

Statistics and learning: Big Data

Learning Decision Trees and an Introduction to Boosting

Sébastien Gadat

Toulouse School of Economics

February 2017

Keywords

- ▶ Decision trees
- ▶ Divide and Conquer
- ▶ Impurity measure, Gini index, Information gain
- ▶ Pruning and overfitting
- ▶ CART and C4.5

Contents of this class:

The general idea of learning decision trees

Regression trees

Classification trees

Boosting and trees

Random Forests and trees

Introductory example

	Alt	Bar	F/S	Hun	Pat	Pri	Rai	Res	Typ	Dur	Wai
x_1	Y	N	N	Y	0.38	\$\$\$	N	Y	French	8	Y
x_2	Y	N	N	Y	0.83	\$	N	N	Thai	41	N
x_3	N	Y	N	N	0.12	\$	N	N	Burger	4	Y
x_4	Y	N	Y	Y	0.75	\$	Y	N	Thai	12	Y
x_5	Y	N	Y	N	0.91	\$\$\$	N	Y	French	75	N
x_6	N	Y	N	Y	0.34	\$\$	Y	Y	Italian	8	Y
x_7	N	Y	N	N	0.09	\$	Y	N	Burger	7	N
x_8	N	N	N	Y	0.15	\$\$	Y	Y	Thai	10	Y
x_9	N	Y	Y	N	0.84	\$	Y	N	Burger	80	N
x_{10}	Y	Y	Y	Y	0.78	\$\$\$	N	Y	Italian	25	N
x_{11}	N	N	N	N	0.05	\$	N	N	Thai	3	N
x_{12}	Y	Y	Y	Y	0.89	\$	N	N	Burger	38	Y

Please describe this dataset *without* any calculation.

Introductory example

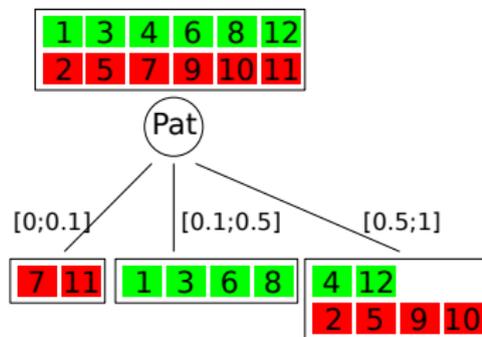
	Alt	Bar	F/S	Hun	Pat	Pri	Rai	Res	Typ	Dur	Wai
x_1	Y	N	N	Y	0.38	\$\$\$	N	Y	French	8	Y
x_2	Y	N	N	Y	0.83	\$	N	N	Thai	41	N
x_3	N	Y	N	N	0.12	\$	N	N	Burger	4	Y
x_4	Y	N	Y	Y	0.75	\$	Y	N	Thai	12	Y
x_5	Y	N	Y	N	0.91	\$\$\$	N	Y	French	75	N
x_6	N	Y	N	Y	0.34	\$\$	Y	Y	Italian	8	Y
x_7	N	Y	N	N	0.09	\$	Y	N	Burger	7	N
x_8	N	N	N	Y	0.15	\$\$	Y	Y	Thai	10	Y
x_9	N	Y	Y	N	0.84	\$	Y	N	Burger	80	N
x_{10}	Y	Y	Y	Y	0.78	\$\$\$	N	Y	Italian	25	N
x_{11}	N	N	N	N	0.05	\$	N	N	Thai	3	N
x_{12}	Y	Y	Y	Y	0.89	\$	N	N	Burger	38	Y

Why is Pat a better indicator than Typ?

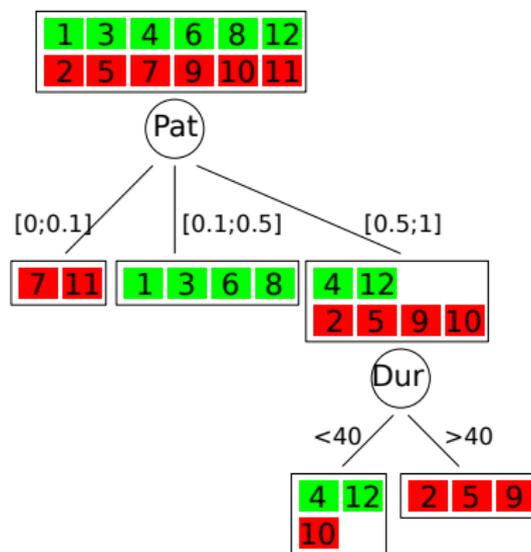
Deciding to wait... or not

1	3	4	6	8	12
2	5	7	9	10	11

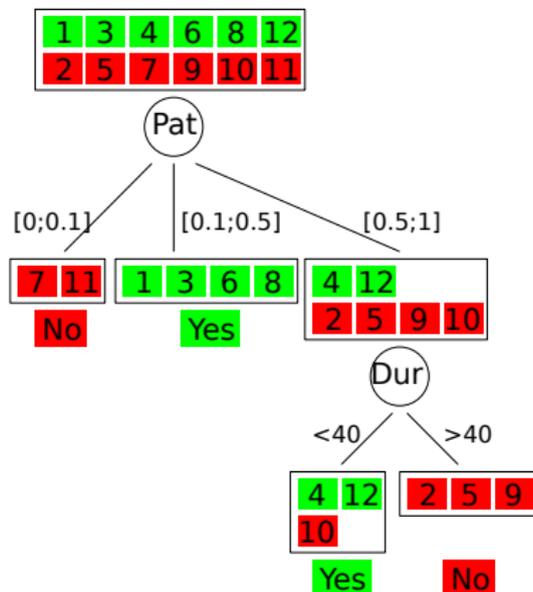
Deciding to wait... or not



Deciding to wait... or not



Deciding to wait... or not



Decision trees

Ingredients:

- ▶ Nodes
Each node contains a *test* on the features which *partitions* the data.
- ▶ Edges
The outcome of a node's test leads to one of its child edges.
- ▶ Leaves
A terminal node, or leaf, holds a *decision value* for the output variable.

Decision trees

Ingredients:

- ▶ Nodes
Each node contains a *test* on the features which *partitions* the data.
- ▶ Edges
The outcome of a node's test leads to one of its child edges.
- ▶ Leaves
A terminal node, or leaf, holds a *decision value* for the output variable.

We will look at binary trees (\Rightarrow binary tests) and single variable tests.

Binary attribute: node = attribute

Continuous attribute: node = (attribute, threshold)

Decision trees

Ingredients:

- ▶ Nodes
Each node contains a *test* on the features which *partitions* the data.
- ▶ Edges
The outcome of a node's test leads to one of its child edges.
- ▶ Leaves
A terminal node, or leaf, holds a *decision value* for the output variable.

We will look at binary trees (\Rightarrow binary tests) and single variable tests.

Binary attribute: node = attribute

Continuous attribute: node = (attribute, threshold)

How does one build a good decision tree?

For a regression problem?

For a classification problem?

A little more formally

A tree with M leaves describes a covering set of M hypercubes R_m in X . Each R_m hold a decision value \hat{y}_m .

$$\hat{f}(x) = \sum_{m=1}^M \hat{y}_m I_{R_m}(x)$$

Notation:

$$N_m = |\{x_i \in R_m\}| = \sum_{i=1}^q I_{R_m}(x_i)$$

The general idea: divide and conquer

Example Set T , attributes x_1, \dots, x_p

FormTree(T)

1. Find best split (j, s) over T // Which criterion?
2. If $(j, s) = \emptyset$,
 - ▶ node = FormLeaf(T) // Which value for the leaf?
3. Else
 - ▶ node = (j, s)
 - ▶ split T according to (j, s) into $(T1, T2)$
 - ▶ append FormTree($T1$) to node // Recursive call
 - ▶ append FormTree($T2$) to node
4. Return node

The general idea: divide and conquer

Example Set T , attributes x_1, \dots, x_p

FormTree(T)

1. Find best split (j, s) over T // Which criterion?
2. If $(j, s) = \emptyset$,
 - ▶ node = FormLeaf(T) // Which value for the leaf?
3. Else
 - ▶ node = (j, s)
 - ▶ split T according to (j, s) into $(T1, T2)$
 - ▶ append FormTree($T1$) to node // Recursive call
 - ▶ append FormTree($T2$) to node
4. Return node

Remark

This is a greedy algorithm, performing local search.

The R point of view

Two packages for tree-based methods: `tree` and `rpart`.

Regression trees – criterion

We want to fit a tree to the data $\{(x_i, y_i)\}_{i=1..q}$ with $y_i \in \mathbb{R}$.

Criterion?

Regression trees – criterion

We want to fit a tree to the data $\{(x_i, y_i)\}_{i=1..q}$ with $y_i \in \mathbb{R}$.

Criterion? Sum of squares: $\sum_{i=1}^q (y_i - \hat{f}(x_i))^2$

Regression trees – criterion

We want to fit a tree to the data $\{(x_i, y_i)\}_{i=1..q}$ with $y_i \in \mathbb{R}$.

Criterion? Sum of squares: $\sum_{i=1}^q \left(y_i - \hat{f}(x_i) \right)^2$

Inside region R_m , best \hat{y}_m ?

Regression trees – criterion

We want to fit a tree to the data $\{(x_i, y_i)\}_{i=1..q}$ with $y_i \in \mathbb{R}$.

Criterion? Sum of squares: $\sum_{i=1}^q (y_i - \hat{f}(x_i))^2$

Inside region R_m , best \hat{y}_m ? $\hat{y}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i = \bar{Y}_{R_m}$

Node impurity measure:

$$Q_m = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{y}_m)^2$$

Regression trees – criterion

Best partition: hard to find. But locally, best split?

Regression trees – criterion

Best partition: hard to find. But locally, best split? Solve $\operatorname{argmin}_{j,s} C(j, s)$

$$\begin{aligned}
 C(j, s) &= \left[\min_{\hat{y}_1} \sum_{x_i \in R_1(j,s)} (y_i - \hat{y}_1)^2 + \min_{\hat{y}_2} \sum_{x_i \in R_2(j,s)} (y_i - \hat{y}_2)^2 \right] \\
 &= \left[\sum_{x_i \in R_1(j,s)} (y_i - \bar{Y}_{R_1(j,s)})^2 + \sum_{x_i \in R_2(j,s)} (y_i - \bar{Y}_{R_2(j,s)})^2 \right] \\
 &= N_1 Q_1 + N_2 Q_2
 \end{aligned}$$

Overgrowing the tree?

- ▶ Too small: rough average.
- ▶ Too large: overfitting.

	Alt	Bar	F/S	Hun	Pat	Pri	Rai	Res	Typ	Dur
x_1	Y	N	N	Y	0.38	\$\$\$	N	Y	French	8
x_2	Y	N	N	Y	0.83	\$	N	N	Thai	41
x_3	N	Y	N	N	0.12	\$	N	N	Burger	4
x_4	Y	N	Y	Y	0.75	\$	Y	N	Thai	12
x_5	Y	N	Y	N	0.91	\$\$\$	N	Y	French	75
x_6	N	Y	N	Y	0.34	\$\$	Y	Y	Italian	8
x_7	N	Y	N	N	0.09	\$	Y	N	Burger	7
x_8	N	N	N	Y	0.15	\$\$	Y	Y	Thai	10
x_9	N	Y	Y	N	0.84	\$	Y	N	Burger	80
x_{10}	Y	Y	Y	Y	0.78	\$\$\$	N	Y	Italian	25
x_{11}	N	N	N	N	0.05	\$	N	N	Thai	3
x_{12}	Y	Y	Y	Y	0.89	\$	N	N	Burger	38

Overgrowing the tree?

Stopping criterion?

- ▶ Stop if $\min_{j,s} C(j, s) > \kappa$? Not good because a good split might be hidden in deeper nodes.
- ▶ Stop if $N_m < n$? Good to avoid overspecialization.
- ▶ Prune the tree after growing. *cost-complexity pruning*.

Cost-complexity criterion:

$$C_\alpha = \sum_{m=1}^M N_m Q_m + \alpha M$$

Once a tree is grown, prune it to minimize C_α .

- ▶ Each α corresponds to a unique cost-complexity optimal tree.
- ▶ Pruning method: *Weakest link pruning*, left to your curiosity.
- ▶ Best α ? Through cross-validation.

Regression trees in a nutshell

- ▶ Constant values on the leaves.
- ▶ Growing phase: greedy splits that minimize the squared-error impurity measure.
- ▶ Pruning phase: Weakest-link pruning that minimize the cost-complexity criterion.

Regression trees in a nutshell

- ▶ Constant values on the leaves.
- ▶ Growing phase: greedy splits that minimize the squared-error impurity measure.
- ▶ Pruning phase: Weakest-link pruning that minimize the cost-complexity criterion.

Further reading on regression trees:

- ▶ MARS: Multivariate Adaptive Regression Splines.
Linear functions on the leaves.
- ▶ PRIM: Patient Rule Induction Method.
Focuses on extremas rather than averages.

A bit of R before classification tasks

Let's load the "Optical Recognition of Handwritten Digits" database.

```
> optical <- read.csv("optdigits.tra", sep="," ,  
header=FALSE)  
> colnames(optical)[65] <- "class"
```

Classification trees

Suppose $y_i \in \{True; False\}$. Let's fit a tree to $\{(x_i, y_i)\}_{i=1..q}$.

Best \hat{y}_m in node m ?

Classification trees

Suppose $y_i \in \{True; False\}$. Let's fit a tree to $\{(x_i, y_i)\}_{i=1..q}$.

Best \hat{y}_m in node m ?

Proportion of class k observations in node m :

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

Class of node m : $\hat{y}_m = \operatorname{argmax}_k \hat{p}_{mk}$

Classification trees

Node impurity measure?

Misclassification error

$$Q_m = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i \neq \hat{y}_m) = 1 - \hat{p}_{m\hat{y}_m}$$

Gini index (CART)

$$Q_m = \sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk})$$

Information or deviance (C4.5)

$$Q_m = - \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}$$

Classification trees

Node impurity measure?

Misclassification error

$$Q_m = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i \neq \hat{y}_m) = 1 - \hat{p}_{m\hat{y}_m}$$

Gini index (CART)

$$Q_m = \sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk})$$

Information or deviance (C4.5)

$$Q_m = - \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}$$

Splitting criterion? Minimize $N_1 Q_1 + N_2 Q_2$

Classification trees

Node impurity measure?

Misclassification error

$$Q_m = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i \neq \hat{y}_m) = 1 - \hat{p}_m \hat{y}_m$$

Gini index (CART)

$$Q_m = \sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk})$$

Information or deviance (C4.5)

$$Q_m = - \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}$$

Splitting criterion? Minimize $N_1 Q_1 + N_2 Q_2$

Pruning? Cost-complexity (often using the misclassification error) criterion.

$$C_\alpha = \sum_{m=1}^M N_m Q_m + \alpha M$$

Classification trees in a nutshell

- ▶ Class values on the leaves.
- ▶ Growing phase: greedy splits that maximize the Gini index reduction (CART) or the information gain (C4.5)
- ▶ Pruning phase: Weakest-link pruning that minimize the cost-complexity criterion.

Classification trees in a nutshell

- ▶ Class values on the leaves.
- ▶ Growing phase: greedy splits that maximize the Gini index reduction (CART) or the information gain (C4.5)
- ▶ Pruning phase: Weakest-link pruning that minimize the cost-complexity criterion.

Further reading on classification trees:

- ▶ EC4.5 and YaDT: Implementation improvements for C4.5
- ▶ C5.0: C4.5 with additional features.
- ▶ Loss matrix.
- ▶ Handling missing values.

A bit of R

```
> help(tree)
> optical.tree <- tree(factor(class) ~., optical,
split="deviance")
> optical.tree.gini <- tree(factor(class) ~., optical,
split="gini")
> plot(optical.tree); text(optical.tree)
> help(prune.tree)
> optical.tree.pruned <- prune.tree(optical.tree,
method="misclass", k=10)
> help(cv.tree)
> optical.tree.cv <- cv.tree(optical.tree, ,
prune.misclass)
> plot(optical.tree.cv)
```

Why should you use Decision Trees?

Advantages

- ▶ Easy to read and interpret.
- ▶ Learning the tree has complexity linear in p .
- ▶ Can be rather efficient on well pre-processed data (in conjunction with PCA for instance).

However

- ▶ No margin or performance guarantees.
- ▶ Lack of smoothness in the regression case.
- ▶ Strong assumption that the data can fit in hypercubes.
- ▶ Strong sensitivity to the data set.

But...

- ▶ Can be compensated by ensemble methods such as Boosting or Bagging.
- ▶ Very efficient extension with Random Forests

Boosting and trees

Motivation

AdaBoost with trees is the best off-the-shelf classifier in the world.
(Breiman 1998)

Not so true today but still accurate enough.

What is Boosting?

Key idea

Boosting is a procedure that combines several “weak” classifiers into a powerful “committee”.

Committee-based or ensemble methods literature in Machine Learning.

Most popular boosting alg. (Freund & Schapire, 1997): AdaBoost.M1.

Warning

For this part, we take a very practical approach. For a more thorough and rigorous presentation, see (for instance) the reference below.

R. E. Schapire. **The boosting approach to machine learning: An overview.** Nonlinear Estimation and Classification, 2002.

The main picture

Weak classifiers

$h(x) = y$ is said to be a *weak* (or a PAC-weak) classifier if it performs better than a random guessing on the training data.

AdaBoost

AdaBoost constructs a strong classifier as a linear combination of weak classifiers $h_t(x)$:

$$f(x) = \sum_{t=1}^T \alpha_t h_t(x)$$

The AdaBoost algorithm

Given $\{(x_i, y_i)\}, x_i \in X, y_i \in \{-1; 1\}$.

The AdaBoost algorithm

Given $\{(x_i, y_i)\}$, $x_i \in X$, $y_i \in \{-1; 1\}$.

Initialize weights $D_1(i) = 1/q$

The AdaBoost algorithm

Given $\{(x_i, y_i)\}$, $x_i \in X$, $y_i \in \{-1; 1\}$.

Initialize weights $D_1(i) = 1/q$

For $t = 1$ to T :

- ▶ Find $h_t = \operatorname{argmin}_{h \in \mathcal{H}} \sum_{i=1}^q D_t(i) I(y_i \neq h(x_i))$

The AdaBoost algorithm

Given $\{(x_i, y_i)\}$, $x_i \in X$, $y_i \in \{-1; 1\}$.

Initialize weights $D_1(i) = 1/q$

For $t = 1$ to T :

- ▶ Find $h_t = \operatorname{argmin}_{h \in \mathcal{H}} \sum_{i=1}^q D_t(i) I(y_i \neq h(x_i))$
- ▶ If $\epsilon_t = \sum_{i=1}^q D_t(i) I(y_i \neq h_t(x_i)) \geq 1/2$ then stop

The AdaBoost algorithm

Given $\{(x_i, y_i)\}$, $x_i \in X, y_i \in \{-1; 1\}$.

Initialize weights $D_1(i) = 1/q$

For $t = 1$ to T :

- ▶ Find $h_t = \operatorname{argmin}_{h \in \mathcal{H}} \sum_{i=1}^q D_t(i) I(y_i \neq h(x_i))$
- ▶ If $\epsilon_t = \sum_{i=1}^q D_t(i) I(y_i \neq h_t(x_i)) \geq 1/2$ then stop
- ▶ Set $\alpha_t = \frac{1}{2} \log \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

The AdaBoost algorithm

Given $\{(x_i, y_i)\}$, $x_i \in X, y_i \in \{-1; 1\}$.

Initialize weights $D_1(i) = 1/q$

For $t = 1$ to T :

- ▶ Find $h_t = \operatorname{argmin}_{h \in \mathcal{H}} \sum_{i=1}^q D_t(i) I(y_i \neq h(x_i))$
- ▶ If $\epsilon_t = \sum_{i=1}^q D_t(i) I(y_i \neq h_t(x_i)) \geq 1/2$ then stop
- ▶ Set $\alpha_t = \frac{1}{2} \log \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
- ▶ Update

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

Where Z_t is a normalisation factor.

The AdaBoost algorithm

Given $\{(x_i, y_i)\}$, $x_i \in X, y_i \in \{-1; 1\}$.

Initialize weights $D_1(i) = 1/q$

For $t = 1$ to T :

- ▶ Find $h_t = \operatorname{argmin}_{h \in \mathcal{H}} \sum_{i=1}^q D_t(i) I(y_i \neq h(x_i))$
- ▶ If $\epsilon_t = \sum_{i=1}^q D_t(i) I(y_i \neq h_t(x_i)) \geq 1/2$ then stop
- ▶ Set $\alpha_t = \frac{1}{2} \log \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
- ▶ Update

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

Where Z_t is a normalisation factor.

Return the classifier

$$H(x) = \operatorname{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right)$$

Iterative reweighting

$$D_{t+1}(i) = \frac{D_t(i)e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

- ▶ Increase the weight of incorrectly classified samples
- ▶ Decrease the weight of correctly classified samples
- ▶ Memory effect: a sample misclassified several times has a large $D(i)$
- ▶ h_t focusses on samples that were misclassified by h_0, \dots, h_{t-1}

Properties

$$\frac{1}{q} \sum_{i=1}^q I(H(x_i) \neq y_i) \leq \prod_{t=1}^T Z_t$$

- ▶ To minimize training error at each step t , minimize this upper bound.
→ This is where $\alpha_t = \frac{1}{2} \log \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$ comes from.
- ▶ This is equivalent to maximizing the margin!

AdaBoost is not Boosting

Many variants of AdaBoost:

- ▶ Binary classification AdaBoost.M1, AdaBoost.M2, . . . ,
- ▶ Multiclass AdaBoost.MH,
- ▶ Regression AdaBoost.R,
- ▶ Online, . . .

And other Boosting algorithms.

Why should you use Boosting?

AdaBoost is a meta-algorithm: it “boosts” a weak classif. algorithm into a committee that is a strong classifier.

- ▶ AdaBoost maximizes margin
- ▶ Very simple to implement
- ▶ Can be seen as a feature selection algorithm
- ▶ In practice, AdaBoost often avoids overfitting.

AdaBoost with trees

Your turn to play: will you be able to implement AdaBoost with trees in R?

Random Forests and trees

Motivation:

- ▶ Aggregation for stabilizing tree inference.
- ▶ Introduce independence between trees to make the aggregation step robust
- ▶ Simple remark: the variance of an average of B i.i.d. random variable is

$$\frac{\sigma^2}{B}.$$

When the variables are correlated with a coefficient ρ , we then have a variance of

$$\rho\sigma^2 + \frac{(1 - \rho)\sigma^2}{B}$$

- ▶ Idea of bagging: sub-sample in the training set to obtain an aggregation with ρ small and B large.

Random Forests and trees

To avoid pruning trees and bypass over-fitting, a common way is to randomly subsample the training set, the set of variables, and then average. It then leads to the so-called Random Forest algorithm.

Algorithm 1 Random forest algorithm

Input Training Set \mathcal{D} - Number of bags B - integer m

B **Iterates** Sample a bootstrap training set \mathcal{D}_b among the n observations

Sample a subset of m variables among the p variables

Compute a classification tree T_b

Output: Prediction with the average decision rule $B^{-1} \sum_{b=1}^B T_b$.

Random Forests and trees

Important parameters/features for the algorithm:

- ▶ m : number of variables that are sampled to build each individual tree. If p is the total number of variables, it should be chosen like

$$m \simeq \sqrt{p}.$$

- ▶ Important remark: it is possible with RF to produce a **selection** of the good variables. Important variables are the ones that are the most selected among the whole population of trees at each node.

Everything is possible with the Random Forest package of Breiman...