Introduction to Machine Learning and Deep Learning

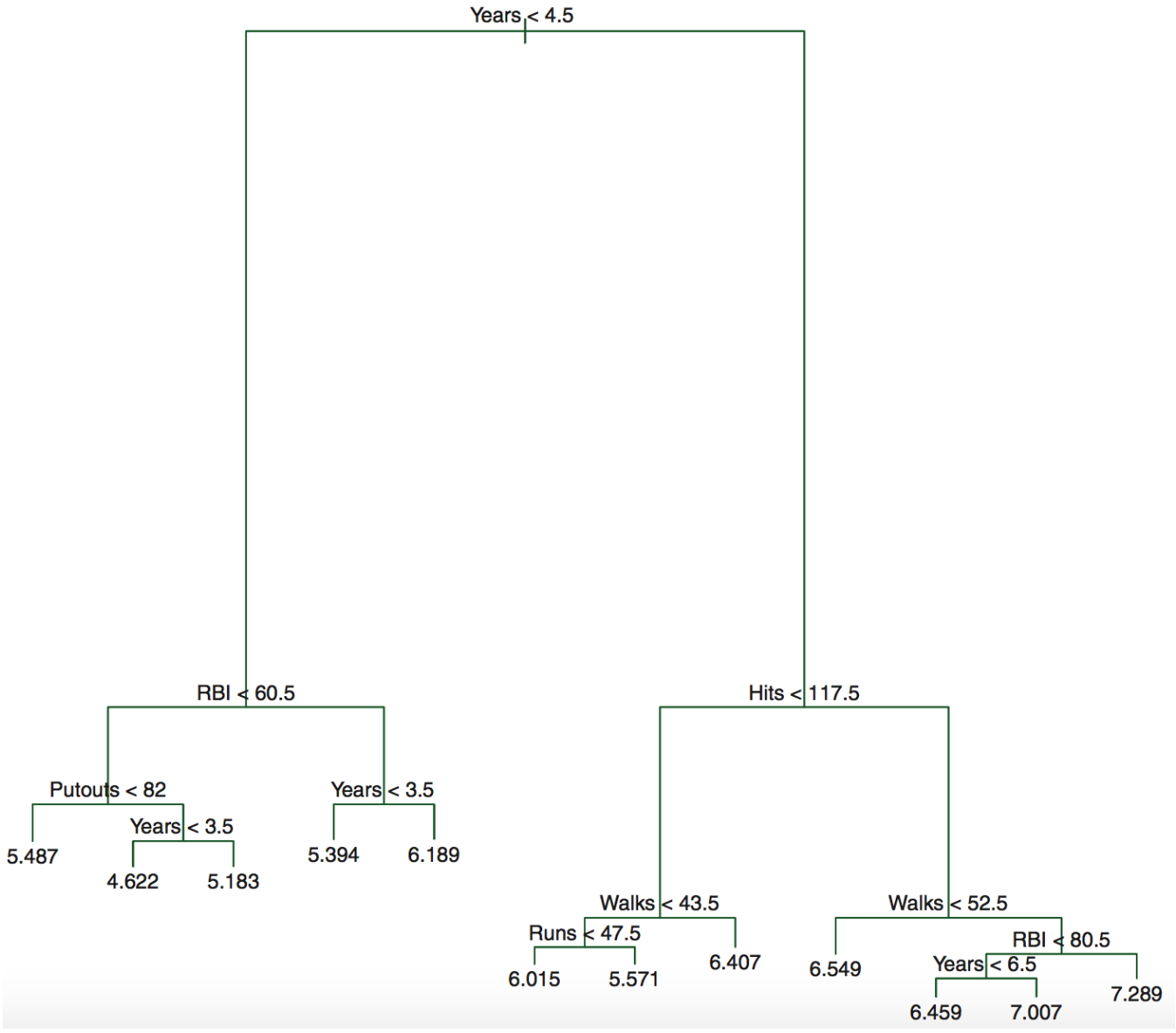
CNRS Formation Entreprise

Stéphane Gaïffas – Karine Tribouley

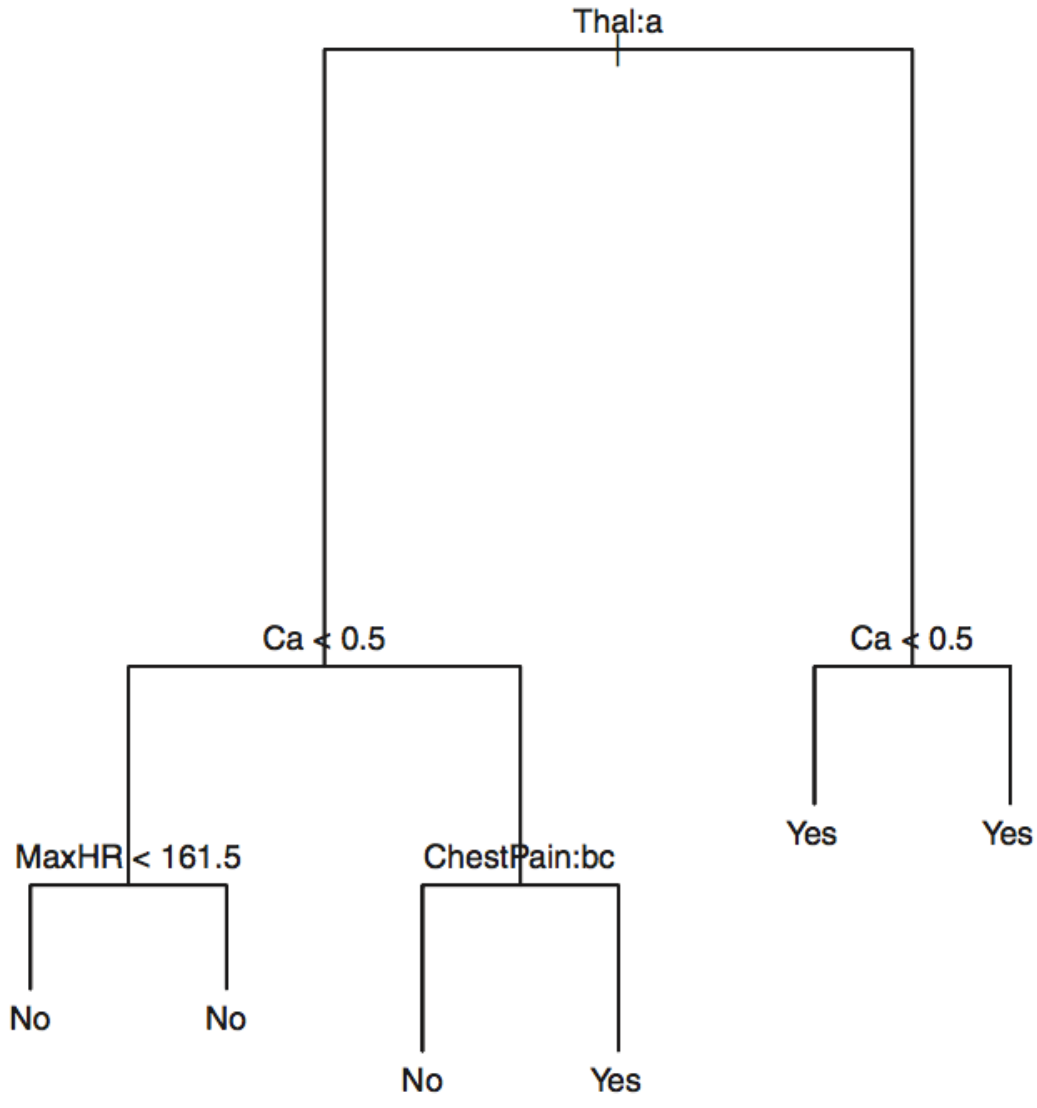
Today

- Classification and regression trees (CART)
- Bagging
- Random forests
- Boosting, adaboost, gradient boosting

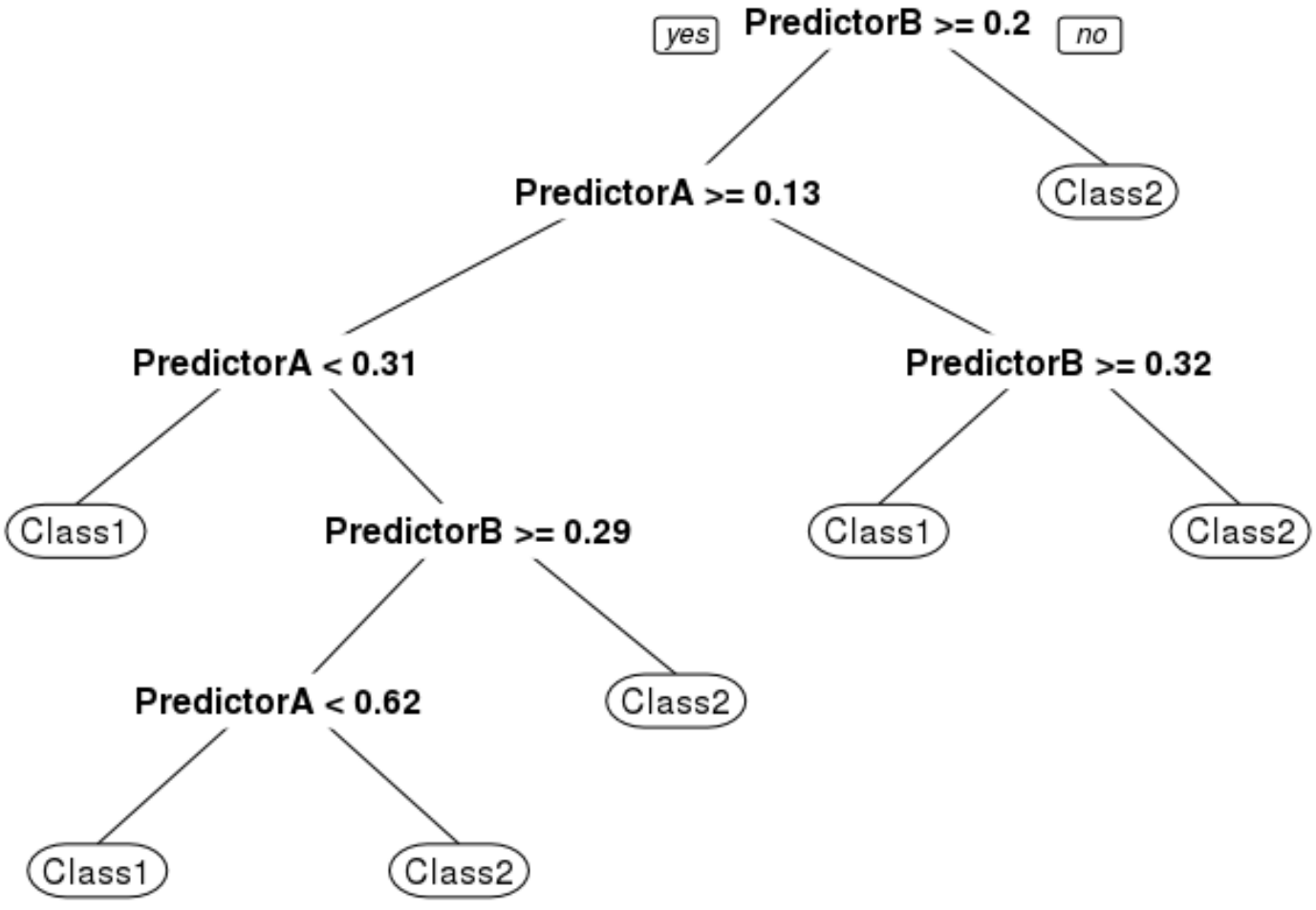
This is a regression tree



This is classification tree



CART = Classification and Regression Trees



Tree principle

- Construct a recursive partition using a tree-structured set of "questions" (splits around a given value of a variable)
- A simple average to predic class probabilities (classification)
- Average in each leaf (regression)

Remarks

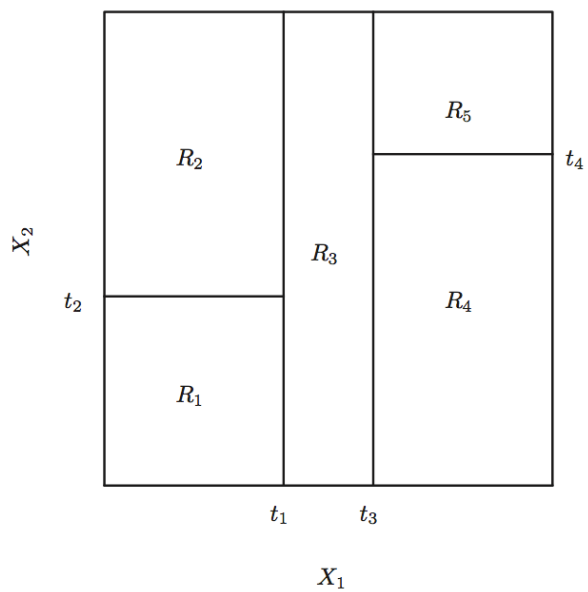
- Quality of the prediction depends on the tree (the partition)
- Issue: finding the optimal tree is hard!

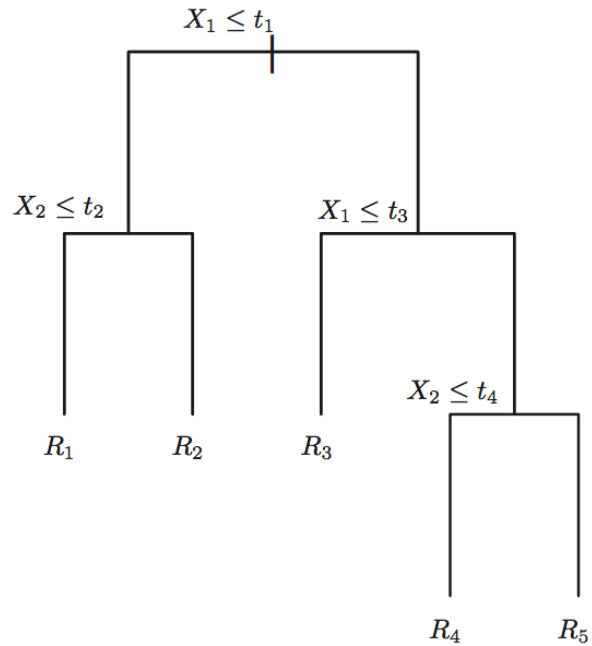
Heuristic construction of a tree

- Greedy approach: a top-down step in which branches are created (branching)

Heuristic construction of a tree

- Start from a single region containing all the data (called *root*)
- Recursively splits nodes (i.e. rectangles) along a certain dimension (feature) and a certain value (threshold)
- Leads to a *guillotine* partition of the feature space
- Which is equivalent to a *binary tree*





Remarks

- Greedy: no regret strategy on the choice of the splits!
- Quite the opposite of gradient descent!
- Heuristic: choose a split so that the two new regions are as **homogeneous** possible...

Homogeneous means:

- For classification: class distributions in a node should be close to a Dirac mass
- For regression: labels should be very concentrated around their mean in a node (for regression)

How to quantify **homogeneousness**?

- Variance (regression)
- Gini index, Entropy (classification)
- Among others

Splitting

- We want to split a node N into a left child node N_L and a right child node N_R
- The childs depends on a cut: a (feature, threshold) pair denoted by (j, t)

Introduce

$$N_L(j,t) = \{x \in N : x_j < t\} \quad \text{and} \quad N_R(j,t) = \{x \in N : x_j \geq t\}.$$

Finding the best split means finding the best (j, t) pair

- Compare the *impurity* of N with the ones of $N_L(j, t)$ and $N_R(j, t)$ for all pairs (j, t)
- Using the *information gain* of each pair (j, t)

For **regression** impurity is usually the **variance** of the node

$$V(N) = \sum_{i:x_i \in N} (y_i - \bar{y}_N)^2 \quad \text{where} \quad \bar{y}_N = \frac{1}{|N|} \sum_{i:x_i \in N} y_i$$

with $|N| = \#\{i : x_i \in N\}$ and **information gain** is given by

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

For **classification** with labels $y_i \in \{1, \ldots, K\}$. First, we compute the classes distribution $p_N = (p_{N,1}, \ldots, p_{N,K})$ where

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

Then, we can consider an impurity measure I such as:

$$G(N) = G(p_N) = \sum_{k=1}^{K} p_{N,k}(1 - p_{N,k}) \quad \text{(Gini index)}$$

$$H(N) = H(p_N) = -\sum_{k=1}^{K} p_{N,k} \log_2(p_{N,k}) \quad \text{(Entropy index)}$$

and for $I = G$ or $I = H$ we consider the information gain

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

CART builds the partition iteratively

For each leaf N of the current tree

- Find the best (feature, threshold) pair (j, t) that maximizes $\mathrm{IG}(j, t)$
- Create the two new childs of the leaf
- Stop if some stopping criterion is met
- Otherwise continue

Stopping criterions

- Maximum depth of the tree
- Maximum number of leaves
- All leafs have less then a chosen number of samples
- Impurity in all leafs is small enough
- Testing error is increasing
- Etc...

Remarks

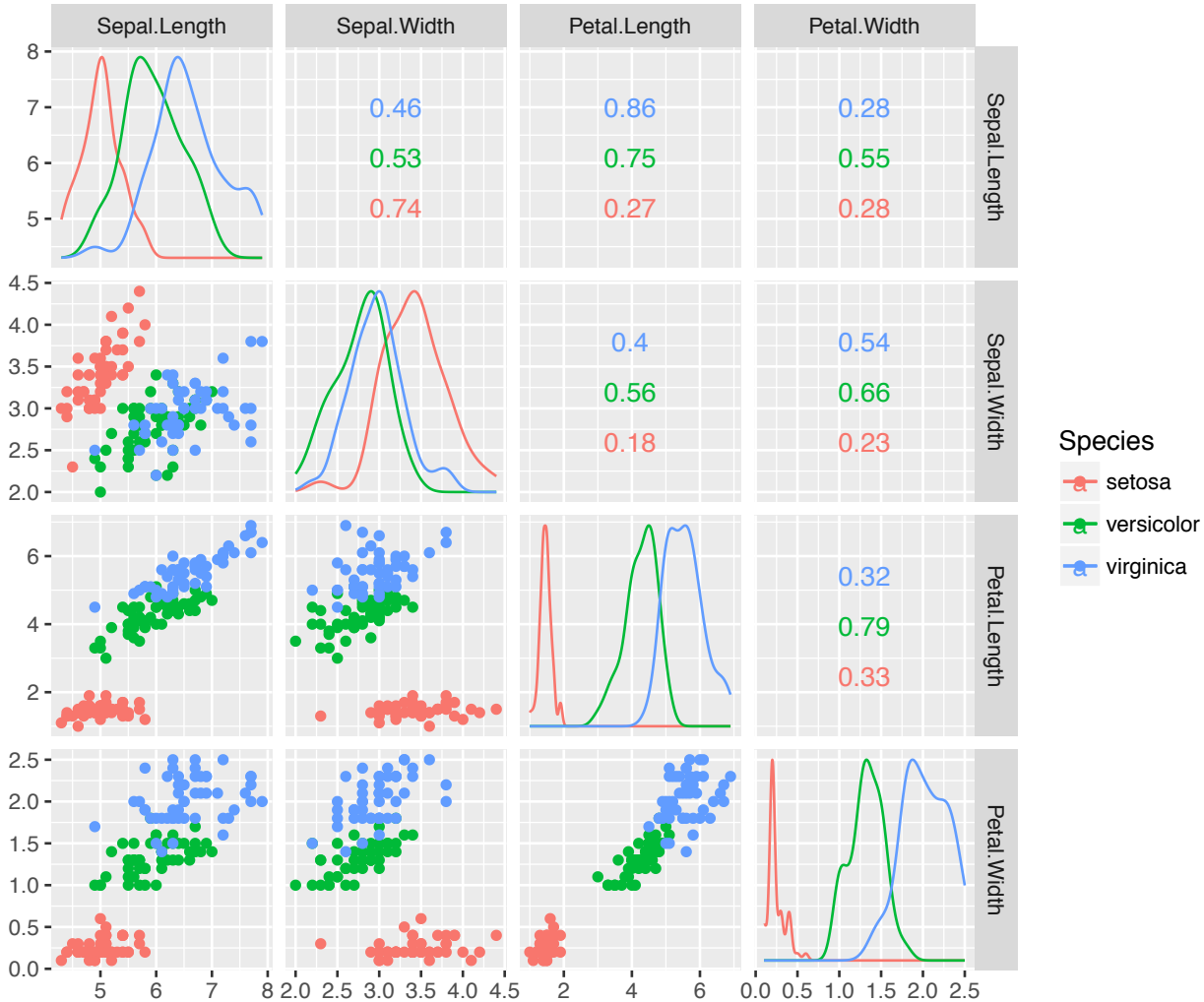
- CART with Gini is probably the most used technique
- Other criterions are possible: χ^2 homogeneity, other losses, than least-squares, etc.

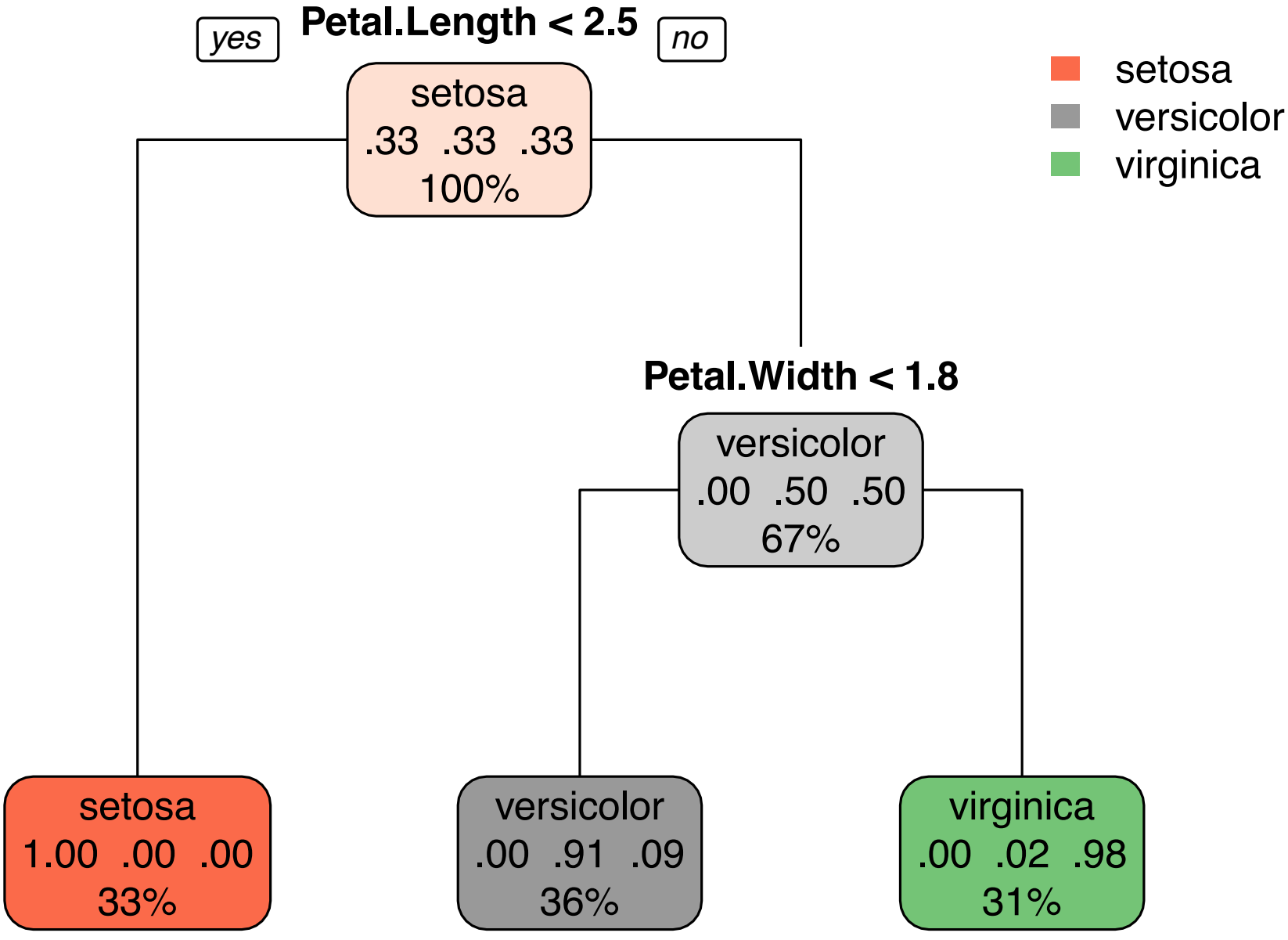
More remarks

- Usually overfits (maximally grown tree typically overfits)
- Usually leads to poor prediction
- Leads to nice interpretable results
- Is the basis of more powerful techniques: random forests, boosting (**ensemble methods**)

Example on iris dataset

- using `R` with the `rpart` and `rpart.plot` library



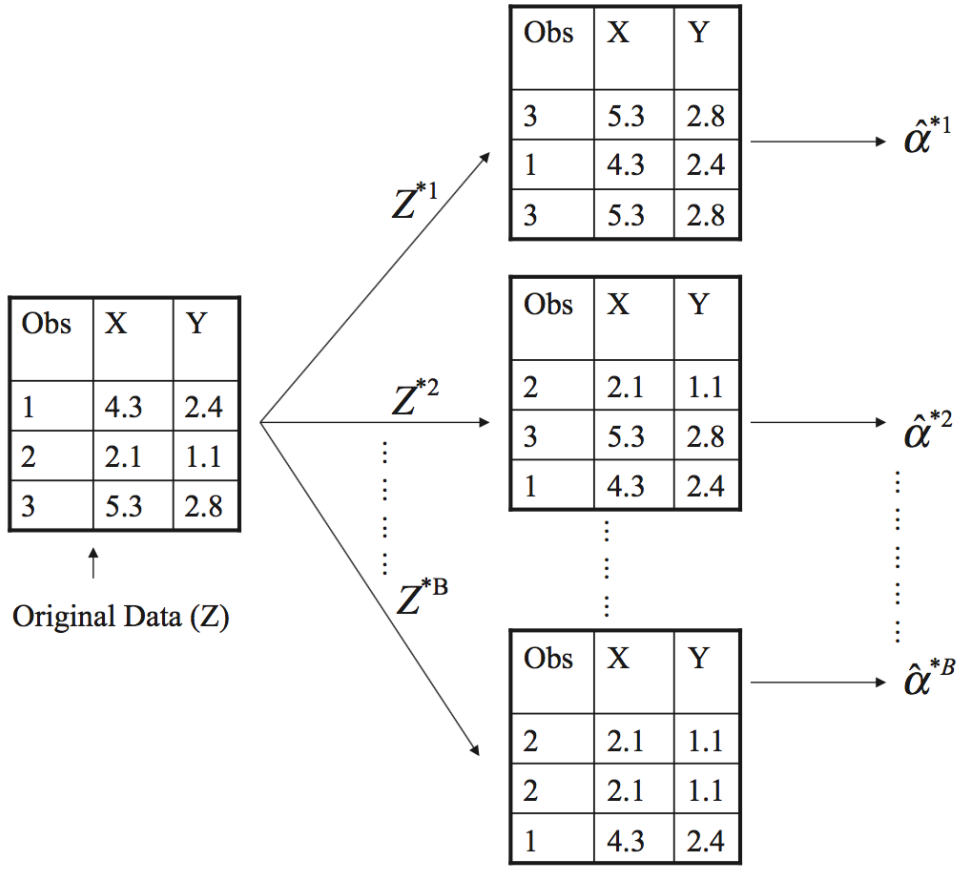

yes
Petal.Length < 2.5
no
setosa
.33 .33 .33
100%
setosa
versicolor
virginica
Petal.Width < 1.8
versicolor
.00 .50 .50
67%
setosa
1.00 .00 .00
33%
versicolor
.00 .91 .09
36%
virginica
.00 .02 .98
31%

Bagging

- Decision trees suffer from a high variance
- Bagging is a general-purpose procedure to (hopefully) reduce the variance of any statistical learning method
- Particularly useful for decision trees
- Bagging = Boostrap Aggregation = **aggregate (average) classifiers fitted on bootstrapped samples**

Boostrap

- Generation of "new" datasets



- Sampling **with replacement** from the dataset

Algorithm 1 Bagging

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

Random forests combine the following ingredients

1. Regression or classification trees (as "weak" learners). These trees can be maximally grown in the random forest algorithm

2. Bagging of such trees: each tree is trained over a sampling with replacement of the dataset

- Instead of a single tree, it uses an average of the predictions given by several trees: reduces noise, and variance of a single tree
- Even more true is the trees are not correlated: baggings helps in reducing the correlation of the trees

3. Columns subsampling: only a random subset of features is tried when **looking for splits**

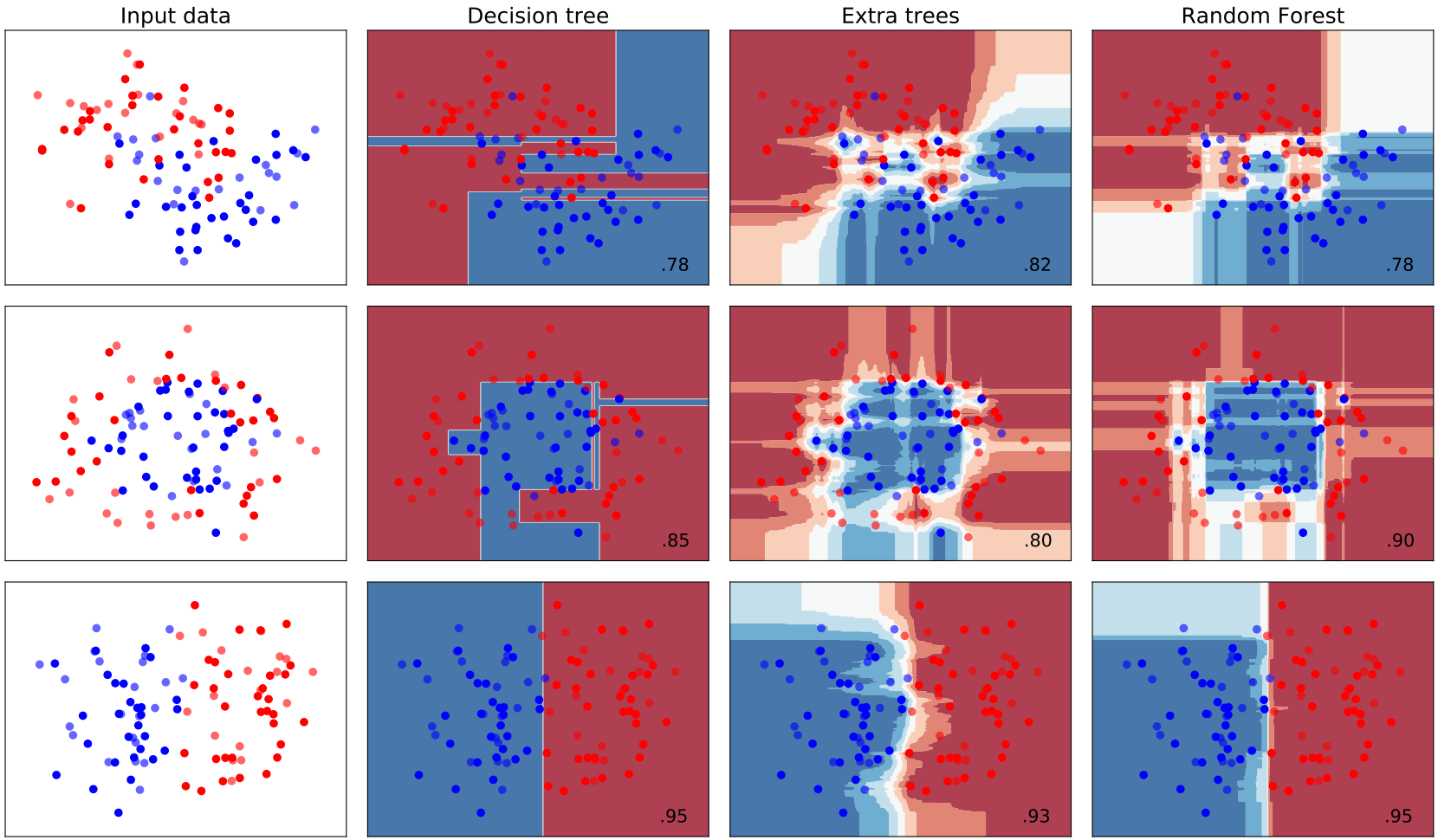
- Also called "features bagging"
- Reduces even further the correlation between trees
- In particular when some features are particularly strong

ExtraTrees: extremely randomized trees

- A variant of random forests
- Features are selected using some impurity (Gini, Entropy)
- Thresholds are selected uniformly at random in the feature range
- Faster, random thresholds are some form of regularization
- (trees are combined and averaged: not so bad to select the thresholds at random...)

Using `scikit-learn`: decision funtions of

- `tree.DecisionTreeClassifier`
- `ensemble.{ExtraTreesClassifier,RandomForestClassifier}`



Boosting for the binary classification problem

- Features $x_i \in \mathbb{R}^d$
- Labels $y_i \in \mathcal{Y}$ with $\mathcal{Y} = \{-1, 1\}$ for binary classification and $\mathcal{Y} = \mathbb{R}$ for regression

Weak classifier

- We have a finite set of "weak classifiers" H
- Each weak classifier $h : \mathbb{R}^d \to \mathcal{Y}$ from H is a **very simple** classifier

Weak classifier examples

- Decision tree that splits a **single feature once**: ""horizontal" or "vertical" hyperplanes. It's called a **stump**
- A **stump** is a tree of **depth** 1
- For regression: linear regression model using one or two features
- For classification: logistic regression using one or two features

A stump

A strong learner

Combine linearly the weak learners to get a stronger one

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

where $\eta_b \in \mathbb{R}$.

- This principle is called **boosting**
- Each $b = 1, \ldots, B$ is called a **boosting step**
- It a **ensemble** method: it combine many weak learners to improve performance

Gradient boosting

- General-purpose method: many choice of losses, many choice of weak learners
- An example is AdaBoost: ADAptive BOOSTing, for binary classification

Look for $h_1, \ldots, h_B \in H$ and $\eta_1, \ldots, \eta_B \in \mathbb{R}$ such that

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

minimizes an empiral risk

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

where ℓ is a loss (least-squares, logistic, etc.)

Problem.

- Even given $\eta_1, \ldots, \eta_B \in \mathbb{R}$ we need to minimize over $|H|^B$ to find the $h_1, \ldots, h_B$
- Size of H is typically $O(d)$ (number of features)

Gradient boosting is a greedy algorithm

Iterative algorithm: at step $b+1$ look for g_{b+1} and η_{b+1} such that

$$(\hat{\eta}_{b+1}, \hat{h}_{b+1}) = \underset{\eta \in \mathbb{R}, h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(y_i, \hat{g}_b(x_i) + \eta h(x_i))$$

and put

$$\hat{g}_{b+1} = \hat{g}_b + \hat{\eta}_{b+1} \hat{h}_{b+1}.$$

This defines a sequence $(\eta_1, g_1), \ldots, (\eta_B, g_B)$.

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 H

Gradient boosting. Replace

$$\operatorname*{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{n} \ell(y_i, \hat{g}_b(x_i) + \eta h(x_i))$$

by

$$\operatorname*{argmin}_{u \in \mathbb{R}^n} \sum_{i=1}^{n} \ell(y_i, \hat{g}_b(x_i) + \eta u)$$

and don't minimize, but do a step in the direction given by:

$$\hat{\delta}_b = \Big(\nabla_u \sum_{i=1}^{n} \ell(y_i, \hat{g}_b(x_i) + u) \Big)_{|u=0}$$

where $\ell'(y, z) = \partial \ell(y, z)/\partial z$. If $\ell(y, z) = \frac{1}{2}(y - z)^2$ this is given by

$$\hat{\delta}_b = \Big[\hat{g}_b(x_1) - y_1 \ \cdots \ \hat{g}_b(x_n) - y_n \Big]^\top$$

Gradient boosting

Obviously $\hat{\delta}_b \neq \begin{bmatrix} h(x_1) & \cdots & h(x_n) \end{bmatrix}^\top$ for all $h \in H$.

So, we find an approximation from H:

$$(\hat{h}, \hat{\nu}) = \underset{h \in H, \nu \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(\nu h(x_i) - \hat{\delta}_b(i) \right)^2.$$

For least-squares this means

$$(\hat{h}, \hat{\nu}) = \underset{h \in H, \nu \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(\nu h(x_i) - \hat{g}_b(x_i) + y_i \right)^2,$$

meaning that we look for a weak learner $\hat{h}$ that corrects the current residuals obtained from the current $\hat{g}_b$

Gradient boosting

1: Put $\hat{g}_0 = \hat{m}$ with $\hat{m} = \text{argmin}_{m \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, m)$
2: **for** $b = 1, \ldots, B$ **do**
3: $\quad \hat{\delta}_b \leftarrow -\Big(\nabla_u \ell(y_i, \hat{g}_b(x_i) + u)\Big)_{|u=0}$ for $i = 1, \ldots, n$
4: $\quad \hat{h}_{b+1} \leftarrow \hat{h}$ with

$$(\hat{h}, \hat{\nu}) = \underset{h \in H, \nu \in \mathbb{R}}{\text{argmin}} \sum_{i=1}^{n} \big(\nu h(x_i) - \hat{\delta}_b(i)\big)^2.$$

5: $\quad \hat{\eta}_{b+1} \leftarrow \text{argmin}_{\eta \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, \hat{g}_b(x_i) + \eta \hat{h}_{b+1}(x_i))$
6: **end for**
7: **return** a boosting learner $g_B(x) = \sum_{b=1}^{B} \eta_b h_b(x)$.

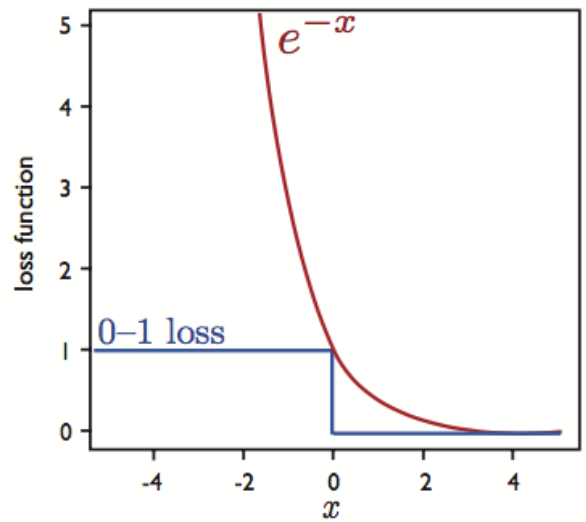
Adaboost

- Gradient boosting with the exponential loss $\ell(y, z) = e^{-yz}$ (binary classification only)
- Construct recursively

$$\hat{g}_{b+1}(x) = \hat{g}_b(x) + \eta_{b+1} h_{b+1}(x)$$

where

$$(h_{b+1}, \eta_{b+1}) = \underset{h \in H, \eta \in \mathbb{R}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} e^{-y_i(\hat{g}_b(x_i) + \eta h(x_i))}$$

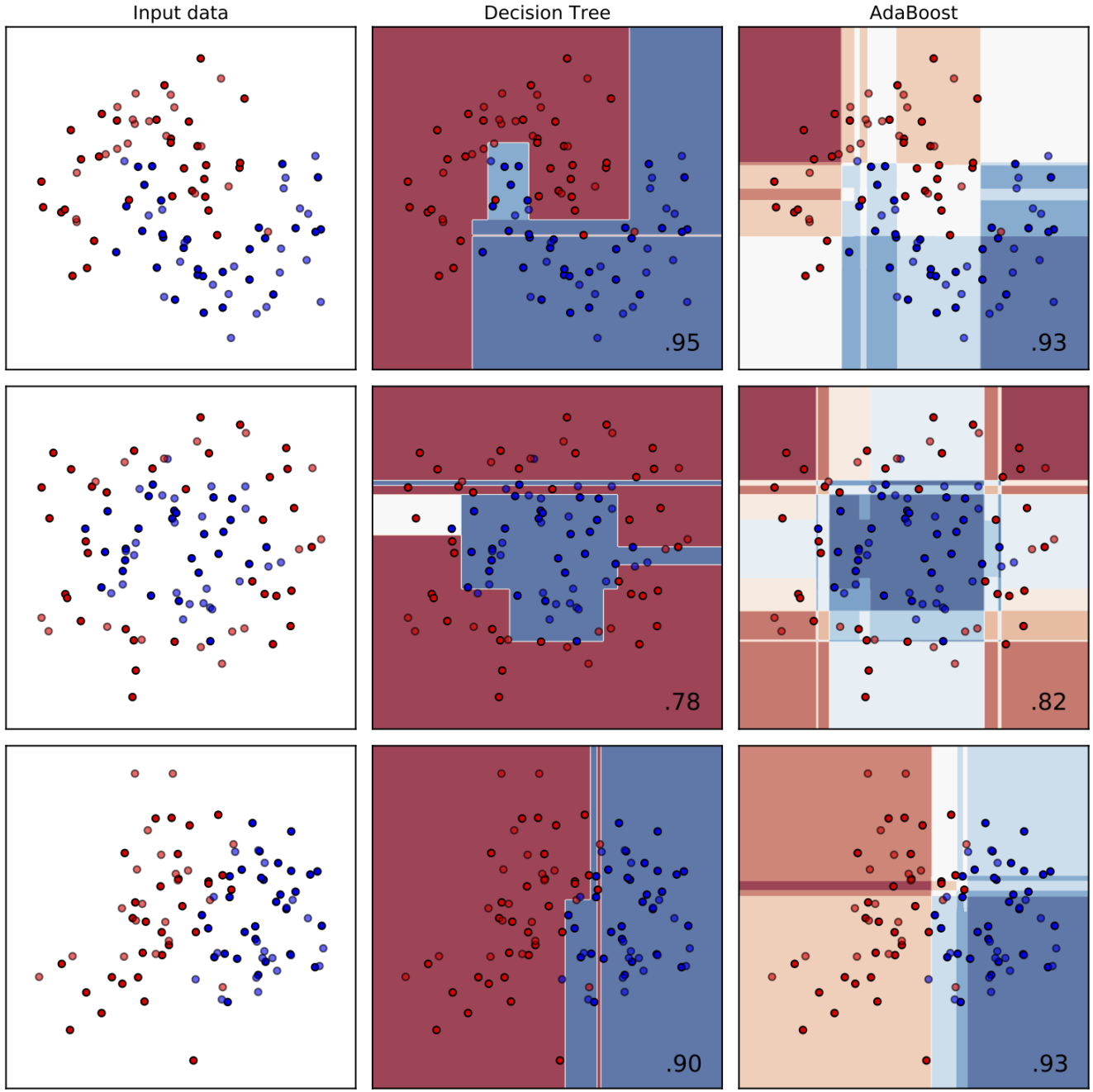


Adaboost

- Quite suprisingly, leads to an **easy** and **explicit** algorithm

Algorithm 2 Adaboost

Put $p_1(i) = 1/n$ for $i = 1, \ldots, n$
for $b = 1, \ldots B$ **do**
  $h_b \in \operatorname{argmin}_{h \in H} \sum_{i=1}^{n} p_b(i) \mathbf{1}_{y_i h(x_i) < 0}$
  $\varepsilon_b \leftarrow \sum_{i=1}^{n} p_b(i) \mathbf{1}_{y_i h_b(x_i) < 0}$
  $\eta_b \leftarrow \frac{1}{2} \log\left(\frac{1-\varepsilon_b}{\varepsilon_b}\right)$
  $Z_b \leftarrow 2\sqrt{\varepsilon_b(1-\varepsilon_b)}$
  $p_{b+1}(i) \leftarrow p_b(i) \dfrac{e^{-\eta_b y_i h_b(x_i)}}{Z_b}$
end for
return A boosting classifier $g_B(x) = \operatorname{sgn}\left(\sum_{b=1}^{B} \eta_b h_b(x)\right)$


Input data
Decision Tree
AdaBoost
.95
.93
.78
.82
.90
.93

Choice of hyper-parameters

For dictionary H

- If H contain trees, choose their depth (usually small... no more than four or five)
- If H contains generalized linear models (logistic regression), select the number of features (usually one or two)
- Number of iterations B (usually a few hundreds, few thousands, until test error does not improve)

Check `XGBoost`'s and documentation for more details on the hyper-parameters

State-of-the-art implementations of Gradient Boosting

- With many extra clever tricks...

`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`

Grand mother's recipe

Grand mother's recipe. If you have "tabular" data (no image, signal, text, etc.)

- Spend time on **feature engineering**
- Spend time on **feature engineering**
- ...
- Spend time on **feature engineering**
- Try out RF or GBM methods before diving into a deep learning method (tomorrow)

This afternoon: practical session

- Prediction of load curves
- Continuous labels / time series
- Using several things, including `XGBoost` and `LightGBM`

Thank you!