# Introduction to Machine Learning and Deep Learning with Python

Day 2

Tabea Rebafka

CNRS Formation Entreprise
28 – 30 June, 2023, Paris

## Outline

## Outline of Day 2

## General context: supervised learning

**Context**

- For each individual $i = 1, \dots, n$, we observe
  - a **feature vector** $x_i \in \mathbb{R}^d$ with $x_i = (x_i^1, \dots, x_i^d) = (x_i^j)_{1 \le j \le d}$
  - an **outcome** $y_i$ with values
    - ★ in $\mathbb{R}$ (**regression problem**)
    - ★ in a finite number of classes $\{1, \dots, K\}$ (**classification problem**)
- Data $\mathcal{D} = \{(x_i, y_i), i = 1, \dots, n\}$ are considered to be i.i.d realizations of some random $(X, Y)$
- We want to learn the relation between $Y$ and $X$
- Given a new feature vector $x_{\text{new}}$, we want to predict its unobserved outcome $y_{\text{new}}$, which is either a real value or a class label.

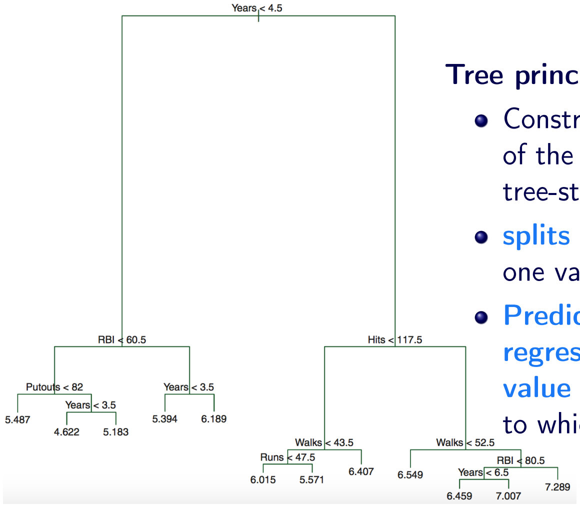
## Supervised learning with decision trees I

**Tree-based methods**

- stratification or segmentation of the predictor space into a number of simple regions
- splitting rules used to segment the predictor space can be summarized in a tree called decision tree
- prediction are based on the values of the training data in the region to which the new observation $x_{\mathrm{new}}$ belongs

## Supervised learning with decision trees II

### Regression tree

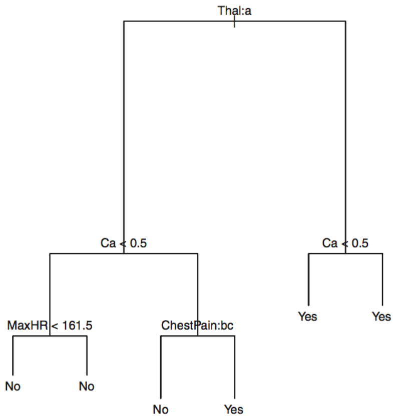


### Tree principle

- Construct a **recursive partition** of the predictor space using a tree-structured set of "questions"
- **splits** around a given value of one variable $x^j$ $(1 \leq j \leq d)$
- **Prediction** for $x_{\text{new}}$ in the **regression** context: **mean value** of training data in the leaf to which $x_{\text{new}}$ belongs

## Supervised learning with decision trees III

### Classification tree



### Tree principle

- Same construction
- Prediction for $x_{\text{new}}$ in the classification context: majority vote in the leaf to which $x_{\text{new}}$ belongs

## Supervised learning with decision trees IV

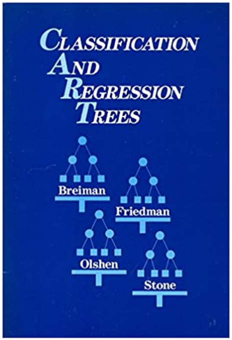
**Decision trees are**

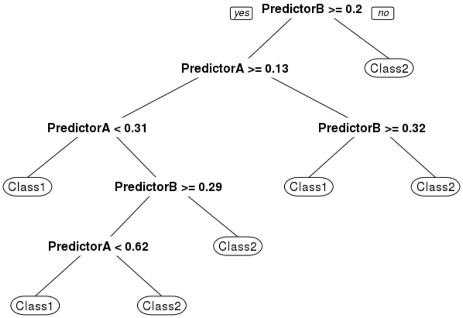
- a **simple** method
- highly **interpretable** (however, beware of unstable results)
- but **not competitive** with the best supervised learning approaches
- **improvement**: combine multiple trees to a single consensus prediction, called **ensemble methods** as
  - bagging
  - random forests
  - boosting

## Supervised learning with decision trees V

### CART = Classification and Regression Trees

- General framework for both types of decision trees



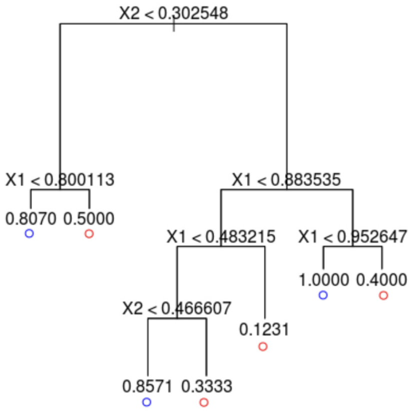


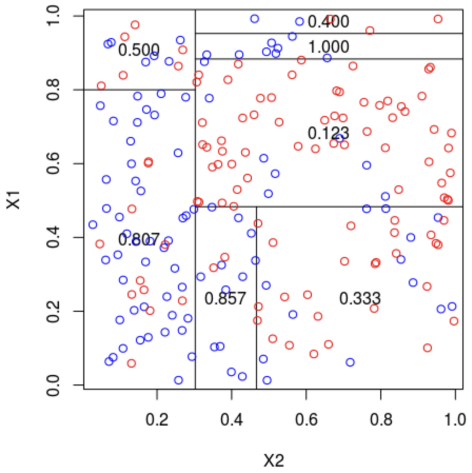
## Construction of decision trees I

**Problem**

- How to construct a decision tree with high prediction accuracy?
- Prediction quality depends on the tree (the partition) and there are many, many possible trees
- Finding the optimal tree is hard!

## Construction of decision trees II





values in the leafs/rectangles = proportion of **blue** training observations in the leaf/rectangle

## Construction of decision trees III

**Construction of a tree**

- Top-down construction where branches are created (branching)
- Start from a single region containing all training data (called root of the tree)
- Recursively choose a feature and a threshold to split nodes into ever smaller rectangular regions
- Leads to a guillotine partition of the feature space
- Which is equivalent to a binary tree

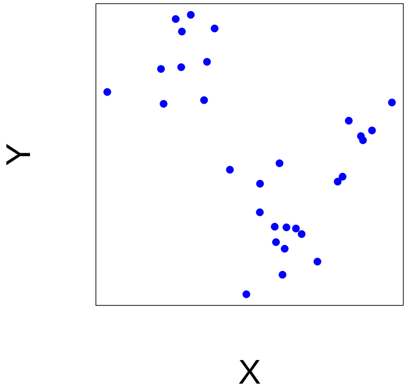
## Construction of decision trees IV

**How to split?**

- Split such that the labels of the training data in the two new regions are as homogeneous as possible
  - For classification: the majority of observations in a node should have the same class label
  - For regression: the outcomes in a node should be very concentrated around their mean value

## Construction of decision trees V

### Quantification of homogeneity



**Regression context**

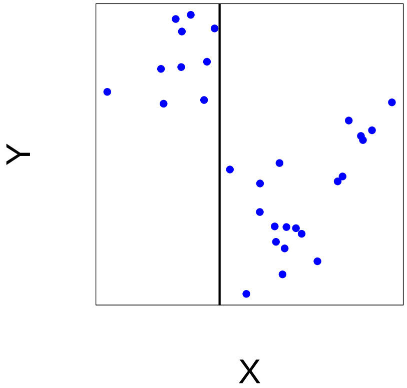
- Variance of training observations in node $N \subset \{1, \dots, n\}$:

$$V(N) = \frac{1}{\#N} \sum_{i \in N} (y_i - \bar{y}_N)^2$$

$$\text{with } \bar{y}_N = \frac{1}{\#N} \sum_{i \in N} y_i$$

## Construction of decision trees VI

### Quantification of homogeneity



**Regression context**

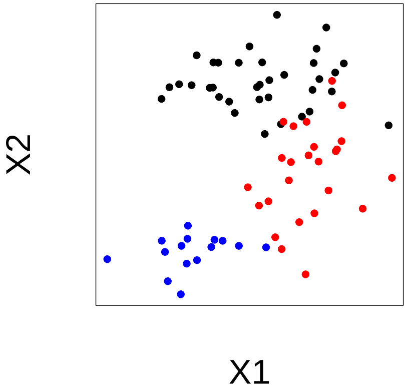
- Variance of training observations in node $N \subset \{1, \ldots, n\}$:

$$V(N) = \frac{1}{\#N} \sum_{i \in N} (y_i - \bar{y}_N)^2$$

$$\text{with } \bar{y}_N = \frac{1}{\#N} \sum_{i \in N} y_i$$

## Construction of decision trees VII

### Quantification of homogeneity



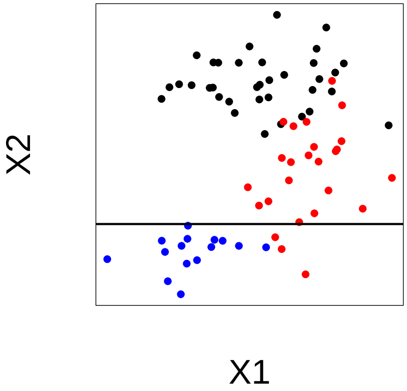
### Classification context

- different measures of **node (im)purity**
- all based on **class frequencies** of training observations in node $N$: For every class $k$

$$p_{N,k} = \frac{\#\{i \in N : y_i = k\}}{\#N}$$

## Construction of decision trees VIII

### Quantification of homogeneity



### Classification context

- different measures of **node (im)purity**
- all based on **class frequencies** of training observations in node $N$: For every class $k$

$$p_{N,k} = \frac{\#\{i \in N : y_i = k\}}{\#N}$$

## Construction of decision trees IX

**Impurity measures for classification**

- Classification error rate: fraction of the observations in node $N$ that do not belong to the most common class:

$$1 - \max_k p_{N,k}$$

$\hookrightarrow$ not sufficiently sensitive for growing a tree

- Gini index

$$G(N) = \sum_{k=1}^{K} p_{N,k}(1 - p_{N,k})$$

- Entropy or cross-entropy

$$H(N) = -\sum_{k=1}^{K} p_{N,k} \log_2(p_{N,k})$$

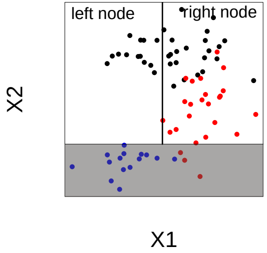
$\hookrightarrow$ $G(N)$ and $H(N)$ are quite similar numerically
$\hookrightarrow$ CART with Gini index is the most used technique

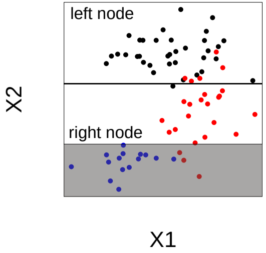
## Construction: Splitting a node I

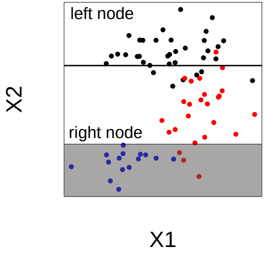
### Node splitting

- Split a node $N$ into a left and a right child node $N_L$ and $N_R$
- Child nodes depend on the chosen **feature** $j$ and the **threshold** $t$ for the split

### Many possible splits







## Construction: Splitting a node II

**Evaluate the quality of a split**

- For a possible split of node $N$ into children $N_L$ and $N_R$, evaluate the **impurity** of the resulting child nodes by

$$\frac{|N_L|}{|N|} I(N_L) + \frac{|N_R|}{|N|} I(N_R)$$

  where $I$ denotes
  - the variance $V$ in the regression context
  - the Gini index $G$ or the entropy $H$ for classification
- weighted sum of impurity of both child nodes

## Construction: Splitting a node III

**Best split**

- Among **all** feature-threshold pairs $(j, t)$ (i.e. for any $j \in \{1, \dots, d\}$ and $t \in \mathbb{R}$), the best split **minimizes** the associated **impurity** measure
- Equivalently, the best split **maximizes** the **information gain** defined as

$$\mathrm{IG}(j,t) = I(N) - \frac{|N_L(j,t)|}{|N|} I(N_L(j,t)) - \frac{|N_R(j,t)|}{|N|} I(N_R(j,t))$$

where $I(N)$ is the impurity of node $N$

## CART algorithm I

**Recursive partitioning**

- CART method builds the partition iteratively
- starting from the root
- until some stopping criterion is satisfied

**Algorithm 1** CART

```
1: while stopping criterion is not met do
2:     for every node $N$ of the current tree do
3:         Find the best feature-threshold pair $(j, t)$ that maximizes $\mathrm{IG}(j, t)$
4:         Create the new child nodes
5:     end for
6: end while
```

## CART algorithm II

**Greedy approach**

- Greedy algorithms make the optimal choice at each step
- No regret strategy on the choice of the splits
- No guarantee to find the optimum

## CART algorithm III

**Recall: Training and test sets**

- The tree is learned on a **training set**
- Split data $\mathcal{D} = \{(y_i, x_i), i = 1, \ldots, n\}$ into a training set and a test set: $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}} = \mathcal{D}$
  - Use $\mathcal{D}_{\text{train}}$ to learn the tree
  - Use $\mathcal{D}_{\text{test}}$ to evaluate error metrics and performance measures

## CART algorithm IV

### Stopping criterions

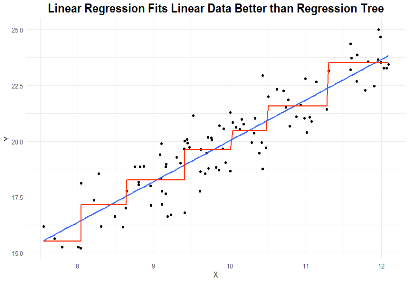
- Common criterions
  - Maximum depth of the tree is attained
  - Maximum number of leaves is attained
  - Number of data points per leaf is below a given threshold
  - Impurity per leaf is small enough
  - Test error increases
- Early stopping is always critical

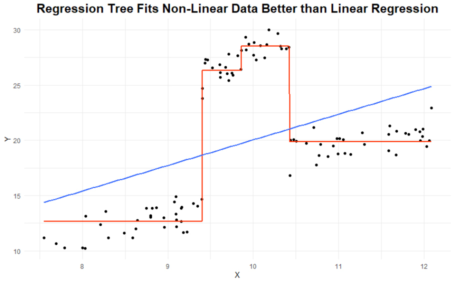
### Alternative to stopping criterion

- Grow a maximal tree, then do **pruning**
- Not optimal either

# Illustration I

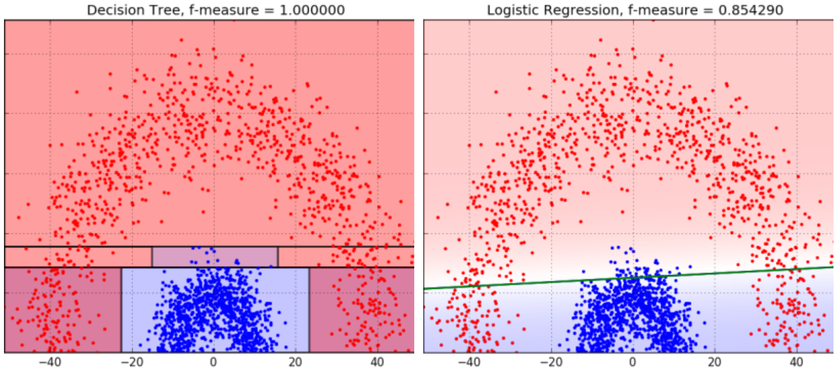
## Regression trees versus linear regression



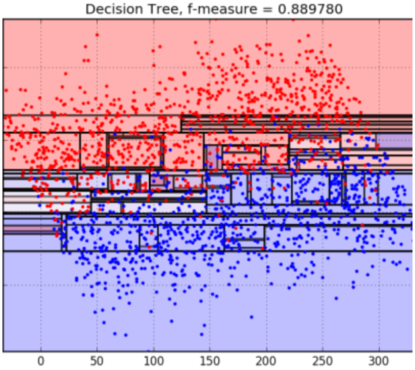


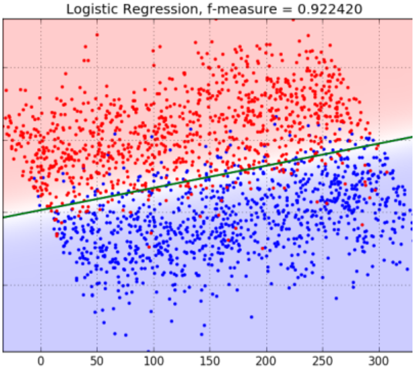
## Illustration II

### Classification tree versus logistic regression



## Illustration III





## Variable importance measure

### Importance of a specific feature

- Use the learned tree to evaluate the importance of each feature for the prediction task
- Different variable importance measures exist
- Naïvely, a feature $j$ that appears in the splits is important
- An overall summary of the importance of feature $j$ is the sum of the information gains obtained over all splits based on feature $j$
- Beware that this measure does not account for correlations among features

## Outline of Day 2

## Bagging I

**Inconvenients of decision trees**

- CART suffers from
  - low prediction accuracy
  - lack of robustness, high variance: new training datasets give rise to rather different trees
- Decision trees are **weak predictors**

$\hookrightarrow$ Instead of looking for one optimal tree, grow many, many trees and combine them to a more accurate predictor

$\hookrightarrow$ Substantial improvements of prediction accuracy can be obtained by aggregating many decision trees (**ensemble methods**)

## Bagging II

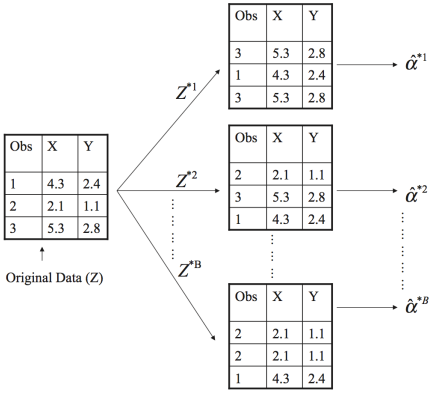
**Bagging approach**

- General-purpose procedure to reduce the variance of **any statistical learning method**
- Bagging = Boostrap Aggregation
- Particularly useful for decision trees
- Principle of **bagging**:
  - create many artificial datasets via **bootstrap**
  - learn many weak predictors on the bootstrapped samples
  - average/combine all predictors to obtain a highly accurate predictor
- **Variance reduction:** Let $Z_1, \ldots, Z_n$ be iid with variance $\sigma^2$. The variance of the mean $\bar{Z}_n$ is $\sigma^2/n$. In other words, averaging a set of observations reduces variance.

## Boostrap I

### Bootstrap

- general **resampling** method to create new artificial datasets
- use bootstrapped samples to **evaluate the accuracy** (e.g. bias, variance, confidence interval) of any estimator or predictor



### Bootstrap datasets

- obtained by sampling **with replacement** from the original dataset: randomly choose observations $(x_i, y_i)$
- generally contain some observations repeatedly
- same size as the original dataset

## Boostrap II

### General bootstrap approach

- Notations
  - $\hat{f}$ : any statistical procedure or estimator that can be learned on a sample ; $\hat{f}$ can be thought of an approximation of some "true" value or the "best" method $f$
  - $\hat{f}_{\mathcal{D}}$: estimator $\hat{f}$ trained on the original data $\mathcal{D}$
  - $\mathcal{D}^{(b)}, b = 1, \dots, B$: bootstrap datasets
  - $\hat{f}^{(b)}$: estimator learned on $\mathcal{D}^{(b)}$ (**bootstrap replicates**)
- **The bootstrap replicates $\{\hat{f}^{(b)}\}_{1 \leq b \leq B}$ are a sample of a distribution that is close to the distribution of $\hat{f}$**
- Denote $\bar{f}_B = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{(b)}$ the bootstrap mean
- Bootstrap approximation of the bias $\mathbb{E}(\hat{f}) - f$ and the variance of $\hat{f}$:

$$\widehat{bias}_{\text{boot}}(\hat{f}) = \bar{f}_B - \hat{f}_{\mathcal{D}} \qquad \widehat{var}_{\text{boot}}(\hat{f}) = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{f}^{(b)} - \bar{f}_B\right)^2$$

## Bagging of regressors or classifiers I

**Algorithm 2** Bagging of regressors or classifiers

```
1: for $b = 1, \dots, B$ do
2:     Sample with replacement $\mathcal{D}^{(b)}$ from the original dataset $\mathcal{D}$
3:     Fit a regressor (resp. classifier) $\hat{f}^{(b)}$ on $\mathcal{D}^{(b)}$
4: end for
5: Aggregate $\hat{f}^{(1)}, \dots, \hat{f}^{(B)}$ to get a single regressor (resp. classifier) using
   an average (resp. majority vote)
```

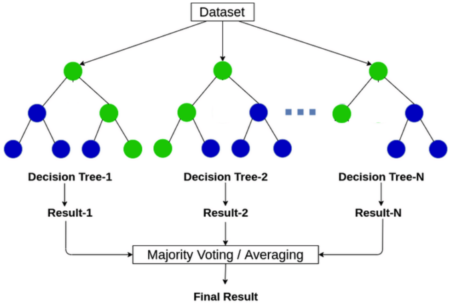
**Bagging for decision trees**

- Trees are grown deep and not pruned. Each individual has high variance, but low bias.
- Aggregation of trees does not lead to a simple aggregated tree $\hookrightarrow$ interpretability gets lost

## Bagging of regressors or classifiers II

### Prediction in a bagged model

- For a new point $x_{\text{new}}$, **every** bootstrap tree makes a prediction
- Aggregate all predictions to a single one:
  - in regression: average of all predictions
  - in classification: majority vote



## Bagging of regressors or classifiers III

**Out-of-bag error**

- straightforward way to estimate the **test error** of a bagged model
- no cross validation, no validation set approach
- on average, a bootstrap sample only uses two-thirds of the data
- the remaining one-third are the **out-of-bag (OOB)** observations
- for every $(x_i, y_i)$, there are around one third of the bootstrap trees that were learned without using this observation

## Bagging of regressors or classifiers IV

**Computation of the out-of-bag error**

- For $i = 1, \dots, n$
  - use all bootstrap trees that where learned when $(x_i, y_i)$ was OOB and compute the prediction $\hat{y}_i^{(b)}$ with the corresponding tree
  - average the predicted responses (or take a majority vote) to obtain a single prediction $\hat{y}_i$
- use all predictions $\hat{y}_i, i = 1, \dots, n$ to compute
  - in regression: the mean-squared error
  - in classification: a classification error

$\hookrightarrow$ The OOB error is a valid estimate of the test error for the bagged model

## Bagging of regressors or classifiers V

**Drawbacks of bagging**

- Trees on bootstrap samples are relatively similar $\hookrightarrow$ correlated trees
- If, for instance, the data contain a dominant feature for prediction, this feature appears at the top split of all trees
- Bagging has relatively large variance: when the whole bagging procedure is repeated on different training sets, quite different predictors are obtained

## Outline of Day 2

## Random forests I

### Random forests

- Extension of bagging
- Aim: reduce variance
- Idea: force trees learned on bootstrap samples to be more different for a better exploration of the model space $\hookrightarrow$ **decorrelate trees**

### Modification when growing a tree

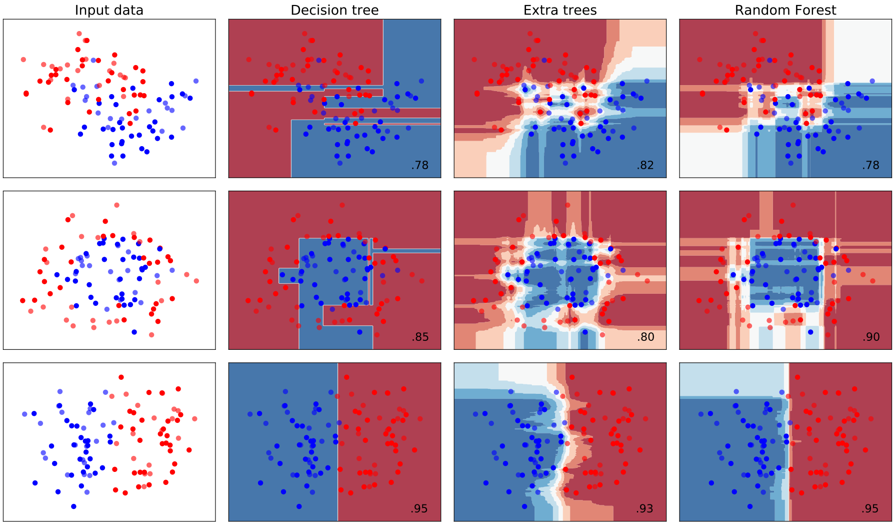
- Do **column subsampling** also called **feature bagging**
- When looking for splits **only a random subset of features is tried**
- Reduces the correlation between trees, especially when some features are particularly strong

## Random forests II

**Extremely randomized trees or Extra trees**

- A variant of random forests and bagging
- Feature selection for the splits is done in the bagging fashion by minimizing an impurity measure (Gini, Entropy)
- **Thresholds are selected uniformly at random in the feature range**
- Computation is fast
- Leads to accurate predictors

# Random forests III



## Outline of Day 2

## Boosting I

### Ensemble learning

- Idea: combine many weak learners to form a strong one
- Bagging and random forests:
  - use independent learners
  - treat all weak learners equally
- Boosting:
  - weak learners are not independent
  - associate different weights to weak learners

## Boosting II

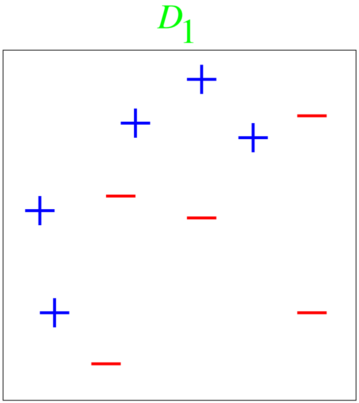
### Boosting

- Sequential approach
- Each weak learner depends on the previously built learners
- Idea: correct for mistakes of the current learner
- Reweight the data points before feeding them to the model
- A simple reweighting approach: replicate certain observations in order to focus the classifier on predicting well on them

## AdaBoost I

### AdaBoost

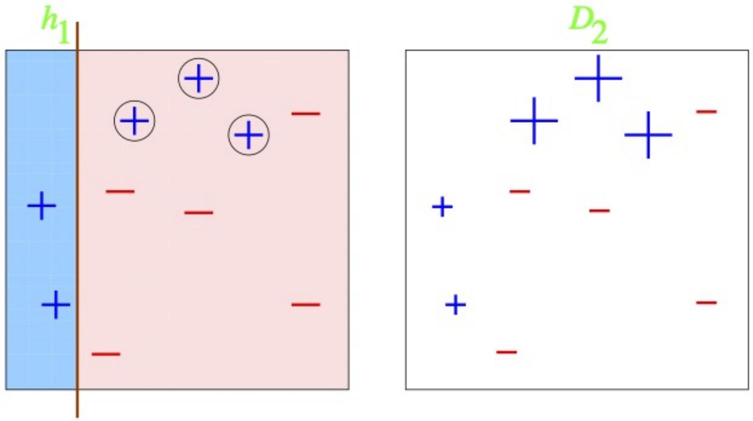
- = adaptive boosting
- for binary classification



- weak classifiers: vertical or horizontal half-planes
- $D_b$ : weights for data points ; start with equal weights

## AdaBoost II

### First round

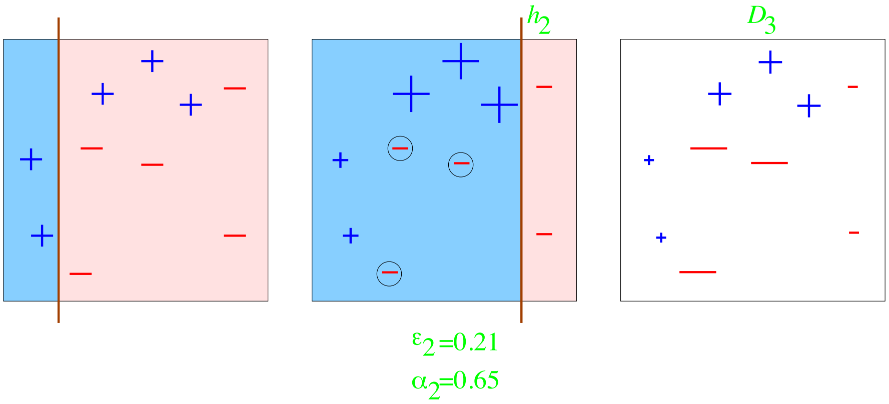


$\varepsilon_1 = 0.30$ (weighted) error rate

$\alpha_1 = 0.42$ weight of the classifier in the ensemble

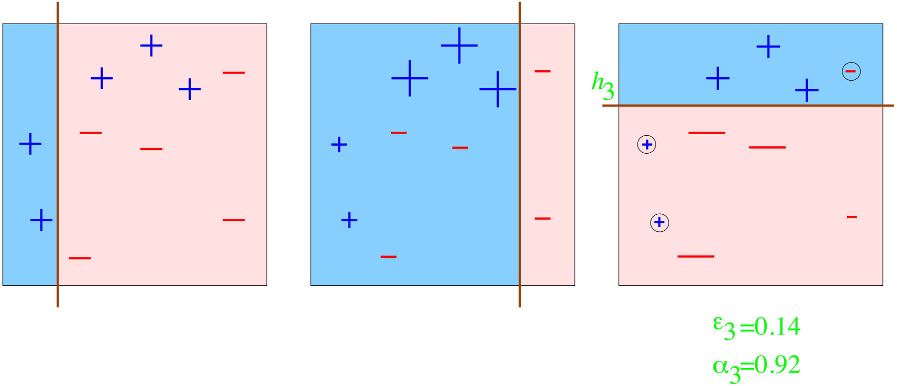
## AdaBoost III

### Second round



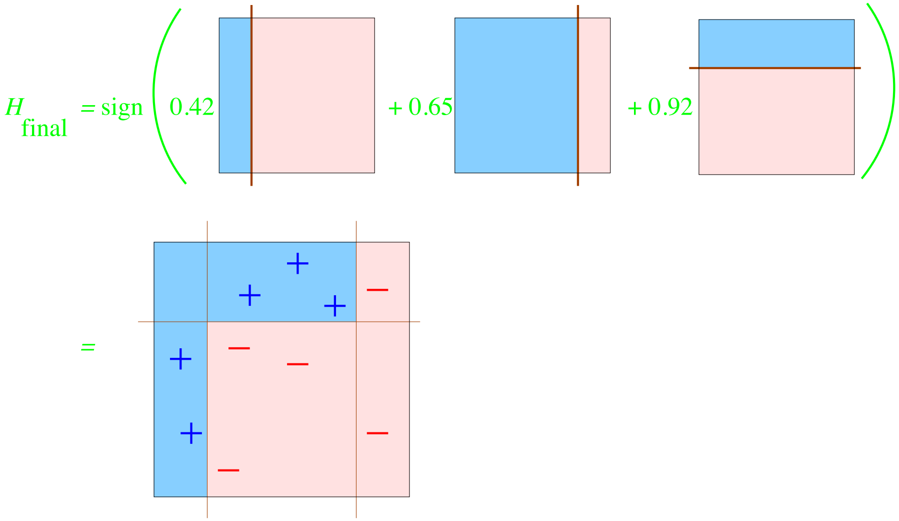
## AdaBoost IV

### Third round



## AdaBoost V

### Final classifier


$H_{\text{final}} = \text{sign}\left(0.42 \quad + 0.65 \quad + 0.92 \right)$
$=$
+
+
+
−
+
−
−
+
−
−

## AdaBoost VI

**Algorithm 3** AdaBoost

```
1: Initial weights of the data points: $D_i^{(1)} = \frac{1}{n}$ for $i = 1, \dots, n$
2: for $b = 1, \dots, B$ do
3:     Train classifier: $h_b \in \arg\min_{h\in\mathcal{H}} \sum_{i=1}^n D_i^{(b)} 1_{y_i h(x_i)<0}$
4:     Error rate: $\varepsilon_b = \sum_{i=1}^n D_i^{(b)} 1_{y_i h_b(x_i)<0}$
5:     Classifier weight: $\alpha_b = \frac{1}{2}\log\left(\frac{1-\varepsilon_b}{\varepsilon_b}\right)$ (log odds ratio)
6:     Normalizing constant: $Z_b = 2\sqrt{\varepsilon_b(1-\varepsilon_b)}$
7:     New data weights: $D_i^{(b+1)} = D_i^{(b)} \frac{e^{-\alpha_b y_i h_b(x_i)}}{Z_b}$
8: end for
9: return Boosting classifier: $g_B(x) = \mathrm{sign}\left(\sum_{b=1}^B \alpha_b h_b(x)\right)$
```

## Gradient boosting I

### Boosting as a minimization algorithm

- Goal: an ensemble learner of the form

$$g_B(x) = \sum_{b=1}^{B} \alpha_b h_b(x)$$

  that minimizes an empiral risk

$$\frac{1}{n}\sum_{i=1}^{n} \ell(y_i, g_B(x_i))$$

  where $\ell$ is a loss (least squares, logistic...)
- Minimization over a chosen family $\mathcal{H}$ of weak learners and weights $\alpha_b \in \mathbb{R}$

## Gradient boosting II

### Gradient boosting

- General-purpose method: many choices for the loss $\ell$, many choices of weak learners
- AdaBoost for binary classification:
  - exponential loss $\ell(y, z) = e^{-yz}$
  - decision trees of depth 1, called stumps

### Problem

- How to find weak learners $h_b$ and coefficients $\alpha_b$?
- Even given $\alpha_1, \dots, \alpha_B \in \mathbb{R}$, minimize over $|\mathcal{H}|^B$ to find the $h_1, \dots, h_B$
- Size of $\mathcal{H}$ is typically $O(d)$ (number of features)

## Gradient boosting III

### Greedy algorithm

- At iteration $b+1$: update current learner $g_b$ by

$$g_{b+1} = g_b + \alpha_{b+1} h_{b+1}$$

where $h_{b+1}$ and $\alpha_{b+1}$ are solutions of

$$\arg\min_{\alpha\in\mathbb{R}, h\in\mathcal{H}} \sum_{i=1}^{n} \ell(y_i, g_b(x_i) + \alpha h(x_i))$$

### Still a problem

- Exact minimization at each step is too hard

### Gradient boosting idea

- Replace exact minimization by a gradient step
- Approximate the gradient step by an element of $\mathcal{H}$

## Gradient boosting IV

**Algorithm 4** Gradient boosting

```
1: Put $g_0 = \arg\min_{m\in\mathbb{R}} \sum_{i=1}^n \ell(y_i, m)$
2: for $b = 1, \dots, B$ do
3:     Get gradient directions: $\delta_{b,i} = -\Big(\nabla_u \ell(y_i, g_b(x_i) + u)\Big)\Big|_{u=0}$
4:     Find learner $h_{b+1}$ that approximates direction $\delta_b$:
           $$(h_{b+1}, \nu_b) = \underset{h\in\mathcal{H},\nu\in\mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^n \big(\nu h(x_i) - \delta_{b,i}\big)^2$$
5:     Classifier weight: $\alpha_{b+1} = \arg\min_{\alpha\in\mathbb{R}} \sum_{i=1}^n \ell(y_i, g_b(x_i) + \alpha h_{b+1}(x_i))$
6:     Update ensemble learner: $g_b = g_{b-1} + \alpha_b h_b$
7: end for
8: return Boosting learner $g_B$
```

## Implementation of gradient boosting I

**Hyperparameters**

- Family $\mathcal{H}$ of weak learners:
  - If $\mathcal{H}$ contains decision trees, choose their depth (usually small... no more than four or five)
  - If $\mathcal{H}$ contains generalized linear models (logistic regression), select the number of features (usually one or two)
- Number of iterations $B$:
  - overfitting is possible (contrary to bagging and random forests)
  - use cross validation, stop when test error does no longer improve
  - usually a few hundreds, few thousands
- Check documentation of `GradientBoostingRegressor` of the `sklearn.ensemble` module for more details on the hyperparameters

## Implementation of gradient boosting II

### Many implementations available

... with many clever extra tricks

- `GradientBoostingRegressor`
  - Part of the `sklearn.ensemble` module
- `XGBoost`
  - `https://xgboost.readthedocs.io/en/latest/`
  - `https://github.com/dmlc/xgboost`
  - Has been state of the art for years
- `LightGBM`
  - `https://lightgbm.readthedocs.io/en/latest/`
  - `https://github.com/Microsoft/LightGBM`
  - Faster (and better ?) than `XGBoost`
- `CatBoost`
  - `https://catboost.ai/en/docs/`
  - One of the most recent implementations
  - Adapted for categorical features

## Grand mother's recipe I

BUTTER SCOTCH PIE — Good
1 cup milk
1 cup brown sugar (scant)
1 egg, 1 tbsp. flour
" A little Salt
1 tablespoon butter.

## Grand mother's recipe II

For standard tabular data (no image, signal, text etc.)

- Spend time on feature engineering
- Spend time on feature engineering
- ...
- Spend time on feature engineering
- Always try out random forests or gradient boosting **before** diving into a deep learning method (tomorrow)

**This afternoon: practical session**

- Prediction of electricity demand based on load curves
- Using random forests and `LightGBM`

Thank you!