CLASSIFICATION TREES

Jelena Jovanovic

Email: jeljov@gmail.com

Web: http://jelenajovanovic.net

Acknowledgement:

These slides are based on the materials from the "Applied Modern Statistical Learning Techniques" course (link), as well as on Chapter 8 of the "Introduction to Statistical Learning" book (link)

Example: Classification of baseball players

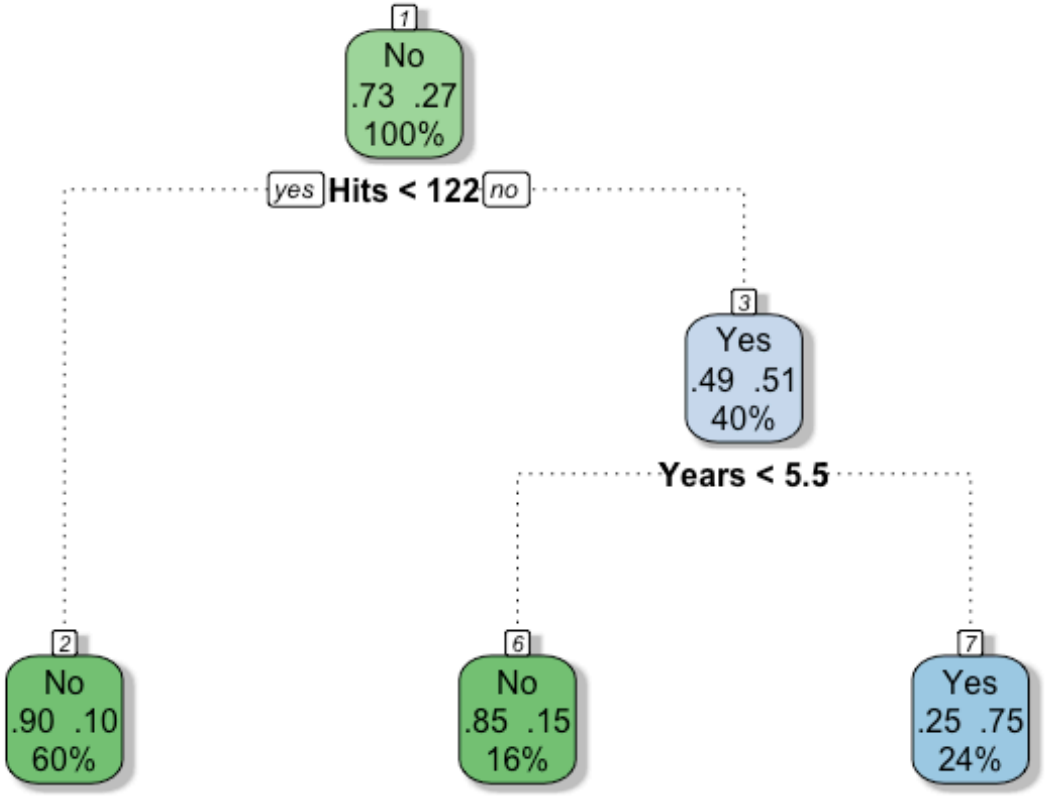
The task is to classify baseball players into those who are well paid and those who are not (WellPaid), based on

- the number of hits in the previous year (Hits), and
- the number of years the player has spent in the main league (Years)

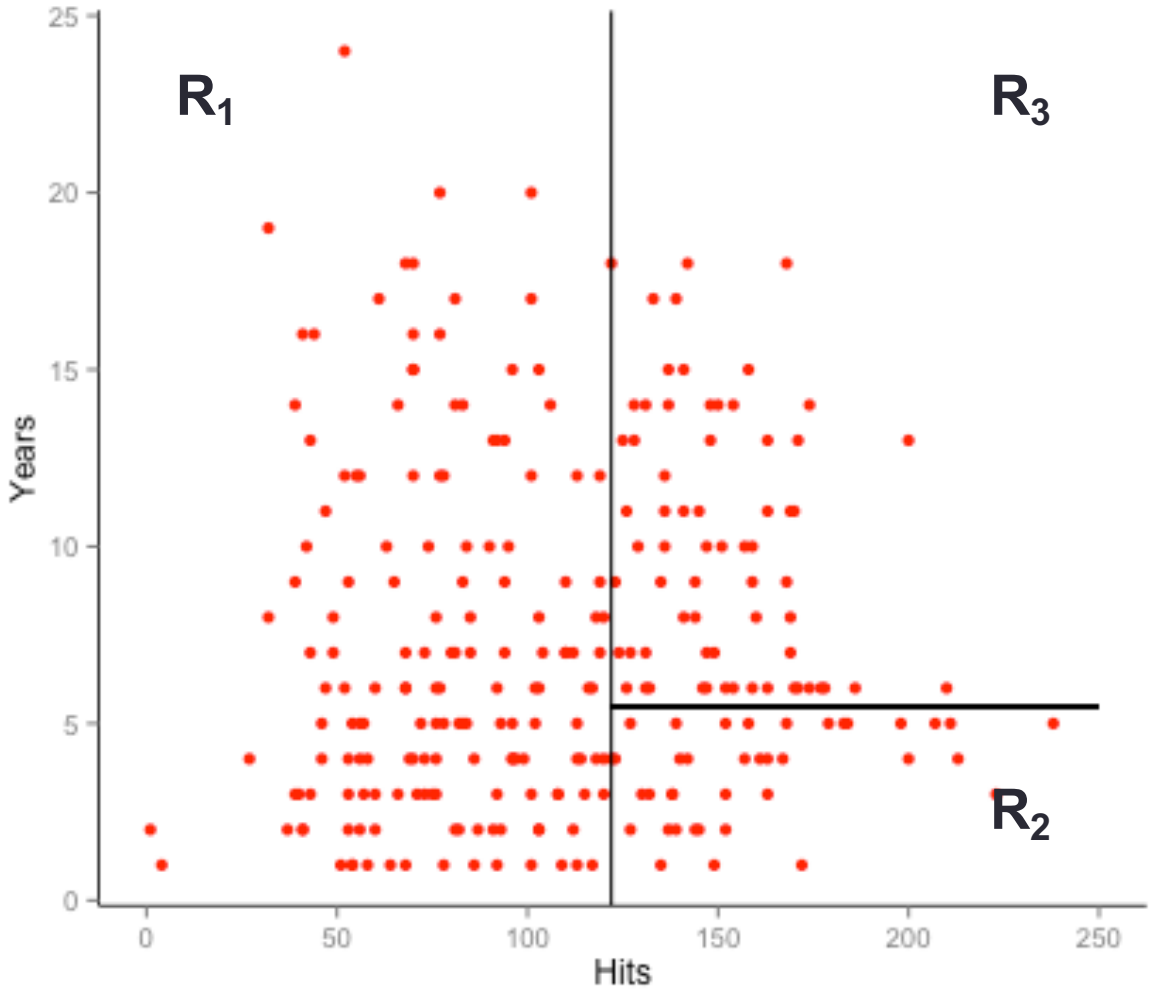
```
Console ~/R Studio Projects/Intelligent Systems Fall 2015/
> str(hitters.subset)
'data.frame':   263 obs. of  3 variables:
 $ Hits    : int  81 130 141 87 169 37 73 81 92 159 ...
 $ Years   : int  14 3 11 2 11 2 3 2 13 10 ...
 $ WellPaid: Factor w/ 2 levels "No","Yes": 1 1 1 1 2 1 1 1 2 1 ...
> head(hitters.subset)
                 Hits Years WellPaid
-Alan Ashby        81    14       No
-Alvin Davis      130     3       No
-Andre Dawson     141    11       No
-Andres Galarraga  87     2       No
-Alfredo Griffin  169    11      Yes
-Al Newman         37     2       No
>
```

Example: Classification of baseball players

- The obtained classification tree indicates that well paid are those players who scored at least 122 hits in the previous season and who have been playing in the major league for at least 5.5 years
- The probability that a player with the given features is well paid is 0.75
- Such players form 24% of all the players for whom the data is available (the training set)



Another way of visualizing the classification tree…



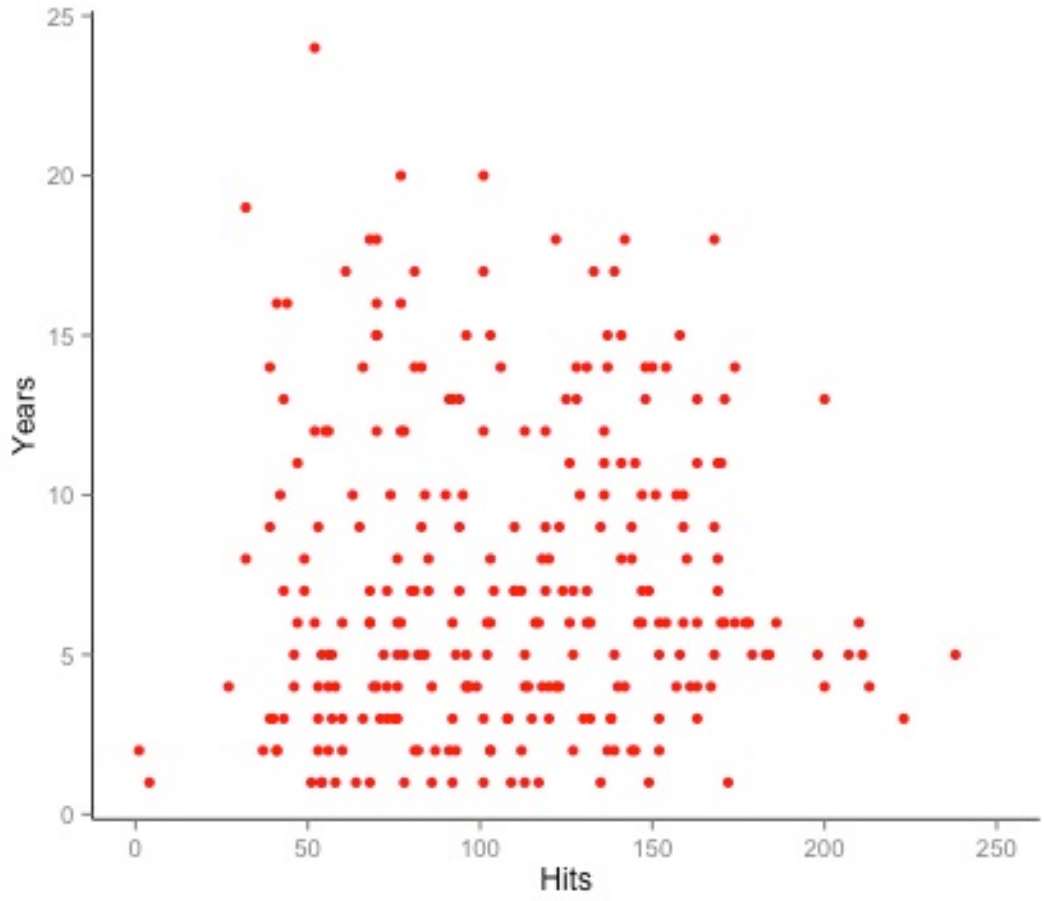
The main idea behind classification trees

- Divide the *predictor space* into multiple distinct (non-overlapping) regions R_1, R_2, …, R_n
  - predictor space is a *p*-dimensional space comprising all possible values of the *p* attributes (x_1,x_2,…,x_p) that describe the observations we have
- A new observation X will be assigned to one of the regions R_1…R_n based on the values of its attributes (x_1,x_2,…,x_p)
- The class of the new observation X will be the most dominant class (*the majority class*) in the region R_j that X was assigned to

Splitting the predictor space

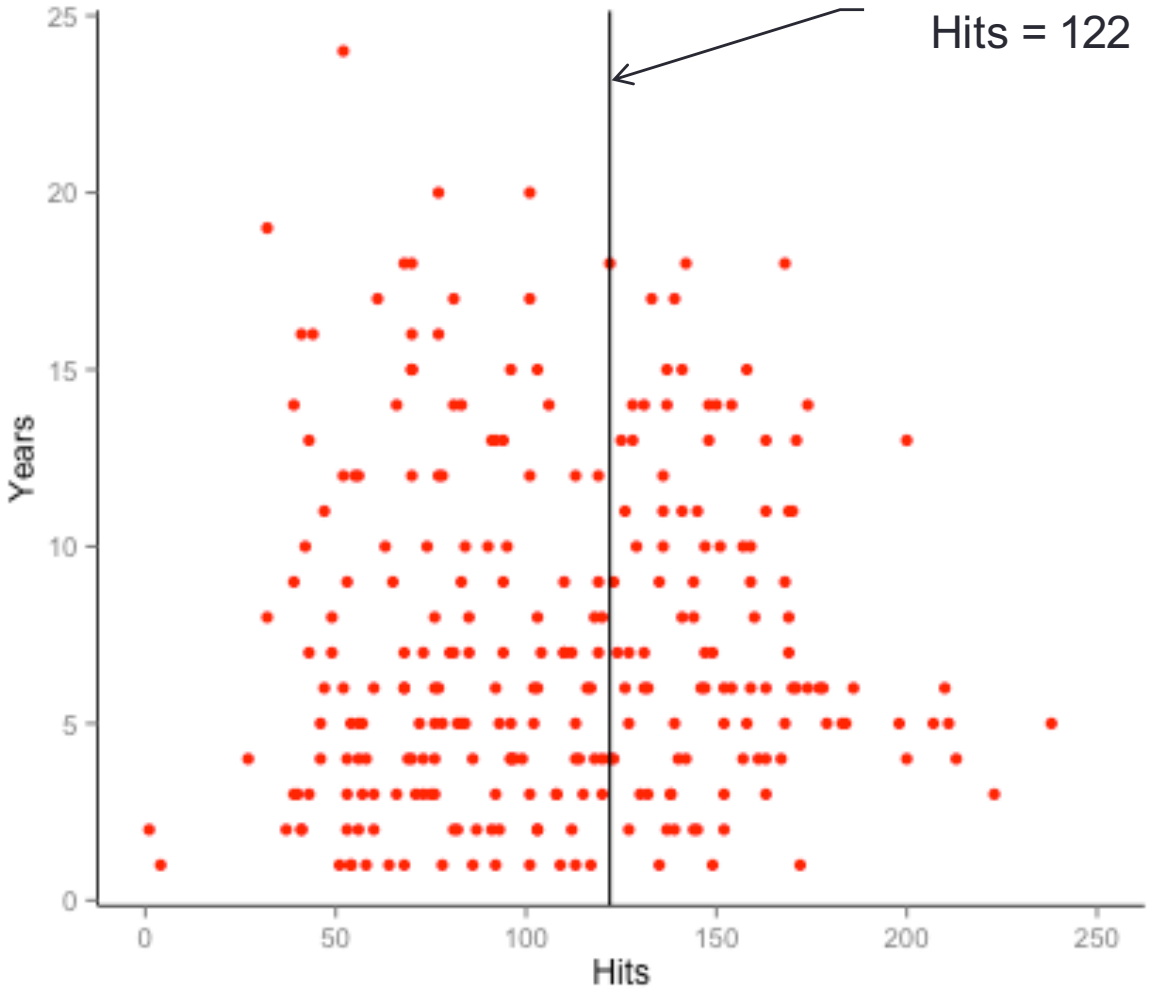
The division of the predictor space into regions R_j is an iterative process where each iteration includes:

- selection of the attribute x_i where the split will occur
- selection of the attribute x_i's value that will serve as the 'threshold'



Splitting the predictor space

For the first split, in the baseball players example, attribute Hits was chosen and its value of 122 is set as the threshold



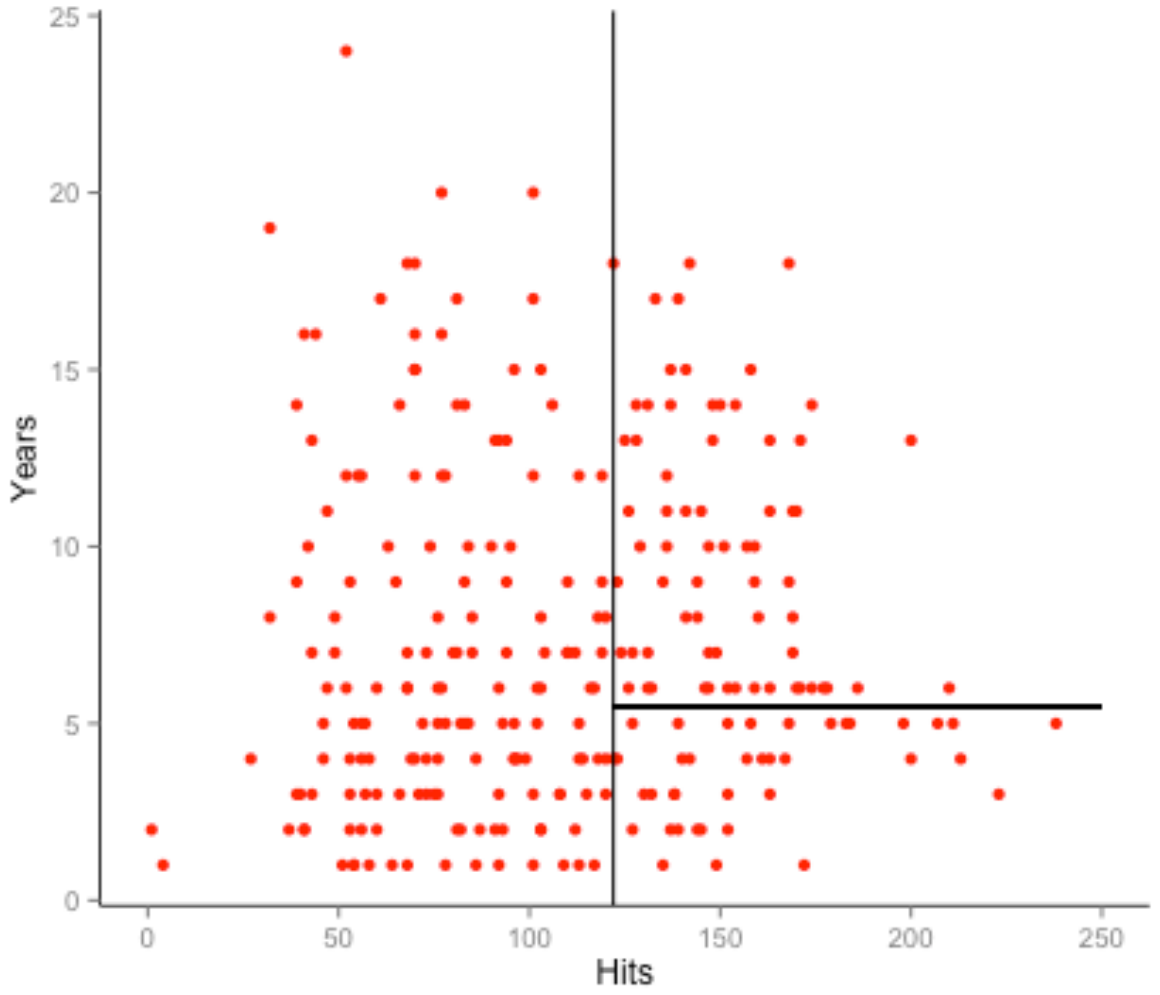
Splitting the predictor space

The first split:

Hits = 122

If Hits > 122, the next split will be on the Years attribute:

Years= 5.5



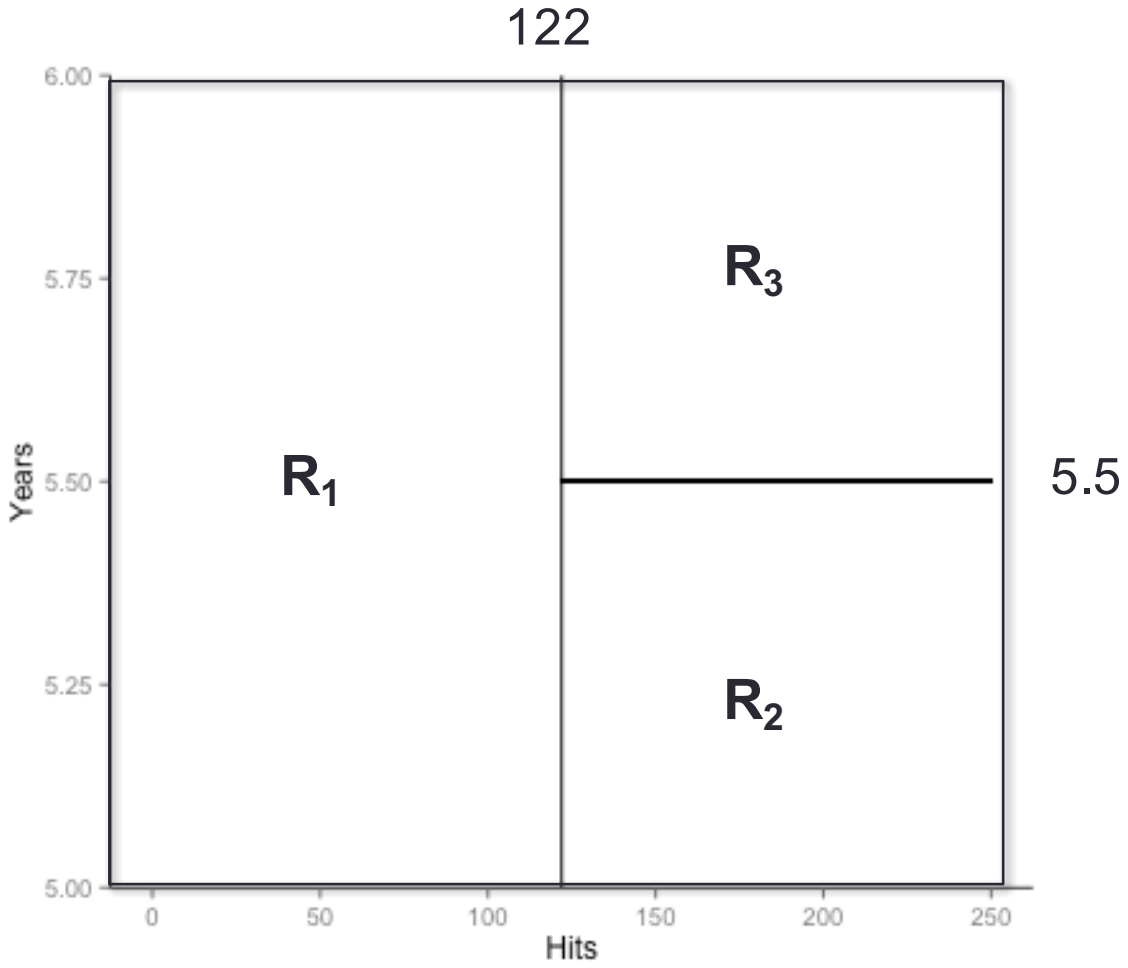
Splitting the predictor space

The first split:

Hits = 122

If Hits > 122, the next split is:
Years = 5.5

Leading to the division of the predictor space into 3 regions



Splitting the predictor space

Questions that naturally occur:

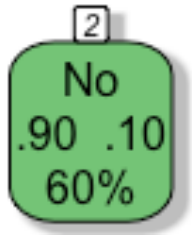
- Where to split and how?
  - In other words, how to create regions R_1, R_2,…,R_n?
- How to determine the class to be assigned to the observations in each of the regions R_1,..,R_n?

How to determine the class of observations in each region?

Using the *majority class* principle:

The class to be associated with the region R_j is the class of the majority of observations (from the training set) that belong to region R_j

In the baseball players example, 90% of observations in region R1 are players who are not well paid => any new player who ends up in region R1 will be classified as not well paid player



Where and how to split?

The goal is to identify regions R_1, R_2,…,R_n that would minimize *Classification Error Rate* (CER)

CER represents the proportion of observations (from the training set) in the given region that do not belong to the dominant (majority) class of that region

$$CER = 1 - \max_k \hat{p}_{ik}$$

$\hat{p}_{ik}$ is the proportion of (training) observations in region *i* that belong to the class *k*

Where and how to split?

- To identify the regions that minimize the classification error, we apply an approach that is based on *recursive binary splitting* of the predictor space
- This approach can be characterized as:
  - *top-down*
  - *greedy*

Recursive, binary split of the predictor space

- The approach is *top-down*
  - Starts from the root of the tree, where all (training set) instances belong to one (common) region, and then, iteratively, spits the predictor space into regions
- It is a *greedy* approach
  - At each step, the best split is selected based only on the current state of the tree
  - The algorithm does not consider the outcomes of possible future steps – it does *not* try to find a split that might lead to the best results in some future step

Recursive, binary split of the predictor space

The algorithm considers each attribute x_j (j=1,p) and each cut-point s_j for that attribute, and

chooses the combination (x_j, s_j) that splits the predictor space into two regions $\{X|x_j > s_j\}$ and $\{X|x_j < s_j\}$ in such a way that the resulting tree has the lowest classification error

Where and how to split?

Besides the *Classification Error Rate*, the following metrics are often used as the criteria for splitting the predictor space:

- Gini index
- Cross-entropy

Gini index

- It is defined as follows:

$$G = \sum_{k=1}^{K} \hat{p}_{ik} (1 - \hat{p}_{ik})$$

$\hat{p}_{ik}$ represents the proportion of (training set) observations in region *i* that belong to class *k*

- It is often described as the measure of *node purity*
  - 'pure' nodes are those with high percentage of observations belonging to the same class
  - small Gini value is an indication of a 'pure' node

Cross-entropy

- Defined as follows:

$$D = -\sum_{k=1}^{K} \hat{p}_{ik} \, log \hat{p}_{ik}$$

- As Gini index, Cross-entropy represents a measure of the node's 'purity'
  - the smaller the value, the 'purer' the given node is

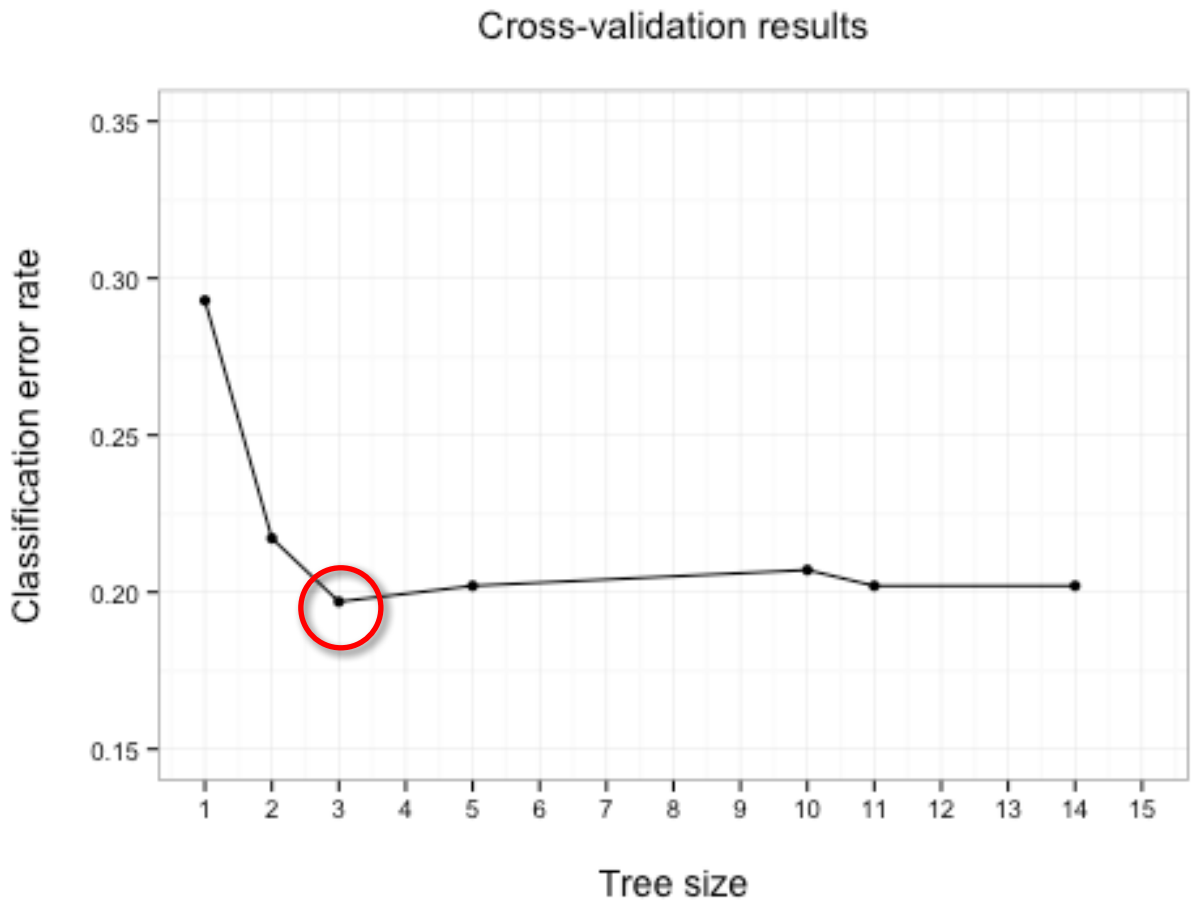
Tree pruning

- A large tree, that is, one with many terminal nodes, may tend to over-fit the training data
- This problem can be resolved by “pruning” the tree, that is, cutting off some of the terminal nodes
- How do we know how far back to prune the tree?

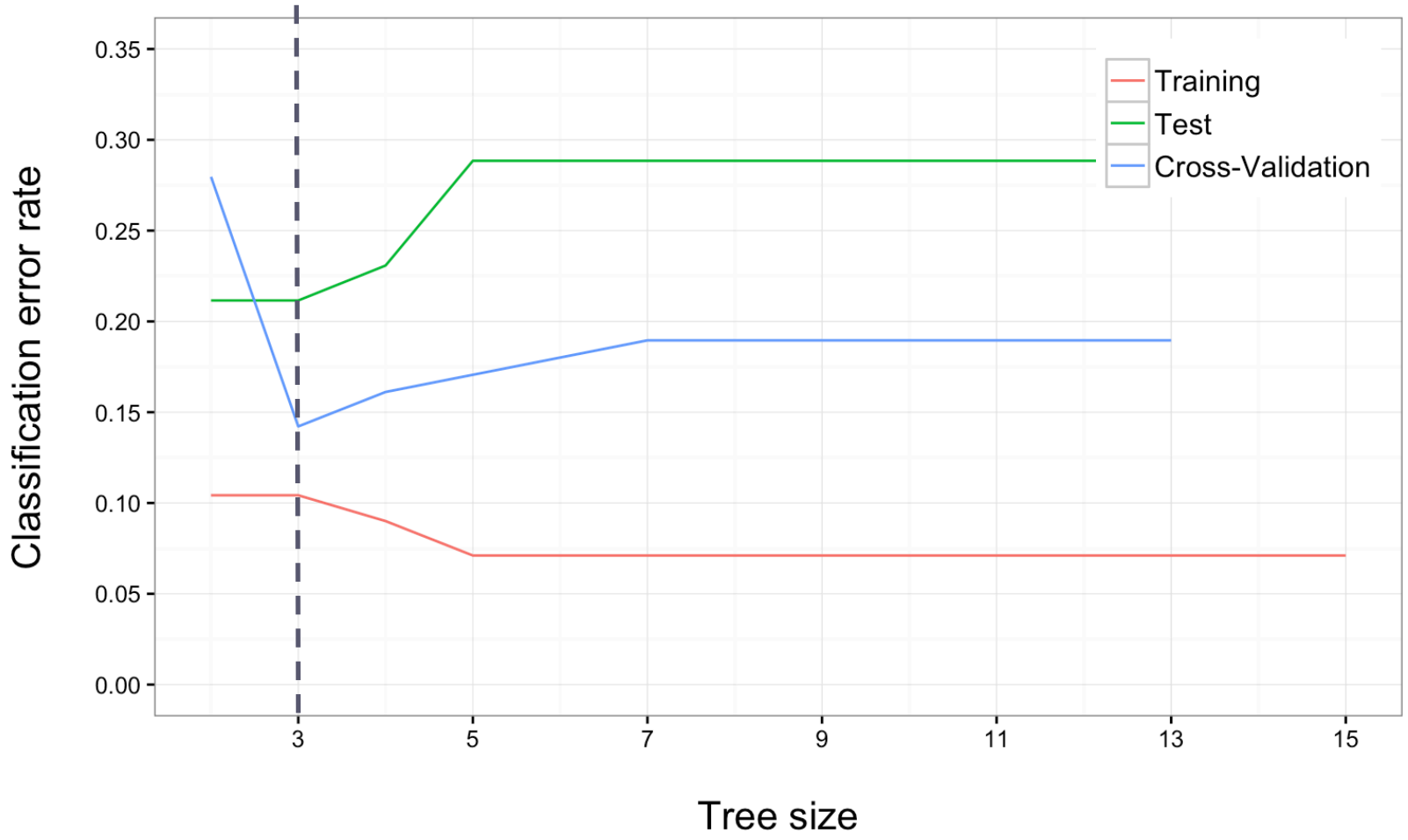
It is recommended to use *cross validation* to determine which tree has the lowest error rate

Tree pruning through cross validation

In the example of classifying baseball players, cross validation identifies the tree of size 3 (i.e., with 3 terminal nodes) as the one with the lowest classification error



Tree pruning through cross validation



The chart confirms that the tree size identified through cross validation (n=3), leads to the lowest error rate on the test set

Pros and Cons of Decision Trees

- Pros:
  - Trees can be plotted graphically, and can be easily interpreted even by non-experts
  - They work fine on both classification and regression problems
  - They can be applied even on attributes with missing values
- Cons:
  - They are not as performant as some other methods of supervised learning (they have lower prediction accuracy)