Tree based methods:

CART - Random Forest

B. Michel

Ecole Centrale de Nantes

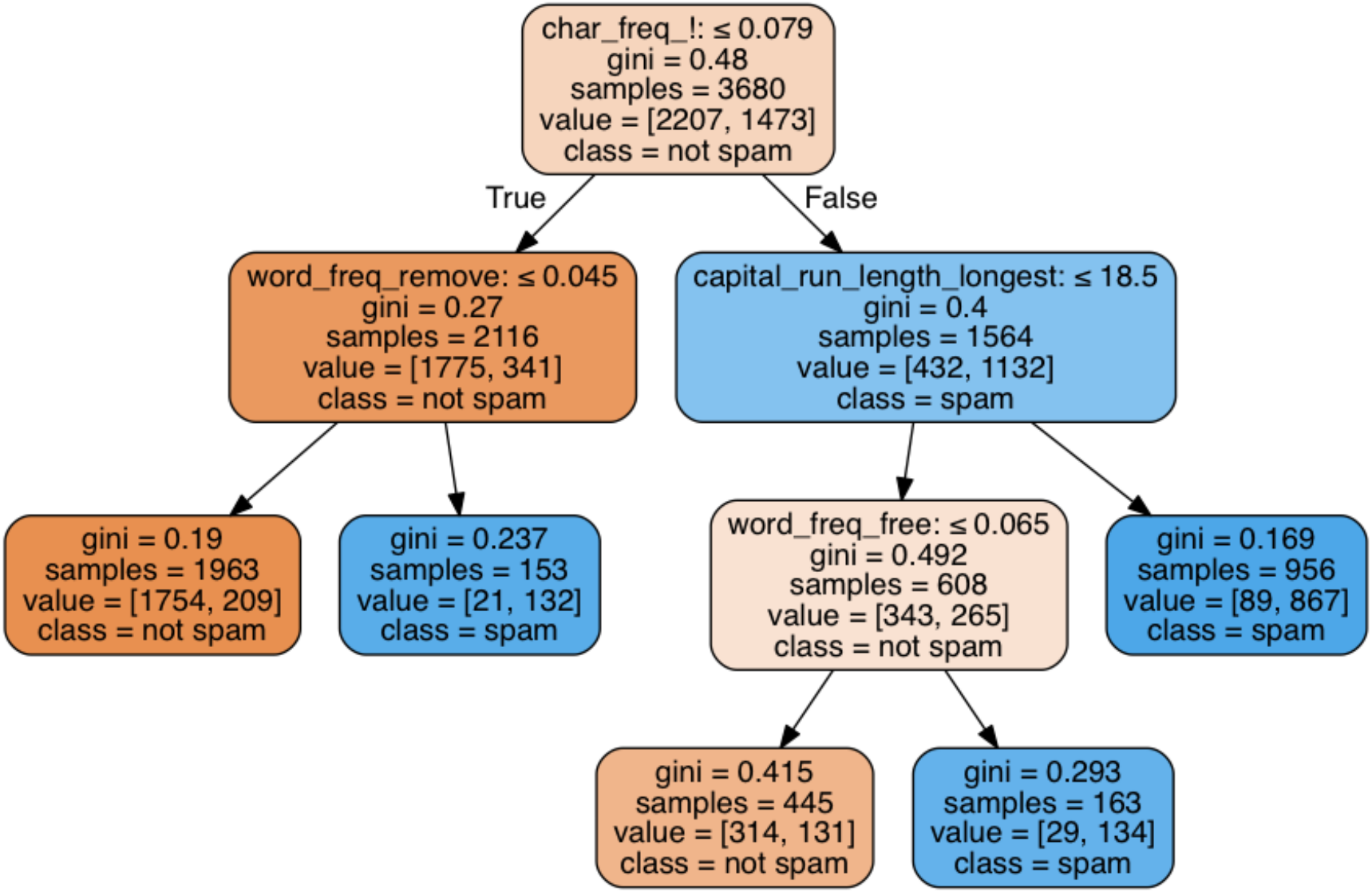
Statistical Learning

Outline

Splitting rules and decision trees

- For x_j a continuous variable: a splitting rule is of the form $x_j \geq t$ for $t \in \mathbb{R}$.
- For x_j a categorical variable: a splitting rule is of the form $x_i = a_j$ or $x_j \in \{a_j, b_j, c_j\}$ where a_j, b_j and c_j are possible values of x_j.
- We consider the simple class of predictors defined by (a succession of) splitting rules.
- The set of splitting rules used to segment the predictor space can be summarized in a tree.

Splitting rules and decision trees

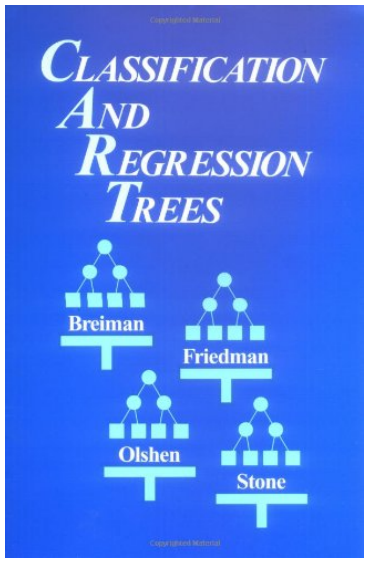


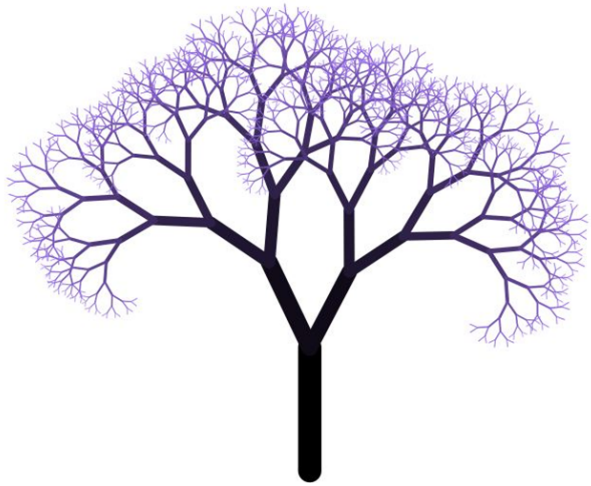
[Spam dataset (see further)]

Outline

CART: Classification and regression tree

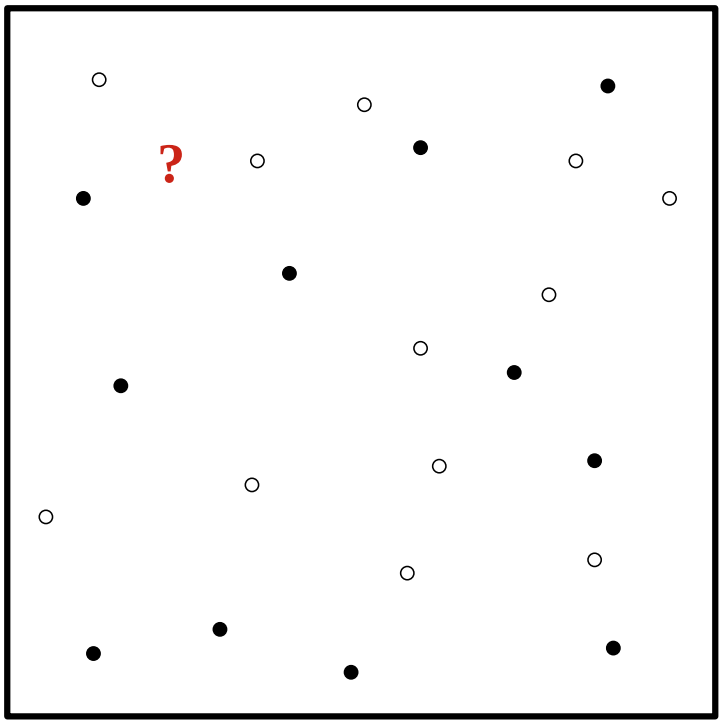
- CART : Classification And Regression Trees
- Introduced by Breiman et al. (1984)





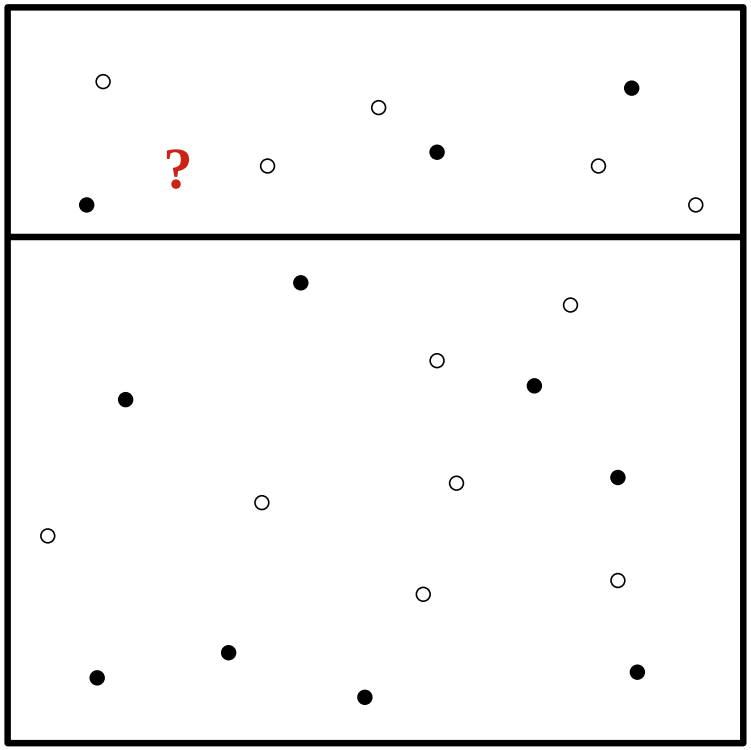
Binary tree for two continuous variables

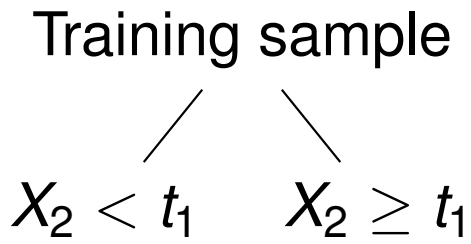
- Let's consider a classification problem with response Y and continuous inputs X_1 and X_2.



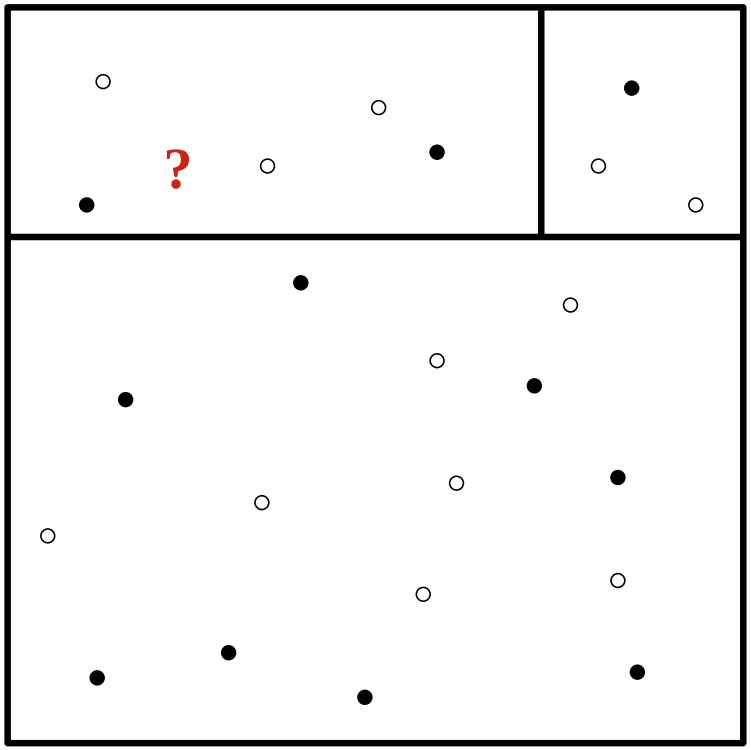
- We want to define a binary tree (associated to a sequence of splitting rules) for this classification problem.

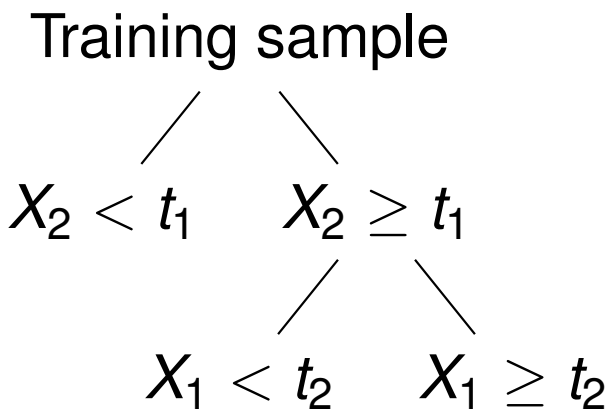
Binary tree for two continuous variables



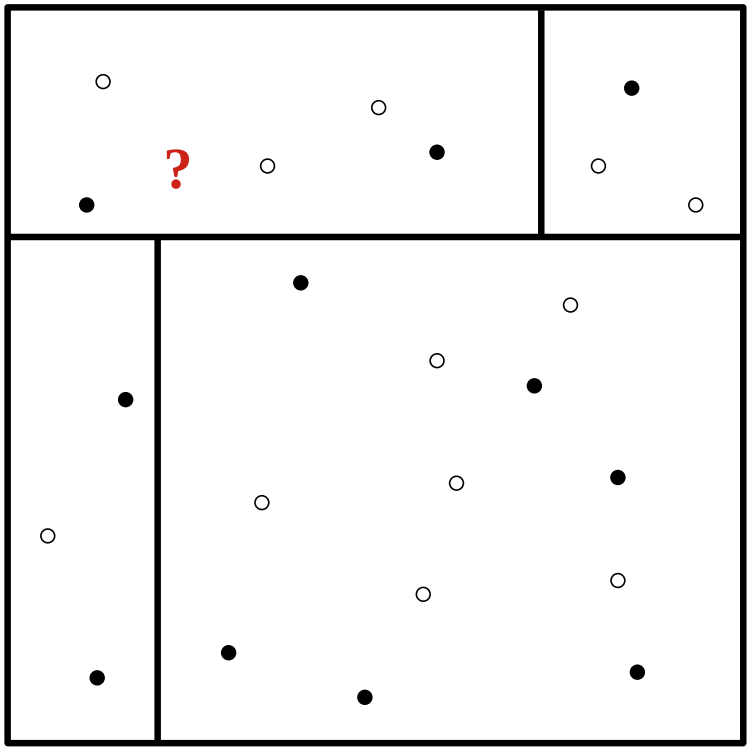


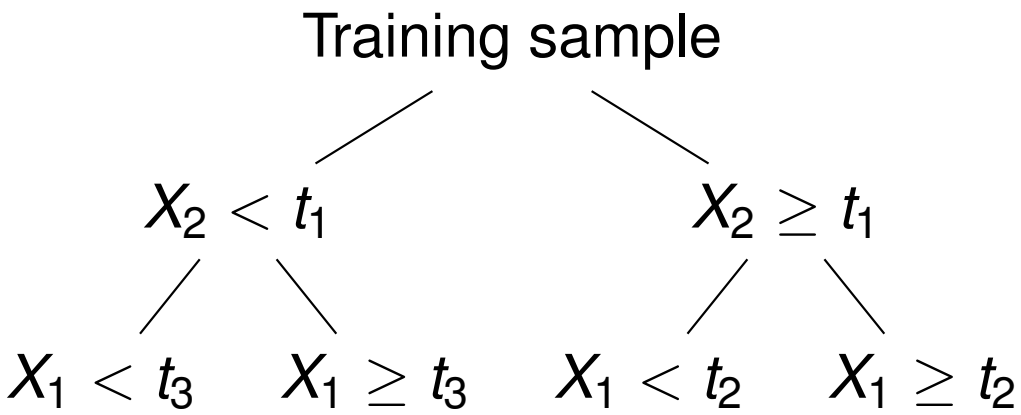
Binary tree for two continuous variables



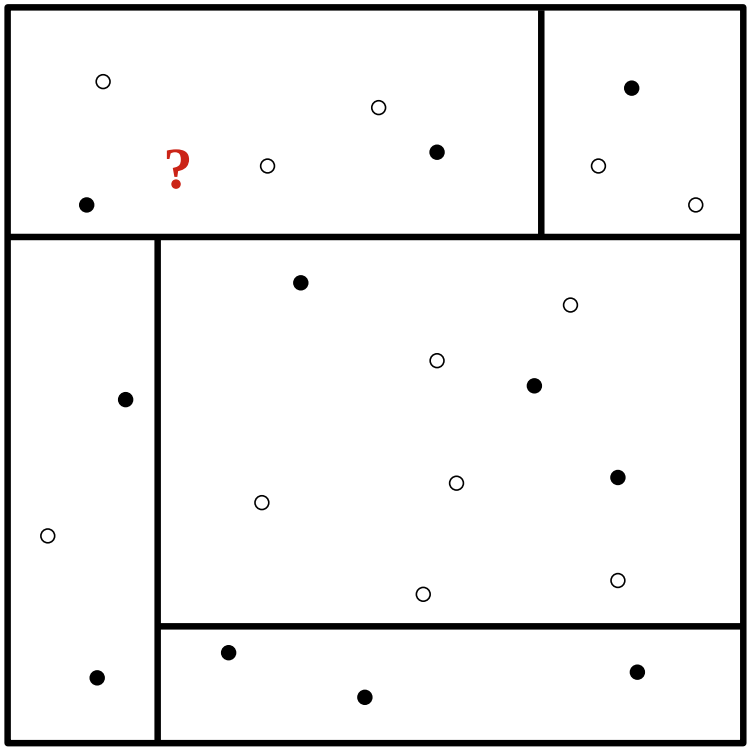


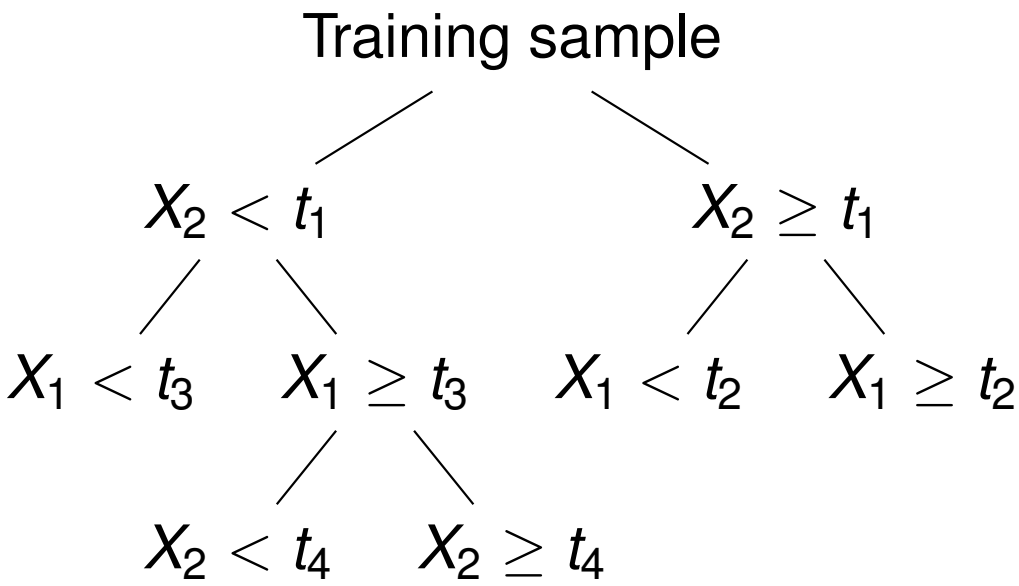
Binary tree for two continuous variables





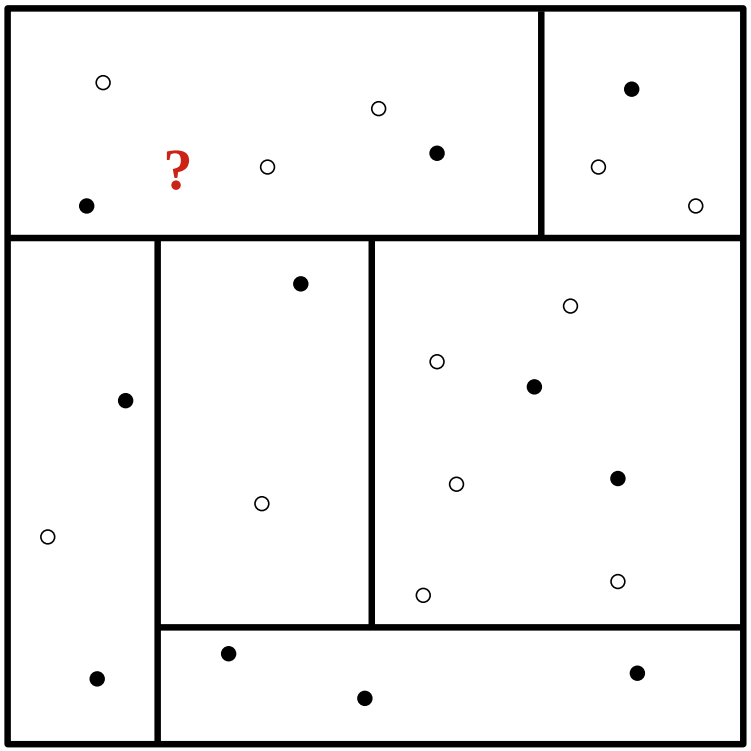
Binary tree for two continuous variables

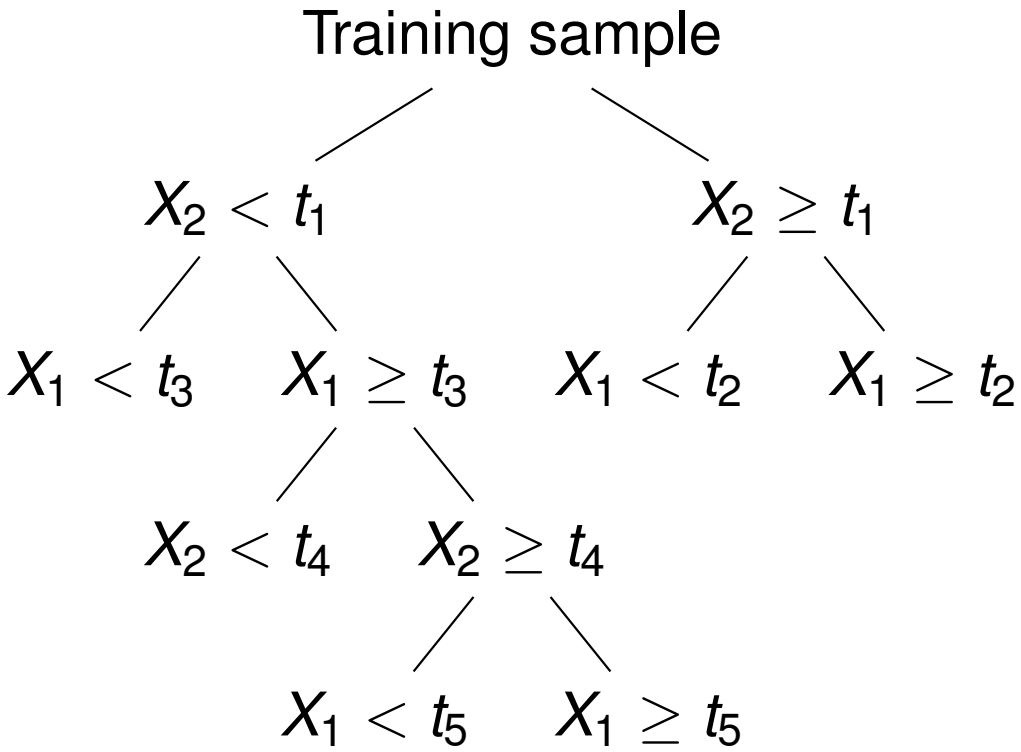




Binary tree for two continuous variables

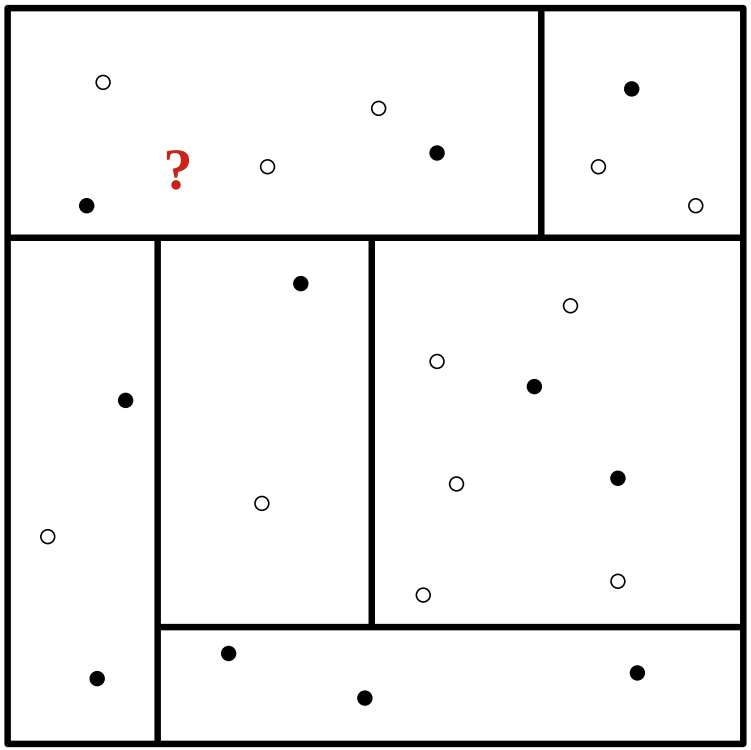
Here is one example of binary tree associated to a sequence of splitting rules:

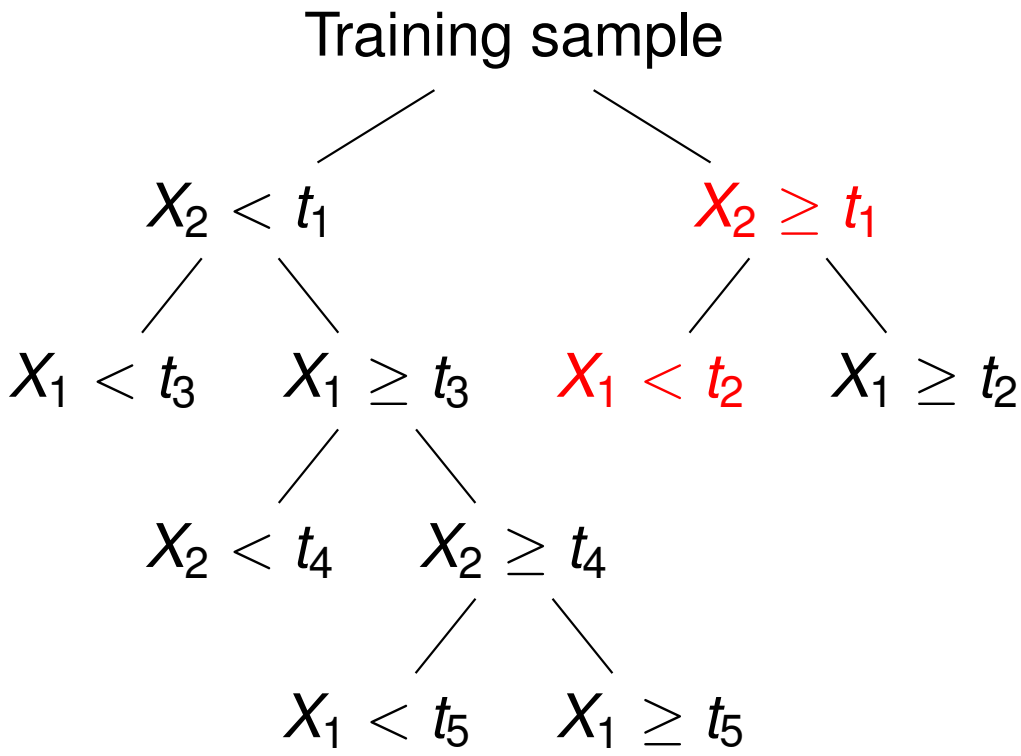




Recursive binary splitting and prediction rule

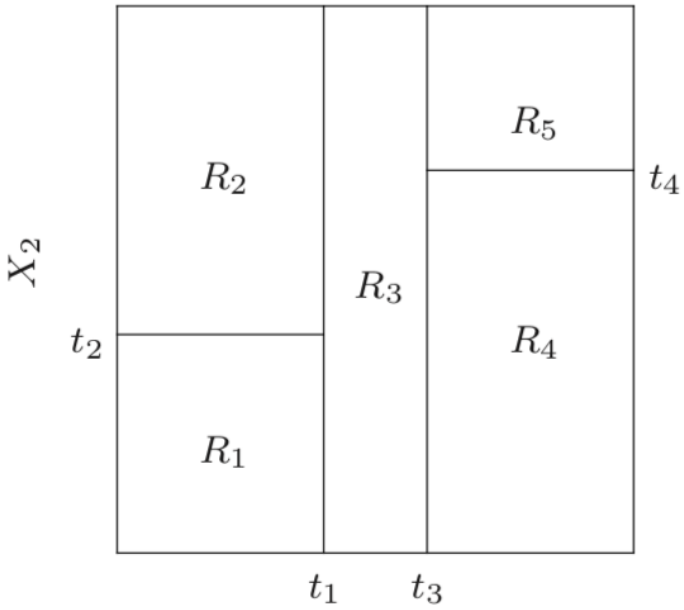
- How can we make prediction based on this decision tree ?
- A given query point x belongs to a region $R(x)$ of the partition.





- Based on the observations (X_i, Y_i)'s, we propose a prediction for $\hat{y}(x)$ which is based on the observations belonging to $R(x)$

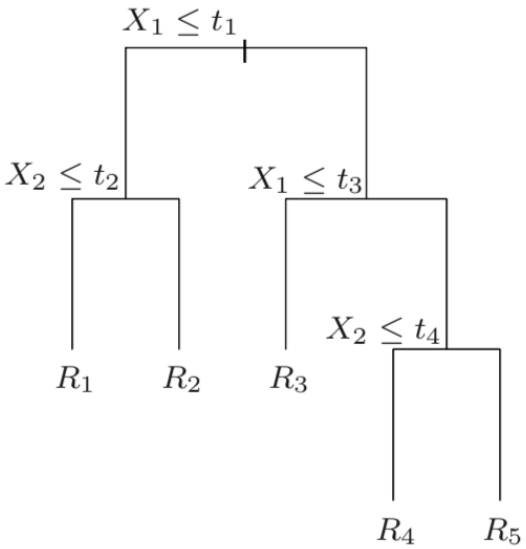
Recursive binary splitting and prediction rule

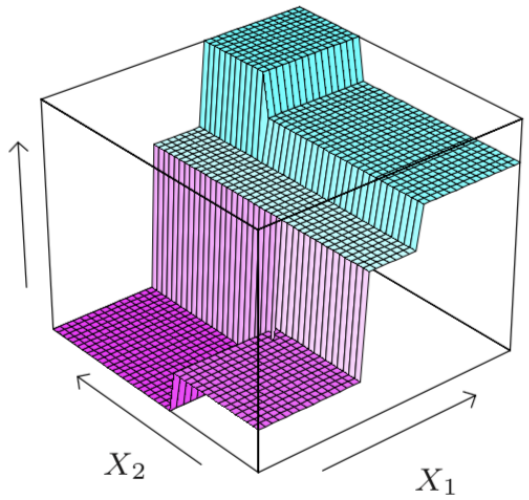


For both classification and regression we consider decision rules which are regressograms :

$$f(x) = \sum_m c_m \mathbb{1}_{x \in R_m}$$

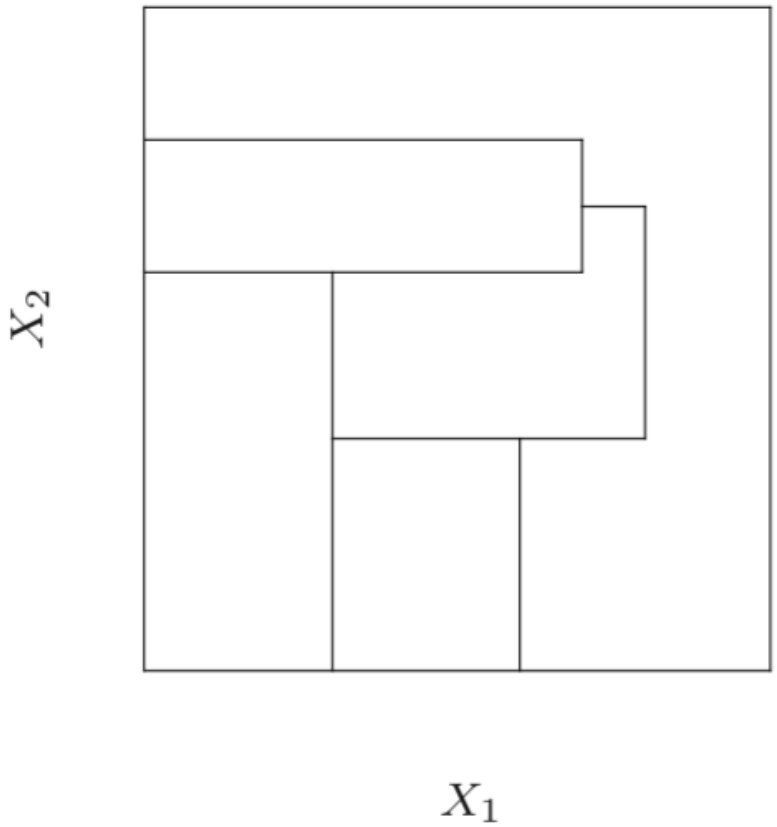
The regions R_m's of the partition correspond to terminal nodes (or leaves) in the binary tree.

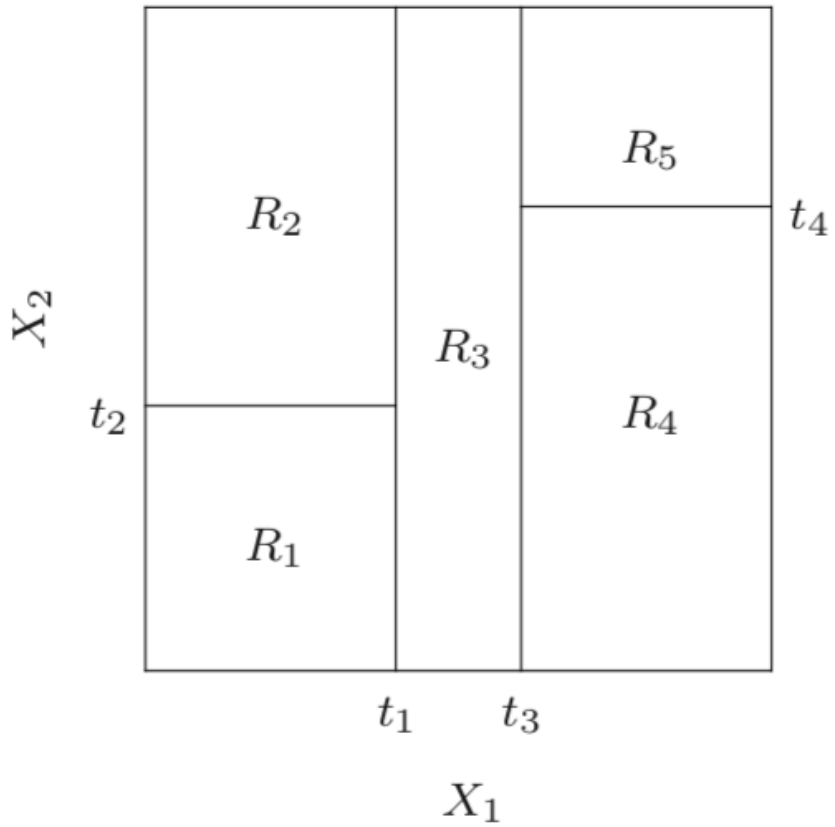




Partition and recursive binary splitting

All partitions of the feature space cannot be obtained by recursive binary splitting:





Outline

Binary tree for regression

- N observations (x_i, y_i), $i = 1, \ldots, n$.
- We need to decide on the splitting variables, on the split points and on the prediction associated the final tree.
- For a partition into M regions $R_1, \ldots, R_M$, response is given by the regressogram function

$$f(x) = \sum_m c_m \mathbb{1}_{x \in R_m}$$

- **Prediction:** for the quadratic loss, we know that we have to take for c_m

$$\hat{c}_m = \overline{\{y_i \mid x_i \in R_m\}}.$$

Binary tree for regression (step 0)

- Finding the best binary partition is generally computationally infeasible. We proceed with a **greedy algorithm**.
- **Step 0:**
  - Starting with all of the data, considering a splitting variable j and a split point t, we define the pair of half-planes

$$R_1(j,t) = \{x \,|, x_j \leq t\} \quad \text{and} \quad R_2(j,t) = \{x \,|, x_j > t\}.$$

  - We seek the "best" splitting variable j and "best" split point t:

$$\min_{j,t} \min_{c_1} \sum_{x_i \in R_1(j,t)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,t)} (y_i - c_2)^2$$
$$= \min_{j,t} \sum_{x_i \in R_1(j,t)} (y_i - \hat{c}_{j,t,1})^2 + \sum_{x_i \in R_2(j,t)} (y_i - \hat{c}_{j,t,2})^2,$$

where $\hat{c}_{j,s,1} = \overline{\{y_i \,|\, x_i \in R_1(j,t)\}}$ and $\hat{c}_{j,t,2} = \overline{\{y_i \,|\, x_i \in R_2(j,t)\}}$.

Binary tree for regression (step 0)

- For each candidate splitting variable X_j, the determination of the optimal split point t_j can be done very quickly because we only need to consider spit points in the set of observations $\{X_{i,j}\ ,\ i = 1 \ldots n\}$.

- By scanning through all of the inputs, determination of the best pair (j, t_j) is feasible.

- Having found the best split, we partition the data into the two resulting regions.

Binary tree for regression (recursion)

- The same process is repeated par recursion on the terminal nodes (each one corresponding to a regions R_m) :
  - For each region R_m we seek the "best" splitting variable j and the "best" split point t_j
  - We split the region R_m according to this best split.

$$\min_{j,s} \sum_{x_i \in R_1(m,j,s)} (y_i - \hat{c}_{m,j,s,1})^2 + \sum_{x_i \in R_2(m,j,s)} (y_i - \hat{c}_{m,j,s,2})^2$$

where

$$\hat{c}_{m,j,s,1} = \overline{\{y_i \,|\, x_i \in R_1(m,j,s)\}}$$

and

$$R_1(m,j,s) = \{x \in R_m | x_j \leq s\}.$$

- The splitting process is stopped only when some given criterion is reached (for instance minimum node size - say 5- is reached).

Binary tree for regression: node impurity

- In a binary tree T, we define the (quadratic) node impurity of the terminal node m as

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

where

$$N_m = |\{x_i \mid x_i \in R_m\}|$$

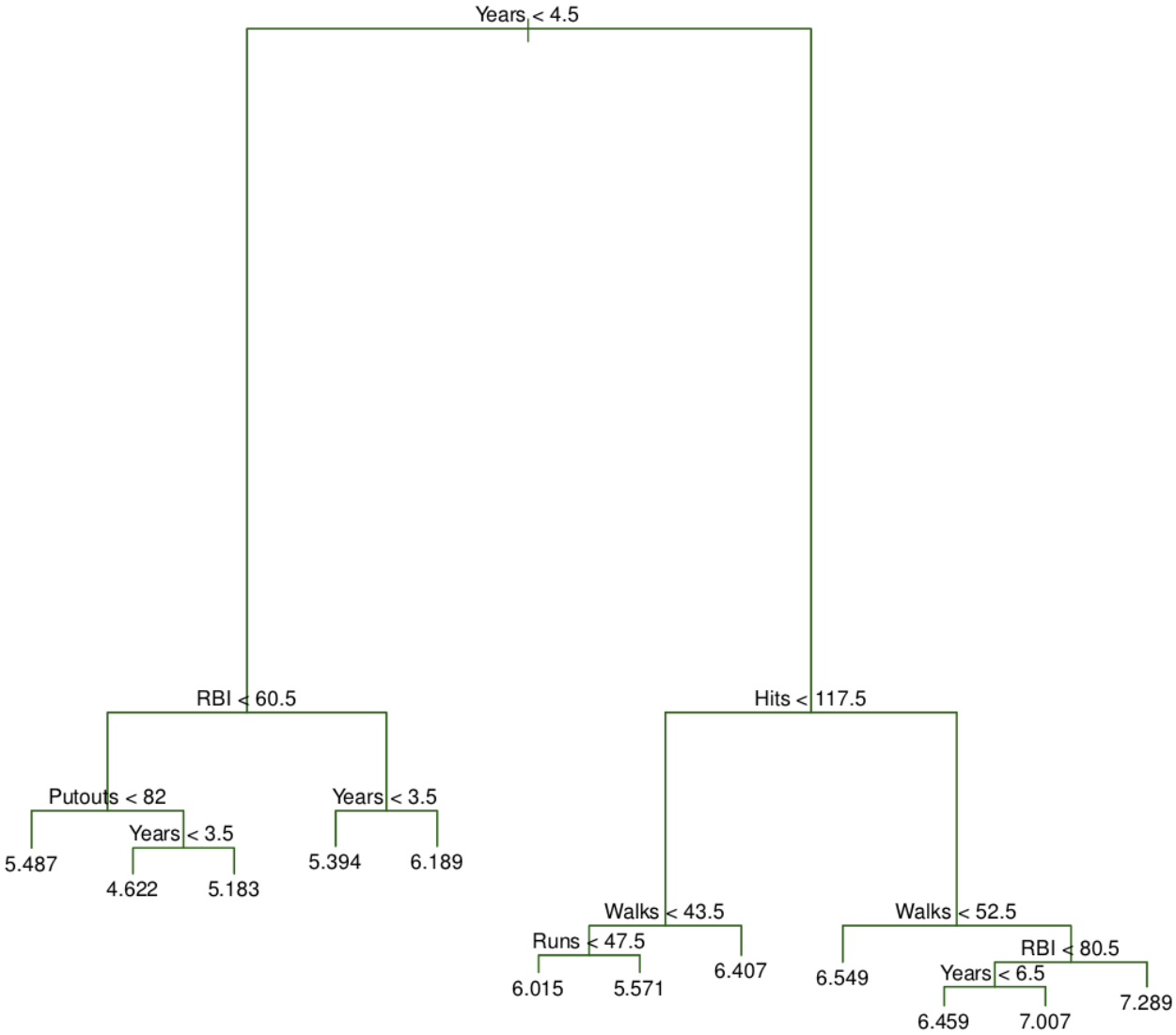
and

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i.$$

- At each step of the algorithm, we grow the binary tree by splitting a given terminal node m (with a splitting variable j and a split point t) into two child nodes m_L and m_R that minimizes

$$N_{m_L} Q_{m_L}(T) + N_{m_R} Q_{m_R}(T).$$

Regression Tree: baseball example



[Source: EOSL slides]

Outline

Binary tree for classification

- Target variable is now a categorical variable Y with values in $\{1, \ldots, K\}$.
- In a node m, representing a region R_m with N_m observations, the proportion of class k is

$$\hat{p}_{mk} := \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{1}_{y_i = k}.$$

- **Prediction** for node m (majority rule):

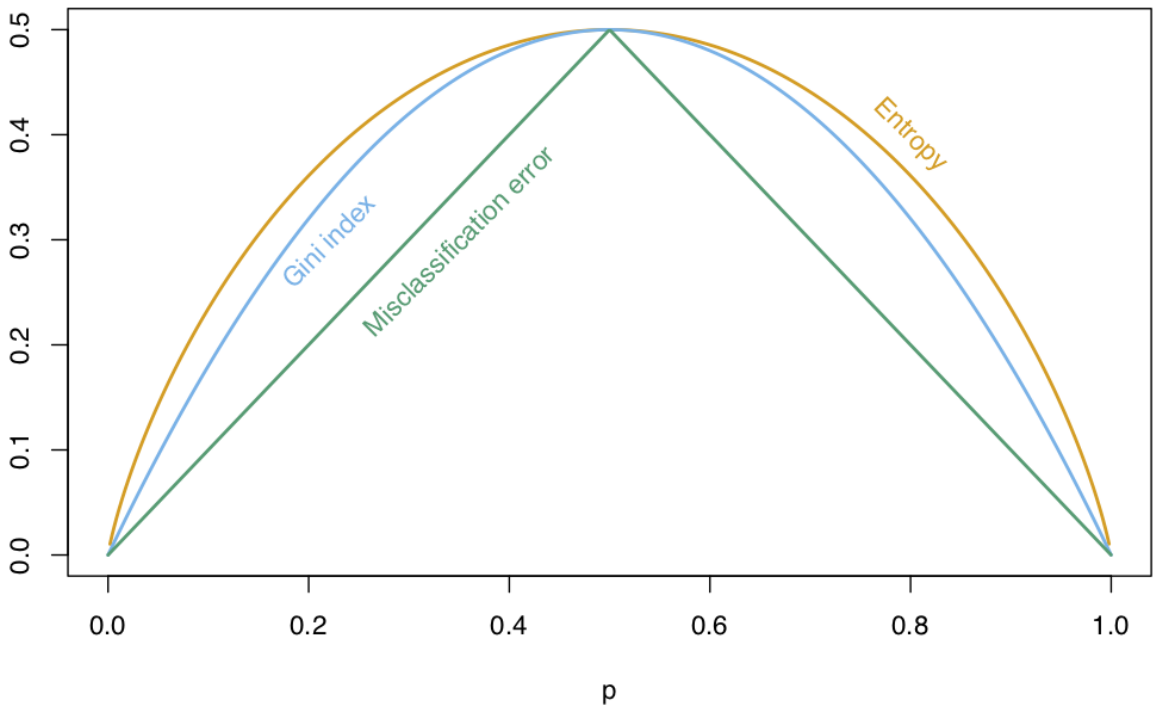
$$\hat{k}(m) = \arg\max_k \hat{p}_{mk}.$$

- Same algorithm as before, we only need to adapt the definition of node impurity.

Binary tree for classification: node impurity

Measures of node impurity for node m:

- Misclassification error: $Q_m(T) = 1 - p_{m\hat{k}(m)}$
- Gini index: $Q_m(T) = \sum_k \hat{p}_{mk}(1 - \hat{p}_{mk})$
- Cross-entropy or deviance: $Q_m(T) = -\sum_k \hat{p}_{mk} \log \hat{p}_{mk}$



[Node impurity for **binary** classification (K = 2). Source: EOSL]

Binary tree for classification: node impurity

- Cross-entropy and the Gini index are more sensitive to changes in the node probabilities than the misclassification rate.
- Example : In a two-class problem with 400 observations in each class, suppose one split creates nodes (300, 100) and (100, 300), while the other created nodes (200, 400) and (200, 0).
  - Both splits produce a misclassification rate of 0.25.
  - Second split produces a pure node and is probably preferable.
  - Both the Gini index and cross-entropy are lower for the second split.

CART: Pros and Cons

- Pros:
  - Trees are very easy to explain (in particular to non experts): they can be displayed graphically, and are easily interpreted
  - Trees can easily handle qualitative predictors.
  - Small computational cost.
- Cons:
  - Trees are not competitive with many others supervised learning approaches in terms of prediction accuracy.
  - Local splitting.
  - Lack of robustness: trees are sensitive to outliers.
- By aggregating many decision trees, the predictive performance of trees can be substantially improved: boosting and Random Forests.

Outline

Outline

Bootstrap

- The basic idea of Bootstrap is to randomly draw datasets with replacement from the training data, each sample the same size as the original training set.
- This is done B times, producing B bootstrap datasets.
- Bootstrap can be used for assessing the accuracy of a parameter estimate or a prediction.
- Bootstrap can also be used to improve the prediction itself.

Bootstrap

- Suppose that we fit a prediction rule to our training data $Z = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.
- Let $\hat{f}(x)$ be a prediction at input x (using Z).
- Let $Z_b^\star$ be a bootstrap sample: $Z_b^\star$ is composed of n pairs (x_I, y_I) where I is sampled i.i.d uniformly in $\{1, \ldots, n\}$.
- Each bootstrap sample gives a prediction $\hat{f}_b^\star(x)$.

Bagging for regression

- **Bagging** is an **ensemble method**.
- For regression, **Bootstrap aggregation** or **bagging** averages the predictions:

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

- The bagging estimate is a Monte Carlo approximation of $\mathbb{E}^{\star}\hat{f}^{\star}(x)$ where the expectation is taken under the bootstrap distribution, conditionally to Z.
- Key idea: averaging the bootstrap samples tends to reduce the variance of the prediction.
- Bagging can reduce the variance of unstable procedures like trees, leading to improved prediction.

Consensus bagging for classification (K classes)

- Let $\hat{f}(x) = (\mathbb{1}_{\hat{G}(x)=1}, \ldots, \mathbb{1}_{\hat{G}(x)=K})$ be the **one hot encoding** of the prediction $\hat{G}(x)$ given by tree classifier at x:

$$\hat{G}(x) = \arg \max_{k=1\ldots K} \hat{f}(x).$$

- We consider the bagged quantity

$$\hat{f}^{\star}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{\star}_{b}(x) = (p_1(x), \ldots, p_K(x))$$

  which is the K-vector of the proportions of (bootstrap) trees predicting class k at x.
- The **consensus bagged classifier** selects the class with the most votes from the B trees

$$\hat{G}_{\text{bag}}(x) = \arg \max_{k=1\ldots K} \hat{f}_{\text{bag}}(x).$$

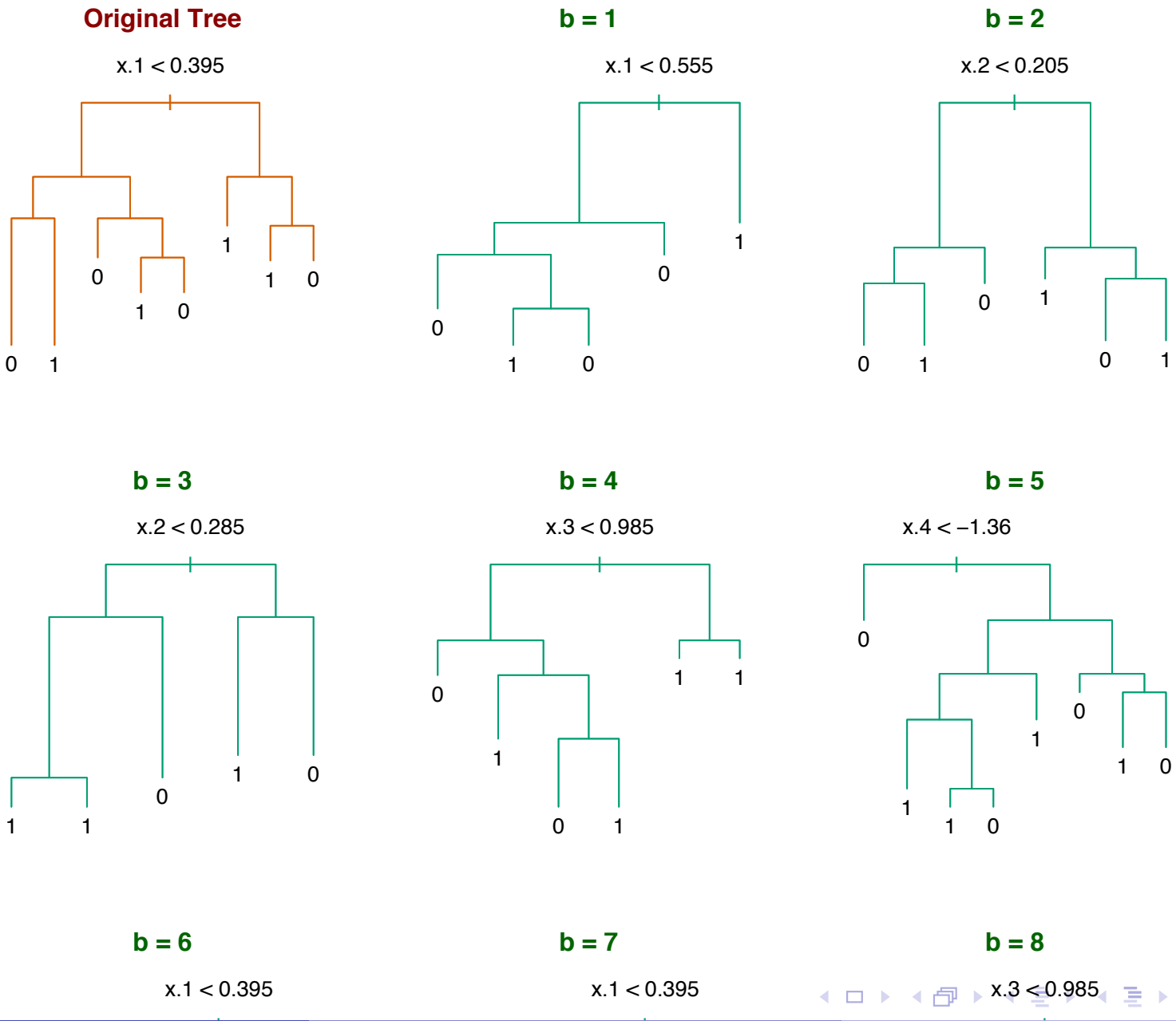
- ⚠ the $p_k(x)$ do not estimate the class-probability at x !

Bagging: simulated example

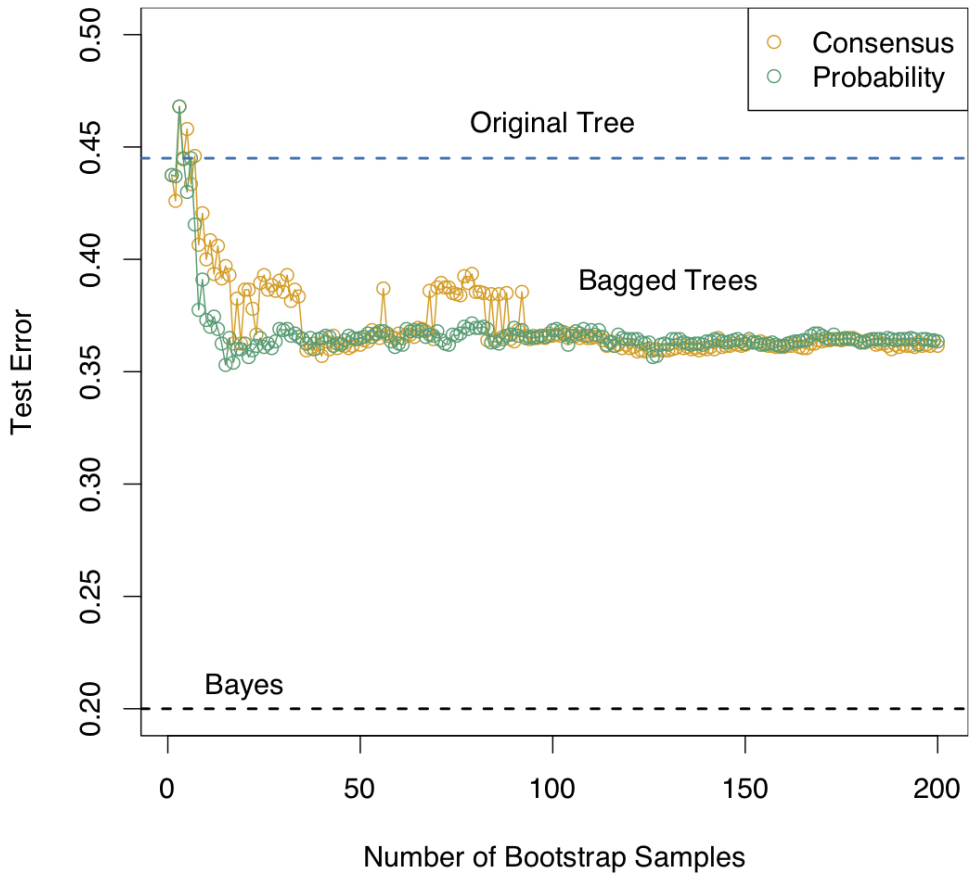
Simulated example from EOSL:

- A sample of size n = 30, with two classes.
- $p = 5$ features, each having a standard Gaussian distribution with pairwise correlation 0.95.
- Response Y generated according to $P(Y = 0|x_1 > 0.5) = P(Y = 1|x_1 \leq 0.5) = 0.8$.
- The Bayes error is 0.2.
- We fit classification trees to the training sample and to each of 200 bootstrap samples.
- No pruning for the bootstrap trees.
- A large test sample is generated from the same population to estimate the generalization error.

Bagging: simulated example



Bagging: simulated example (continue)



[Source: EOSL]

Bagging reduces the variance

- General idea: for $Z_1, \ldots, Z_n$ i.i.d. :

$$\text{Var}\left(\frac{1}{n}\sum_{i=1}^{n} Z_i\right) = \frac{1}{n}\text{Var}(Z_1).$$

- Bagging is interesting for predictors with high variance but low bias: relevant for fully developed trees (without any pruning).
- Bagging a regression tree generally improves the prediction performances.
- **Wisdom of Crowds**: Bagging a not too bad classifier can make it better.
- But bagging a bad classifier can make it worse !

Outline

From Bagged trees to Random Forests

- Take-home message for Bagging:
  - Bagging works especially well for high-variance, low-bias procedures, such as trees.
  - Improvement with bagging is through variance reduction.
  - For regression: fit regression trees for bootstrap samples and average the result.
  - For classification: majority vote for the predicted class.
- **Random forests** (RF) is a substantial modification of bagging that builds a large collection of **de-correlated trees**, and then averages them.
- For many problems the performance of random forests is very similar to boosting.
- Random forests are popular, and are implemented in a variety of packages (R, python ...)

RF improve the variance reduction of bagging

- Pairs of (predictions) of bagged trees are correlated.
- For B i.d. random variables (identically distributed, but not necessarily independent), each with variance σ^2, with positive pairwise correlation ρ, the variance of the average is

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

- The correlation of pairs of bagged trees limits the benefits of averaging.
- How to reduce the correlation between the trees in bagged trees, without increasing the variance too much ?
- RF improve the variance reduction of bagging with adding some randomness:

Before each split, select $m \leq p$ of the input variables at random as candidates for splitting.

Random Forest Algorithm

Algorithm 1 Random Forest algorithm.

for $b = 1$ to B **do**

1. Draw a bootstrap sample $Z^\star$ of size n from the training data.
2. Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{\min}$ is reached:
   1. Select m variables at random from the p variables.
   2. Pick the best variable/split-point among the m.
   3. Split the node into two daughter nodes.

end for

Output the ensemble of trees $\{T_b\}_{b=1\ldots B}$

To make a prediction at a new point x:

- Regression: **average** the tree predictions.
- Classification: **majority vote** over the class predictions of the trees.

Illustration: Spam Data Set [EOSL]

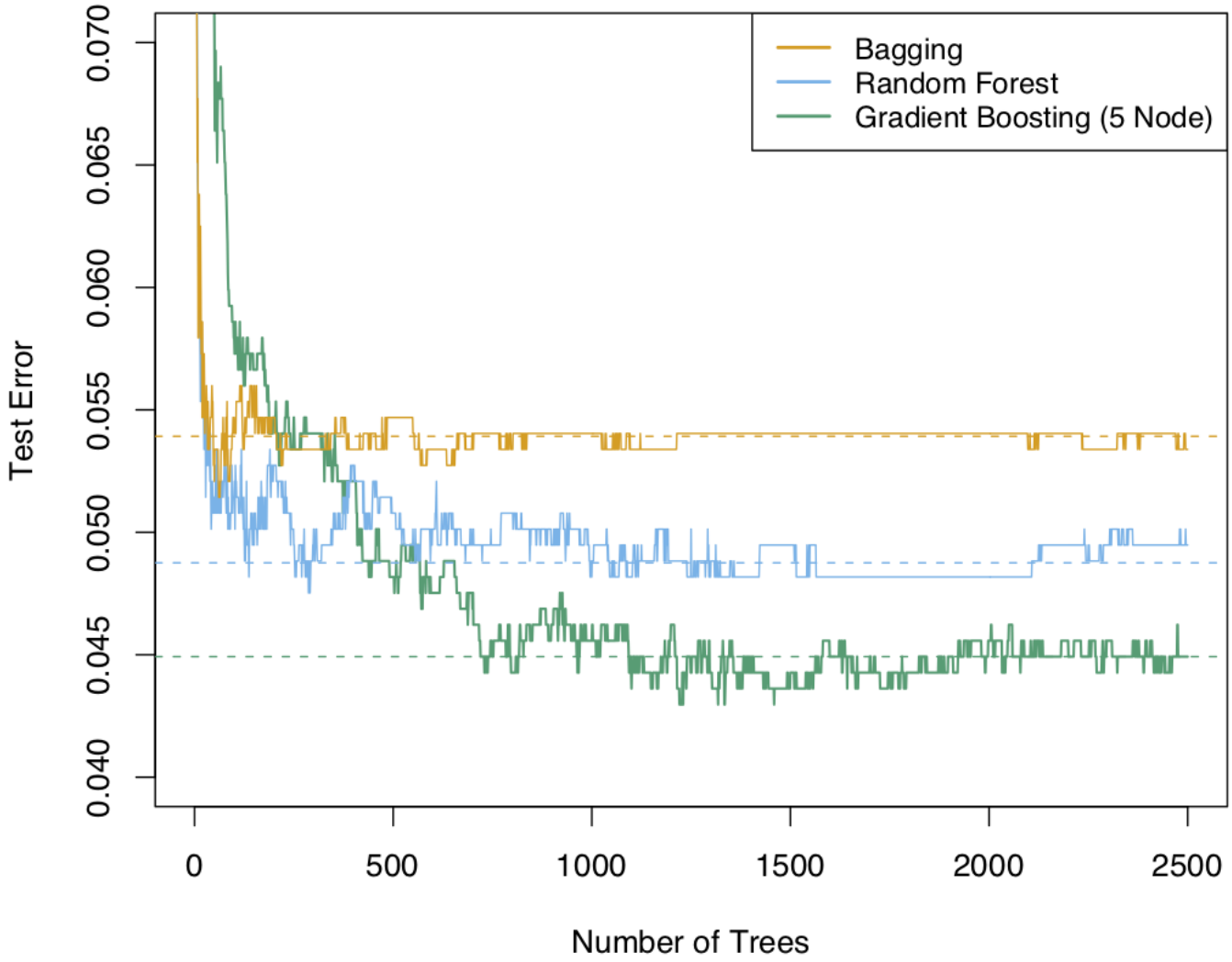
- The data consists of information from 4601 email messages.
- Classification task: predict whether the email was "spam" or "not spam".
- For the data we have the true outcome : email or spam is available, along with the relative frequencies of 57 of the most commonly occurring words and punctuation marks in the email message.

TABLE 1.1. *Average percentage of words or characters in an email message equal to the indicated word or character. We have chosen the words and characters showing the largest difference between* `spam` *and* `email`.

	george	you	your	hp	free	hpl	!	our	re	edu	remove
spam	0.00	2.26	1.38	0.02	0.52	0.01	0.51	0.51	0.13	0.01	0.28
email	1.27	1.27	0.44	0.90	0.07	0.43	0.11	0.18	0.42	0.29	0.01

[Source: EOSL]

Illustration: Spam Data Set [EOSL]



[Source: EOSL]

Random Forest parameters

- Number of trees: need to be large enough (a few hundreds).
- the number of variables picked at random for each node:
  - for regression: def. value is $m = p/3$.
  - for classification : $m = \sqrt{p}$.
- Minimum node size :
  - for classification: def. value is one
  - for regression : def. value is five.

Of course these parameters should be treated as tuning parameters in practice.

Out of bag samples

- All the observations are not represented in the bootstrap samples.
- Let $\hat{f}$ be the RF predictor.
- Out-of-bag (OOB) error, also called out-of-bag estimate :
  - For each observation $z_i = (x_i, y_i)$, construct its random forest predictor $\hat{f}_i$ by aggregating (average or majority vote) only those trees corresponding to bootstrap samples in which z_i did not appear.
  - Evaluate the error for this prediction: $\mathcal{R}_{\text{oob}}(\hat{f}, z_i) = \ell(\hat{f}_i(x_i), y_i)$
  - Average all the oob errors: $\mathcal{R}_{\text{oob}}(\hat{f}) = \frac{1}{n} \sum_{i=1 \ldots n} \mathcal{R}_{\text{oob}}(\hat{f}, z_i)$.
- Out-of-bag estimates avoid the need for an independent validation dataset.

Variable Importances

- **Splitting Importance Variable.** At each split in each tree, the improvement in the split-criterion is the importance measure attributed to the splitting variable, and is accumulated over all the trees in the forest separately for each variable.
- **Permutation importance.** Values for the jth variable are randomly permuted in the oob samples. The oob error is computed again and compared to the original oob error:

$$\mathcal{R}_{\text{oob},j}(\hat{f}) - \mathcal{R}_{\text{oob}}(\hat{f})$$

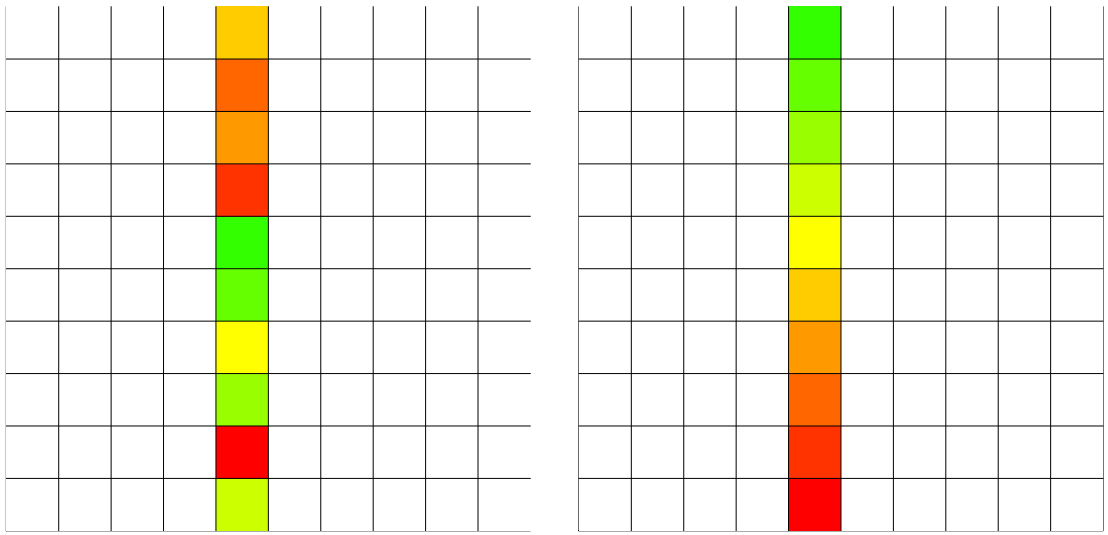
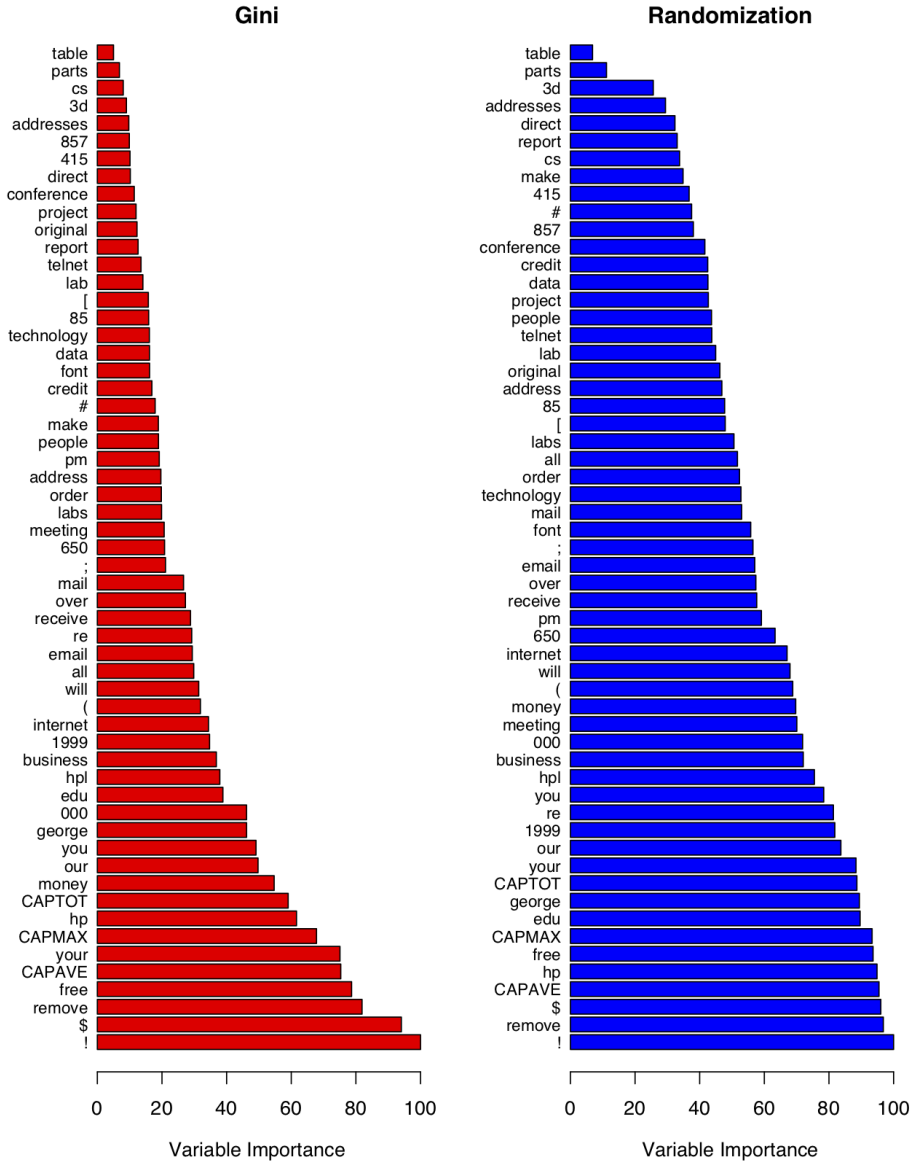


Illustration: Spam Data Set [EOSL]



[Source: EOSL]

Outline

Boosting

- Like Bagging, Boosting is a family of machine learning algorithms that combine weak learners to define strong learners: **ensemble method**.
- Contrary to Bagging, Boosting defines a strong predictor iteratively.
- As for Bagging, Boosting is a general approach that can be applied to many statistical learning (regression, classification ...)

Weak learners and strong learners

- **Weak learners:**
  - Each weak learner is a (simple) prediction function $h : \mathbb{R}^d \mapsto \mathbb{R}$ or $\{-1, 1\}$.
  - Examples : regression trees and classification trees with small depth, regression linear models with only a few variables, logistic regression with only a few variables.
- **Strong learners:**
  - Boosting is a way of fitting an additive expansion in a set of elementary "basis" functions:

$$f^{(B)}(x) = \sum_{b=1}^{B} \beta^{(b)} h^{(b)}(x)$$

    where the $h^{(b)}$ are weak learners in a given set $\mathcal{H}$.
  - The prediction at x is given by
    - ★ $f^{(B)}(x)$ for regression
    - ★ $H^{(B)}(x) := \text{sign}(f^{(B)}(x))$ for classification.

Forward Stagewise additive modeling

For some lost ℓ and a training set $(x_i, y_i)_{i=1,\dots,n}$ we want to solve:

$$\arg \min_{\beta^{(1)},\dots\beta^{(B)}\in\mathbb{R},\, h^{(1)},\dots,h^{(B)}\in\mathcal{H}} \quad \sum_{i=1}^{n} \ell\left(y_i, \sum_{b=1}^{B} \beta^{(b)} h^{(b)}(x_i)\right)$$

In general, this is is a difficult optimization problem. Alternative :

Algorithm 2 Forward Stagewise Additive Modeling Algorithm.

Initialize $f^{(0)}(x) = 0$.

for $b = 1$ to B **do**

1. Compute

$$(\beta^{(b)} h^{(b)}) = \arg \min_{\beta\in\mathbb{R},\, h\in\mathcal{H}} \quad \sum_{i=1}^{n} \ell\left(y_i, f^{(b-1)}(x_i) + \beta h(x_i)\right)$$

2. Set $f^{(b)}(x) = f^{(b-1)}(x) + \beta^{(b)} h^{(b)}(x)$.

end for

Adaboost and Gradient Boosting

- We want to fit an additive model in a forward stagewise manner:
  1. Compute

$$(\beta^{(b)} h^{(b)}) = \arg \min_{\beta \in \mathbb{R},\, h \in \mathcal{H}} \sum_{i=1}^{n} \ell \left(y_i, f^{(b-1)}(x_i) + \beta h(x_i) \right)$$

  2. Set $f^{(b)}(x) = f^{(b-1)}(x) + \beta^{(b)} h^{(b)}(x)$.
- AdaBoost: by reweighing the data points.
- Gradient Boosting: gradient descent approach.

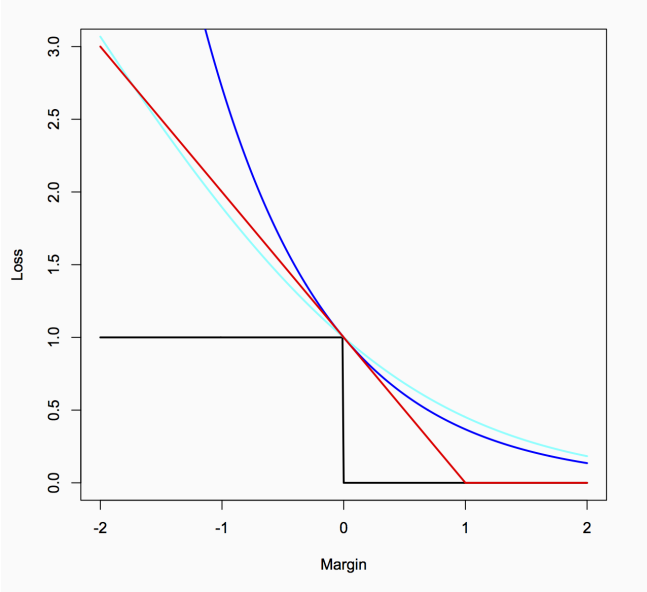
From Adaboost to XGBoost

- Adaboost: the first successful boosting algorithm [Freund et al., 1996, Freund and Schapire, 1997]
- Formulate Adaboost as gradient descent with exponential loss function[Breiman et al., 1998, Breiman, 1999]
- Generalize Adaboost to Gradient Boosting in order to handle a variety of loss functions [Friedman et al., 2000, Friedman, 2001]
- XGBoost : a scalable implementation of gradient tree boosting system widely recognized for its outstanding results in numerous data challenges [Chen and Guestrin, 2016].

Outline

Exponential loss

- Classification framework, observations $(x_i, y_i)_{i=1\dots n}$.
- Exponential loss: $\rho(u) = \exp(-u)$.
- Surrogate risk:
  $\mathcal{R}^\rho(f) := \mathbb{E}\rho(Yf(X))$.
- Surrogate empirical risk:
  $\hat{\mathcal{R}}_n^\rho(f) := \frac{1}{n}\sum_{i=1}^n \rho(Y_i f(X_i))$



- Let $f_\rho^\star \in \operatorname{argmin}_{f\in\mathcal{F}} \mathcal{R}^\rho(f)$ and $\hat{f}_\rho \in \operatorname{argmin}_{f\in\mathcal{F}} \hat{\mathcal{R}}_n^\rho(f)$.
- Motivation for introducing the exponential loss:
  - Convex loss makes the optimization easier.
  - The sign of the target function $f_\rho^\star$ (approximated by $\hat{f}_\rho$) is equal to the sign of the Bayes predictor (for the misclassification loss).

Adaboost: two presentations

- Adaboost = Forward Stagewise Algorithm with exponential loss.
- AdaBoost = gives more and more weight to the data points x_i, which are wrongly classified at the stage b.

Weighted misclassification error

- Classification framework: Adaboost is an ensemble method for weak classifiers $h \in \mathcal{H}$, $h : \mathbb{R}^p \mapsto \{-1, 1\}$.
- For a family of positive weights ω_i, $i = 1 \ldots n$, and $h \in \mathcal{H}$, we define the weighted misclassification error:

$$\text{err}(\omega, h) := \frac{\sum_{i=1}^{n} \omega_i . \mathbb{1}_{h(x_i) \neq y_i}}{\sum_{i=1}^{n} \omega_i}$$

Adaboost Classifier

Algorithm 3 Adaboost Algorithm.

1: Initialize the observation weights $\omega_i^{(1)} = 1/n$, $i = 1 \dots n$.
2: **for** $b = 1$ to B **do**
3: Find

$$h^{(b)} \in \arg\min_{h \in \mathcal{H}} \text{err}(\omega^{(b)}, h)$$

4: Compute

$$\beta^{(b)} := \frac{1}{2} \log \left(\frac{1 - \text{err}(\omega^{(b)}, h^{(b)}))}{\text{err}(\omega^{(b)}, h^{(b)})} \right)$$

5: Update the weights: for $i = 1 \dots n$:

$$\omega_i^{(b+1)} = \omega_i^{(b)} \exp \left(2\beta^{(b)} \mathbb{1}_{h^{(b)}(x_i) \neq y_i} - \beta^{(b)} \right).$$

6: **end for**
7: Output the classifier $H^{(B)}(x) := \text{sign}(\sum_{b=1}^{B} \beta^{(b)} h^{(b)}(x))$.

Adaboost: some remarks (1)

$$\beta^{(b)} = \frac{1}{2} \log \left(\frac{1 - \text{err}(\omega^{(b)}, h^{(b)}))}{\text{err}(\omega^{(b)}, h^{(b)})} \right)$$

and

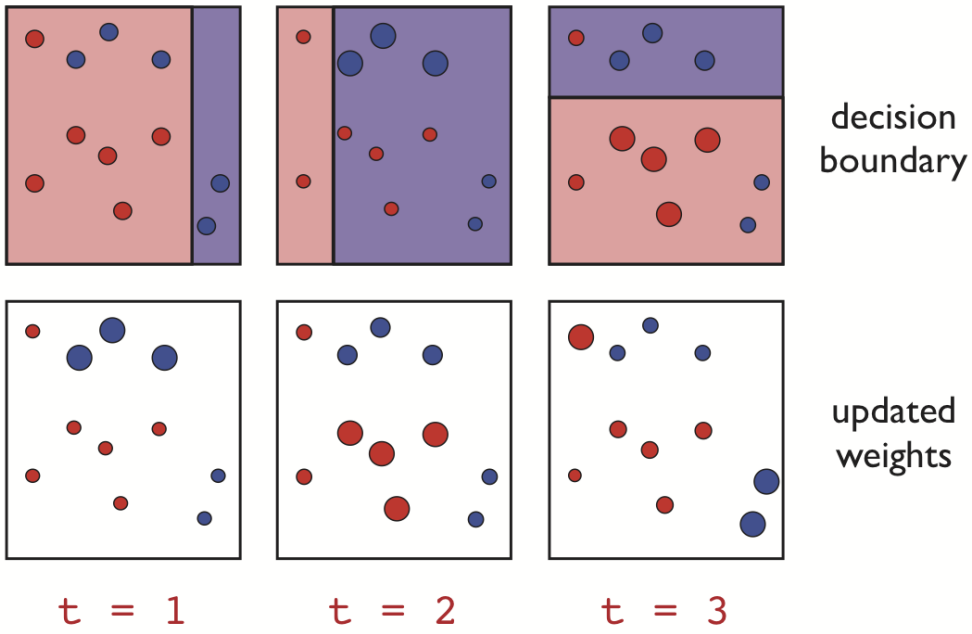
$$\omega_i^{(b+1)} = \omega_i^{(b)} \exp \left(2\beta^{(b)} \mathbb{1}_{h^{(b)}(x_i) \neq y_i} - \beta^{(b)} \right)$$

- $\beta^{(b)} > 0$ iif err < 0.5 :
  - For any classifier with accuracy higher than 50%, the weight $\beta^{(b)}$ is positive.
  - For a classifier with less than 50% accuracy, the weight $\beta^{(b)}$ is negative : we combine its prediction by flipping the sign.
- $\omega_i^{(b+1)} = \omega_i^{(b)} \exp(\pm\beta^{(b)})$: $\omega_i^{(b+1)}$ is larger than $\omega_i^{(b)}$ for misclassified points (if $\beta^{(b)} > 0$).
- At step b, if $\text{err}(\omega^{(b)}, h^{(b)}) = 0$ then the algorithm will stop and output $H_b(x)$.

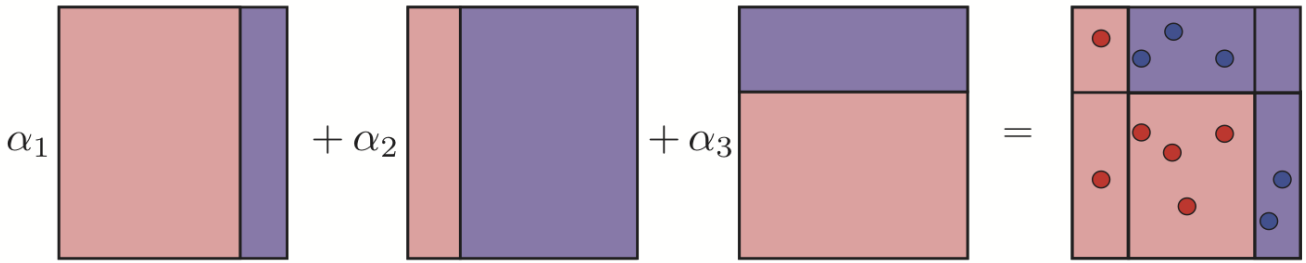
Adaboost: some remarks (2)

- Unlike fitting a single complex predictor to the data, which amounts to fitting the data hard and potentially overfitting, Boosting learns slowly.
- Adaboost is also Forward Stagewise Algorithm with exponential loss.
- Several versions of Adaboost, in particular for multiclass classification and for regression.
- The term Boosting generally refers to Adaboost.
- Adaboost with trees: at each step we fit a regression or classification tree (Line 3) with small depth, for instance a "decision stump" (with only two terminal nodes)

Illustration: weights update with Adaboost



(a)



(b)

[Source: Foundations of Machine Learning.]

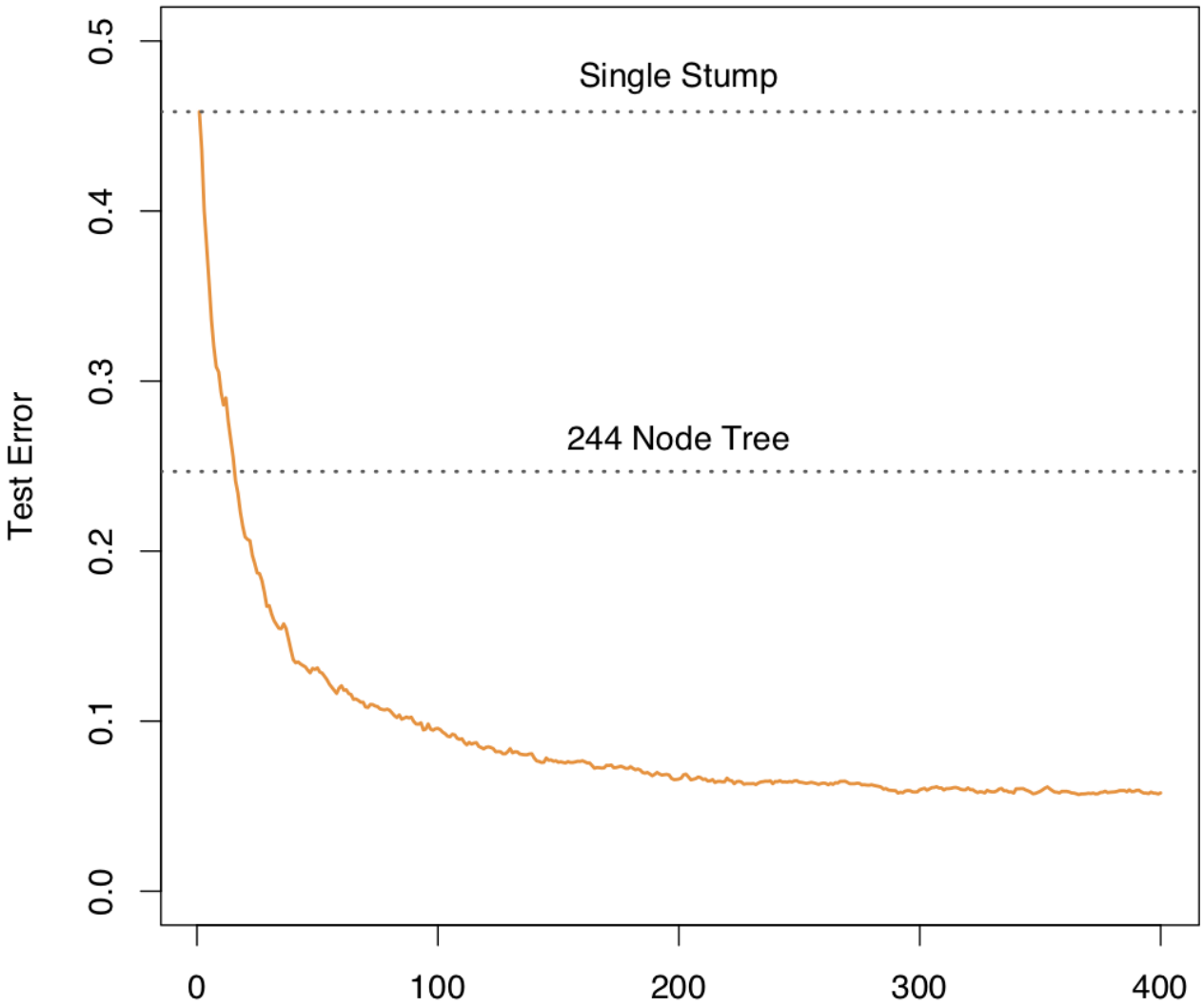
Adaboost on simulated data [EOSL p.339]

- Generative model:
  - Features: $X_1, \ldots, X_{10}$ are standard independent Gaussian variables.
  - Target variable Y is defined by

$$Y = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_j^2 > \text{median}(\chi^2_{10}) = 9.34 \\ -1 & \text{otherwise} \end{cases}$$

- Adaboost:
  - weak classifiers are "decision stumps".
  - classes are approximately balanced.
  - 2 000 training observations.
  - 10 000 test observations.

Adaboost on simulated data [EOSL p.339]



[Source: [EOSL]

Outline

Forward stagewise and gradient descent

- We want to fit an additive model in a forward stagewise manner:
  1. Compute

$$(\beta^{(b)} h^{(b)}) = \arg \min_{\beta \in \mathbb{R},\, h \in \mathcal{H}} \sum_{i=1}^{n} \ell \left(y_i, f^{(b-1)}(x_i) + \beta h(x_i) \right)$$

  2. Set $f^{(b)}(x) = f^{(b-1)}(x) + \beta^{(b)} h^{(b)}(x)$.

- Two main ideas:
  - Gradient descent to solve the optimization problem
  - fit a weak predictor on the gradient step.

Reminder: Gradient Descent

For minimizing a convex and differentiable function

$$J : \mathbb{R}^n \to \mathbb{R},$$

the Gradient Descent (or Steepest Descent) introduces the sequence $\theta^{(0)}, \ldots, \theta^{(b)}, \ldots$ such that

$$\theta^{(b)} = \theta^{(b-1)} - \nu \nabla J \left(\theta^{(b-1)} \right).$$

ν is the step length.

Gradient for the Forward Stagewise Algorithm

- Without the constraint "$h \in \mathcal{H}$" in the Forward Stagewise Algorithm, the problem is to minimize

$$u \in \mathbb{R}^n \to \sum_{i=1}^{n} \ell\left(y_i, f^{(b-1)}(x_i) + u_i\right)$$

- For applying the first step of the Gradient Descent, we need to compute

$$\begin{pmatrix} \frac{\partial \ell(y_1, f(x_1))}{\partial f(x_1)} \\ \vdots \\ \frac{\partial \ell(y_i, f(x_i))}{\partial f(x_i)} \\ \vdots \\ \frac{\partial \ell(y_n, f(x_n))}{\partial f(x_n)} \end{pmatrix}_{f(x)=f^{(b-1)}(x)} := \begin{pmatrix} \frac{\partial \ell(y_1, f(x_1)+u_1)}{\partial u_1}(u_1 = 0) \\ \vdots \\ \frac{\partial \ell(y_i, f(x_i)+u_i)}{\partial u_i}(u_i = 0) \\ \vdots \\ \frac{\partial \ell(y_n, f(x_n)+u_n)}{\partial u_n}(u_n = 0) \end{pmatrix}$$

Regression and quadratic loss

- Let us consider the regression framework and the quadratic loss

$$\ell(y_i, v_i + u_i) = \frac{1}{2}(y_i - v_i - u_i)^2.$$

Then

$$\frac{\partial \ell(y_i, v_i + u_i)}{\partial u_i} = -(y_i - v_i - u_i)$$

and

$$\frac{\partial \ell(y_i, v_i + u_i)}{\partial u_i}(u_i = 0) = -(y_i - v_i)$$

- In this case, the gradient step is in the direction of the residuals:

$$-\begin{pmatrix} \vdots \\ y_i - f^{(b-1)}(x_i) \\ \vdots \end{pmatrix} =: \delta^{(b-1)}$$

Naive Gradient Descent is not sufficient

- Similar computations with other losses in regression (absolute value, Huber Loss),
- Similar computations for classification with the cross entropy loss (also called deviance), see further.
- The gradients $\delta^{(b-1)}$ are called *pseudo residuals*.
- Remember our initial problem:

$$\arg \min_{\beta \in \mathbb{R},\, h \in \mathcal{H}} \quad \sum_{i=1}^{n} \ell \left(y_i, f^{(b-1)}(x_i) + \beta h(x_i) \right)$$

- The previous naive Gradient Descent approach gives the gradients at the observation points $(x_i, y_i) : \delta^{(b-1)}(i)$, $i = 1 \ldots n$.
- It would yield a predictor that could be only computed on the training set !

Weak learner fitted on the gradient

- Solution: at each step of the Gradient Descent, fit a weak learner on the direction of the gradient !

$$h^{(b)} = \arg\min_{h \in \mathcal{H}} \quad \sum_{i=1}^{n} \left[\delta^{(b-1)}(i) - h(x_i) \right]^2 .$$

- Then we find $\hat{\beta}^{(b)}$ such that

$$\hat{\beta}^{(b)} = \arg\min_{\beta \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, f^{(b-1)}(x_i) + \beta h^{(b)}(x_i)).$$

- We use the quadratic loss to fit the gradient whatever the ML task (regression, classification ...).

Gradient Boosting Regressor with quadratic loss

Algorithm 4 Gradient Boosting Regressor with quadratic loss.

1: Initialize $f^{(0)}(x) := \hat{\mu}$ with $\hat{\mu} = \arg\min_{\mu \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, \mu)$.

2: **for** $b = 1$ to B **do**

3: For $i = 1, \ldots, n$, compute the residuals: $\delta^{(b-1)}(i) = f^{(b-1)}(x_i) - y_i$.

4: Compute $\hat{h}^{(b)} = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} \left[\delta^{(b-1)}(i) - h(x_i)\right]^2$.

5: Compute $\hat{\beta}^{(b)} = \arg\min_{\beta \in \mathbb{R}} \sum_{i=1}^{n} \ell\left(y_i, f^{(b-1)}(x_i) + \beta \hat{h}^{(b)}(x_i)\right)$.

6: Compute $f^{(b)}(x) = f^{(b-1)}(x) + \hat{\beta}^{(b)} \hat{h}^{(b)}(x)$.

7: **end for**

8: Output Boosting Predictor $f^{(B)}(x) = \sum_{b=1}^{B} \hat{\beta}^{(b)} \hat{h}^{(b)}(x)$.

Generic Gradient Boosting Algorithm

Algorithm 5 Gradient Boosting with general loss.

1: Initialize $f^{(0)}(x) := \hat{\mu}$ with $\hat{\mu} = \arg\min_{\mu \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, \mu)$

2: **for** $b = 1$ to B **do**

3: Compute the gradients $\delta^{(b-1)} = \begin{pmatrix} \vdots \\ \frac{\partial \ell(y_i, f(x_i))}{\partial f(x_i)} \\ \vdots \end{pmatrix}_{f(x)=f^{(b-1)}(x)}$.

4: Compute $\hat{h}^{(b)} = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} \left[\delta^{(b-1)}(i) - h(x_i)\right]^2$.

5: Compute $\hat{\beta}^{(b)} = \arg\min_{\beta \in \mathbb{R}} \sum_{i=1}^{n} \ell\left(y_i, f^{(b-1)}(x_i) + \beta \hat{h}^{(b)}(x_i)\right)$.

6: Compute $f^{(b)}(x) = f^{(b-1)}(x) + \hat{\beta}^{(b)} \hat{h}^{(b)}(x)$.

7: **end for**

8: Output Boosting Predictor $f^{(B)}(x) = \sum_{b=1}^{B} \hat{\beta}^{(b)} \hat{h}^{(b)}(x)$.

Gradient Boosting for Multiclass classification

- Boosting procedure on K score functions $f_k(x)$ (one for each class k).
- Class conditional probabilities $p_k(x) = P(Y = k|x)$ are computed all together through a soft max transformation:

$$\hat{p}_k(x) := \frac{\exp(f_k(x)}{\sum_{l=1}^{K} \exp(f_l(x))}$$

- Loss function: cross-entropy between $y \in \{1, \dots K\}$ and a K-vector of proportions p

$$\ell(y, p) := -\sum_{k=1}^{K} \log p_k \mathbb{1}_{y=k}.$$

Note that :

  - the loss is close to zero when $y = k$ with p_k close to one.
  - the loss is large when $y = k$ with p_k close to zero.

Gradient Boosting for Multiclass classification

- Boosting procedure on K score functions $f_k(x)$ (one for each class k).
- Class conditional probabilities $p_k(x) = P(Y = k|x)$ are computed all together through a soft max transformation:

$$\hat{p}_k(x) := \frac{\exp(f_k(x)}{\sum_{l=1}^{K} \exp(f_l(x))}$$

- Loss function: cross-entropy between $y \in \{1, \dots K\}$ and a K-vector of proportions p

$$\ell(y, p) := -\sum_{k=1}^{K} \log p_k \mathbb{1}_{y=k}.$$

We have $\ell(y, \hat{p}(x)) = -\sum_{k=1}^{K} f_k(x)\mathbb{1}_{y=k} + \log\left(\sum_{l=1}^{K} \exp(f_l(x))\right)$.

- At each iteration:
  - Compute $n \times K$ derivatives:

$$\frac{\partial \ell\left(y_i, (p_1(x_i), \dots, p_K(x_i))\right)}{\partial f_k(x_i)} = \mathbb{1}_{y_i=k} - p_k(x_i)$$

  - Fit a K (weak) predictors on the K gradients by least squares.

Gradient Tree Boosting

- Gradient tree boosting specializes Gradient Boosting for small regression trees (with M-terminal nodes).
- Note that we use regression trees (as weak learners) also for classification tasks.
- At each iteration b, a regression tree is fitted on the gradient (quadratic node impurity criterion):

$$T^{(b)}(x) = \sum_{m=1}^{M} \bar{y}_{bm} \mathbb{1}_{x \in R_{bm}} \quad \text{where} \quad \bar{y}_{bm} = \text{mean}(\delta^{(b-1)}(i) | x_i \in R_{bm})$$

- For all $m \in \{1, \dots M\}$, find

$$\hat{\beta}^{(bm)} = \arg\min_{\beta \in \mathbb{R}} \sum_{x_i \in R_{bm}} \ell(y_i, f^{(b-1)}(x_i) + \beta)$$

- Update separately in each corresponding region with global learning rate ν:

$$f^{(b)}(x) = f^{(b-1)}(x) + \nu \sum_{m=1}^{M} \beta^{(bm)} \mathbb{1}_{R_{bm}}(x).$$

Gradient Tree Boosting

Algorithm 6 Gradient Tree Boosting Algorithm for general loss.

1: Initialize $f^{(0)}(x) := \hat{\mu}$ with $\hat{\mu} = \arg\min_{\mu \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, \mu)$.

2: **for** $b = 1$ to B **do**

3: For $i = 1, \ldots, n$, compute the gradients $\delta^{(b)}(i)$.

4: Fit a regression tree $T^{(b)}$ with terminal nodes $\{R_{b,m}\}_{m=1}^{M}$ (on the pseudo-residual $\delta^{(b)}(i)$.

5: Compute $\hat{\beta}^{(bm)} = \arg\min_{\beta \in \mathbb{R}} \sum_{x_i \in R_{bm}} \ell(y_i, f^{(b-1)}(x_i) + \beta)$

6: Compute $f^{(b)}(x) = f^{(b-1)}(x) + \nu \sum_{m=1}^{M} \hat{\beta}^{(bm)} \mathbb{1}_{R_{bm}}(x)$.

7: **end for**

8: Output Boosting Predictor $f^{(B)}(x)$.

Some comments on Gradient Tree Boosting

- For classification we fit K regression trees on the K gradients.
- Possible regularization by limiting the minimum number of observations in the terminal nodes of the trees.
- Parameters : number of terminal nodes, depth, learning rate, number of iterations (early stopping)...
- Terminology : GBM is Gradient Tree Boosting (GBM stands for Gradient Boosting Machine).
- Extensions and efficient implementations of gradient boosting:
  - Stochastic Boosting Gradient: subsample on the training sample (c.f. SGD) [Friedman 99]
  - XGBoost (see further)
  - LightGBM (Microsoft Research framework for gradient boosting)
  - CatBoost (high-performance open source library for gradient boosting on decision trees)

Outline

XGBoost

- XGBoost is a scalable implementation of gradient tree boosting system widely recognized for its outstanding results in numerous data challenges [Chen and Guestrin, 2016].

- Two key points for XGBoost:
  - Additional regularization term in the objective function of the forward stagewise method:

$$\mathcal{L}^{(b)}(\beta, h) := \sum_{i=1}^{n} \ell\left(y_i, f^{(b-1)}(x_i) + \beta h(x_i)\right) + \Omega(f^{(b)})$$

    where

$$\Omega(f) = \gamma M + \frac{1}{2}\lambda \|c\|^2$$

    and $f = \sum_{m=1}^{M} c_m \mathbb{1}_{R_m}$.
  - Optimization method of order 2 for minimizing the regularized loss (GMB is order 1).

XGBoost: node impurity

As a consequence, XGBoost has to consider an alternative impurity node criterion (at each step):

$$Q^{(b)}(\beta, f) := \frac{1}{2} \sum_{m=1}^{M} \frac{(\sum_{i \in R_{bm}} g_i)^2}{\sum_{i \in R_{bm}} h_i + \lambda}$$

where

$$g_i = \left(\frac{\partial \ell(y_i, f(x_i))}{\partial f(x_i)} \right)_{f(x_i)=f^{(b-1)}(x_i)}$$

and

$$h_i = \left(\frac{\partial^2 \ell(y_i, f(x_i))}{\partial f(x_i)^2} \right)_{f(x_i)=f^{(b-1)}(x_i)}$$

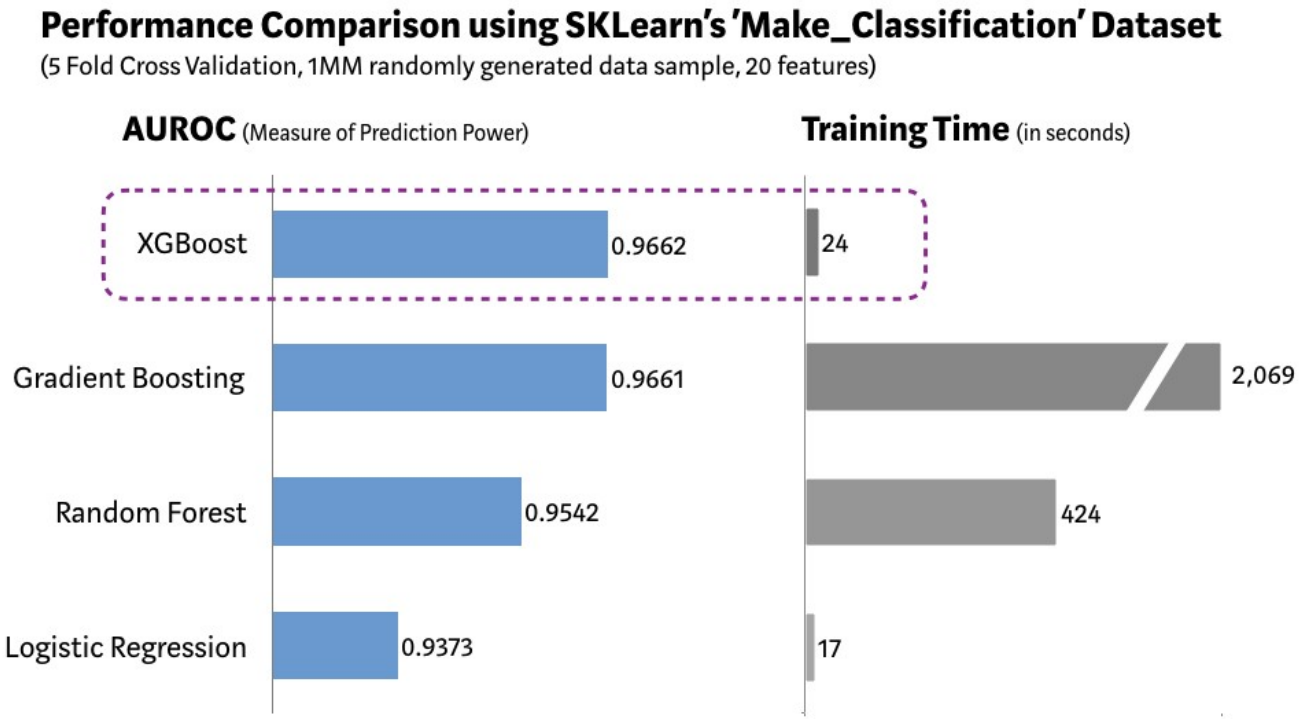
are first and second order gradient statistics on the loss function.

XGBoost package

- XGBoost proposes an approximated greedy algorithm.
  - It is difficult to execute the top down greedy algorithm for fitting a regression tree when the data does not fit entirely into memory (with this Impurity note or the standard ones).
  - XGBoost :
    - ★ based on candidate split points.
    - ★ based on percentile of the features.
    - ★ require to sort the data: block structure and parallel computing.

XGBoost package

- XGBoost proposes an approximated greedy algorithm.
- High Performances



[Source: towardsdatascience.com]

XGBoost package

- XGBoost proposes an approximated greedy algorithm.
- High Performances
- Sckit-learn wrapper : XGBClassifier.
- More about XGBoost: read the original paper.