedure in order to reduce the complexity and avoid overfitting.

Outline

- Introduction
- Classification and Regression Trees
  - Construction of a maximal binary tree
  - Homogeneity criterion
  - Pruning the maximal tree
  - Practical remarks
- Random Forests
- Boosting
- Conclusion

Principles for constructing a tree

- Recursive binary split
  $\rightarrow$ Split a region in two, then split subregions in two, then ...
- Splits are defined by one variable
  $\rightarrow$ Very easy numerically: p optimizations in 1-dimension
- Clustering idea
  $\rightarrow$ Find a split that give the most homogeneous groups

- We observe p quantitative or qualitative explanatory variables X^j and a variable to predict Y which is either qualitative with m modalities $\{\mathcal{T}_\ell; \ell = 1 \ldots, m\}$ or real quantitative, on a sample of n individuals.
- The construction of a binary discrimination tree consists in determining a sequence of nodes.
  - A node is defined by the choice of a variable among the p explanatory variables and of a division which induces a partition into two classes. Implicitly, to each node corresponds a subset of the initial sample to which a dichotomy is applied.
  - A division is defined by a threshold value if the selected variable is quantitative or a split into two groups of modalities if the variable is qualitative.
  - At the root, the initial node corresponds to the whole sample; the procedure is then iterated over each of the subsets.

The algorithm requires:

1. the definition of a criterion allowing to select the best division among all admissible ones for the different variables
2. a rule allowing to decide that a node is terminal: it thus becomes a leaf
3. the predicted value associated to a leaf.

Division criterion

- A division is said to be admissible if the two corresponding son nodes are not empty.
- If the explanatory variable is a quantitative variable with m possible values (or qualitative but ordinal with m modalities), it provides $(m-1)$ possible binary divisions.
- If it qualitative but not ordinal, the number of divisions becomes $2^{(m-1)}-1$.
- The division criterion is based on the definition of an heterogeneity function.
- The objective is to divide the observations which compose a node into two more homogeneous groups with respect to the variable to explain Y.

Division criterion

Optimal division

- Dividing the node κ creates two son nodes: κ_L (left node) and κ_R (right node).
- Given an heterogeneity function D_κ of the node κ, the algorithm retains the division which minimizes

$$D_{\kappa_L} + D_{\kappa_R}.$$

- For each node κ in the construction of the tree:

$$\max_{\{\text{Divisions of } X^j; j=1,p\}} D_\kappa - (D_{\kappa_L} + D_{\kappa_R})$$

- Graphically, the length of each branch can be represented proportionally to the reduction in heterogeneity induced by the division.

Stopping rule and affectation

- The growth of the tree stops at a given node, which therefore becomes a terminal node also called a *leaf*,
  - when it is homogeneous (all the individuals have the same value for Y) or
  - when there is no longer an admissible partition or
  - to avoid unnecessarily fine splittings, when the number of observations it contains is less than some prescribed value (generally chosen between 1 and 5).
- When Y is quantitative (regression trees), the predicted value associated to a leaf is the mean of the values of the Y_i's among the observations belonging to this terminal node.

Outline

Constructing regression trees

For a given region (node) κ with size $|\kappa|$, define the heterogeneity by:

$$D_\kappa = \sum_{i \in \kappa} (y_i - \overline{y}_\kappa)^2 = |\kappa| \frac{1}{|\kappa|} \sum_{i \in \kappa} (y_i - \overline{y}_\kappa)^2$$

Splitting procedure

For a variable x_j, and a split candidate t, define left and right subregions

$$\kappa_L(t,j) = \{x_j \leq t\}, \qquad \kappa_R(t,j) = \{x_j > t\}.$$

Find (j,t) in order to minimize the intra-class variance

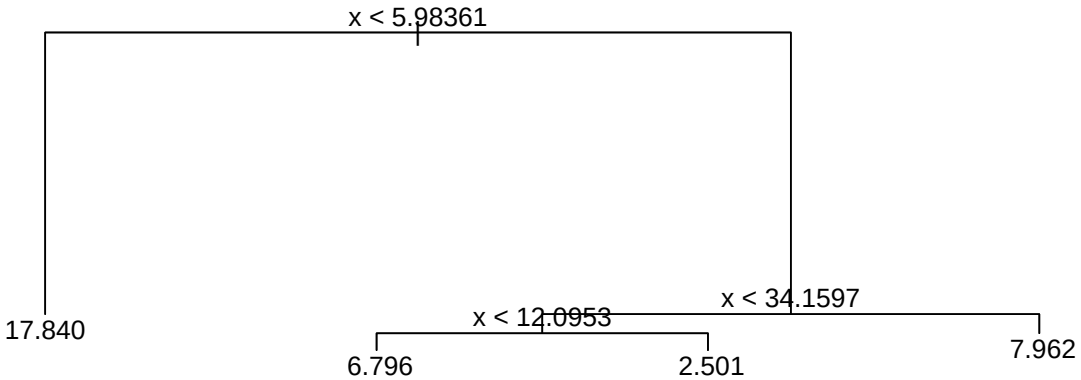
$$J(j,t) = D_{\kappa_L(t,j)} + D_{\kappa_R(t,j)},$$

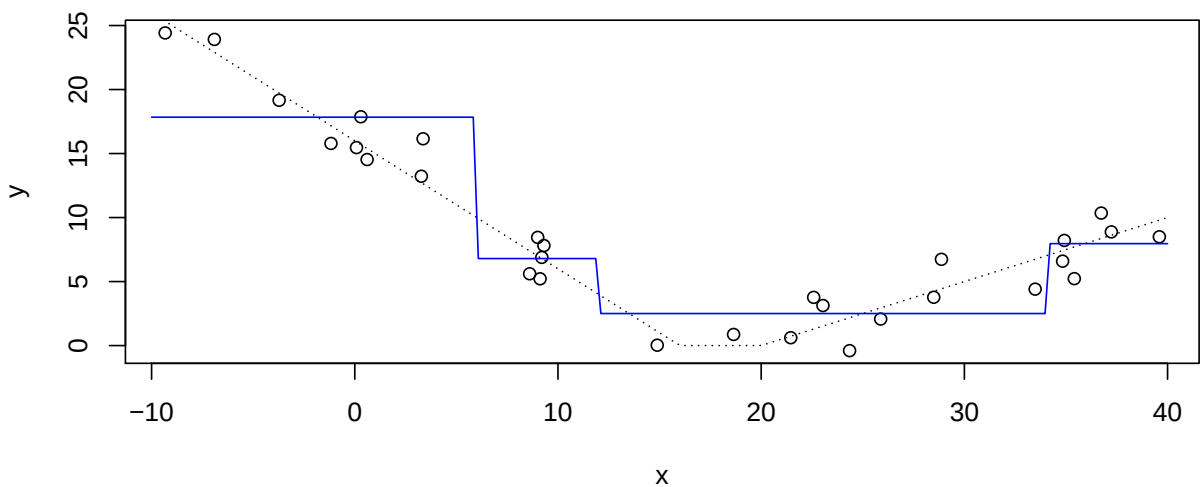
or equiv. to maximize the decrease in heterogeneity (inter-class variance)

$$D_\kappa - J(j,t)$$

NB: To remove ties, the best cut t is taken at the middle of two consecutive data points.

Illustration in 1 dimension





Constructing classification trees

This is the same procedure, with specific notions of heterogeneity

Heterogeneity measures in classification

p_κ^ℓ: proportion of the class $\mathcal{T}_\ell$ of Y in the node κ.

- The Cross-Entropy or deviance is defined by

$$E_\kappa = -\sum_{\ell=1}^{m} p_\kappa^\ell \log(p_\kappa^\ell) \qquad \Rightarrow \qquad D_\kappa = -|\kappa| \sum_{\ell=1}^{m} p_\kappa^\ell \log(p_\kappa^\ell)$$

  Maximal in $(\frac{1}{m}, \ldots, \frac{1}{m})$, minimal in $(1, 0, \ldots, 0), \ldots, (0, \ldots, 0, 1)$
  (by continuity, we assume that $0 \log(0) = 0$)

- Gini concentration: $D_\kappa = |\kappa| \sum_{\ell=1}^{m} p_\kappa^\ell (1 - p_\kappa^\ell)$

Illustration with two classes ($m = 2$)

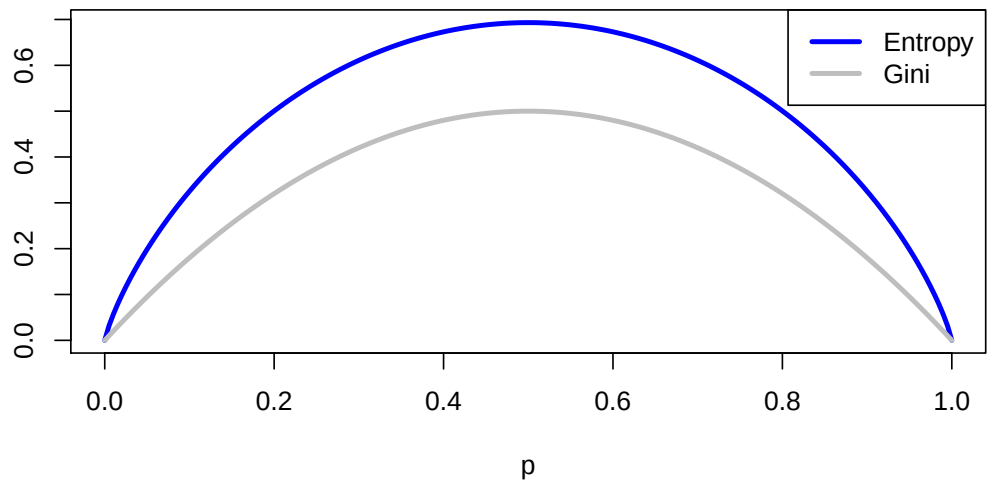


Figure: Heterogeneity criterions for classification. Both are minimal for $p = 0$ or $p = 1$, and maximal for $p = 1/2$.

Outline

- Introduction
- Classification and Regression Trees
  - Construction of a maximal binary tree
  - Homogeneity criterion
  - Pruning the maximal tree
  - Practical remarks
- Random Forests
- Boosting
- Conclusion

Stopping rule, pruning, optimal tree

- We need a tradeoff between maximal tree (overfits) and the constant tree (too rough)
- There exists a nice theory to find an optimal tree, minimizing prediction error penalized by complexity (number of leaves)
- When aggregating trees (random forest), simpler procedures are often preferred (see why after), e.g. fixing the number of leaves.

Pruning and optimal tree

Pruning: notations

- We look for a parcimonious tree
- *Complexity* of a tree: $|A| =$ numbers of leaves in A
- Adjustment error of A:

$$D(A) = \sum_{\kappa=1}^{|A|} D_\kappa$$

D_κ: heterogeneity of leaf κ

Sequence of embedded trees

- Adjustment error penalized by the complexity:

$$Crit_\gamma(A) = D(A) + \gamma \times |A|$$

- When $\gamma = 0$: $A_{\max}$ (maximal tree) minimizes $Crit_\gamma(A)$
- When γ increases, the division for which the improvement of D is smaller than γ, is cancelled; hence
  - two leaves are gathered (prunned)
  - there father node becomes a terminal node
  - A_J becomes A_{J+1}.
- After iteration of this process, we get a sequence of trees:

$$A_{\max} \supset A_1 \supset A_2 \supset \cdots A_K$$

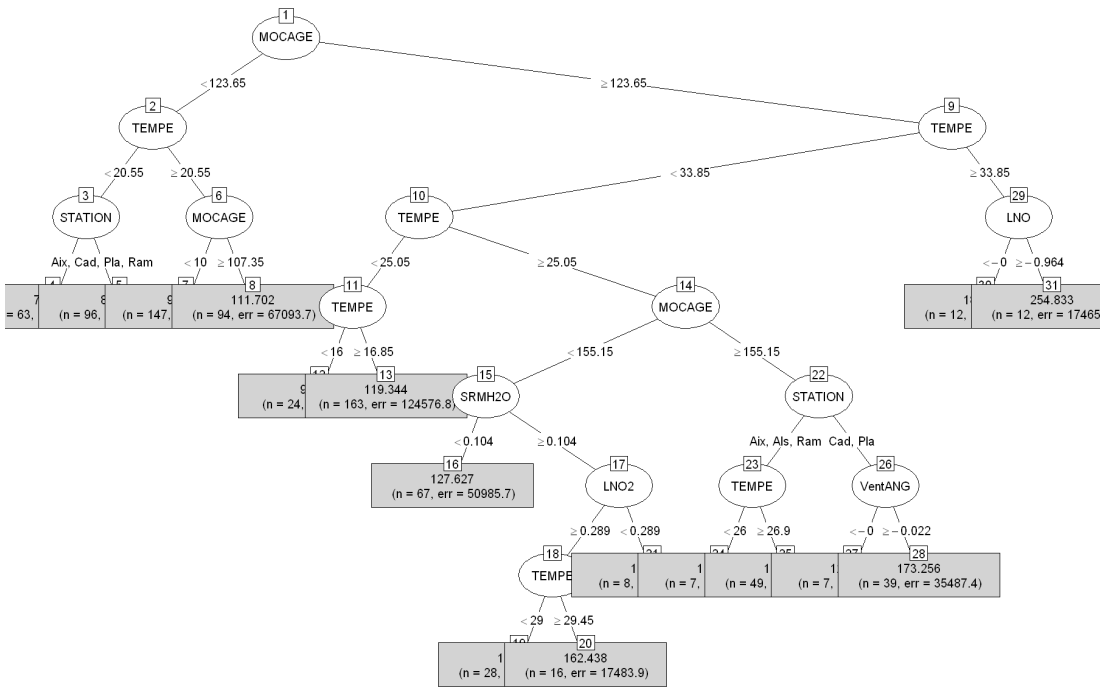
Optimal tree

Algorithm to select the optimal tree

- Maximal tree $A_{\max}$
- Construction of Breiman's sequence $A_1 \dots A_K$ of nested trees associated with the sequence of penalization parameters $(\gamma_1, \dots, \gamma_K)$
- V-fold cross validation error:
  for $v = 1, \dots, V$ **do**
  - For each sample (composed of $V - 1$ folds), estimation of the sequence of trees associated with the sequence of penalization parameters $(\gamma_1, \dots, \gamma_K)$
  - Estimation of the error on the validation fold.

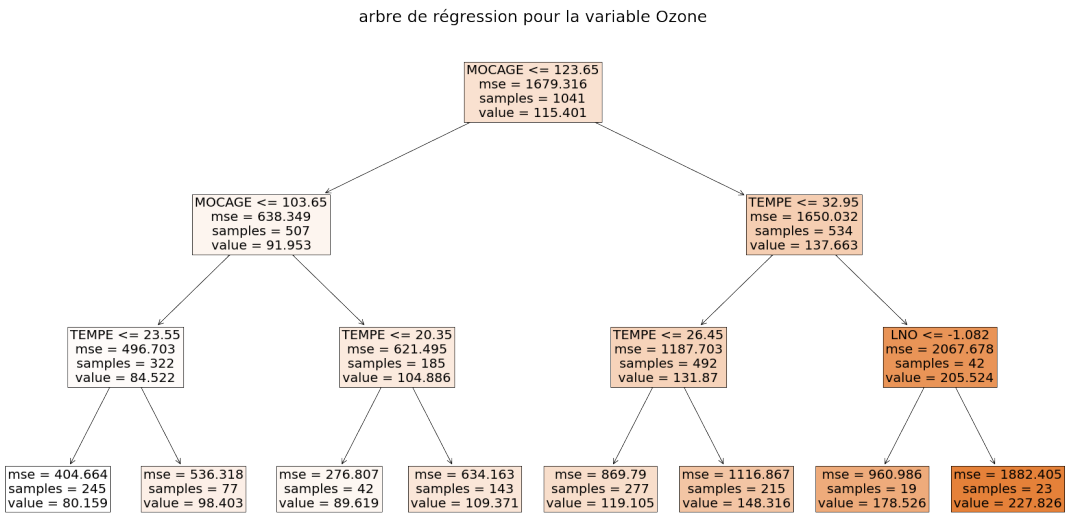
  EndFor
- For each value $(\gamma_1, \dots, \gamma_K)$, computation of the means of these errors.
- Determination of the optimal value γ_{Opt}, corresponding to the minimal error mean.
- Retain the tree corresponding to γ_{Opt} in Breiman's subsequence $A_1 \dots A_K$

Example of pruned tree for Ozone data



Ozone: Regression tree pruned by cross-validation(R software)

Example of pruned tree for Ozone data



Ozone: Regression tree pruned by cross-validation(Python-sklearn)

Outline

- Introduction
- Classification and Regression Trees
  - Construction of a maximal binary tree
  - Homogeneity criterion
  - Pruning the maximal tree
  - Practical remarks
- Random Forests
- Boosting
- Conclusion

Missing values

- CART is tolerant to missing data for the prediction phase.
- Assume that the dataset has some missing predictor values for some (or all) of the variables.
- Instead of discarding observations with missing values, or imputing the missing values, CART proposes two better strategies.
  - First, for categorical variables, we can add a category for "missing".
  - The second approach is to construct surrogate variables that will be considered if the value of a variable is missing. We choose as usual the best predictor and split at one node, the first surrogate is the second best, and so on.. When an observation is sent down the tree (during the training or prediction phase), if the value of a predictor is missing at one node, we use the first surrogate; if this one is missing, we use the second and so on..

Instability of trees

- A major drawback of trees is their high variance. They are not robust, in the sense that a small change in the data can lead to very different sequences of splits.
- This is why we have to be careful with the interpretation. This is due to the hierarchical procedure: an error in the choice of a split in the top of the tree cannot be corrected below.
- This instability is the price to pay to have a simple and interpretable model.
- We will see in the next chapter how to aggregate trees to obtain Random Forests and reduce the variance of the prediction rule.

Outline

Introduction

- We present algorithms based on a random construction of a family of models: bagging for bootstrap aggregating (Breiman 1996) and the random forests of Breiman (2001) which proposes an improvement of *bagging* specific to models defined by binary trees (CART).
- The principle of *bagging* applies to any modeling method (regression, CART, neural networks..) but are mostly interesting, and significantly reduces the prediction error, only in the case of *unstable* models.
- It is mainly implemented in association with binary trees as a basic models.
- The already underlined instability of trees appears as a property favoring the reduction of variance by aggregation of these models.

Outline

Principle and algorithm

- Let Y be a quantitative or qualitative variable, $X^1, \ldots, X^p$ the explanatory variables, $\hat{f}(\mathbf{x})$ a predictor, with $\mathbf{x} = (x^1, \ldots, x^p) \in \mathbb{R}^p$.
- We denote by n the number of observations and

$$\mathbf{Z} = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$$

  a sample with distribution F.
- Considering B *independent samples* denoted $\{\mathbf{Z}_b\}_{b=1,B}$, a predictor by model aggregation is defined below in the case where the variable to explain Y is:
  - quantitative: $\widehat{f}_B(.) = \frac{1}{B} \sum_{b=1}^{B} \widehat{f}_{\mathbf{Z}_b}(.)$ (mean)
  - qualitative: $\widehat{f}_B(.) = \arg\max_j \text{card} \left\{ b \mid \widehat{f}_{\mathbf{Z}_b}(.) = j \right\}$ (majority vote)

Principle and algorithm

- The principle is elementary, averaging the predictions of several independent models allows to reduce the variance and therefore to reduce the prediction error.
- However, it is unrealistic to consider B independent samples. This would require too much data.
- These samples are therefore replaced by B bootstrap samples each obtained by n draws with replacement according to the empirical measure $\widehat{F}_n$.
- This leads to the following algorithm.

Bagging - Algorithm

Bagging - Algorithm

- Let $\mathbf{x}_0$
- Let $\mathbf{Z} = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ a learning sample
- For $b = 1$ to B
  - Draw a bootstrap sample of size n with replacement $\mathbf{z}_b$.
  - Estimate $\widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0)$ with the bootstrap sample.
- Compute the mean $\widehat{f}_B(\mathbf{x}_0) = \frac{1}{B}\sum_{b=1}^{B} \widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0)$ or the result of a majority vote.

Bagging - Principle

Bootstrap AGGregatING

- Variance reduction: by aggregating independent predictions
  - Aggregation: average (regression), majority vote (classification)
- Bootstrap trick: get new data from themselves by resampling!
  - Caution: new data remain (slightly) dependent on the initial ones

Bagging - Introductive example

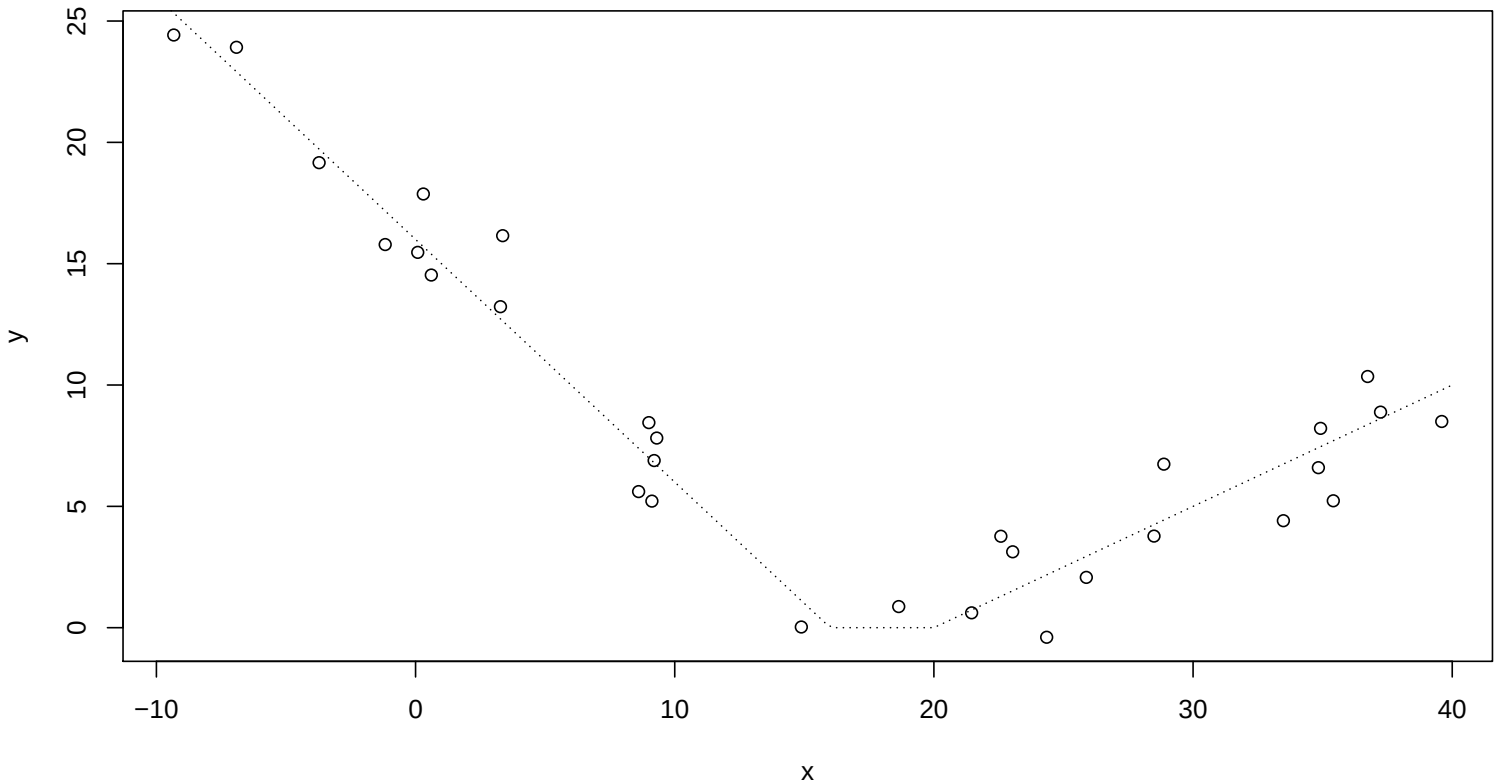


Figure: Original data

Bagging - Introductive example

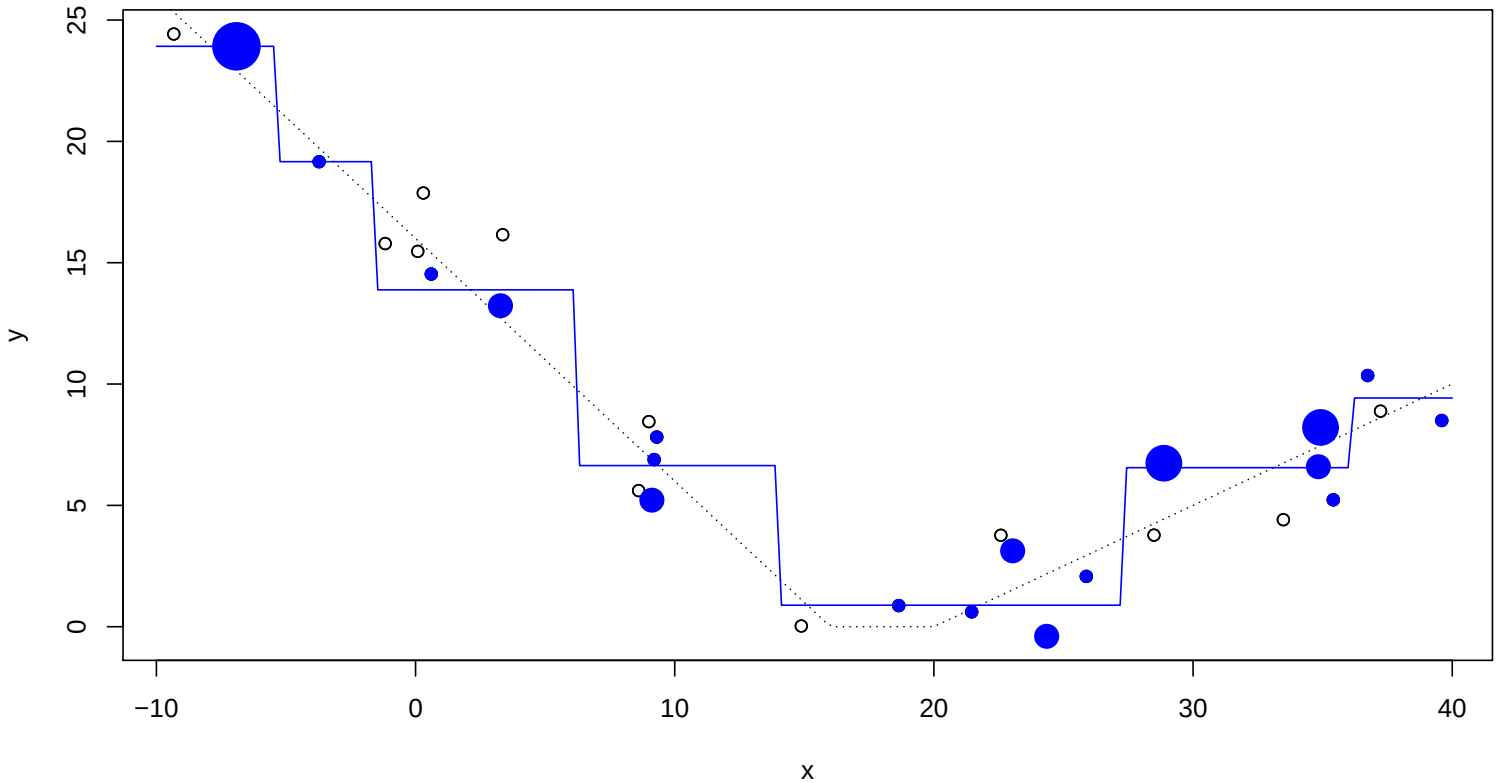


Figure: Bootstrap sample $n^o 1$ (in blue), and corresp. prediction with tree. The point size is proportional to the number of replicates.

Bagging - Introductive example

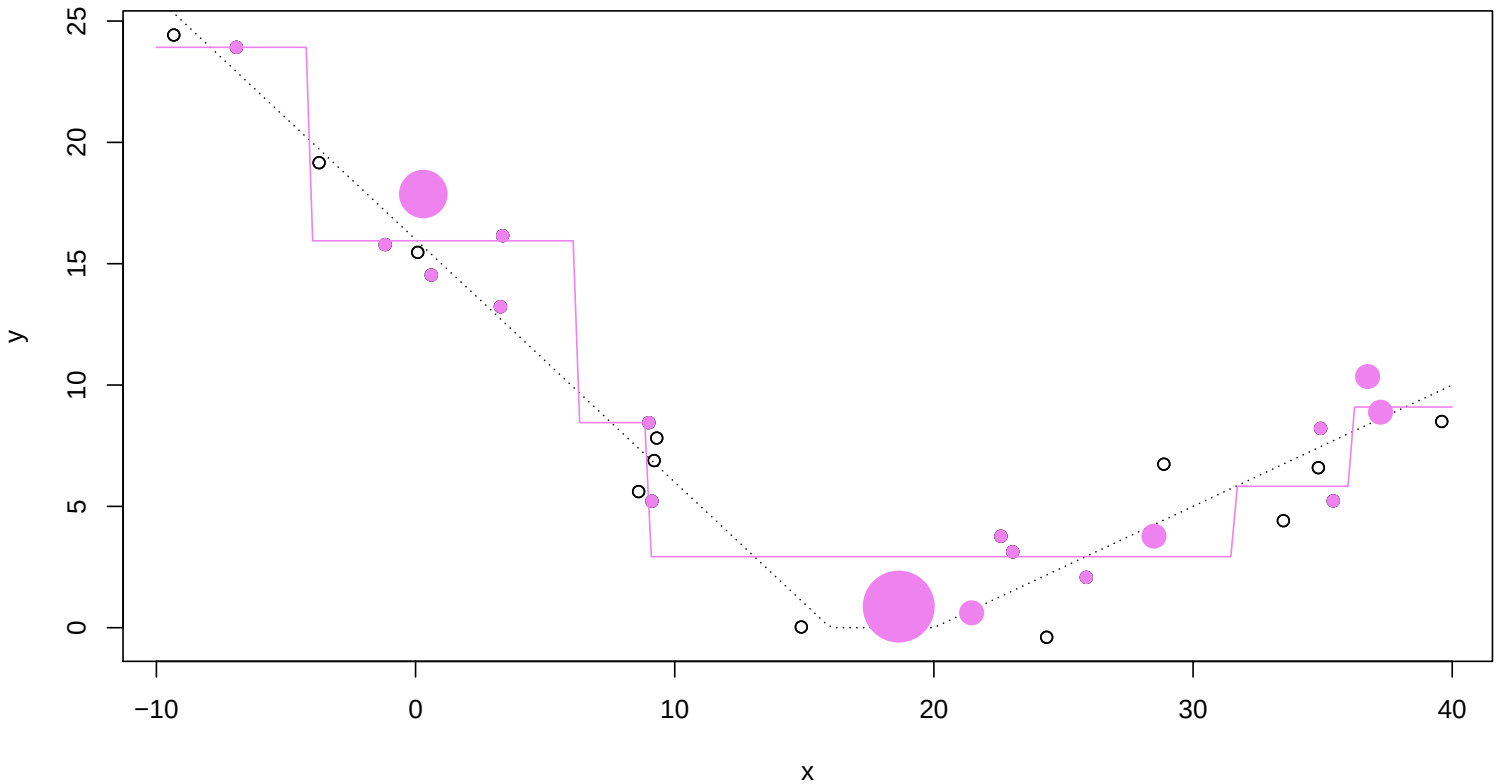


Figure: Bootstrap sample $n^o 2$ (in violet), and corresp. prediction with tree. The point size is proportional to the number of replicates.

Bagging - Introductive example

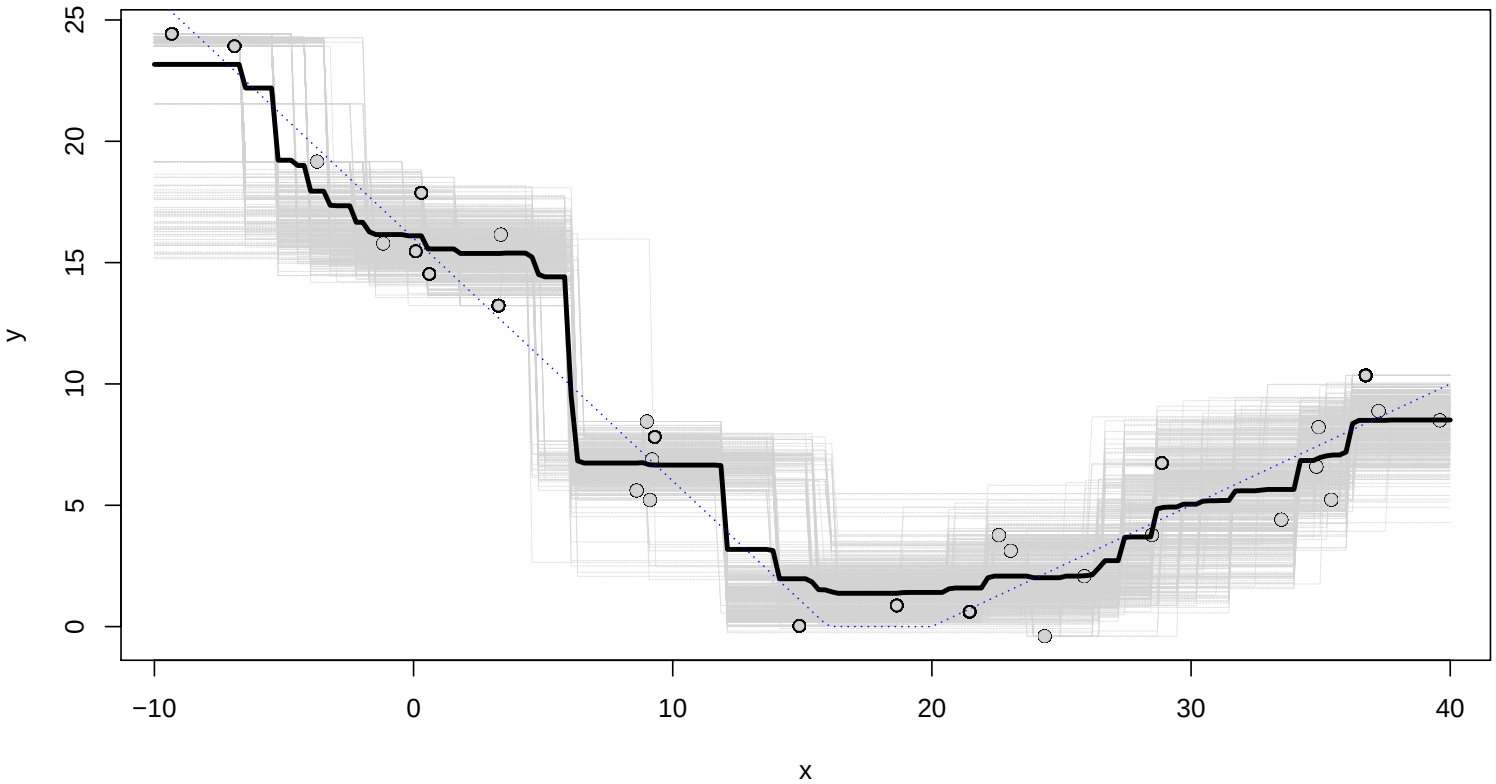


Figure: 500 bootstrap samples (grey), corresp. predictions with tree, and their average (bold line).

Bagging - Theory

- The B boostrap samples are built on the same learning sample $\mathbf{Z} = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ and therefore the estimators $\widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0)$ are not independent.
- We assume that,
  - $\forall b$, $\mathbb{E}(\widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0)) = f(\mathbf{x}_0)$,
  - $\forall b$, $\text{Var}(\widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0)) = V(\mathbf{x}_0)$
  - $\forall b \neq b'$, $\text{Corr}(\widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0), \widehat{f}_{\mathbf{z}_{b'}}(\mathbf{x}_0)) = \rho(x_0)$.
- Then, in the regression case, we obtain

$$\mathbb{E}(\widehat{f}_B(\mathbf{x}_0)) = f(\mathbf{x}_0)$$

$$\text{Var}(\widehat{f}_B(\mathbf{x}_0)) = \rho(x_0)V(\mathbf{x}_0) + \frac{(1-\rho(x_0))}{B}V(\mathbf{x}_0)$$

Bagging - Theory

$$\begin{aligned} \mathrm{Var}(\widehat{f}_B(\mathbf{x}_0)) &= \rho(x_0)V(\mathbf{x}_0) + \frac{(1-\rho(x_0))}{B}V(\mathbf{x}_0) \\ &\to \rho(x_0)V(\mathbf{x}_0) \text{ as } B \to +\infty \end{aligned}$$

- If the correlation term $\rho(x_0)$ is small, the variance of the aggregated predictor $\widehat{f}_B(\mathbf{x}_0)$ is much smaller than the one of a single predictor.
- This underlines the importance of finding low correlated predictors $(\widehat{f}_b(\mathbf{x}_0))_{1\leq b\leq B}$, which is at the core of the **Random forests** algorithm.

Outline

- In the specific case of binary decision tree models (CART), Breiman (2001) proposes an improvement of the *bagging* by adding a random component.
- The objective is to make the aggregated trees more independent by adding randomness in the choice of the variables which are involved in the prediction.
- Since the initial publication of the algorithm, this method has been widely tested and compared with other procedures, it becomes in many machine learning articles the method to beat in terms of prediction accuracy.
- Theoretical convergence properties, difficult to study, have been published quite recently (Scornet et al. 2015).
- However, when the underlying problem is linear, it can also lead to bad results.

Algorithm

The bagging is applied to binary decision trees by adding a random selection of m explanatory variables among the p variables.

Random Forests algorithm

- Let $\mathbf{x}_0$
- Let $\mathbf{Z} = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ a learning sample
- For $b = 1$ to B
  - Take a bootstrap sample $\mathbf{z}_b$
  - Estimate a tree on this sample with **randomization** of the variables: the search for each optimal division is preceded by a random selection of a subset of m predictors.
- Calculate the mean estimate $\widehat{f}_B(\mathbf{x}_0) = \frac{1}{B}\sum_{b=1}^{B} \widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0)$ or the result of a majority vote.

Parameters of the algorithm

- The pruning strategy can, in the case of random forests, be quite elementary.
- Indeed, pruned trees may be strongly correlated because they may involve the same variables appearing to be the most explanatory.
- In the default strategy of the algorithm, it is simply the minimum number of observations per leaf which limits the size of the tree, it is set to 5 by default.
- We therefore aggregate rather complete trees, which are considered of low bias but of high variance.

Parameters of the algorithm

- The random selection of a reduced number of m potential predictors at each stage of the construction of the trees significantly increases the variability by highlighting other variables.
- Each tree is obviously less efficient, sub-optimal, but, united being strength, aggregation ultimately leads to good results.
- The number m of variables drawn randomly can, according to the examples, be a sensitive parameter with default choices are not always optimal:
  - $m = \sqrt{p}$ in a classification problem,
  - $m = p/3$ in a regression problem.
- The iterative evaluation of the *out-of-bag* error makes it possible to control the number B of trees in the forest as well as to optimize the choice of m. It is nevertheless a cross-validation procedure which is preferably used to optimize m.

Bagging - Out-Of-Bag data

Out-Of-Bag (OOB) data

For each bootstrap sample:

- Let $U_1^\star, \ldots, U_N^\star$ be random variables representing the boostrapped indices. The probability that a given data z_i is not chosen is:

$$\mathbb{P}\left(z_{U_1^\star} \neq z_i, \ldots, z_{U_N^\star} \neq z_i\right) = \left(1 - \frac{1}{N}\right)^N \underset{N \to +\infty}{\longrightarrow} e^{-1} \approx 0.367$$

- The non-chosen data are called Out-Of-Bag (OOB). They can be used as a test set inside the bootstrap loop

The OOB error is obtained by averaging prediction errors over OOB data

Bagging - Out-Of-Bag data

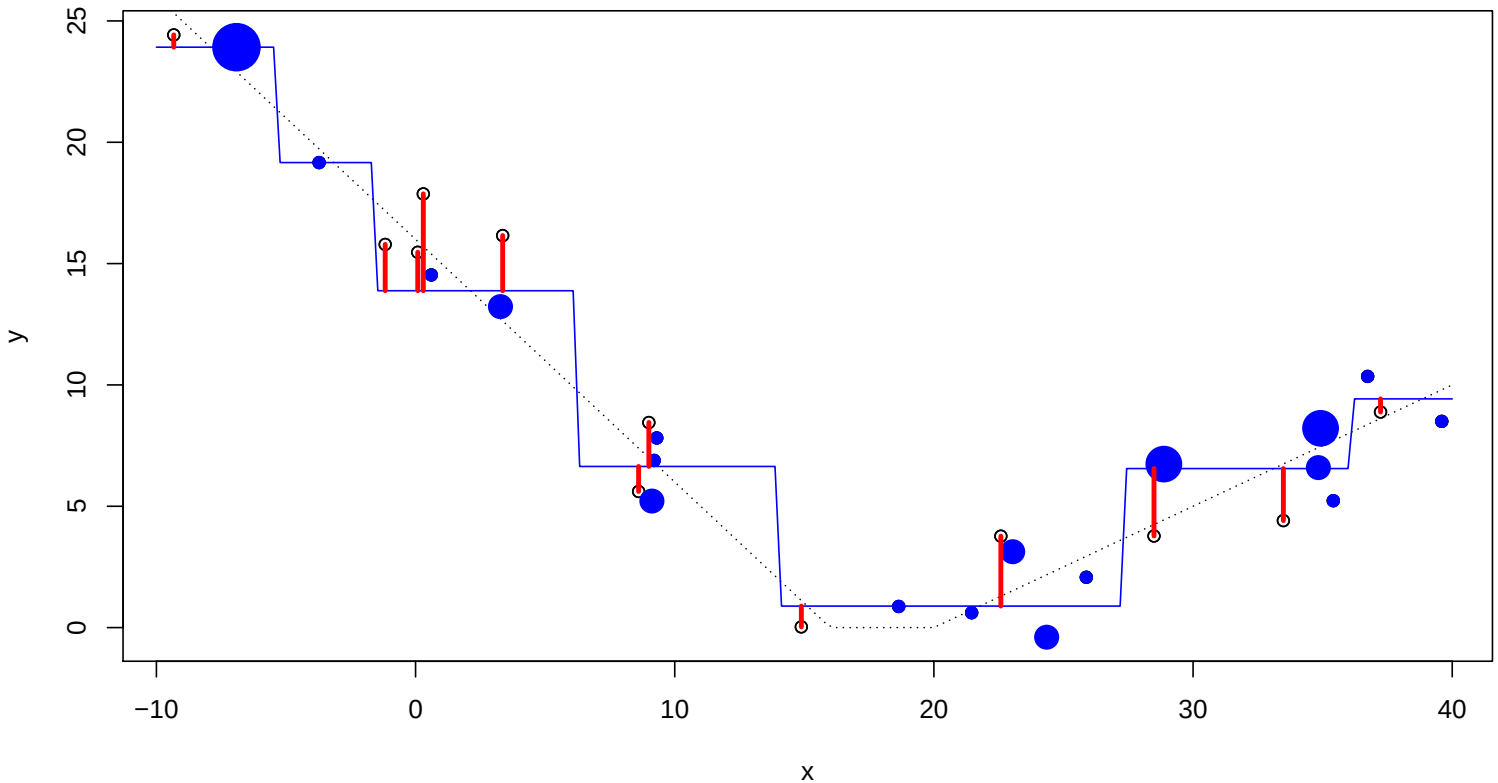


Figure: Residuals for the OOB bootstrap sample $n^o 1$ (red bars).

Bagging - Out-Of-Bag data

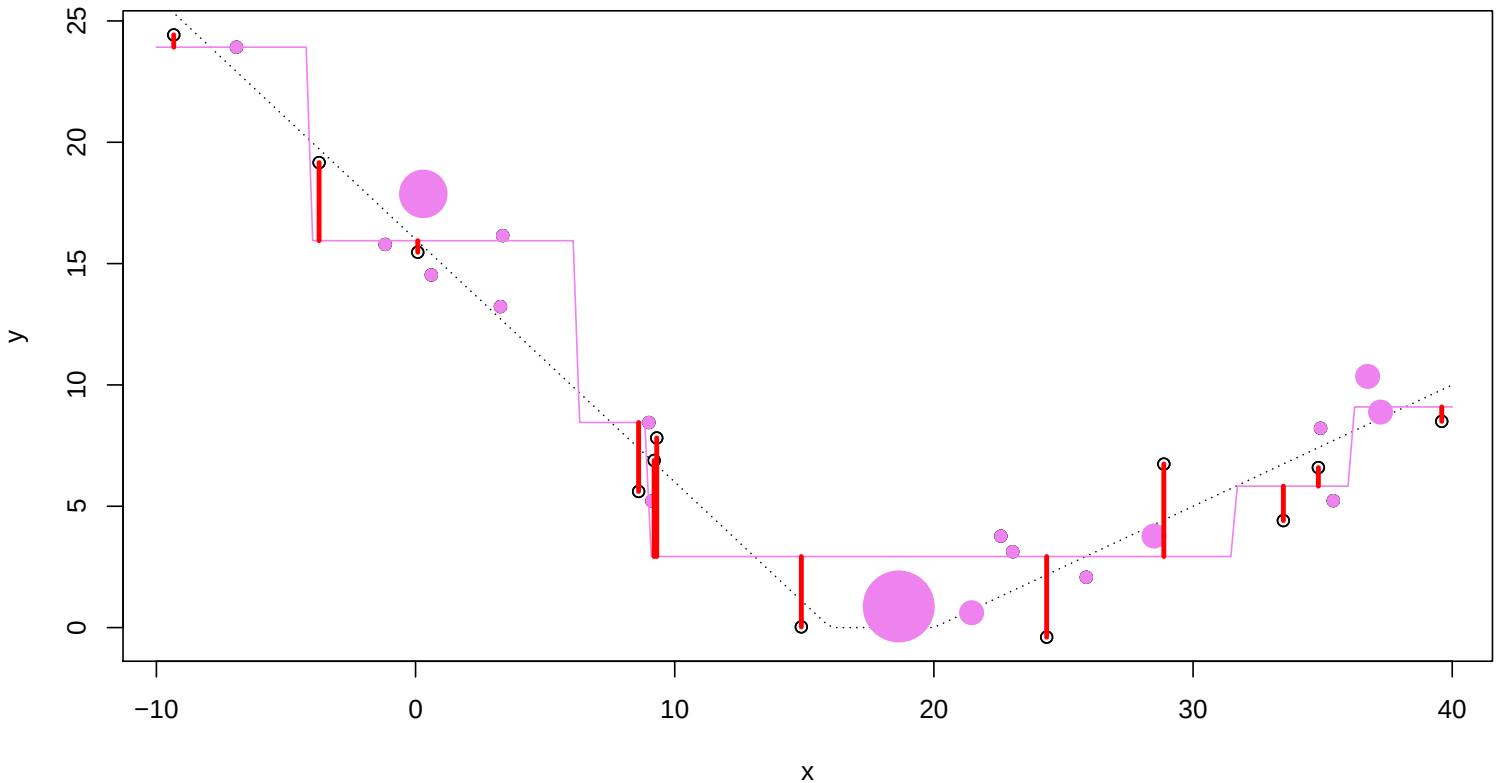


Figure: Residuals for the OOB bootstrap sample $n^o 2$ (red bars).

Outline

Variables importance

- Random forests generally have a good accuracy, they are easily implementable, parallelisable but not easy to interpret like any model built by aggregation, leading to a black-box model.
- To favor interpretation, indexes of importance for each explanatory variable have been introduced.
- This is obviously all the more useful as the variables are very numerous.
- Two criteria have been proposed to evaluate the importance of the variable X^j.

Interpretation - Variable importance

How can we quantify the importance of a variable X^j in random forest?

Mean decrease Impurity: MDI (Breiman, 2003)

- Average the decrease of heterogeneity when X^j is chosen as a split.

Mean Decrease Accuracy: MDA (Breiman, 2001)

- Based on a permutation of variables:
  - Compute the OOB error for the subsample of OOB data.
  - Compare with the OOB error when permuting at random the values of the variable X^j in the learning sample (but keeping the output Y unchanged).

Mean decrease Impurity

More formally, with previous notations, the MDI of the variable X^j is defined by

$$MDI(X^j) = \frac{1}{Mn} \sum_{l=1}^{M} \sum_{\kappa \in \mathcal{T}_l, j_\kappa^* = j} \left[D_\kappa - \left(D_{\kappa_L}(t_\kappa^*, j_\kappa^*) + D_{\kappa_R}(t_\kappa^*, j_\kappa^*) \right) \right],$$

- $\{\mathcal{T}_l, 1 \leq l \leq M\}$ is the collection of trees in the forest,
- (t_κ^*, j_κ^*) the split retained at node κ:
  - j_κ^* corresponds to the optimal variable selected for the split
  - t_κ^* corresponds to the optimal threshold along the j_κ^* variable.

Mean decrease Accuracy

- Consider a variable X^j and denote by $\mathcal{D}_{l,n}$ the out-of-bag data set of the l-th tree and $\mathcal{D}^j_{l,n}$ the same data set where the values of X^j have been randomly permuted.
- Denote by $m_n(.,\Theta_l)$ the l-th tree estimate and

$$R_n\left[m_n(.,\Theta_l),\mathcal{D}\right] = \frac{1}{|\mathcal{D}|}\sum_{i,(X_i,Y_i)\in\mathcal{D}}(Y_i - m_n(X_i,\Theta_l))^2.$$

- The MDA is defined by

$$MDA(X^j) = \frac{1}{M}\sum_{l=1}^{M}\left\{R_n\left[m_n(.,\Theta_l),\mathcal{D}^j_{l,n}\right] - R_n\left[m_n(.,\Theta_l),\mathcal{D}_{l,n}\right]\right\}.$$

Example on ozone data

```
> library(randomForest)
> rf.reg <- randomForest(O3obs ., data = datappr, xtest = datestr[, -2],
    ytest = datestr[, "O3obs"], ntree = 500, do.trace = 50, importance = TRUE)
```

Tree	Out -of - bag MSE	%Var(y)	Test set MSE	%Var(y)
50	697.9	40.77	568.5	36.75
100	689.5	40.28	555.9	35.93
150	683.8	39.95	563.2	36.41
200	685.4	40.04	561	36.27
250	678.2	39.62	564.2	36.47
300	675.1	39.44	569.2	36.79
350	676.8	39.54	572.8	37.02
400	674.3	39.39	571.4	36.93
450	673.9	39.37	571.5	36.94
500	674.3	39.39	569.6	36.82

```
> round(importance(rf.reg), 2)

          %IncMSE    IncNodePurity
JOUR        1.98         11011.79
MOCAGE     41.46        388657.27
TEMPE      51.73        409018.57
STATION    21.73         75350.42
VentMOD    12.95         91387.20
VentANG    18.81        124908.37
SRMH2O     16.76        114463.05
LNO2        7.73         84152.34
LNO        10.04         74387.32
```

Details (from R help file of function `importance`)

"The first measure [%IncMSE] is computed from permuting OOB data: For each tree, the prediction error on the out-of-bag portion of the data is recorded (error rate for classification, MSE for regression). Then the same is done after permuting each predictor variable. The difference between the two are then averaged over all trees, and normalized by the standard deviation of the differences.

The second measure [IncNodePurity] is the total decrease in node impurities from splitting on the variable, averaged over all trees. For classification, the node impurity is measured by the Gini index. For regression, it is measured by residual sum of squares."

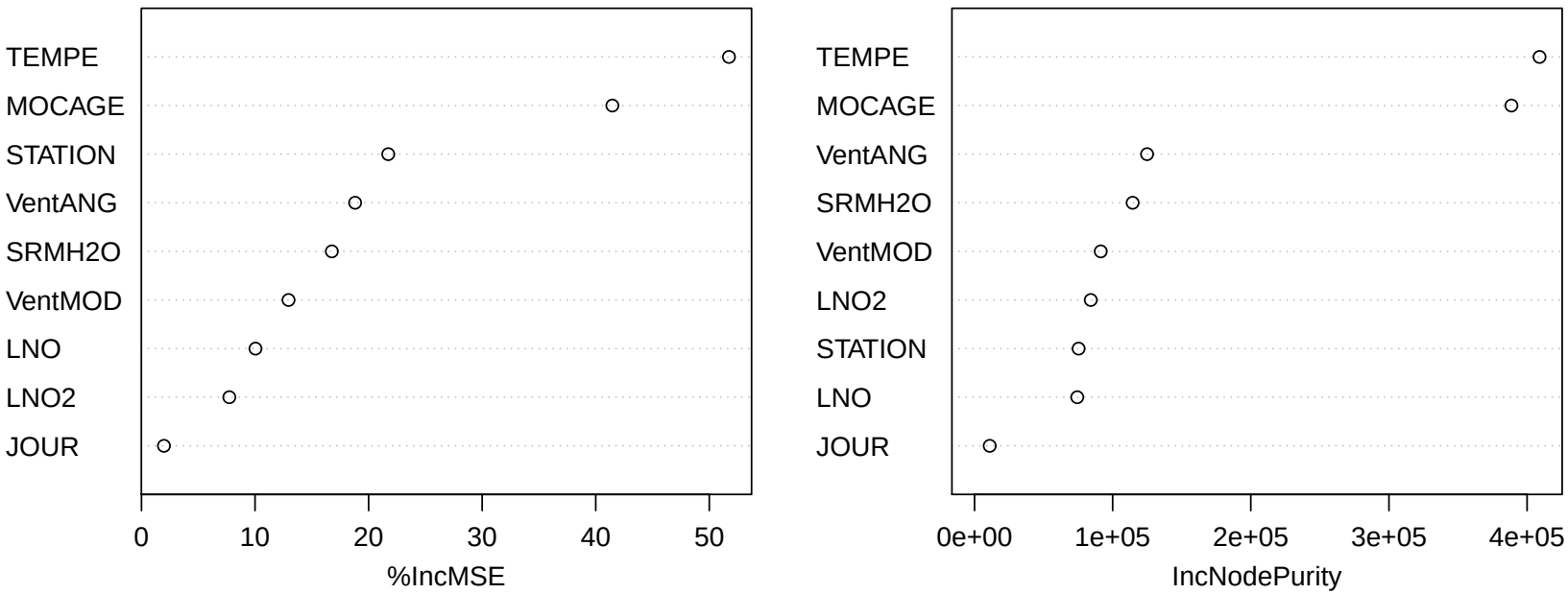


Figure: Variable importance plot, returned by the R function `importance`

Recent work:
C. Bénard, S. da Veiga, E. Scornet, (2022) *MDA for random forests: inconsistency, and a practical solution via the Sobol-MDA*

Outline

Implementation

- The `randomForest` library of R interfaces the original program developed in Fortran77 by Leo Breiman and Adele Cutler which maintains the site dedicated to this algorithm https://math.usu.edu/adele/forests/
- An alternative in R, more efficient in computing time especially with a large volume of data, consists in using the `ranger` library.
- A very efficient and close version of the original algorithm is available in the `Scikit-learn` library of Python.

Outline

- Introduction
- Classification and Regression Trees
- Random Forests
- Boosting
- Conclusion

Boosting principle

- Boosting was originally introduced for binary classification problems but it has been also extended to k class classification and to regression problems.
- The main idea is to combine weak classifiers (or regressors) in a way to get a powerful aggregated procedure.
- The aggregation allows to reduce the variance of the single predictors but also their bias.
- The final prediction rule is obtained as a linear combination of a recursive sequence of predictors, where each predictor is an adaptive version of the previous one, given more weight to the observations that are badly adjusted at the previous step.
- We first present the most popular boosting algorithm, introduced by Freund and Schapire (1996), called AdaBoost for binary classification.

Basic algorithm

AdaBoost

- Adaboost (or adaptive boosting) is a boosting method introduced by Freund and Schapire (1996) to combine several binary classifiers $f_1, \ldots, f_k$ with values in $\{-1, 1\}$.
- $\mathbf{z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ a learning sample, $y_i \in \{-1, 1\}$.
- The aim is to minimize the empirical risk for the exponential loss function over linear combination of the classifiers

$$\hat{f} = \underset{f \in \text{span}(f_1, \ldots, f_k)}{\text{argmin}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \exp(-y_i f(\mathbf{x}_i)) \right\}.$$

AdaBoost

- To approximate the solution, Adaboost computes a sequence of functions $\hat{f}_m$ for $m = 0, \ldots M$ with

$$\begin{aligned} \hat{f}_0 &= 0 \\ \hat{f}_m &= \hat{f}_{m-1} + \beta_m f_{j_m} \end{aligned}$$

where (β_m, j_m) minimizes the empirical risk

$$\underset{\beta \in \mathbb{R}, j=1,\ldots,p}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \exp(-y_i(\hat{f}_{m-1}(\mathbf{x}_i) + \beta f_j(\mathbf{x}_i))) \right\}.$$

- The final classification rule is given by

$$\hat{f} = \operatorname{sign}(\hat{f}_M).$$

AdaBoost

- We denote
$$w_i^{(m)} = \frac{1}{n} \exp(-y_i \hat{f}_{m-1}(\mathbf{x}_i))$$
- We assume that for all $j = 1, \ldots, k$,
$$\widehat{\mathcal{E}_m}(j) = \frac{\sum_{i=1}^n w_i^{(m)} \mathbb{1}_{f_j(\mathbf{x}_i) \neq y_i}}{\sum_{i=1}^n w_i^{(m)}} \in]0, 1[.$$
- Then, we have :
$$j_m = \underset{j=1,\ldots,k}{\operatorname{argmin}} \widehat{\mathcal{E}_m}(j),$$
and
$$\beta_m = \frac{1}{2} \log \left(\frac{1 - \widehat{\mathcal{E}_m}(j_m)}{\widehat{\mathcal{E}_m}(j_m)} \right).$$

AdaBoost

AdaBoost algorithm

- $w_i^{(1)} = 1/n$ for $i = 1, \ldots, n$.
- For $m = 1, \ldots, M$

$$j_m = \underset{j=1,\ldots,p}{\operatorname{argmin}} \widehat{\mathcal{E}_m}(j), \beta_m = \frac{1}{2} \log \left(\frac{1 - \widehat{\mathcal{E}_m}(j_m)}{\widehat{\mathcal{E}_m}(j_m)} \right).$$

$$\begin{aligned} w_i^{(m+1)} &= w_i^{(m)} \exp(-y_i \beta_m f_{j_m}(\mathbf{x}_i)) \text{ for } i = 1, \ldots, n \\ &= \frac{1}{n} \exp(-y_i \hat{f}_{m-1}(\mathbf{x}_i)) \exp(-y_i \beta_m f_{j_m}(\mathbf{x}_i)) \\ &= \frac{1}{n} \exp(-y_i \hat{f}_m(\mathbf{x}_i)) \end{aligned}$$

- $\hat{f}_M(x) = \sum_{m=1}^{M} \beta_m f_{j_m}(x)$.
- $\hat{f} = \text{sign}(\hat{f}_M)$.

Gradient Boosting Models

GBM : Principle 1

- Gradient Boosting Models (Friedman, 2002)
- In the case of a differentiable loss function
- Principle :
  - Construct a sequence of models in such a way that at each step, each model added to the linear combination, appears as a step towards a better solution
  - This step is done in the direction of the gradient of the loss function approximated by a regression tree

Gradient Boosting Models (GBM)

Let us describe the algorithm *Gradient Tree Boosting* for regression

- Initialize $\widehat{f}_0(x) = \arg\min_\gamma \sum_{i=1}^n \ell(Y_i, \gamma)$ for all x.
- For $m = 1$ to M
  - Compute $r_{im} = -\left[\frac{\partial \ell(Y_i, f(X_i))}{\partial f(X_i)}\right]_{f=\hat{f}_{m-1}}$; $i = 1, \ldots, n$
  - Adjust a regression tree δ_m to the data $(X_i, r_{im})_{i=1,\ldots,n}$. The terminal regions are denoted $(R_{jm}, j = 1, \ldots J_m)$.
  - Compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{X_i \in R_{jm}} \ell(Y_i, f_{m-1}(X_i) + \gamma)$.
  - Update $\widehat{f}_m(x) = \widehat{f}_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{1}_{x \in R_{jm}}$.
- Output $\widehat{f}(x) = \widehat{f}_M(x)$.

Extreme Gradient Boosting (XGBoost)

- XGBoost algorithm was introduced by Chen and Guestrin (2016) *XGBoost : A scalable tree boosting system*
- It is based on additional penalization terms to control overfitting.
- XGBoost allows to optimize a lot of parameters, leading to a total control of the gradient Boosting algorithm.
- It also uses implementation tips for parallelization (GPU).
- The implementation is available in R (caret), Python, Julia, Spark ..

XGBoost penalization

- Chen and Guestrin (2016) "*XGBoost : A scalable tree boosting system*" propose to minimize the following objective function :

$$\sum_{i=1}^{n} \ell(f_M(X_i), Y_i) + \sum_{m=1}^{M} \Omega(f_m),$$

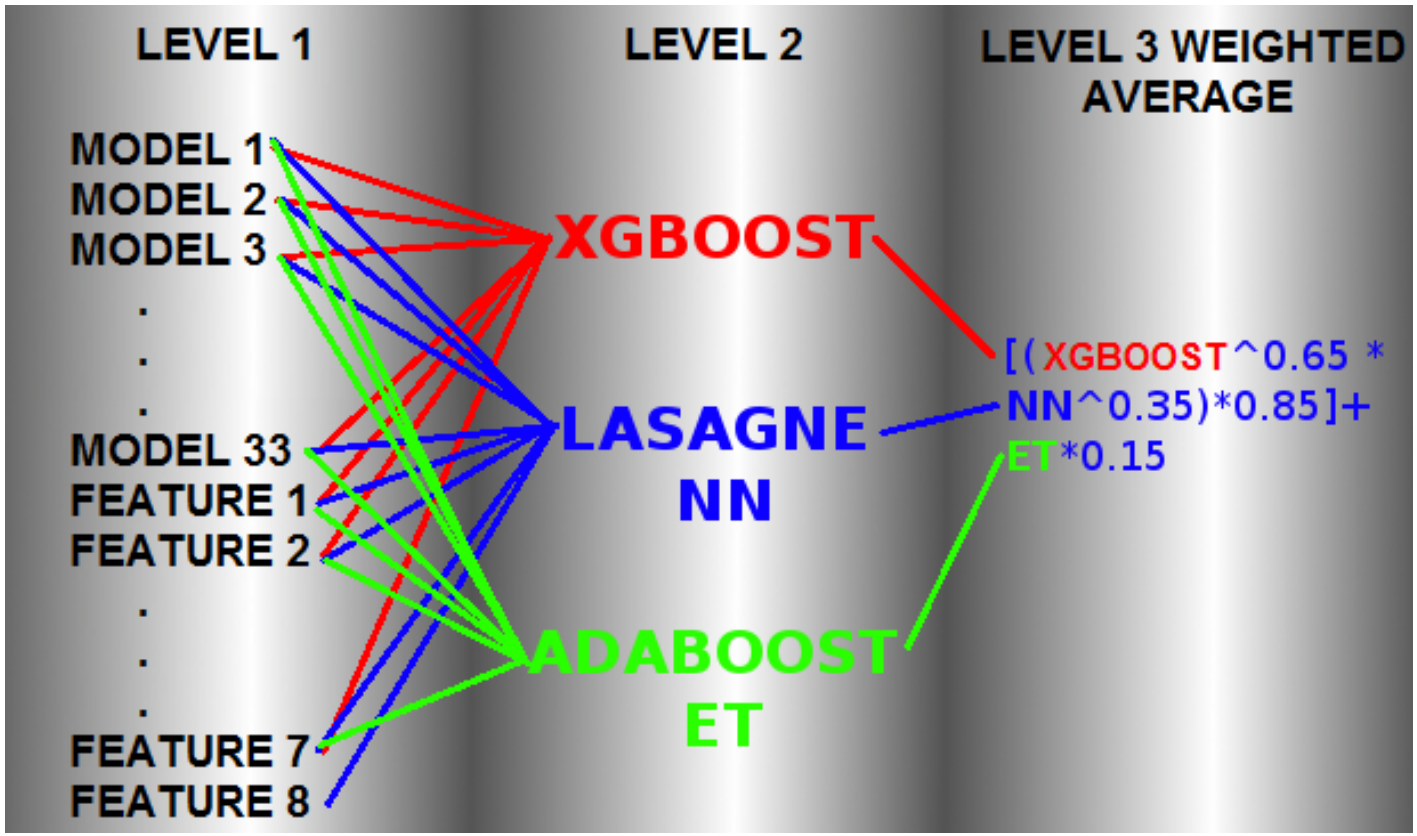
where

$$\Omega(f_m) = \gamma J_m + \lambda \|\boldsymbol{w_m}\|^2,$$

where J_m is the number of leaves of the tree f_m and $\boldsymbol{w_m}$ the vector containing the values assigned to the leaves of the tree f_m.

- The strength of XGBoost lies in the implementation tips for parallelization, making it much faster and more accurate than traditional Gradient Boosting algorithms.
- It is often the winning algorithm of Kaggle competitions.

Extreme Gradient Boosting (XGBoost)



Kaggle: Identify people who have a high degree of Psychopathy based on Twitter usage

Outline

- Introduction
- Classification and Regression Trees
- Random Forests
- Boosting
- Conclusion

Conclusion

- Trees are easy to interpret.
- There exists efficient algorithms to find the pruned trees.
- Trees are tolerant to missing data.

$\Longrightarrow$ Success of CART for practical applications.

- High instability of the trees: not robust to the learning sample, curse of dimensionality ..
- Prediction accuracy of a tree is often poor compared to other procedures.

$\Longrightarrow$ This is why Random Forests have been introduced.

- They generally have good performances.
- Indices of importance for variables are computed for a better interpretability.

Conclusion

Random Forests are used for different purposes (see the dedicated site: https://math.usu.edu/adele/forests/)

- Similarity or proximity between observations: after building each tree, increment by 1 the similarity or proximity of two observations that are in the same leaf. Sum on the trees of the forest, normalize by the number of trees. We can compute a matrix of dissimilarities that results from them.
- Detection of multidimensional atypical observations: outliers or novelties that correspond to observations which do not belong to known classes. A criterion of "abnormality" with respect to a class is based on the previous notion of proximities of an observation to the other observations of its class.
- Another algorithm, inspired by Random Forests has been developed for anomaly detection, it is called Isolation Forests.
- Random forests are used for the imputation of missing data.
- Adaptations to take into account censored data to model survival times correspond to the Survival Forests algorithm.

Conclusion

- The boosting allows to reduce the variance compared with single procedures but also the bias by aggregation, which generally leads to very performant procedures.
- Trees are easily interpretable. The interpretation is lost by aggregating. Nevertheless, like for Random Forest, indices of importance can be computed : one can average over the M trees of the boosting algorithm the indices of importance computed for each tree. This is crucial to have an idea of the relative importance of each predictor in the model.
- The Boosting is implemented in the R package `gbm` and `xgboost`. It is also implemented in the Scikit Learn library of Python (`GradientBoostingClassifier`, `GradientBoostingRegressor`, `xgboost`).

References

- C. Bénard, S. da Veiga, E. Scornet, (2022) *MDA for random forests: inconsistency, and a practical solution via the Sobol-MDA*, hal
- G. Biau, E. Scornet (2016) *A random forest guided tour* TEST, 25: 197-227.
- L. Breiman, J. Friedman, C. J. Stone, R. A. Olshen (1984). *Classification and regression trees*. Chapman et Hall. CRC Press, Boca Raton.
- L. Breiman, (2001) *Random forests* Machine Learning 45: 5-32.
- L. Breiman, (2003) *Setting up, using and understanding Random Forests* https://www.stat.berkeley.edu
- Chen, T. and Guestrin, C. (2016) *XGBoost : A scalable tree boosting system*, KDD 16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. https://arxiv.org/pdf/1603.02754.pdf
- R. Genuer et J.M. Poggi *Les forêts aléatoires avec R*, Presses Universitaires de Rennes.
- Hastie, T. and Tibshirani, R. and Friedman, J, (2009), *The elements of statistical learning: data mining, inference, and prediction*, Springer.
- R.E. Shapire and Y. Freund *Boosting: Fundation and Algorithms* MIT Press, 2012