5. Tree-based and Ensemble Learning Methods

Jesper Armouti-Hansen

University of Cologne

January 21, 2019

Course website

jeshan49.github.io/eemp2/

Today

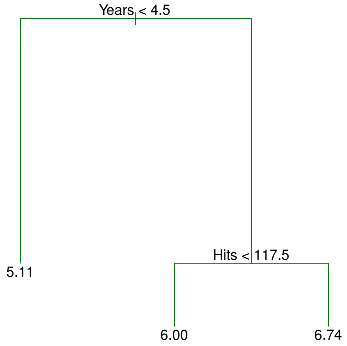
- Lecture[1]:
  - Decision Trees
  - Ensemble Learning:
    - Bagging
    - Random Forest
    - Boosting

[1]Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani

Introduction to Decision Trees

- Decision Trees (DTs) are versatile ML algorithms that can perform both classification and regression tasks
- a DT is a tree-based method – these involve *stratifying* or *segmenting* the feature space into a number of simple regions
- After this split, we typically make predictions based on the mean or mode response value in the regions
- The set of splitting rules used to segment the feature space can be summarized in a tree
- DTs are also the fundamental components of *Random Forests*, which are among the most powerful ML algorithms available
- We will also consider other powerful algorithms applicable to Decision Trees: *boosting* and *bagging*

Decision Trees - Regression



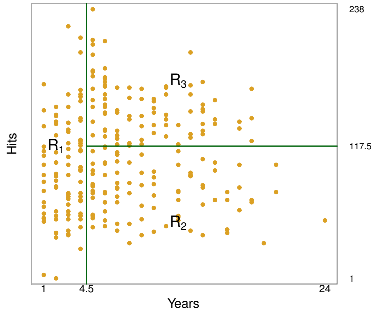


Figure: Right: A regression tree for predicting the log salary of a baseball player, based on years in major league and number of hits made in the previous year- Left: The three-region partition on the feature space from the regression tree (See ISLR pp. 304–5)

Terminology

- The regions (e.g., R_1, R_2, R_3) are known as *terminal nodes* or *leaves* of the three
- The points along the three where the feature space is split are known as *internal nodes* (e.g., Years<4.5 and Hits<117.5)
- The initial node (Years<4.5) is also sometimes referred to as the *root node*
- The segments of the tree that connects the nodes are *branches*
- Note: The tree is usually displayed upside down

Prediction via Stratification of the Feature Space

- How do we build regression trees?
- Roughly speaking, there are two steps:
  1. We divide the feature space, $X_1, \ldots, X_p$ into J distinct and non-overlapping regions, $R_1, \ldots, R_J$
  2. For every observation that lies in R_j, we predict the mean of the response values for the training observations in R_j
- But how do we divide $X_1, \ldots, X_p$ into $R_1, \ldots, R_J$?
  - We find $R_1, \ldots, R_J$ that minimizes

$$\sum_{j=1}^{J} \sum_{i \in R_j} \left(y_i - \hat{y}_{R_j}\right)^2 \tag{1}$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth region

- Unfortunately, that division approach is computational infeasible (it is a NP-Complete problem)
- Instead, one usually apply a *top-down, greedy* approach known as *recursive binary splitting*
- We choose the feature X_j and the cutpoint s such that the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ lead to the largest possible reduction in RSS
- That is, we seek j, s that define the half-spaces

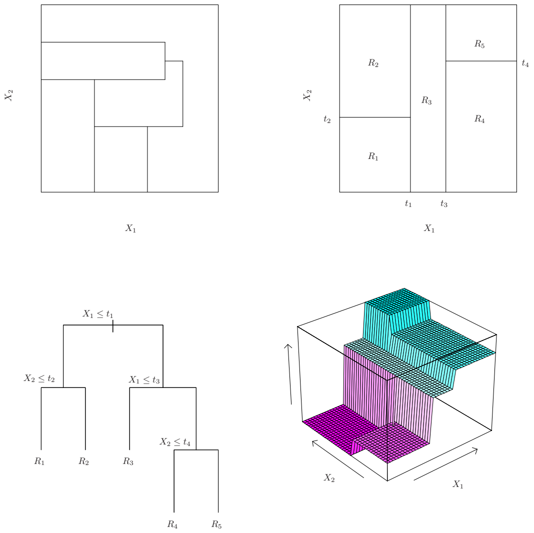
$$R_1(j,s) = \{X|X_j < s\}, R_2(j,s) = \{X|X_j \geq s\} \tag{2}$$

such that we minimize

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2 \tag{3}$$

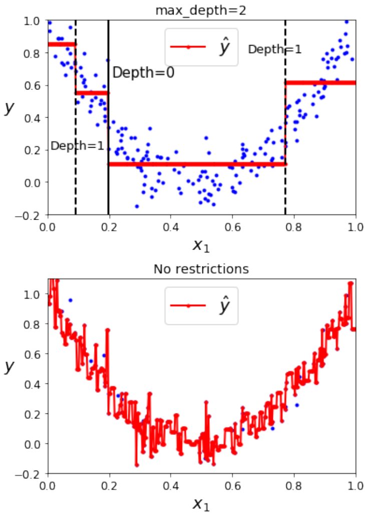
- We iterate the process of splitting the (sub)spaces until a stopping criterion is reached (e.g. *max depth* or *min observations per leaf*)
- If no such criterion is given, the process will continue until no improvement can be made

Example



X_2
X_1
R_2
R_5
t_4
R_3
t_2
R_4
R_1
t_1
t_3
X_2
X_1
$X_1 \le t_1$
$X_2 \le t_2$
$X_1 \le t_3$
$X_2 \le t_4$
R_1
R_2
R_3
R_4
R_5
X_2
X_1

Example – Simple regression with/without restrictions



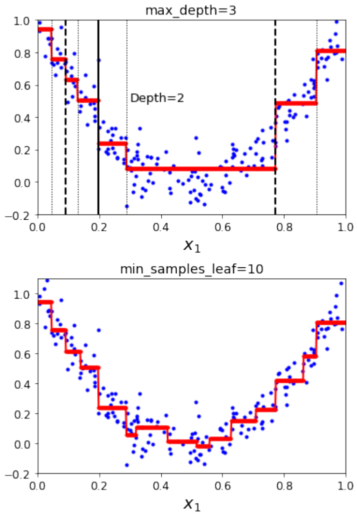


Figure: See HOML p. 178

Cost Complexity Pruning (CCP)

- Rather than considering every possible subtree of a large complex tree T_0, we consider a sequence of subtrees indexed by $\alpha \geq 0$
- For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i: x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \tag{4}$$

is minimized

Algorithm 1 Building a Regression Tree with CCP

```
1: Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer that some minimum number of observations.
2: Apply CCP to the large tree in order to obtain a sequence of best subtrees, as a function of $\alpha$.
3: Use k-fold CV to choose $\alpha$.
4: Return the subtree from Step 2 that corresponds to the chosen value of $\alpha$.
```

Alternatives in Scikit-Learn

- Unfortunately, tree pruning is not yet available in Scikit-Learn

- However, DTs are rarely used on their own, so this is not a major drawback

- We may consider alternative approaches:
  1. Create a grid of candidate values for a range of criteria such as *min depth* and *min observations per leaf* etc.
  2. For each possible combination of values, perform k-fold CV to find the optimum

  - If the number of combinations becomes to large such that grid search CV is infeasible, then draw a random sample of fixed size and perform k-fold CV

- We then may approach the optimal pruned tree

Decision Trees - Classification

- A Decision Tree for classification is build the same way as with regression
- We apply the recursive binary splitting algorithm to sequentially segment our feature space
- Naturally, we will no longer be minimizing the RSS. We could use the misclassification rate:
$$E_m = 1 - \max_k(\hat{p}_{mk}) \tag{5}$$
where $\hat{p}_{mk}$ is the proportion of observations in the mth region belonging to the kth class – our estimate of p_{mk}
- However, it turns out that E_m is suboptimal for tree-growing – We rather need a measure of node impurity

Gini Index – Node impurity

- In practice we use the *Gini index* to build trees
- The *Gini index* value in region R_m is a measure of total variance across the K classes and is given by

$$G_m = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{6}$$

  which is often referred to as *node impurity* – a small value indicates that a node contains predominantly observations from single class
- Thus, when deciding to split X, we seek j and s that define half-spaces

$$R_1(j,s) = \{X|X_j < s\}, R_2(j,s) = \{X|X_j \geq s\} \tag{7}$$

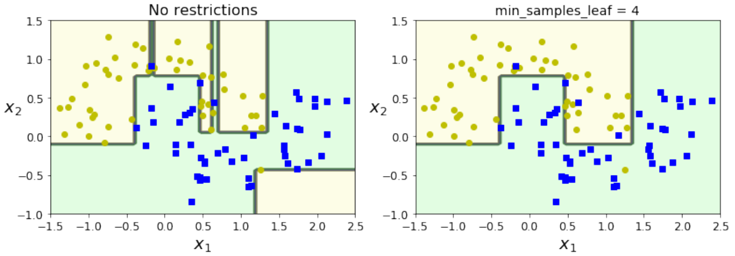
  such that we minimize the weighted sum of node impurity:

$$\frac{N_1}{N} G_1 + \frac{N_2}{N} G_2 \tag{8}$$

- If we cannot gain purity in making this split, the preceding node is declared as a terminal node/leaf

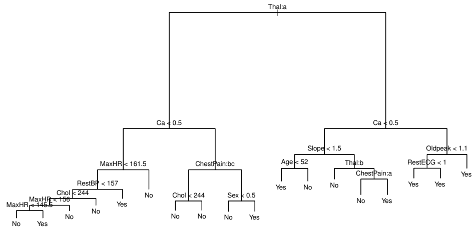
Example – Classification with/without restrictions

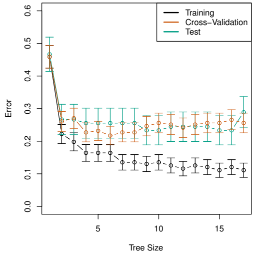
- As with regression trees, we risk overfitting if we do not specify terminal criteria

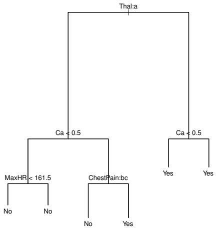


- As with regression trees, we can prune the classification tree with CCP

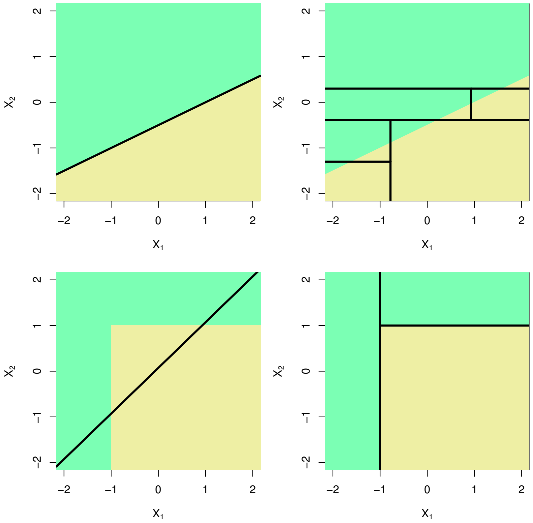
Example – Pruning a Classification Tree with CCP







Trees vs. Linear Models



Instability

Decision Trees are

- simple and easy to interpret, as well as versatile

However,

- they are very sensitive to small variations in the data
    - removing one observation may substantially change the tree
    - thus, they have high variance
- The decision boundaries have to be orthogonal to the feature axis
    - they are sensitive to the rotation of the data (PCA may help)
- In many applications, their predictive ability is below that of other well-known ML methods
- However, as we shall see, we can use DTs as building block to construct more powerful methods

Ensemble Learning

- Consider the following simplified version of the *Condorcet Jury Theorem*:
- Suppose there are N voters on a jury
- For simplicity, let each voter's probability of being correct be p
- Furthermore, let M be the probability that the majority is correct
- Then, under certain conditions,

$$p > \frac{1}{2} \Rightarrow M > p \tag{9}$$

and

$$\lim_{N \to \infty} M \to 1 \tag{10}$$

- Thus, we can think of training a set of methods (or different subsets of the training data) to improve our prediction accuracy – This is (roughly) what ensemble learning methods are about

Voting Classifiers

- **Voting classifiers**: Combining different classifiers to one
- For example, we train a
  1. kNN classifier,
  2. Logistic regression,
  3. LDA, and
  4. Decision Tree

  on our training data set
- To make a final classification, we aggregate these in to a majority-vote classifier
  - Hard voting: For an observation, predict the class with most "votes"
  - Soft voting: If each classifier can predict probabilities, calculate the mean probability of each class, and pick the class corresponding to the highest mean probability
- Since the effectiveness of the previous result is larger for independent voters, choosing diverse classifiers is recommended

Example – Hard Voting Classifier

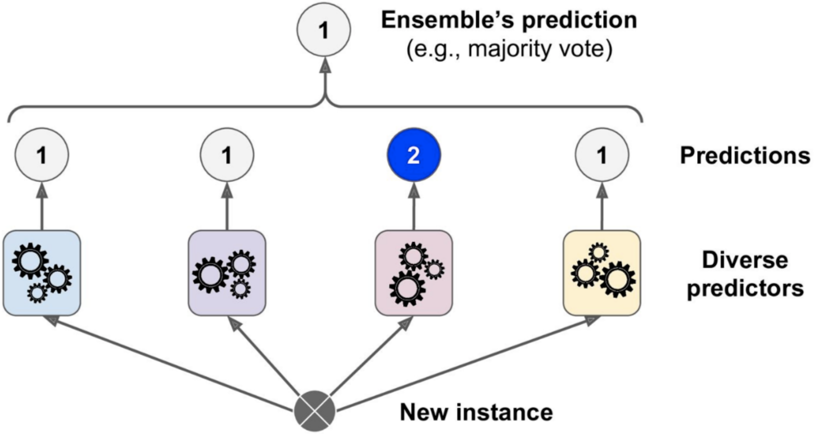


Figure 7-2. Hard voting classifier predictions

Bagging (Bootstrap Aggregating)

- The effectiveness of the voting classifier comes from using a set of diverse (low correlated) classifiers, which in turn lowers the variance of our estimate at a point x_0
  - As an illustration, suppose we have N independent observations $Z_1, \ldots Z_n$ with variance σ^2
  - Then the variance of the mean $\bar{Z}$ is given by σ^2/N
- Another way to decrease the variance would then be to train a method (e.g. a Decision Tree) on B different training samples drawn from the population
- We then train the method on each of the sets, and then predict according to

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^b(x) \tag{11}$$

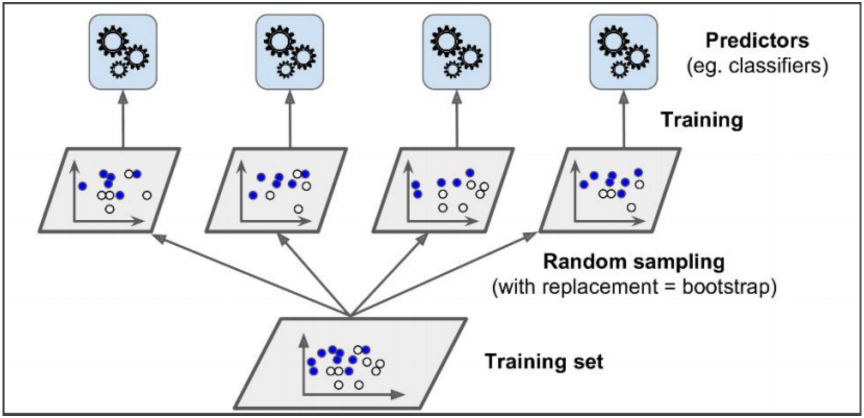
  in the regression setting, and by majority voting in the classification setting

- Usually, we do not have access to multiple training sets
- Instead, we could split up our training set into B partitions and train a model on each set – This is called *pasting*
- Alternatively, we could sample B sets, with replacement, from our training set to create B bootstrap samples with the same size as our training set
- Then we train a method on each set and then predict according to

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{b}(x) \tag{12}$$

  in the regression setting, and by majority voting in the classification setting
- With Decision Trees, we grow each tree deep and leave them unpruned
- Thus, they have high variance, but low bias – Averaging then lowers the variance

Example – Bagging



Example – Bagging

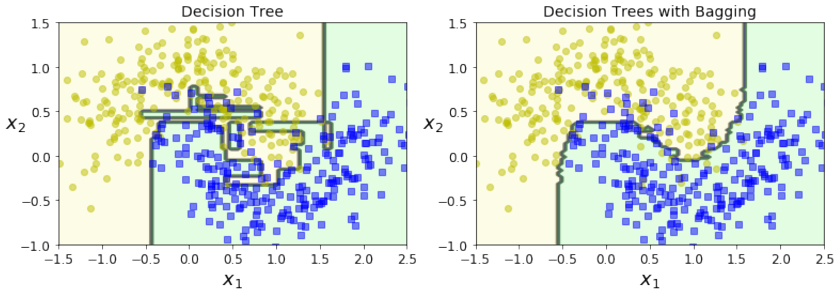


Figure: A single Decision Tree vs. a bagging ensemble of 500 trees (See HOML p. 189)

Out-of-Bag (OOB) error estimation

- Note that the bagged trees are repeatedly fit to bootstrap subsets of the observations
- Thus, on average, each bagged tree makes use of approx. 63.2% of the training observations
- We can predict the response of the ith observation using each bagged tree in which the observation was not used (OOB)
- We will then have approx. $B/3$ predictions for each observation in the training data
- We then average (regression) or use majority voting (classification) to make a prediction
- The resulting OOB error is a valid estimate of the test error and a convenient alternative to CV in this setting

Variable Importance Measures

- Although we can significantly increase accuracy, we loose interpretability when bagging trees
- It is no longer immediately clear which variables are most important to the procedure
- However, it is actually possible to obtain an overall summary of the importance of each predictor using RSS (regression) or the Gini index (classification)
- For example, for bagged regression trees, we can record the total amount that RSS (1) decreases due to splits over a given feature, averaged over all B trees
- If the value is relatively high for a given feature, then this feature is important
- We will see in the tutorial that we can get a graphical representation

Random Forest

- A Random Forest is an ensemble of Decision Trees, generally trained via the bagging method
- However, it provides an improvement over bagged trees by using a small tweak that further decorrelates the trees
    - Recall that ensemble methods increase in accuracy as correlation decreases
- The trick: each time a split in a tree is considered, a random sample of $m < p$ predictors are considered
- Thus, instead of searching for the best feature when splitting a node, it searches for the best among the random sample of m predictors
- The result is greater tree diversity which trades of a higher bias with lower variance compared to bagged trees
- Often, one sets $m = \lceil \sqrt{p} \rceil$

Boosting

- Boosting is an ensemble method that can combine several *weak learners* into one *strong learner*
    - Weak learner: Computationally simple method, that performs slightly above chance
- Idea: Train learners sequentially on the (modified) training data, where each succeeding learner tries to correct its predecessor
- In general, learning methods that learn slowly tend to perform well
- We will consider the two most popular boosting methods:
    - *AdaBoost*
    - *Gradient Boosting*

AdaBoost (Adaptive Boosting)

- A way for a new learner to correct its predecessor it to pay more attention to the training instances that the predecessor underfitted
- Thus, at each iteration, AdaBoost changes the sample distribution by modifying the weights attached to each of the instances
- It increases the weights of the wrongly predicted instances and decreases the ones of the correctly predicted instances
- The following weak learner thus focuses more on the difficult instances
- Once all learners are trained, the AdaBoost makes predictions very much like bagging, except that the learners have different weights depending on their overall accuracy on the weighted training set

AdaBoost – Regression illustration

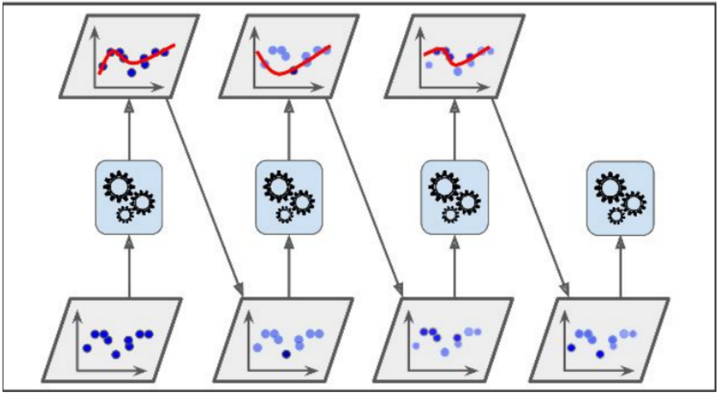
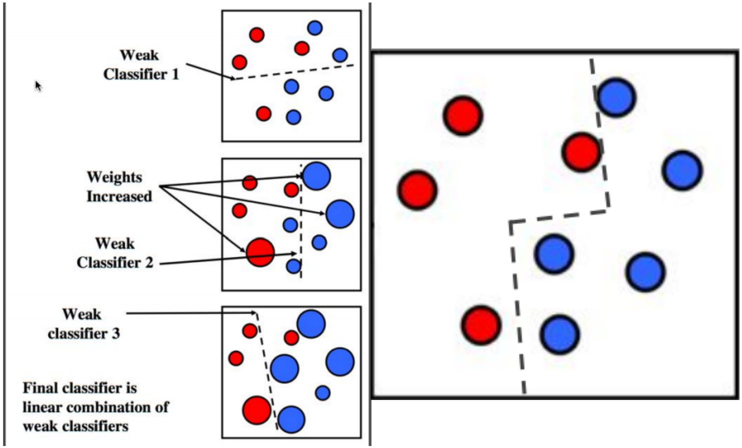


Figure: AdaBoost sequential training with instance weigh updates (See HOML p. 189)

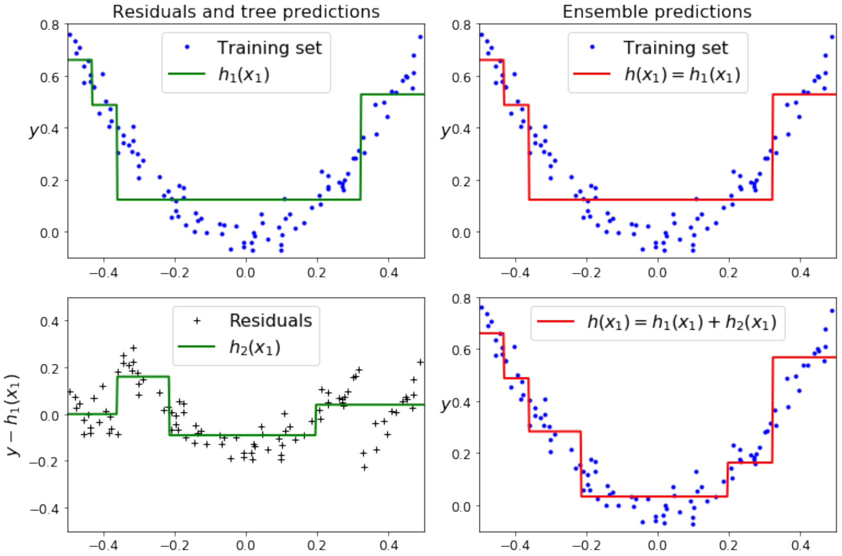
AdaBoost – Classification illustration



Gradient Boosting

- Just like AdaBoost, Gradient Boosting works by sequentially adding learners to an ensemble
- Each of the learners attempts to correct for its predecessor
- However, this method tries to fit the new predictor to the residual errors made by the previous predictor
- That is, if $\hat{f}_1$ is trained on (X, y), then $\hat{f}_2$ is trained on (X, r_1) where $r_1 = y - \hat{f}_1$, and so on
- One may then stop the collection if no further improvement is observed for several rounds, or once a certain number of trees have been reached

Gradient Boosting – Example



References

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112). **Chapter 8**

Géron, A. (2017). Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. **Chapters 6 & 7**