FITPED | AI

Machine Learning

Published on

Work in progress version

Erasmus+ FITPED-AI

Future IT Professionals Education in Artificial Intelligence

Project 2021-1-SK01-KA220-HED-000032095

TABLE OF CONTENTS

Introduction to machine learning

Chapter 1

1.1 Introduction

🕮 1.1.1

Can a machine learn new knowledge?

The objection to artificial intelligence is that machines can hardly be considered intelligent unless they **can learn new knowledge** and **adapt to new situations**. The fact that **systems act as they are prescribed** does not help either.

Machine learning can be defined as follows:

"Learning is any process by which a system improves performance based on experience."

or "Learning is changes in a system that are adaptive in the sense that they allow the system to accomplish the same task or tasks from the same class of tasks a second time more efficiently and effectively." (Simon, 1983)

This learning includes:

- **Skill refinement** - improvement in solving many tasks just by doing them more times.
- **Knowledge acquisition** - knowledge is generally acquired through experience.

🕮 1.1.2

Machine learning algorithms can solve the following groups of problems:

- **A group of problems for which there are no human experts**.

For example, in modern manufacturing facilities, it is necessary to **predict machine failures** before they actually occur based on sensor analysis. Because the machines are new, **there is no expert to give** the programmer in question **all the knowledge** needed to create a computer system. A system built on machine learning **can study the recorded data** and infer prediction rules for subsequent machine failures.

- **A group of problems where experts exist, but are unable to explain their expertise**.

This is the case for many **recognition tasks**, such as speech recognition, handwriting recognition, and natural language understanding. In fact, **all humans demonstrate expert ability** to solve these tasks, but none of them are able to describe in detail the steps they apply in solving them. Fortunately, **humans can provide machines with examples of inputs and correct outputs** for these tasks, so machine learning algorithms can learn to map inputs to correct outputs.

- **A group of problems where circumstances change rapidly**.

In finance, for example, people would like to **predict future stock market developments**, consumer purchases or currency exchange rates. This **data changes quite rapidly**, so even if a programmer could create a good prediction program, it would have to be rewritten frequently. A learning program can relieve the programmer from constant modification and debugging by creating a set of prediction rules learned by learning.

- **A group of applications that must be configured for each user separately**.

Consider, for example, **a program for filtering unwanted e-mail**. Each user will need different filters. It is unreasonable to expect each user to define their own rules, and it is also unfeasible to have a **software engineer available to each user** to update their rules. A system using machine learning is able to learn which emails a user rejects and thus maintain filtering rules automatically.

🕮 1.1.3

What is learning?

To illustrate where the main advantages, but also the issues of machine learning lie, we give an example, the so-called **The Badges Game**. The example was invented by Haym Hirsh, who at a machine learning conference in 1994 assigned a "+" or "-" to each registered participant. The label was assigned by some unknown function known only to the creator of the example. The designation depended only on the first and last name of the participant.

The task for the participants was to identify the unknown function used to generate the +/- sign.

The list looked something like this:

```
+ Naoki Abe                - Myriam Abramson          + David W.
Aha
+ Kamal M. Ali                    - Eric Allender
      + Dana Angluin
- Chidanand Apte          + Minoru Asada                    +
Lars Asker
+ Javed Aslam                     + Haralabos Athanassiou   +
Jose L. Balcazar
+ Timothy P. Barber               + Michael W. Barley
      - Cristina Baroglio
+ Peter Bartlett          - Eric Baum               + Welton
Becket
- Shai Ben-David          + George Berg                     +
Neil Berkman
+ Malini Bhandaru         + Bir Bhanu               + Reinhard
Blasig
- Avrim Blum              - Anselm Blumer                   +
Justin Boyan
+ Carla E. Brodley        + Nader Bshouty                   -
Wray Buntine
- Andrey Burago                   + Tom Bylander
      + Bill Byrne
- Claire Cardie                   + Richard A. Caruana
      + John Case
+ Jason Catlett                   + Nicolo Cesa-Bianchi
      - Philip Chan
+ Mark Changizi                   + Pang-Chieh Chen         -
Zhixiang Chen
+ Wan P. Chiang                   - Steve A. Chien          +
Jeffery Clouse
+ William Cohen                   + David Cohn              -
Clare Bates Congdon
```

For a full list, see:

https://www.seas.upenn.edu/~cis5190/fall2018/assets/lectures/lecture-0/game.html

We do not need to build a machine learning model to solve this problem. However, it is important to think about how we would formalize this problem as a learning problem and what are the difficulties that arise in doing so.

When solving, it is important to remember that only the first and last names of the participants will not be enough even for a machine learning algorithm. New variables need to be generated from those names, e.g., the length of the full name, the length, i.e., the number of characters of the first name and the last name, the first character of the first name and its numbered code, the last character of the last name and its numbered code, the number of consonants, the number of vowels, and so on.

If we subsequently have the above mentioned variables - properties/attributes - calculated, we can deploy a machine learning algorithm that learns to add +/- tags for each name.

It is important to note that we do not need to know the exact function that Mr. Hirch created. We only try to estimate it, i.e. we try to copy the results of this feature as closely as possible. Mathematicians call this "good estimation" of the behaviour of a function as approximation. The algorithm of machine learning will therefore seek to approximate the function of Mr. Hirsh.

📝 1.1.4

What is a function for approximating values?

- It is the replacement of given values with appropriate close numbers based on a function that is not entirely accurate, but it is still good to be usable.
- It is the replacement of given values with appropriate close numbers based on a function that exactly corresponds to the substituted values.

🕮 1.1.5

In the previous example, we stated that machine learning only tries to approximate the real function. We still don't know the original Hirsh function. For some names from an existing dataset, it is possible to approximate the +/- marks using rules such as:

- if the length (number of characters) of the name is less than or equal to 5 yes + otherwise -

Name	Label
Claire Cardie	–
Peter Bartlett	+
Eric Baum	+
Haym Hirsh	+
Shai Ben-David	+
Michael I. Jordan	–

How were the labels generated?

```
If length of first name <= 5, then + else -
```

or

- if the numerical code of the last letter of the name is smaller than the numerical code of the last letter of the surname yes + otherwise -

Name	Label
Claire Cardie	–
Peter Bartlett	+
Eric Baum	+
Haym Hirsh	–
Leslie Pack Kaelbling	+
Yoav Freund	–

How were the labels generated?

```
If last letter of first name is before last letter of last name:
    label = +
else
    label = -
```

🕮 1.1.6

The purpose of **The Badges Game** example was not to solve the problem with the unknown function for adding +/- badges to conference participants. With an example, we wanted to show that machine learning algorithms only try to approximate the real function. At the same time, we wanted to show that working to solve a problem using machine learning is not about "headlessly" deploying a randomly selected algorithm and expecting excellent results. Most of the work consists in preparing the dataset, adding new features, i.e. attributes, in carefully selecting a machine learning algorithm, evaluating the algorithm, and understanding the results.

For the application of machine learning, it is necessary to implement the following steps in most cases:

Data preprocessing

Extracting symptoms

Creating a model

Making a prediction

Model testing and modification

Typical questions when applying machine learning are:

How to represent input data?

What deep background knowledge do we need?

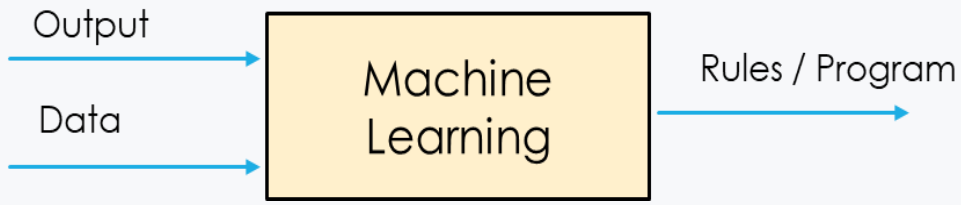
How does learning take place?

🕮 1.1.7

So what is the difference between traditional programming and machine learning?

In traditional programming, we know the problem, we know the rules to solve it, and if we apply these rules to the input data, we get the result.

In machine learning, we know the input data and we also know what the result should be. We are looking for a model, i.e. for example rules, which can generally calculate the result from the input data.



In the following text, we present examples of how machine learning can improve a task based on experience (training data) with respect to a measure (metric) of performance.

Task: Checkers game

Performance metric: Percentage of games won against any opponent

Data: Playing practice games against each other

Task: Recognizing handwritten words

Performance metric: Percentage of correctly classified words

Data: Database of annotated images of handwritten words

Task: Categorizing email messages as spam or ham.

Performance metric: Percentage of email messages correctly classified.

Data: A database of emails that have been manually annotated

Task: Driving on highways using sensors

Performance metric: Average distance traveled before human-judged error

Data: A sequence of images and steering commands recorded while observing a human driver.

📝 1.1.8

If we want to use machine learning to categorize which news belongs to fake news, the so-called fake news, what input (data) and output (output) data do we need to build such a classifier?

- A database of messages, along with information from manual annotators (people judging the messages) about which message is fake and which is genuine.
- Rules written by the state government by which institutions determine which message is fake.
- 67 / 5 000 Výsledky prekladov Výsledok prekladu A list of people compiled by the state government who spread false information.
- Database of politicians who lie.

1.2 Types of machine learning

🕮 1.2.1

Perhaps the easiest way to acquire knowledge is to memorize data about how to accomplish a task. It is information that will make it possible to accomplish a similar task better in the future. We call this method **swotting**.

For example, Samuel (1963) used the swotting method in a program that played checkers. He used a mini-max search of the space of the checkers game. The time complexity allowed only a few levels of depth to be searched each time, then a static evaluation function was used. Based on it, a move evaluation is made and the root evaluation is remembered at the same time.

Sometimes in the future, when browsing the game space, the situation arises that at the last level of the search, among the generated nodes, there is a node for which we have memorized the evaluation, which is the result of searching the space under the node. If this information did not exist, we would have to consider only the evaluation according to the static evaluation function. By remembering the previous evaluation, we seem to increase the depth of the search, i.e., we improve the quality of the search.

🕮 1.2.2

The evaluation function is often constructed by combining information from multiple sources. The programs take into account several factors, e.g., the advantage in the number of stones, or the mobility of the stones when playing checkers.

Based on these, a single number is calculated to evaluate the desirability of the position. In the game of checkers, for example, Samuel used an evaluation function in the form of a polynomial

in and +in$_{1122}$ +... and+in and$_{nn}$

Similarly, pattern recognition programs classify input data into appropriate categories. When creating such programs, it is often difficult to know in advance what weight to assign to each feature. One possibility is to start with some estimate of the weights, and then let the program adjust this estimate according to the experience it will gain. Qualities that appear to be good predictors of a successful solution of the project will increase the weights, the unsuccessful ones will decrease or not be taken into account.

Learning understood in this way is also called **learning by adapting parameters**.

🕮 1.2.3

To design a learning method by adapting parameters, it is necessary to know:

- which weight is to be increased or decreased,
- when the weight is to be changed,
- how much the weight should be changed.

If information is available on whether the rating function has estimated the configuration well, then the weights of those attributes that predicted the final result will increase and the weights of those that were wrong will decrease. For example, when classifying patterns, the program receives information about the correct classification.

This is more complicated with gaming programs. At most, the program gets information at the end about who won. However, many moves contributed to the final result, of which several could have been erroneous. For example, Samuel took an approach where the evaluation function generates its own feedback. It was based on the consideration that the sequences of steps that lead to better positions can be considered good. The weights of the attributes that recommended them will increase.

Samuel's program was also taught by playing against itself, i.e. one copy of it played with unchanging weights and the other copy had the weights changed. At the end of the game, the attributes of the program that won were taken. The process of learning by adapting parameters is limited in nature, since it does not make any use of knowledge about the structure of the problem.

🕮 1.2.4

The basic division of machine learning is into individual types of learning. Types of learning depend on feedback when learning. The following types of learning have settled in the literature:

- **Supervised learning** – immediate availability of sensations about both inputs and outputs.
- **Learning by reward and punishment** (enhanced learning / reinforcement learning) – the agent receives information about the evaluation of the action, but not about what the correct action should have been.
- **Learning without a teacher** (unsupervised learning) – the agent does not receive any information about what the correct actions should be.

📝 1.2.5

Assign the correct name of the learning type to the characteristics of learning.

the algorithm receives information about the evaluation of the action, but not about what the correct action should have been - _____

the algorithm has immediate availability of sensations about both inputs and outputs - _____

the algorithm does not receive any information about what the correct actions should be - _____

- Supervised learning
- Unsupervised learning
- Reinforcement learning

🕮 1.2.6

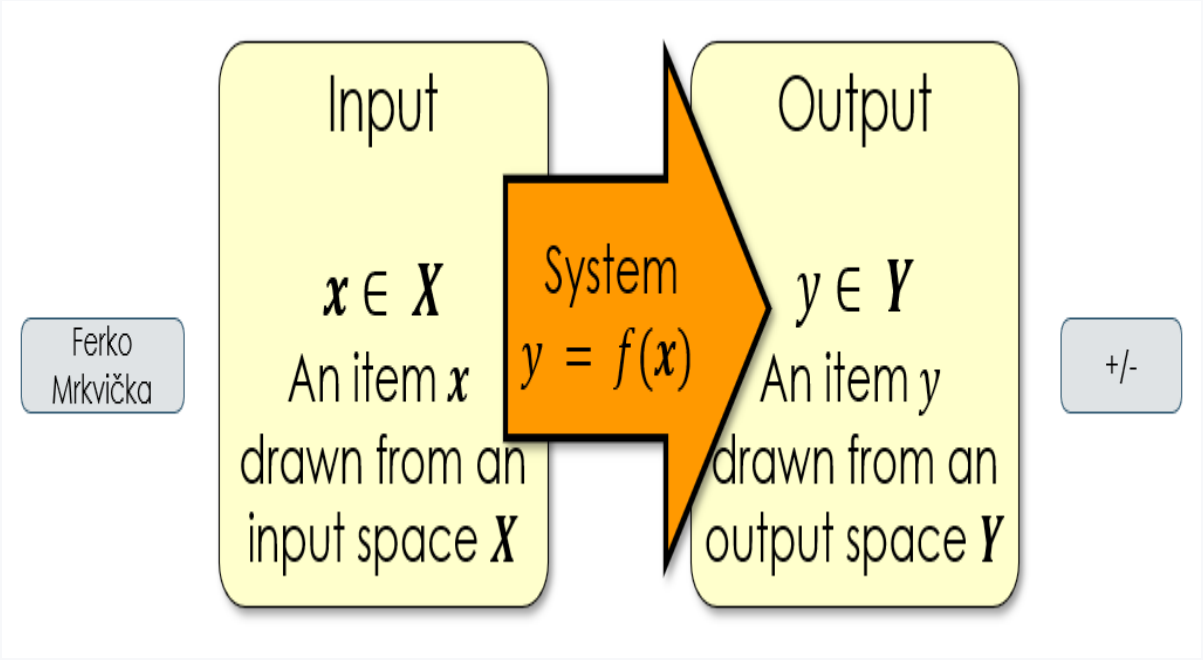
Learning with a teacher

Consider systems that apply the function ***f() to*** input **x** and return output ***y = f(x)***.

When learning with a teacher, ***f(x) is*** learned from examples.

We usually use machine learning when we do not know the ***f(x)*** function that we want the system to apply and we cannot "invent" it. In fact, the function can be simple.

For Mr. Hirsh's problem, the entry into office is the name of the conference participant, and the output of the f***(x)*** function is the **+** sign or **-**



When learning with the teacher, the algorithm searches for the best function that approximates the true values according to the data. The space of all functions that the algorithm "takes into account" is called the **hypothesis space**.

🕮 1.2.7

As a result of machine learning, a model (mostly a formula) is created that approximates the data. This model is created by a machine learning algorithm from historical data. For example, we consider a banking application that advises the bank whether or not to give a loan according to the characteristics of the customer. To create the model, historical data on previous loans will be used. In these historical data, the bank has the characteristics of its previous customers, including information on whether the customer in question repaid the loan provided. This historical data is called examples and is traditionally used to create a model. The example set is an input dataset.

Thus, the machine learning model created is a generalization of the examples with which the system was initially familiar. If we create a model, we need to answer the following questions:

- Is the model which we have created good?
- How good is it?

🕮 1.2.8

To answer questions about the quality of the model, it is necessary to simulate an estimate of our model as follows:

1. Remove some examples from a dataset
2. Create a model on remaining examples
3. Predict (estimate) deleted examples

This means that we provide the machine learning algorithm with only a fraction of the examples we have and we use them to train and build the model. We will call these examples **the training examples** or the **training set**.

We use the remaining examples to test our model. These are examples that were not used when creating the model. We will call these examples **test examples** or a **test set**.

It should be noted that even in the remaining examples, we also have the corresponding outputs for individual inputs. For example, for a banking application, we also have information in the test set whether the client has repaid the loan or not. Therefore, if we bring examples of the test set to the input of our model, we can find out the predicted result by the model and compare it with the real historical result.

🕮 1.2.9

When learning with a teacher, the learning algorithm receives the correct function value for the relevant inputs. Thus, the ordered pair (x,f(x)) represents an "example", where x is the input and ***f(x)*** is the output of the function for ***x***. If we have a given collection of examples of the function f, the function h should be returned so that it is an estimate of **the function f**. The **function h** is called the hypothesis.

The number of classes is fixed and is determined by the user. The systems do not use any other domain-specific information other than training examples. Despite the fact that the structure of the model is simple, the operations they perform (generalization, compression, and organization of data) are the basis of learning.

Many problems that at first glance do not look like classification problems can be transformed into classification problems. In the following examples, we show what constitutes the input ***x*** and the output ***f (x)***, i.e., what constitutes an **example** for each classification task:

Diagnosis of the disease

x: Patient characteristics (symptoms, laboratory tests)

f(x) : Disease (or maybe: recommended treatment)

Part-of-speech tagging

x: English/Slovak sentence

f(x) : Parts of speech in a sentence

Face recognition

x: Bitmap image of a person's face

f(x) : Name and surname of the person (or maybe: property)

Automatic control

x: Bitmap image of the road surface in front of the car

f(x) : Degrees of steering wheel rotation

1.2.10

In a min-max search, Samuel (1963) proposed a method for memorizing the valuation of a newly-developed node/root. If at some point in the future, when browsing the space, a situation arises that at the last level of the search there is also a node among the generated nodes for which we have memorized the valuation, this rating can be used.

This method of machine learning is called:

- bickering

- learning without a teacher
- logit regression
- linear regression

📝 1.2.11

Choose which claims apply to Learning with a teacher - Supervised learning

- For classification tasks, the number of classes is fixed and is determined by the user.
- The method does not use any other domain-specific information except for training examples.
- To evaluate the success of the method, the predicted values are compared with the actual values of the test set.
- Not suitable for regression types of tasks.

1.3 Evaluating machine learning models

🕮 1.3.1

The machine learning process itself consists of the following steps:

1. Understanding the domain, taking into account prior knowledge and objectives
2. Data integration, selection, cleansing, pre-processing
3. Creating models
4. Interpretation of results
5. Deployment of discovered knowledge/models

📝 1.3.2

List the individual steps/phases of machine learning in the correct order.

- <|span style="color: rgb(0, 0, 0);">Data integration, selection, cleansing, pre-processing<|/span>
- <|span style="color: black;">Deployment of discovered knowledge/models<|/span>

- <|span style="color: black;">Interpretation of results<|/span>
- <|span style="color: rgb(0, 0, 0);">Creating models<|/span>
- <|span style="color: rgb(0, 0, 0);">Understanding the domain, taking into account prior knowledge and objectives<|/span>

🕮 1.3.3

When creating models and interpreting the results, it is necessary to assess the suitability of the model, its correctness and accuracy. In the case of several models, it is necessary to choose a better model.

This brings us to the problem of How to measure accuracy? Which model is better? There is no clear answer to these questions.

For example, if we wanted to create a model for diagnosing a certain disease, and we know that 10 out of 10,000 samples are positive.

We would create multiple models. At first glance, the following statements about the models we have created seem correct:

A: "the classification model has a success rate of 80% "

B: "classification model is 400% better than random selection "

C: "the classification model perfectly captures all positive cases "

However, if we look at these claims in more detail, we find the following potential issues.

A: "the classification model has a success rate of 80% "

This model looks very promising. However, if we create a model that labels all samples as negative, then with 10 positives out of 10,000 samples we will achieve a success rate of 99.9%. So if we do nothing and say that all samples are negative, we have a 99.9% success rate.

B: "classification model is 400% better than random selection "

Such a model would also look very promising at first glance. However, with 10 positives out of 10 000 samples, one positive out of 1000 samples will be randomly

selected. If the algorithm is 400% better, then it can identify 4 positives out of 1000 samples. But is this estimate enough for us?

C: "the classification model perfectly captures all positive cases "

Of the above, perhaps the best-looking claim for a potential model, However, if we create a classifier with only one rule that each sample is positive, then we will also perfectly capture all positive cases. But would such a model be necessary?

It is clear from the above statements that we have to assess the suitability of a model from different perspectives and that numerical representations of suitability alone are not always sufficient.

🕮 1.3.4

Literature used:

- Eric Eaton: Introduction to Machine Learning (CIS 419/519) - https://www.seas.upenn.edu/~cis5190/fall2017/lectures/01_introduction.pdf
- Dan Roth: Applied Machine Learning (CIS 519/419) - https://www.seas.upenn.edu/~cis5190/fall2020/assets/lectures/lecture-1/Lecture1-intro.pptx
- Emily Fox, Carlos Guestrin: Machine Learning Specialization, University of Washington https://www.coursera.org/specializations/machine-learning
- Pavol Návrat et al: Artificial Intelligence. STU in Bratislava, 2002, Bratislava, 393 pages, ISBN 80-227-1645-6.

1.4 Practical exercise

📝 1.4.1

In this object-lesson, we will learn the basics of Python, especially the so-called DataFrame, which precompiles tables of data. The latter is precisely the most commonly used data structure, which consists of labelled axes (rows and columns).

To load the data file into the DataFrame, we will use the pandas library, which we will import using the following command.

```
import pandas as pd
```

In this demonstration, we will work with Titanic passenger data stored at the URL priscilla.fitped.eu/data/pandas/titanic.csv.

Use the following command to read and write the data file.

```
data =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv')
print(data)
```

Program output:

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                 Name     Sex
Age  SibSp  \
0                              Braund, Mr. Owen Harris    male
22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female
38.0      1
2                               Heikkinen, Miss. Laina  female
26.0      0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female
35.0      1
4                            Allen, Mr. William Henry    male
35.0      0
..                                                 ...     ...
...    ...
886                             Montvila, Rev. Juozas    male
27.0      0
```

```
887                      Graham, Miss. Margaret Edith  female
19.0      0
888          Johnston, Miss. Catherine Helen "Carrie"  female
NaN       1
889                             Behr, Mr. Karl Howell    male
26.0      0
890                               Dooley, Mr. Patrick    male
32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
..     ...               ...      ...   ...      ...
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]
```

We will list the contents of only one specific column according to the following command:

```
dataAge = data['Age']
print(dataAge)
```

Program output:

```
0      22.0
1      38.0
2      26.0
3      35.0
4      35.0
       ...
886    27.0
887    19.0
888     NaN
889    26.0
890    32.0
Name: Age, Length: 891, dtype: float64
```

Sometimes it is necessary to find out the i-th record in the data file. This can be viewed using .iloc

```
zaznam = data.iloc[0]
print(zaznam)
```

Program output:

```
PassengerId                          1
Survived                             0
Pclass                               3
Name           Braund, Mr. Owen Harris
Sex                             male
Age                             22.0
SibSp                              1
Parch                              0
Ticket                      A/5 21171
Fare                             7.25
Cabin                             NaN
Embarked                            S
Name: 0, dtype: object
```

To find out the data file types, use the following command:

```
typy = data.dtypes
print(typy)
```

Program output:

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

and the length of the data file by using the len() function.

```
dlzka = len(data)
print(dlzka)
```

Program output:

```
891
```

When working in python in machine learning tasks, we often need to know the shape of our data (number of rows and columns).

Using the following command, we find that our data file contains 891 columns and 12 rows.

```
tvar = data.shape
print(tvar)
```

Program output:

```
(891, 12)
```

The basic descriptive statistics of the dataset are returned by the *describe()* function.

```
print(data.describe())
```

Program output:

```
       PassengerId    Survived      Pclass         Age
SibSp  \
count   891.000000  891.000000  891.000000  714.000000
891.000000
mean    446.000000    0.383838    2.308642   29.699118
0.523008
std     257.353842    0.486592    0.836071   14.526497
1.102743
min       1.000000    0.000000    1.000000    0.420000
0.000000
25%     223.500000    0.000000    2.000000   20.125000
0.000000
50%     446.000000    0.000000    3.000000   28.000000
0.000000
```

```
75%     668.500000    1.000000    3.000000   38.000000
1.000000
max     891.000000    1.000000    3.000000   80.000000
8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

1.4.2

Loading to DataFrame

Check what you have learned in the previous task.

Load the file from https://priscilla.fitped.eu/data/pandas/banking.csv into the DataFrame. Find out the number of rows and columns of data.

1.4.3

Which solution correctly displays the Name column?

```
import pandas as pd
data =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv')
print(data['Name'])
data =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv')
print(data.'Name')
data = titanic
data['Name']()
```

1.4.4

Next, we will explore some of the features of the *sklearn* library, which is one of the most widely used libraries for machine learning.

First, we need to determine our features (features, or x-data) that will be the input to the machine learning model and the end value (target, or y) that will be the output of the machine learning model.

The following code sample loads the Titanic passenger data into a DataFrame data structure and divides it into features and target, where features are all values except the last column Embarked and target is embarked.

```
import pandas as pd
data =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv')

X, y = data.iloc[:, :-1], data.iloc[:, [-1]]

print(X)
print(y)
```

Program output:

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name     Sex
Age  SibSp  \
0                              Braund, Mr. Owen Harris    male
22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female
38.0      1
2                               Heikkinen, Miss. Laina  female
26.0      0
```

```
3          Futrelle, Mrs. Jacques Heath (Lily May Peel)  female
35.0       1
4                              Allen, Mr. William Henry    male
35.0       0
..                                                   ...     ...
...     ...
886                               Montvila, Rev. Juozas    male
27.0       0
887                        Graham, Miss. Margaret Edith  female
19.0       0
888            Johnston, Miss. Catherine Helen "Carrie"  female
NaN        1
889                               Behr, Mr. Karl Howell    male
26.0       0
890                                 Dooley, Mr. Patrick    male
32.0       0

     Parch            Ticket     Fare Cabin
0        0         A/5 21171   7.2500   NaN
1        0          PC 17599  71.2833   C85
2        0  STON/O2. 3101282   7.9250   NaN
3        0            113803  53.1000  C123
4        0            373450   8.0500   NaN
..     ...               ...      ...   ...
886      0            211536  13.0000   NaN
887      0            112053  30.0000   B42
888      2        W./C. 6607  23.4500   NaN
889      0            111369  30.0000  C148
890      0            370376   7.7500   NaN

[891 rows x 11 columns]
    Embarked
0          S
1          C
2          S
3          S
4          S
..       ...
886        S
887        S
888        S
889        C
890        Q
```

```
[891 rows x 1 columns]
```

When solving machine learning tasks, we divide the data into training and testing data.

Using the training data, we train the machine learning model and then validate it on the test data.

The sklearn library provides a function *train_test_split that* splits the data into two variables, where the first variable (usually referred to as *X_train)* contains the data that will be used later for training, and the second variable (usually referred to as *X_test)* contains the data that will be used to validate the model.

We further divided our features and target data into training and test data in the ratio of 80:20 using the following commands.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
```

⌨ 1.4.5

Division of data into training data and test data

Verify that you can correctly split the data into features and target, as well as training and test data.

Write code to retrieve data from https://priscilla.fitped.eu/data/pandas/banking.csv, set the second to fifth columns as properties, and set the sixth column as the target value.

Split the data into training and test data in a 70:30 ratio. Find the shape in the X_test variable.

Tree-Based Learning

Chapter 2

2.1 Decision Trees

🕮 2.1.1

Decision trees (DTs) are a nonparametric method of learning with a teacher used for **classification** and **regression**. The goal is to build a model that predicts the value of a target variable by learning simple decision rules derived from data functions.

The advantages of decision trees include:

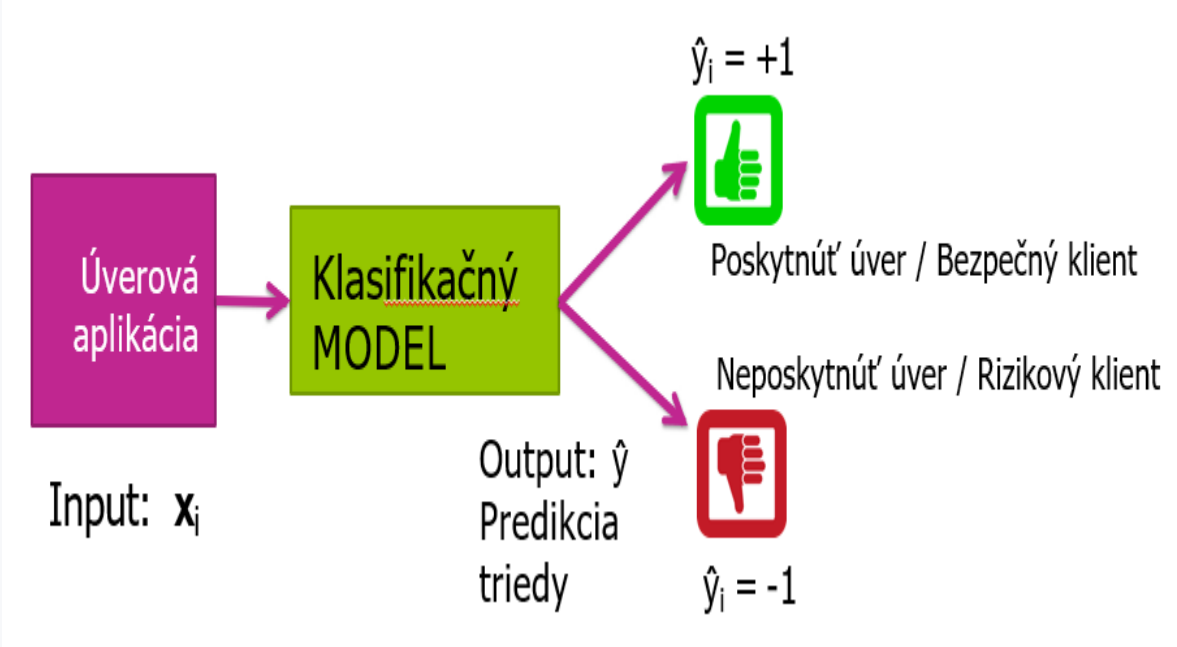
- They are simple to understand and interpret.
- Trees can be visualized.
- It does not require additional data preparation (e.g. data normalization, removal of blanks).
- It can handle both numeric and categorical data (however, the scikit-learn library does not yet support categorical variables).
- They handle the problem of classification into multiple classes.
- They belong to the so-called **white box models**. They are easy to explain and interpret.
- The model can be validated using statistical tests. This allows to be taken into account the reliability of the model.

The disadvantages of decision trees are as follows:

- Decision trees can produce overly complex trees that undergeneralize the data (overfitting).
- Decision trees can be unstable because small deviations in the data can lead to the generation of a completely different tree.
- Decision tree predictions are neither smooth nor continuous. They are piecewise constant approximations.
- If some classes dominate, the wrong trees may be generated.

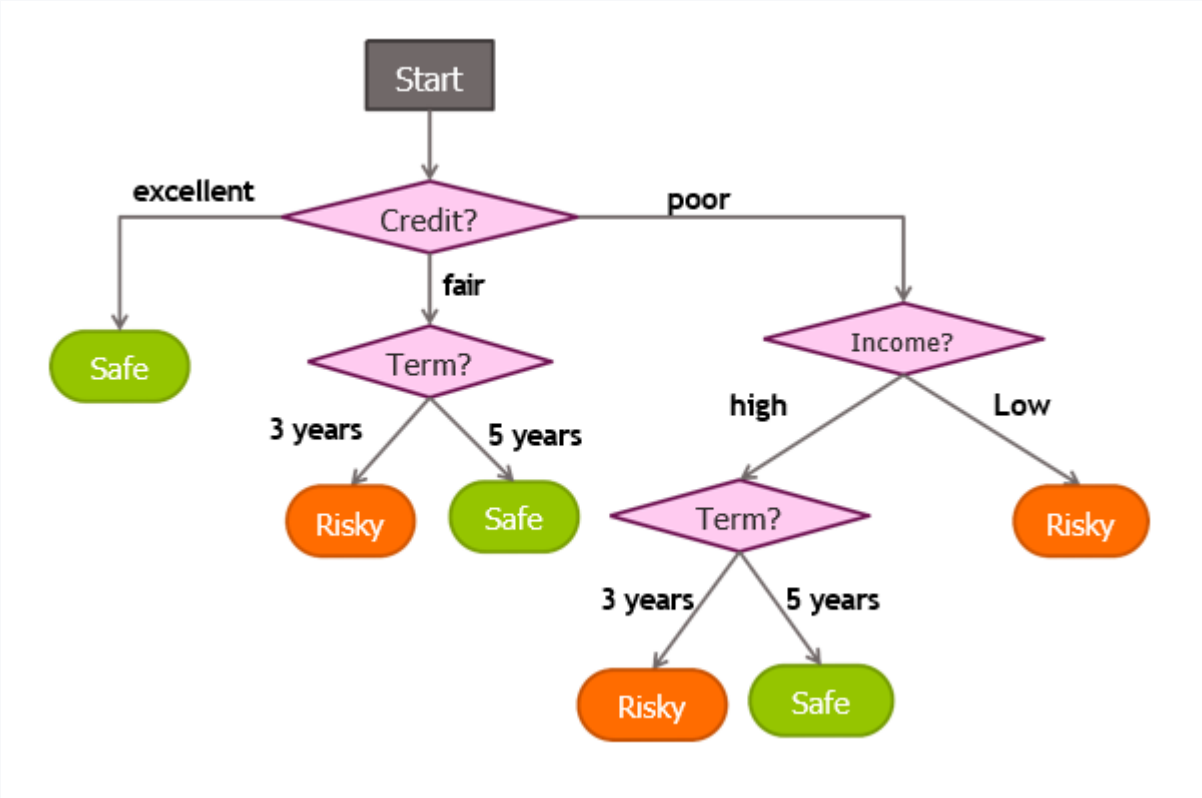
📖 2.1.2

We will demonstrate the creation of decision trees using a credit application as an example. The goal will be to create an application that, after inputting the monitored characteristics of a bank customer, decides whether or not the bank recommends granting credit to that customer. The main part of the application will be a classification model created by us, whose output will be a "Yes" or "No" recommendation.



📖 2.1.3

For example, we can imagine a decision tree classification model as follows:

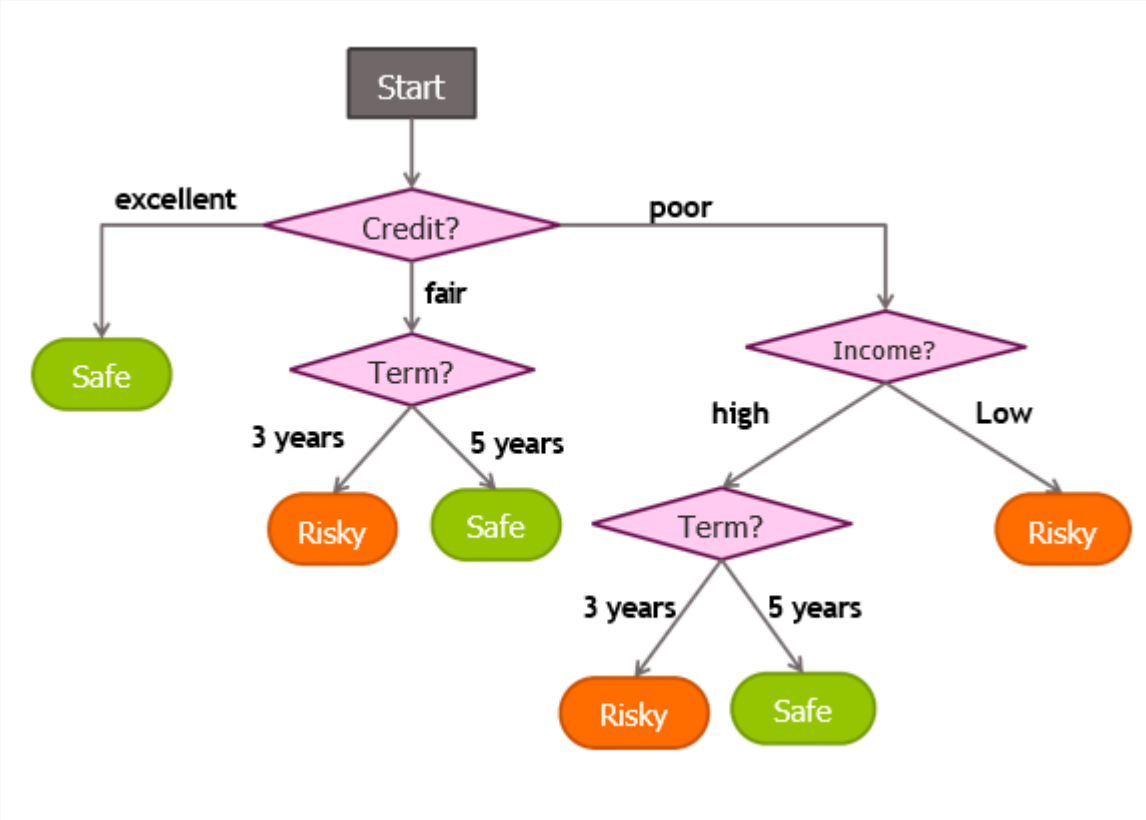


The decision tree represents a visualized set of rules for classification. In our example, a client who has a **fair credit** score and wants a **5-year loan** may apply for a loan at a bank. The created decision tree model finds that if **credit = fair**, then the length of the loan **Term** still needs to be checked. This is **5 years** in the case of our client. The tree then shows that the client is marked as safe and the model recommends giving him credit.

2.1.4

According to the following decision tree model, determine the recommendation for the following client:

Client1 - (Term=5 years; Credit=poor; Income=high)



- Recommendation: Safe
- Recommendation: Risky

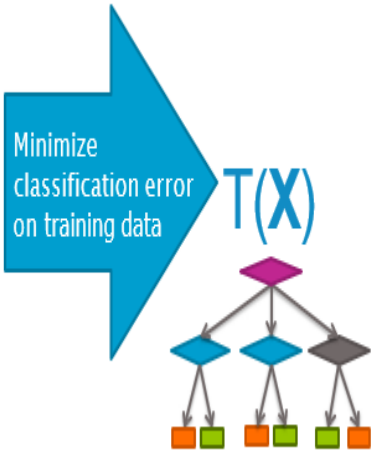
📖 2.1.5

The question remains how to build such a decision tree model. Like all other machine learning models, decision trees will be created from historical data. Based on the historical data of previous clients, and their loans repayment or default, a decision tree will be created from this data.

Klient	Príjem	Konto	Pohlavie	Nezamestnaný	Úver
K1	Vysoký	Vysoké	Žena	Nie	Áno
K2	Vysoký	Vysoké	Muž	Nie	Áno
K3	Nízky	Nízke	Muž	Nie	Nie
K4	Nízky	Vysoké	Žena	Áno	Áno
K5	Nízky	Vysoké	Muž	Áno	Áno
K6	Nízky	Nízke	Žena	Áno	Nie
K7	Vysoký	Nízke	Muž	Nie	Áno
K8	Vysoký	Nízke	Žena	Áno	Áno
K9	Nízky	Stredné	Muž	Áno	Nie
K10	Vysoký	Stredné	Žena	Nie	Áno
K11	Nízky	Stredné	Žena	Áno	Nie
K12	Nízky	Stredné	Muž	Nie	Áno

Our goal will be not only to build a decision tree, but to find the best possible decision tree that will predict future credits with the smallest possible error.

Klient	Príjem	Konto	Pohlavie	Nezamestnaný	Úver
K1	Vysoký	Vysoké	Žena	Nie	Áno
K2	Vysoký	Vysoké	Muž	Nie	Áno
K3	Nízky	Nízke	Muž	Nie	Nie
K4	Nízky	Vysoké	Žena	Áno	Áno
K5	Nízky	Vysoké	Muž	Áno	Áno
K6	Nízky	Nízke	Žena	Áno	Nie
K7	Vysoký	Nízke	Muž	Nie	Áno
K8	Vysoký	Nízke	Žena	Áno	Áno
K9	Nízky	Stredné	Muž	Áno	Nie
K10	Vysoký	Stredné	Žena	Nie	Áno
K11	Nízky	Stredné	Žena	Áno	Nie
K12	Nízky	Stredné	Muž	Nie	Áno



📖 2.1.6

A large number of trees can be generated for the selected dataset. The exponentially large number of possible trees makes learning a decision tree difficult!

When generating a decision tree, it is important to evaluate each tree (the models created) and decide which one is better. Therefore, quantification of the quality of the tree is necessary. This mark can be determined in a number of ways, which we refer to as performance metrics.

Of these, the simplest metric appears to be: **classification error**.

This is calculated as follows:

$$\text{Chyba} = \frac{\#\ \text{nesprávne predikcie}}{\#\ \text{počet príkladov (vzoriek)}}$$

According to the formula, it can be seen that the classification error reaches values from **0** to **1**. The closer to zero, the better classification. The best value is 0 of misclassified samples out of ***n samples***. Therefore, the result will be 0.

The worst value is ***n*** errors out of ***n samples***, it means. ***n/n=1***

📝 2.1.7

I am thinking with a decision tree. Out of 28 examples in the test set, 14 can classify correctly.

What is his classification error?

- 14 / 28 = 0,5
- 1
- 0
- 28/14 = 2
- 14 / (28 + 14) = 0,33333

2.2 A greedy algorithm for decision tree

📖 2.2.1

We describe the steps of the greesy algorithm for building the decision tree. We will consider the following training set.

Klient	Prijem	Konto	Pohlavie	Nezamestnany	Uver
K1	Vysoky	Vysoke	Zena	Nie	Ano
K2	Vysoky	Vysoke	Muz	Nie	Ano
K3	Vysoky	Nizke	Muz	Nie	Nie
K4	Nizky	Vysoke	Zena	Ano	Ano
K5	Nizky	Vysoke	Muz	Ano	Ano
K6	Nizky	Nizke	Zena	Ano	Nie
K7	Vysoky	Nizke	Muz	Nie	Ano
K8	Vysoky	Nizke	Zena	Ano	Ano
K9	Nizky	Stredne	Muz	Ano	Nie
K10	Vysoky	Stredne	Zena	Nie	Ano
K11	Nizky	Stredne	Zena	Ano	Nie
K12	Nizky	Stredne	Muz	Nie	Ano

The training set contains historical records of the ability to repay the loan in the past. The set contains records of 12 loans, of which 8 were able to be repaid and 4 were not.

📝 2.2.2

From the table on the bank's past clients and their ability to repay the loan, find out how many high-income women (income = high, gender = female) have repaid their loan. Enter the number as the answer for this question.

Klient	Prijem	Konto	Pohlavie	Nezamestnany	Uver
K1	Vysoky	Vysoke	Zena	Nie	Ano
K2	Vysoky	Vysoke	Muz	Nie	Ano
K3	Vysoky	Nizke	Muz	Nie	Nie
K4	Nizky	Vysoke	Zena	Ano	Ano
K5	Nizky	Vysoke	Muz	Ano	Ano
K6	Nizky	Nizke	Zena	Ano	Nie
K7	Vysoky	Nizke	Muz	Nie	Ano
K8	Vysoky	Nizke	Zena	Ano	Ano
K9	Nizky	Stredne	Muz	Ano	Nie
K10	Vysoky	Stredne	Zena	Nie	Ano
K11	Nizky	Stredne	Zena	Ano	Nie
K12	Nizky	Stredne	Muz	Nie	Ano

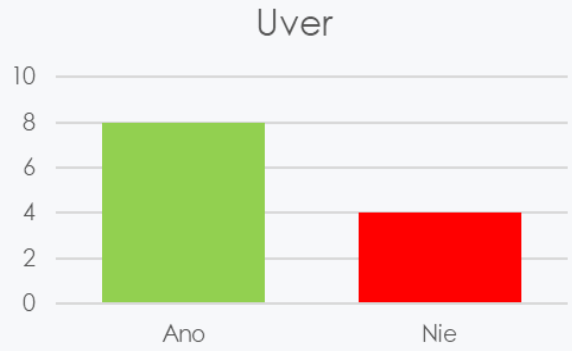
🕮 2.2.3

The first step of the greedy algorithm is:

1. Start with an empty tree and calculate the classification error of the empty tree

In the case of our dataset, where 8 clients have repaid the loan and 4 clients have not, the classification error will be 4/(4+8) = 0.3333

The frequency of distribution of clients to one of the two classes is visualized in the histogram.



The empty tree shows that if we do nothing further and say that all clients will be labelled "Yes", it means, they will be safe clients, then we make a classifier with a classification error of 0.3333.

🕮 2.2.4

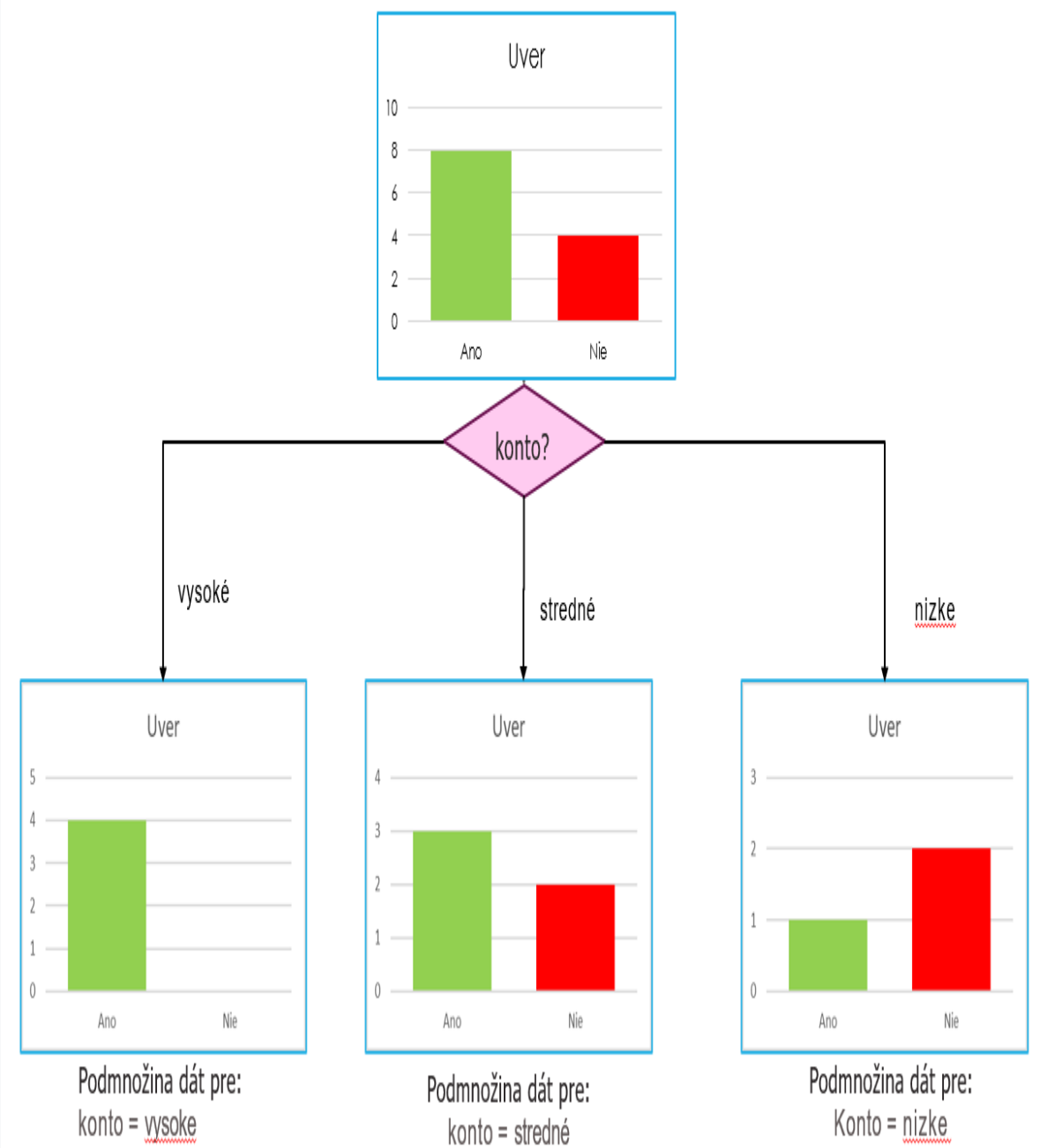
The second step of the greedy algorithm will be:

Split the data by features/attributes.

According to our training set:

Klient	Prijem	Konto	Pohlavie	Nezamestnany	Uver
K1	Vysoky	Vysoke	Zena	Nie	Ano
K2	Vysoky	Vysoke	Muz	Nie	Ano
K3	Vysoky	Nizke	Muz	Nie	Nie
K4	Nizky	Vysoke	Zena	Ano	Ano
K5	Nizky	Vysoke	Muz	Ano	Ano
K6	Nizky	Nizke	Zena	Ano	Nie
K7	Vysoky	Nizke	Muz	Nie	Ano
K8	Vysoky	Nizke	Zena	Ano	Ano
K9	Nizky	Stredne	Muz	Ano	Nie
K10	Vysoky	Stredne	Zena	Nie	Ano
K11	Nizky	Stredne	Zena	Ano	Nie
K12	Nizky	Stredne	Muz	Nie	Ano

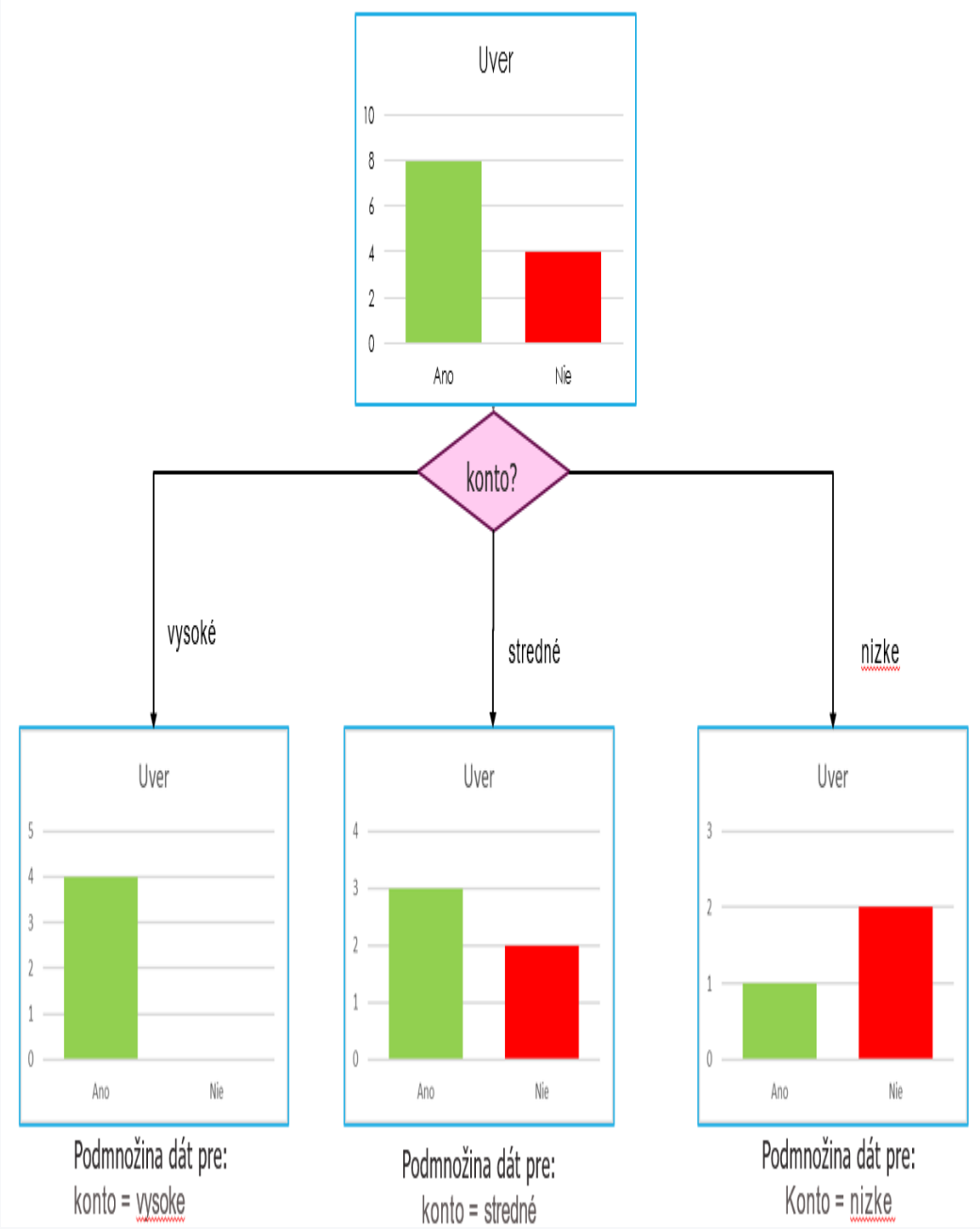
we can split the data according to the **Account(Konto)** property as follows:



For each subset, according to the splitting property of the account, we created historgrams of the representation of the target variable in each subset.

2.2.5

According to the histograms, determine how many "Yes" and "No" loan repayment values were in the group with the medium account, it means for the property **Account = Medium**.



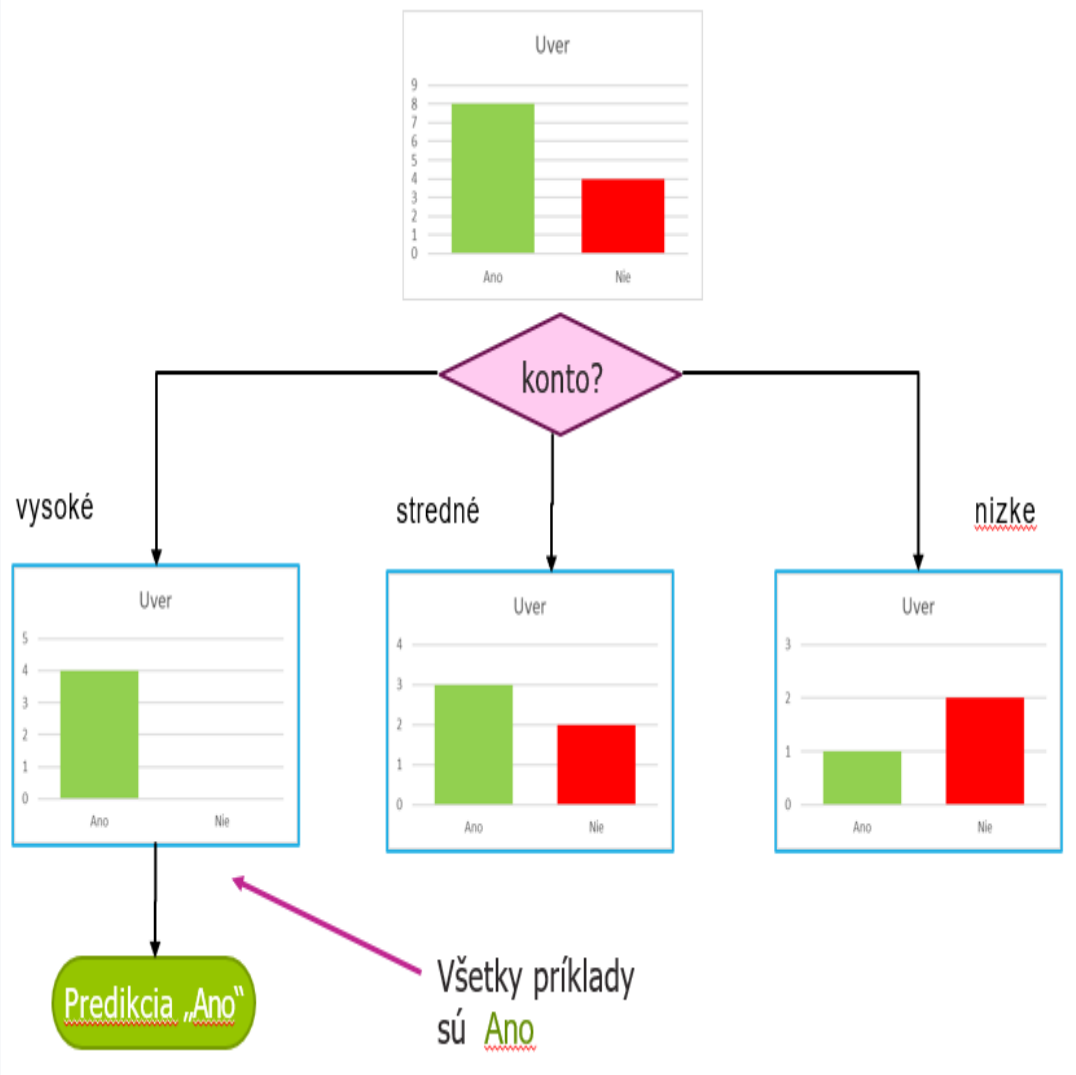
- Yes = 3; No = 2
- Yes = 1; No = 2
- Yes = 4; No = 0

🕮 2.2.6

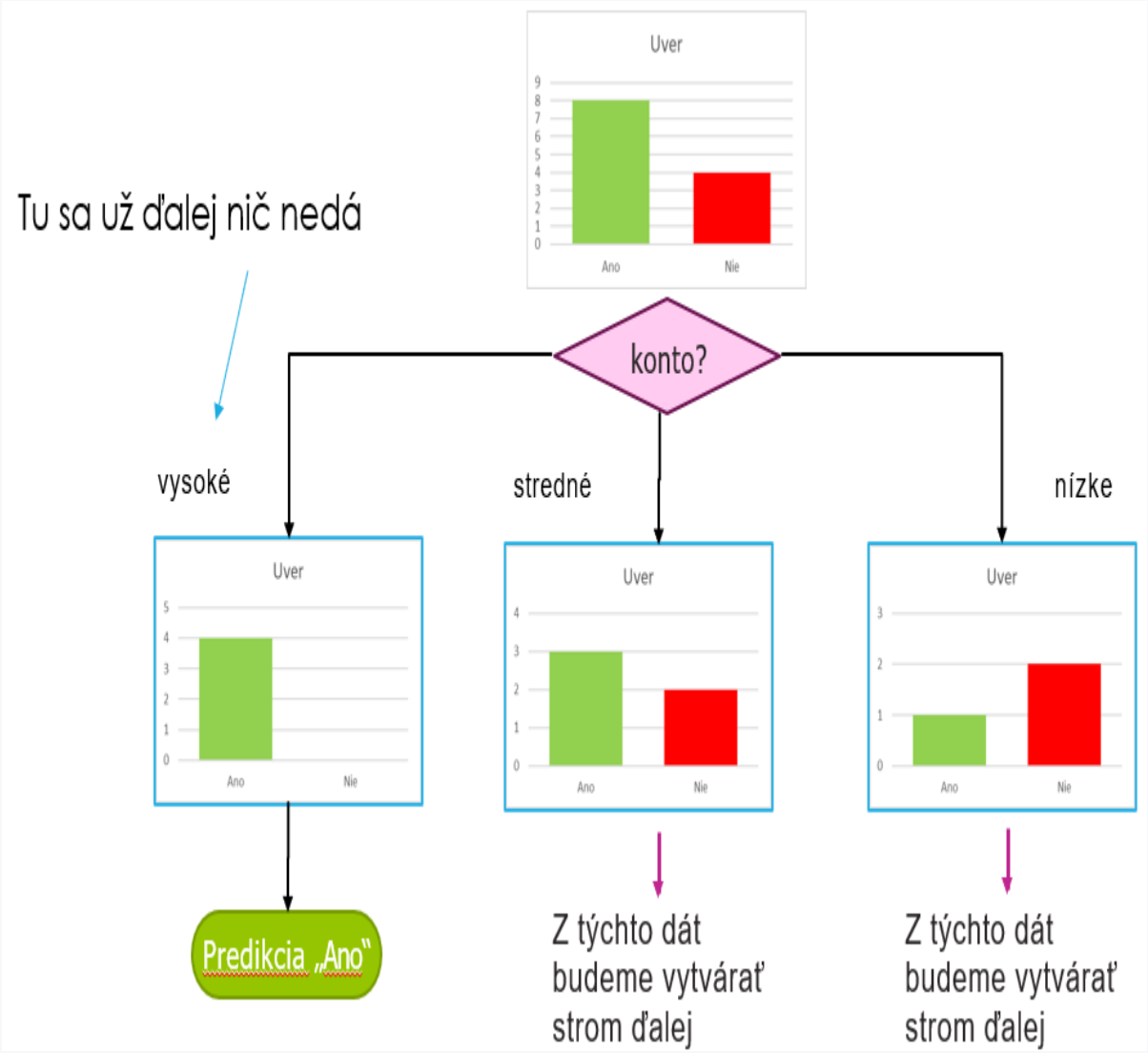
We just created a depth-1 tree with one splitting property - Account. The third step of the greedy algorithm is:

Make a prediction if possible

Note that for the condition **Account = High**, all examples are in the class "**Yes**". This means that in the past, all clients who had a high account have repaid the loan. In this case, we can make a prediction because there is no data from any other class already in this branch of the tree.



In the case of medium and low accounts, there is no clear class. Nevertheless, we can also make a prediction here if necessary. We could make a prediction according to the majority of the class, i.e. for the medium account the prediction would be **Yes** and for the low account it would be **No**. However, this would only create a tree of depth 1. Therefore, a better option is to continue recursively creating a tree from each subset of the data.



2.2.7

We can now summarize the whole greedy algorithm. It looks like this:

```
Step 1: start with empty trees

Step 2: select a property for data splitting

Step 3: create a distribution according to the selected
property

For each distribution of the tree:
                    Step 4: If you cannot go any further, make
a prediction
```

```
                    Step 5: Otherwise go to Step 2 & continue
with recursion of this distribution
```

In the given algorithm we find 2 questionable parts. The first is **Step 2**. It is a problem of feature selection, i.e. Feature split selection.

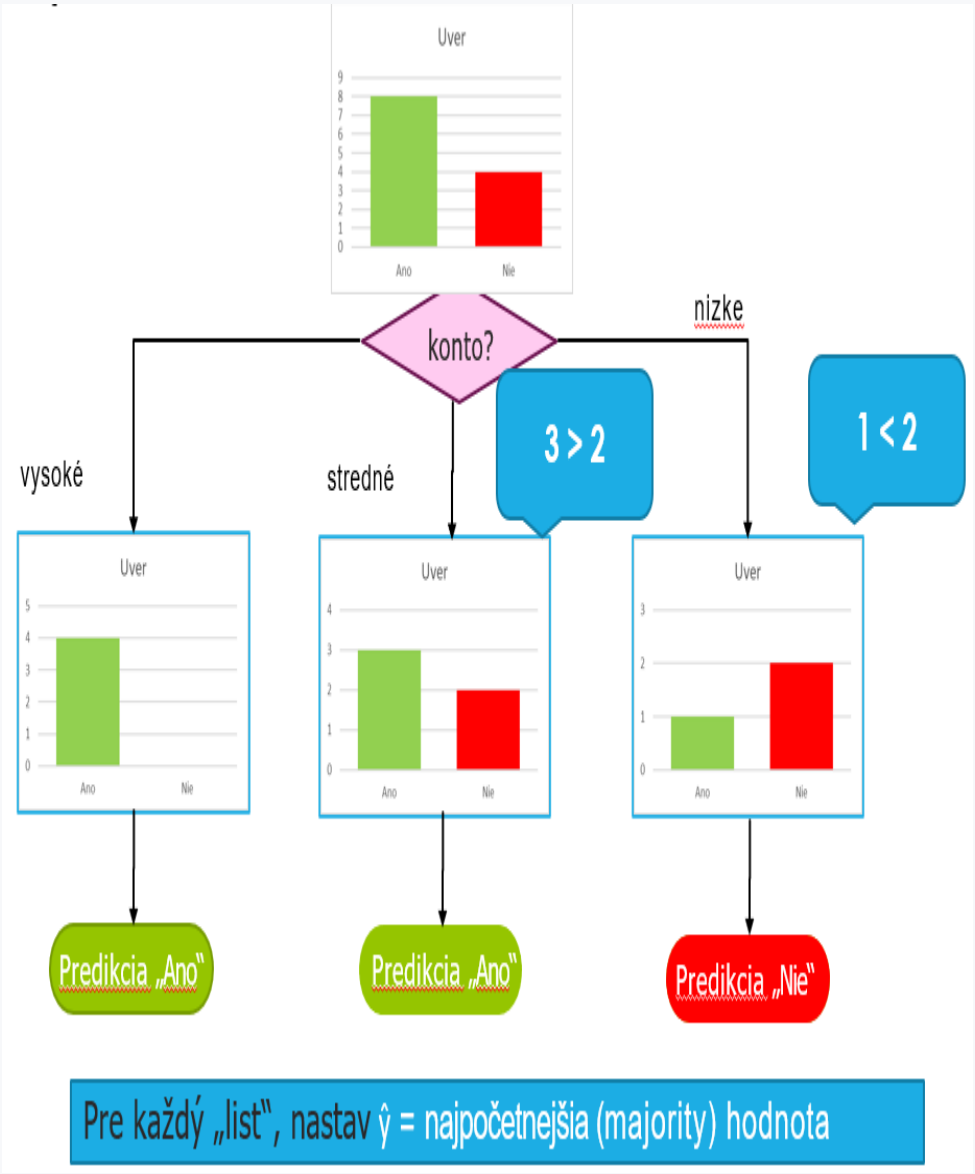
The second questionable part is **Step 4**, which deals with the problem of stopping the tree creation, i.e. stopping condition.

We discuss both of these issues in the next chapter.

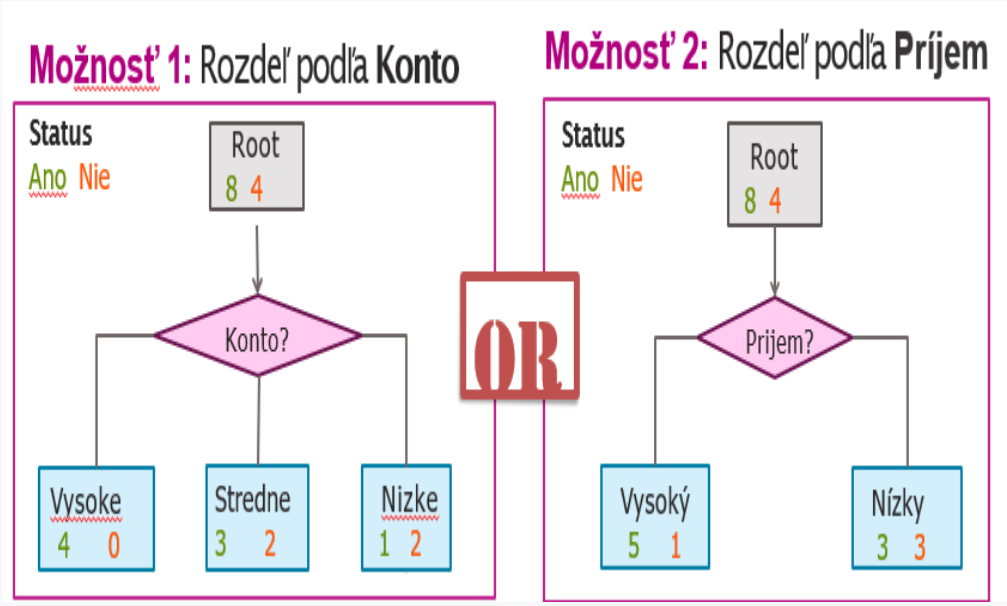
2.3 Choosing the best property for the distribution and stopping conditions of the algorithm

🕮 2.3.1

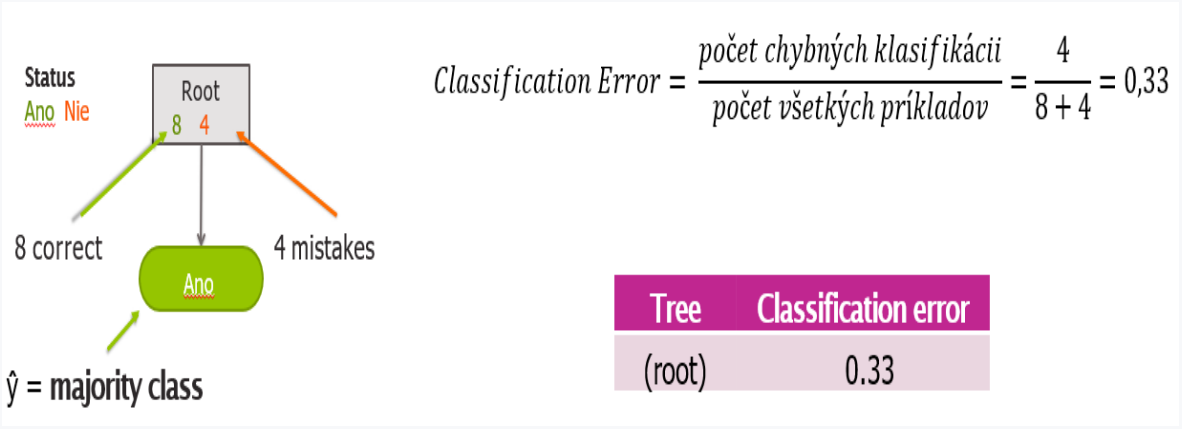
The first problem in the construction of decision trees is the selection of the best feature for distribution. This problem takes advantage of the "computational power of machine learning algorithms". In our example, the algorithm will proceed by creating a simple tree of depth 1 for each feature considered. In such a tree, it must always decide for prediction. Therefore, it will perform the prediction according to the most represented class in the data subset.

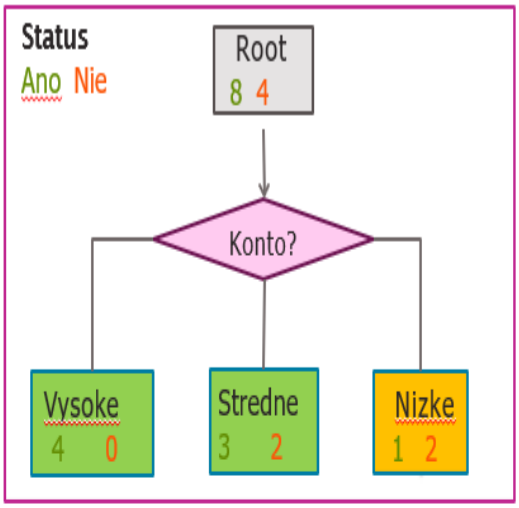


Let us now consider all the trees of depth 1 constructed in this way. To find the most appropriate feature, we use the classification error. For example, we can compare a tree for a distribution according to **Account** and a tree according to **Receipt**.

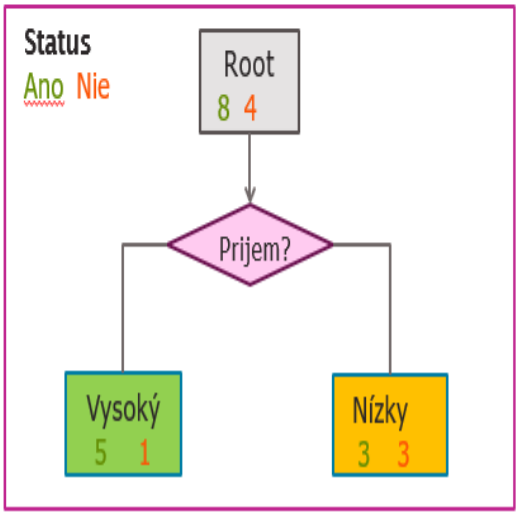


We perform the comparison according to the classification error. Thus, we select the feature whose tree has the smallest classification error. We also compare this with the empty tree.



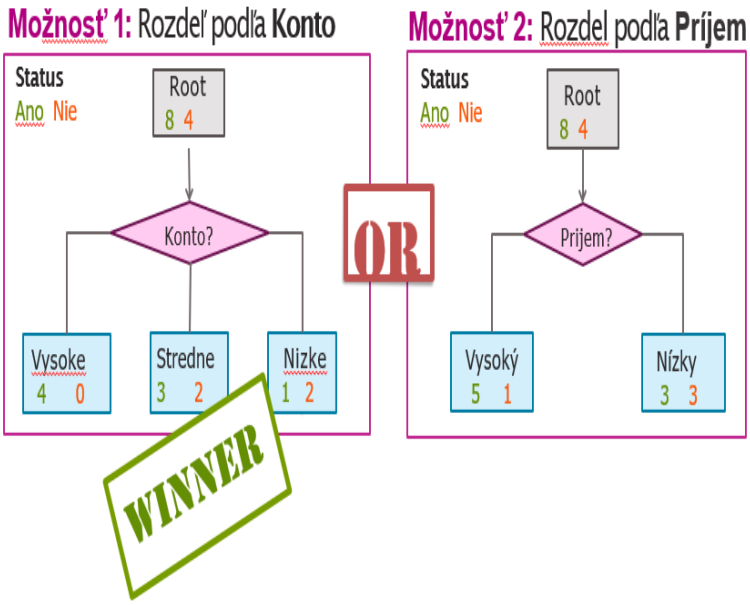


$$Classification\ Error(Konto) = \frac{3}{8+4} = 0{,}25$$



$$Classification\ Error(Prijem) = \frac{4}{8+4} = 0{,}33$$

It remains for us to compare the results of the classification errors.



Tree	Classification error
(root)	0.33
split on Konto	0.25
split on Prijem	0.33

The lowest classification error is obtained by dividing by **ACCOUNT**.

📝 2.3.2

Correctly complete the formula for calculating the classification error and the best and worst possible value of the classification error

$$\text{classification error} = \frac{\text{\_\_\_\_\_}}{\text{\_\_\_\_\_}}$$

The best possible value of the classification error is _____.

The worst possible value of the classification error is _____.

- 0
- number of incorrect predictions
- number of all examples
- 1

🕮 2.3.3

Feature split selection algorithm consists of the following steps:

1.Given a subset of data M (a node in the tree)

2. For each features $h_i(x)$ do:

- Partition the M data according to the function $h_i(x)$ to form a single-level tree
- Calculate the classification error of the following tree

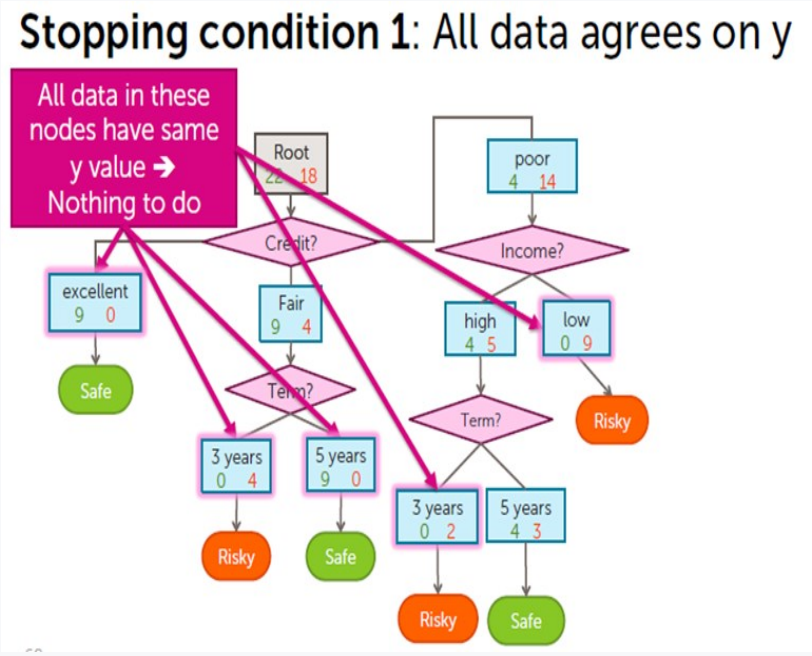
3.Select function $h_*(x)$ with the smallest mistake in classification

🕮 2.3.4

The second problem of the greedy algorithm is the stopping condition

There can be more than one of these conditions, but most often the algorithm terminates if:

1. If there is nothing left in the nodes to split.



2. If we used all of the features

3. If further division does not achieve a lower classification error.

🕮 2.3.5

The goal of the preceding example was to present the algorithm for tree creation as simply as possible. For this reason, we chose a classification error for feature selection. However, this is not used in real tasks. For selecting the best feature, the Gini Index or Information Gain is used in real examples. Also, the classification error is usually not sufficient even when evaluating the success of the developed model. In this case, so-called performance measures are used. There are several of these measures. For example, the fastest and most frequently calculated one is accuracy

Also for the sake of simplicity, we have so far dealt with tree formation from categorical variables only.

The formation of a decision tree requires a number of non-trivial steps. Fortunately, there are several libraries and ready-made methods in well-known programming languages. In these methods, then, e.g., the stopping condition or the measure for selecting a features are only given as method parameters.

📝 2.3.6

In a simple dataset, the features (rich, handsome) of the last four suitors of Gertrude B are recorded. The **relationship** feature tells whether Gertrude B. stayed with a suitor for more than 1 month, i.e. it records long-term relationships.

We want to create a decision tree model that will predict whether Gertrude will stay with her partner for more than a month, i.e. we will model the **relationship** feature. Which feature (**rich** or **handsome**) will be selected as the first feature to create such a decision tree? I.e. which of the two features will be selected as more appropriate for data- feature split selection?

- both features can be selected as appropriate, i.e. likely to be selected at random
- rich
- handsome

2.4 Performance metrics for machine learning models

🕮 2.4.1

Currently, we are already able to create the first machine learning model - a decision tree - based on a greedy algorithm. We can "measure" its success or performance by two basic metrics, classification error and classification accuracy.

For these metrics, the following relationships hold:

$$Accuracy(Správnosť) = \frac{počet\ \mathbf{správnych}\ klasifikácií}{počet\ všetkých\ príkladov}$$

$$Classification\ Error = \frac{počet\ chybných\ klasifikácii}{počet\ všetkých\ príkladov}$$

At the same time, the relationship is as follows:

$$Accuracy = 1 - Classification\ Error$$

📖 2.4.2

However, a number of other metrics are used to evaluate the performance of the model. Why are metrics important?

- Metrics help to capture the quality of the model into a quantitative expression (not all errors are the same).
- They help to refine ML generation effort. In terms of expressing improvements of models, methods, datasets, etc..

· They are useful to quantify differences between:

· actual performance and initial expectations,

· desired performance and actual performance.

· They measure progress over time.

- They are useful for lower-level tasks and pruning. Ideally, the goal of training a model should be a metric, but this is not always possible.
- Metrics are useful and important for evaluation.

📖 2.4.3

In classification tasks, we recognize two types of errors.

Error of the first type, i.e. false positive or false positivity. The error occurs when a negative example labels the classification model as positive.

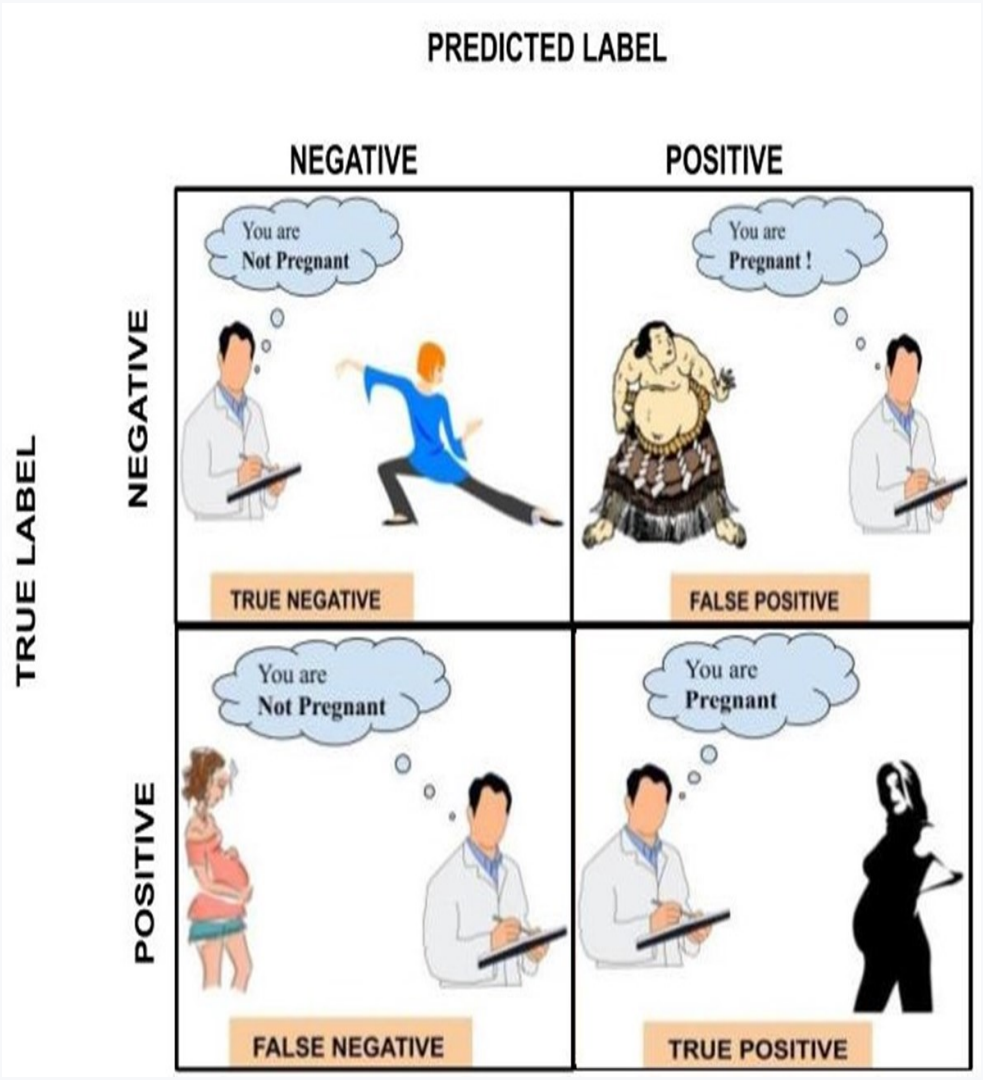
Error of the second type, i.e. false negative, or false negativity, it is an error when the model labels a positive example as negative.

According to these two types of errors, we can calculate the performance metrics of the model-classifier.

📖 2.4.4

In addition to the two types of errors, there are also two types of classification success. It is the case if a positive example is correctly labeled as positive by the classifier or a negative example is correctly labeled as negative by the classifier

We can clearly show both types of success and also the two types of errors in the so-called confusion matrix.

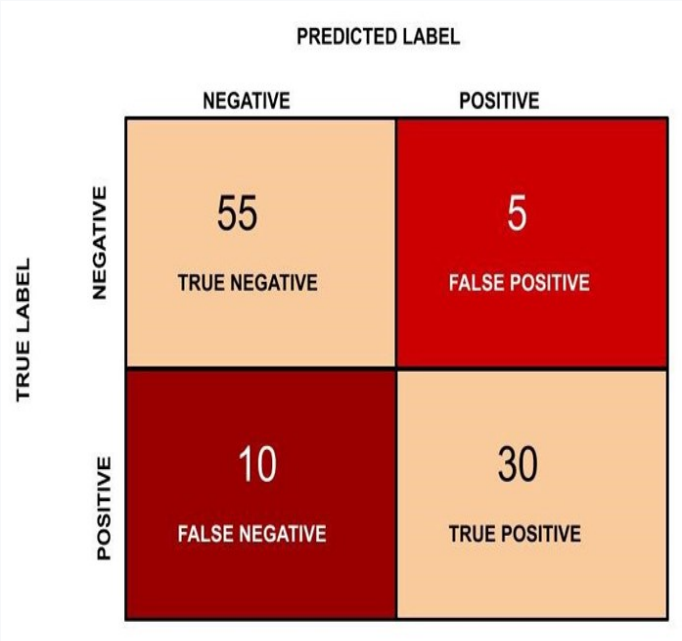


This records the number of examples evaluated correctly or incorrectly by the classifier.

📖 2.4.5

One of the few metrics that we already know is Accuracy.

If we have the following confusion matrix,



then the Accuracy can be calculated as follows:

$$Accuracy(Správnosť) = \frac{počet\ \textbf{správnych}\ klasifikácií}{počet\ všetkých\ príkladov} = \frac{TrueNegative + TruePositive}{TrueNegative + FalsePositive + TruePositive + FalseNegative}$$

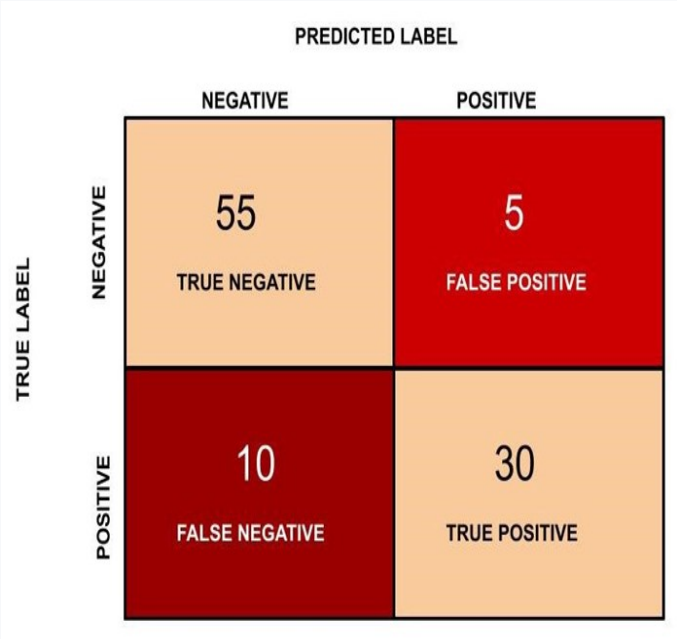
$$Accuracy(Správnosť) = \frac{TN+TP}{TN+FP+TP+FN}$$

For the above mentioned confusion matrix, Accuracy is calculated by substituting into the formula:

$$Accuracy = \frac{55+30}{55+5+30+10} = 0{,}85$$

📖 2.4.6

However, is accuracy a good metric? In the previous example, we used the following confusion matrix:

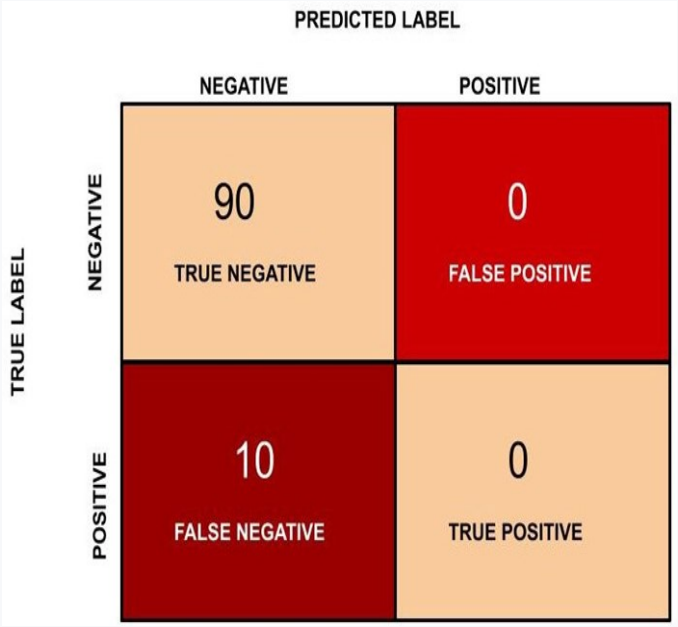


We calculated:

$$Accuracy(Správnosť) = \frac{TN+TP}{TN+FP+TP+FN}$$

$$Accuracy = \frac{55+30}{55+5+30+10} = 0{,}85$$

If we build no classifier and just say that all examples are negative, we get the following confusion matrix:



Based on matrix, we can now calculate the accuracy:

$$Accuracy = \frac{TN+TP}{TN+FP+TP+FN}$$

$$Accuracy = \frac{90+0}{100+0} = 0,9$$

Note that despite the weak classifier, we computed a higher Accuracy.

Accuracy is not a good metric when the dataset is unbalanced.

Using Accuracy in such scenarios, it can lead to misleading interpretation of the results.

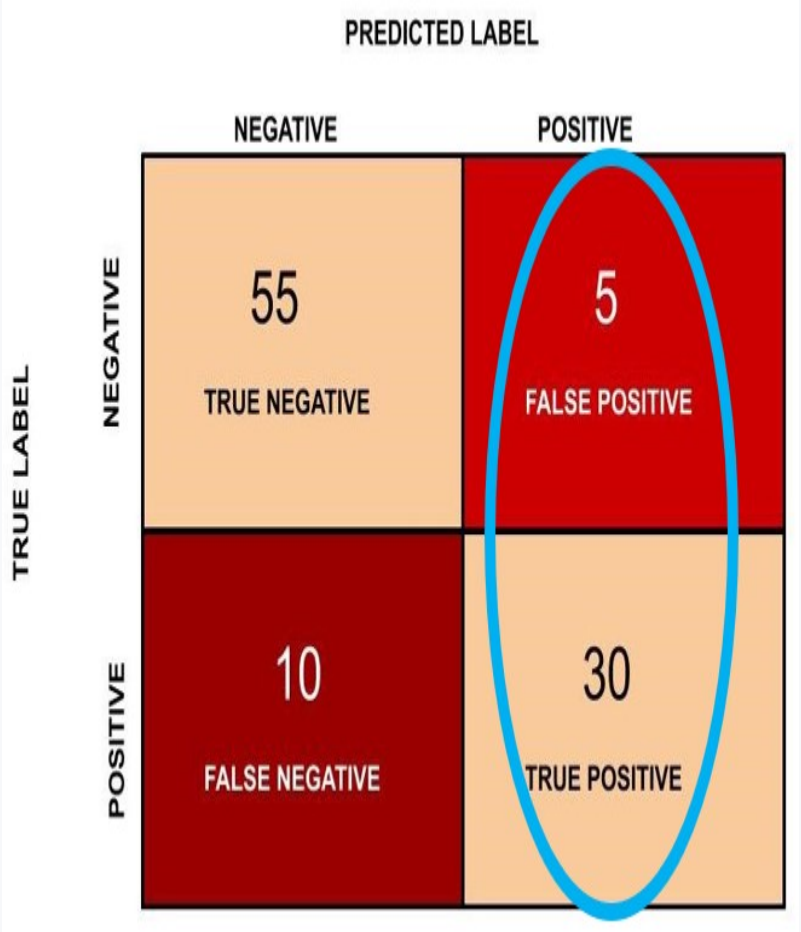
🕮 2.4.7

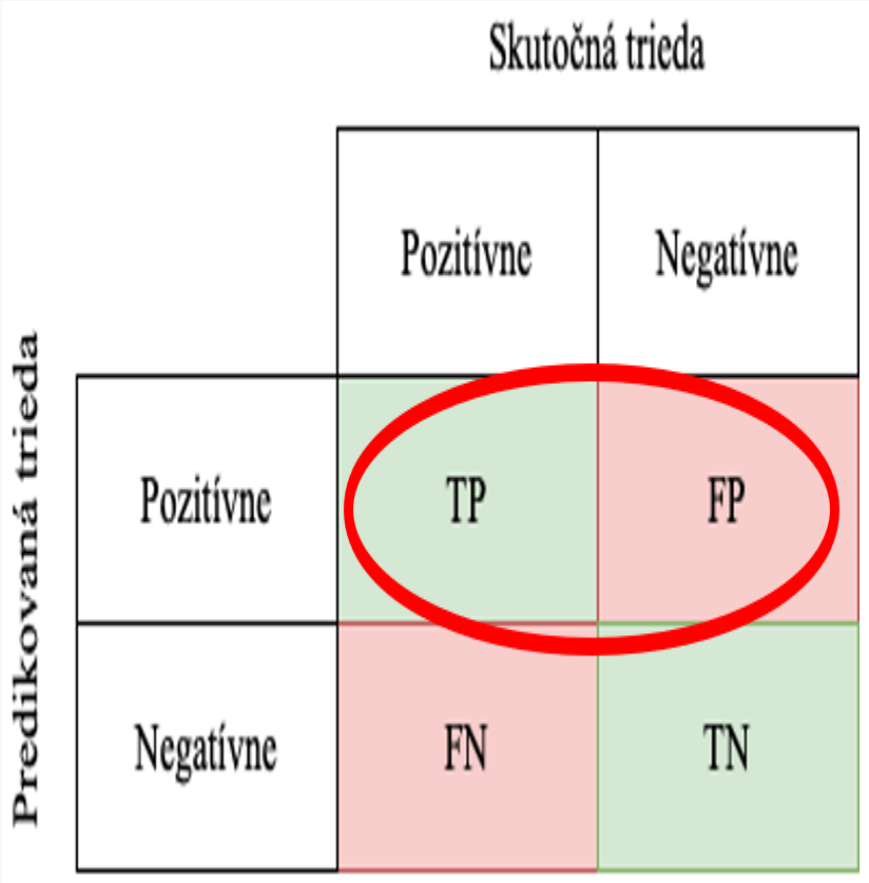
In addition to the other metrics, there is a dataset balance metric. It is called **Prevalence** and it is calculated as the number of positive samples to all samples.

$$Prevalence = \frac{\#positives}{\#positives + \#negatives}$$

It is clear from the formula that the ideal balance of the dataset is closed to 0.5.

Also note the different notation of the confusion matrix. In the following two examples we give examples of different notations.





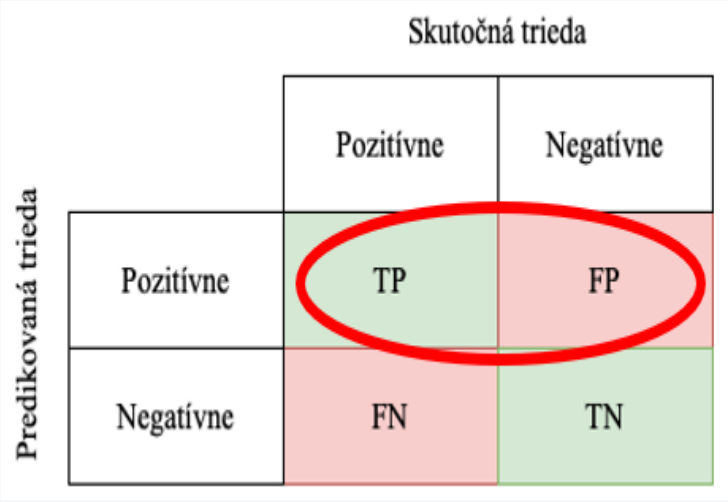
It is always necessary to check the column and row headers in confusion matrix

🕮 2.4.8

Another metric is **Precision**.

We can calculate it as follows:

$$precision = \frac{TP}{TP + FP}$$



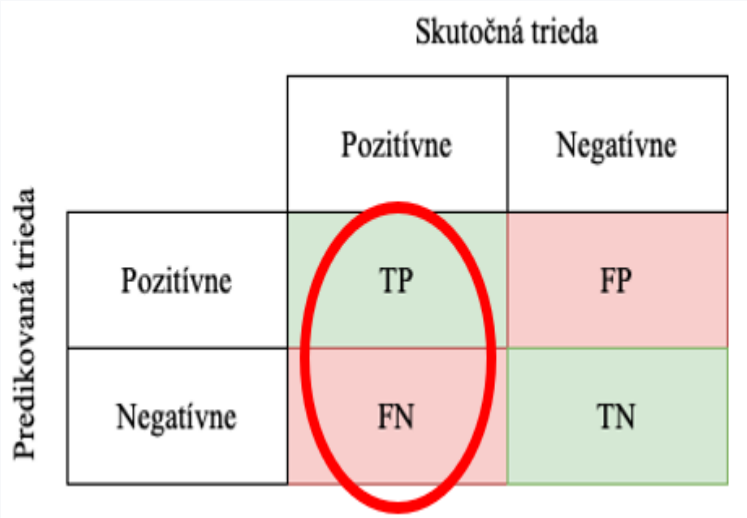
Accuracy should ideally achieve 1 (high) for a good classifier. Accuracy only becomes 1 when the numerator and denominator are equal, i.e. TP = TP + FP, which also means that FP is zero.

In binary classification, classes do not have to be divided into positive/negative, but can be divided into e.g. spam/ham, fake/real, obese/not obese, etc. For this reason, partial accuracies can also be computed.

🕮 2.4.9

Recall is also known as **sensitivity** or as **true positive rate**.

$$recall = \frac{TP}{TP + FN}$$

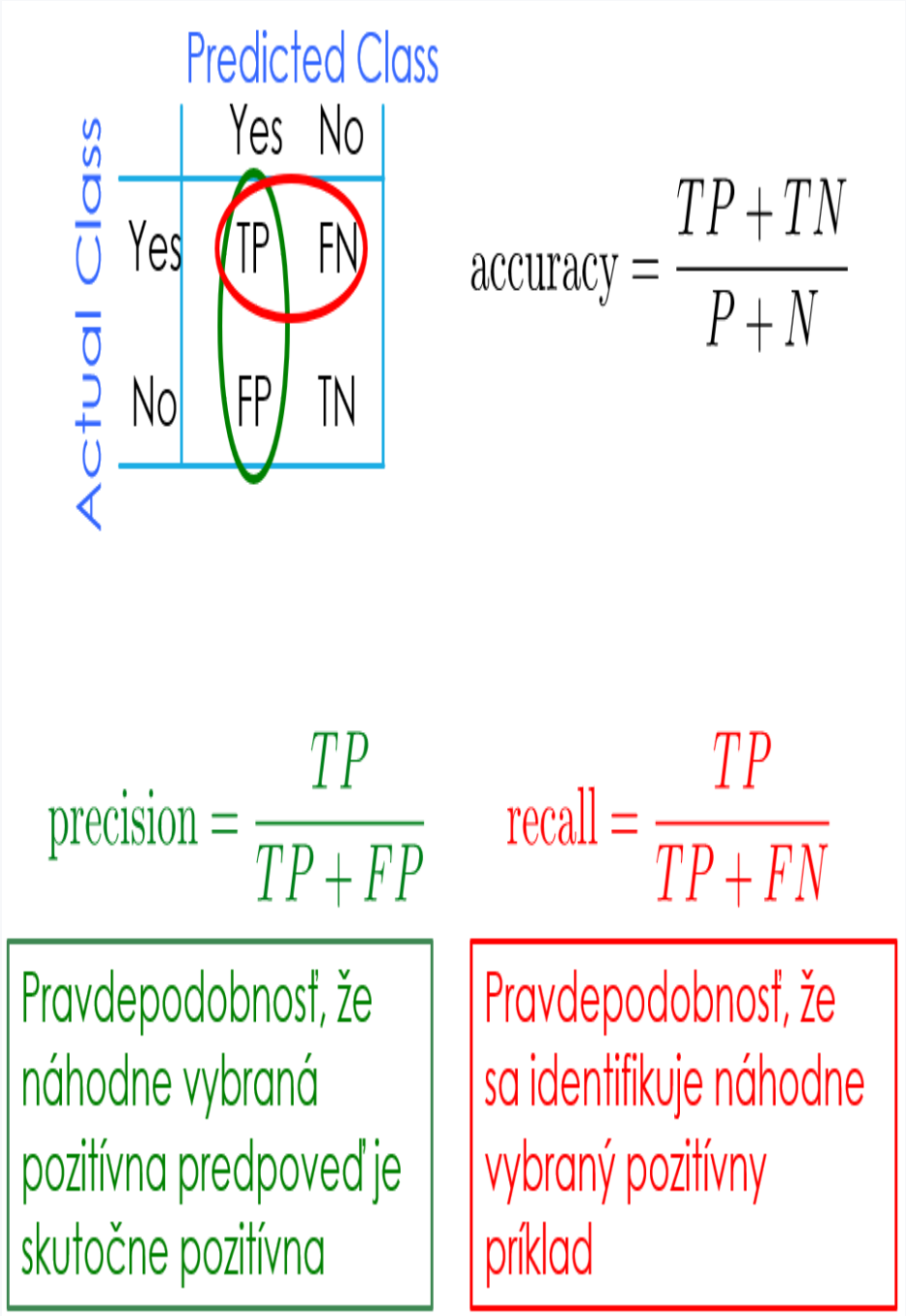


For a good classifier, the value should ideally achieve 1 (high), which means that the FN is zero.

Similarly, to precision, a **partial** recall can be computed.

📖 2.4.10

Review:



2.4.11

Complete the formula for calculating the **accuracy**

As an aid, we provide a confusion matrix

$$\text{Precision} = \frac{\text{TN + FP}}{\text{TP}}$$

- TN + FP
- TP
- TN
- FP
- TP + FP
- TP + TN

📖 2.4.12

The last metric is a **F1 score**.

$$f1\ score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Ideally, in a good classifier, we want both, precision and recall to be 1.

Which also means that FP and FN are 0.

The **F1 score** metric takes both precision and recall into account.

F1 score is the harmonic mean of precision and recall and is a better measure than precision.

2.5 Decision Trees - Practical Example 1

📝 2.5.1

We will demonstrate the creation of decision trees with a practical example in Python. In the practical example, we will try to create a decision tree model for loan prediction. We load the data for our model from the file uvery.csv using the Pandas library.

```
import pandas

loans =
pandas.read_csv('http://priscilla.fitped.eu/data/machine_learn
ing/uvery.csv', sep=';')
```

With simple **head**() and **tail**() commands we can check our data. We can also use the **discribe**() method to display the basic statistics for our file. Since the file contains only categorical variables, the basic statistics will be very simple.

```
print("---------------------------")
print(loans.head())
print("---------------------------")
print(loans.tail())
print("---------------------------")
print(loans.describe())
```

Program output:

```
---------------------------
  Klient  Prijem   Konto Pohlavie Nezamestnany Uver
0     K1  Vysoky  Vysoke     Zena           Nie  Ano
1     K2  Vysoky  Vysoke      Muz           Nie  Ano
2     K3   Nizky   Nizke      Muz           Nie  Nie
3     K4   Nizky  Vysoke     Zena           Ano  Ano
4     K5   Nizky  Vysoke      Muz           Ano  Ano
---------------------------
   Klient  Prijem    Konto Pohlavie Nezamestnany Uver
7      K8  Vysoky    Nizke     Zena           Ano  Ano
8      K9   Nizky  Stredne      Muz           Ano  Nie
9     K10  Vysoky  Stredne     Zena           Nie  Ano
10    K11   Nizky  Stredne     Zena           Ano  Nie
11    K12   Nizky  Stredne      Muz           Nie  Ano
---------------------------
       Klient Prijem   Konto Pohlavie Nezamestnany Uver
count      12     12      12       12           12   12
unique     12      2       3        2            2    2
top        K1  Nizky  Vysoke     Zena          Nie  Ano
freq        1      7       4        6            6    8
```

From the above results, it is easy to see that our set contains 12 examples. The target variable that our model will predict is the variable **Loan** with possible values **Yes** and **No.**

We will create the decision tree model using the **scikit-learn** library. It is one of the most widely used libraries for machine learning. However, in the case of decision tree models, this library cannot handle categorical variables. For this reason, we need to convert the categorical variables to numerical variables in our dataset.

By quick reasoning, a function can be created to convert categorical variables. In our example, we convert a categorical variable feature (i.e., a column in pandas) **Income**.

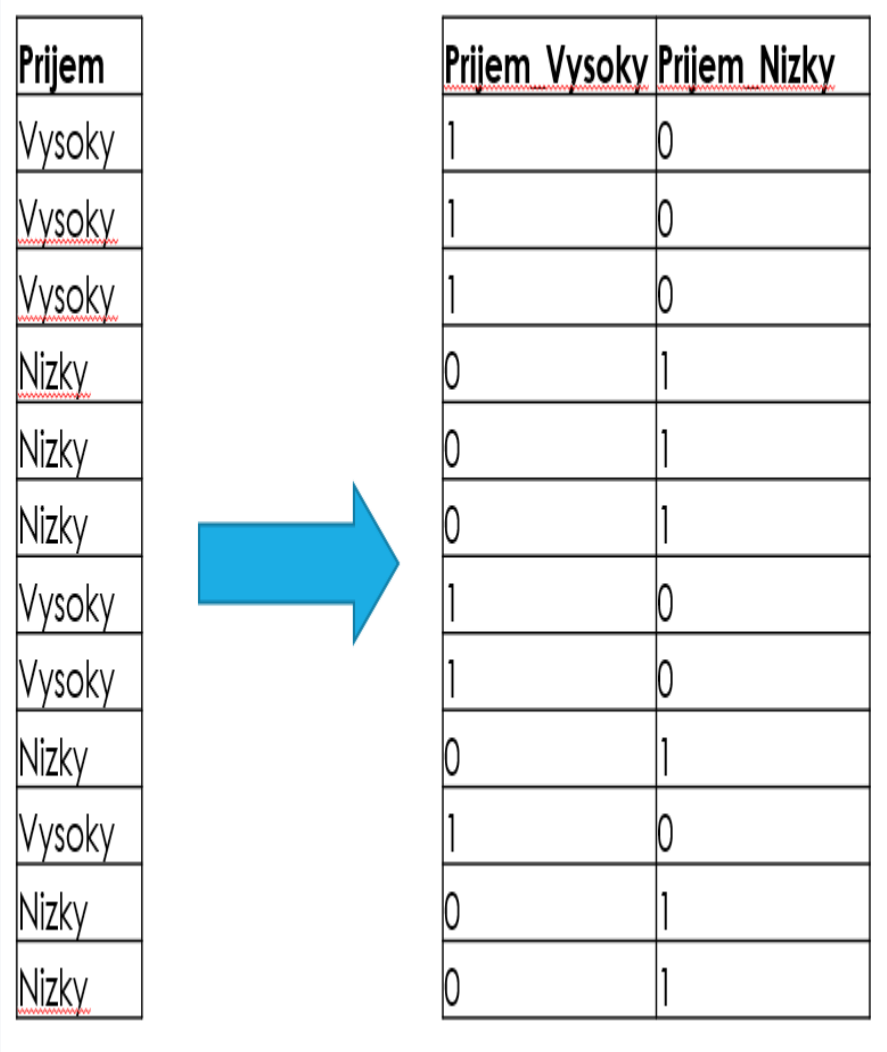
```
loans["Prijem_int"] = loans["Prijem"]
def cat2int(column):
    vals = list(set(column))
    for i, string in enumerate(column):
        column[i] = vals.index(string)
    return column

cat2int(loans['Prijem_int'])

print(loans.head())
```

Program output:

```
  Klient  Prijem   Konto Pohlavie Nezamestnany Uver Prijem_int
0     K1  Vysoky  Vysoke     Zena           Nie  Ano          1
1     K2  Vysoky  Vysoke      Muz           Nie  Ano          1
2     K3   Nizky   Nizke      Muz           Nie  Nie          0
3     K4   Nizky  Vysoke     Zena           Ano  Ano          0
4     K5   Nizky  Vysoke      Muz           Ano  Ano          0
```

Note the new feature **Income_int**. Its interpretation is easy as long as we also have the **Income** feature. However, we need to be aware of several shortcomings of this approach. The first shortcoming is that this conversion does not always set low income to 0, high income to 1. If we already consider multiple categorical values, e.g. slightly higher, medium, very low, etc. the clarity of the numerical values may be unclear. Especially, if we do not see the original **Income** column.

For these reasons, so-called dummies are used to convert categorical variables into numerical variables. Dummies create a new feature (column) for each value of a categorical variable. For the **Income** feature, the dummies will look as follows:



Prijem
Vysoky
Vysoky
Vysoky
Nizky
Nizky
Nizky
Vysoky
Vysoky
Nizky
Vysoky
Nizky
Nizky

Prijem_Vysoky	Prijem_Nizky
1	0
1	0
1	0
0	1
0	1
0	1
1	0
1	0
0	1
1	0
0	1
0	1

In Python, we can create dummies by simply calling the appropriate method.

```
loans =
pandas.get_dummies(loans,columns=["Prijem"],drop_first=False)

print(loans.head())
```

Program output:

```
  Klient Uver Prijem_int  Prijem_Nizky  Prijem_Vysoky
Konto_Nizke  \
0     K1  Ano           1             0               1
0
1     K2  Ano           1             0               1
0
2     K3  Nie           0             1               0
1
```

```
3      K4   Ano            0                1                0
0
4      K5   Ano            0                1                0
0

   Konto_Stredne  Konto_Vysoke  Pohlavie_Muz  Pohlavie_Zena
Nezamestnany_Ano  \
0                0             1             0                1
0
1                0             1             1                0
0
2                0             0             1                0
0
3                0             1             0                1
1
4                0             1             1                0
1

   Nezamestnany_Nie
0                 1
1                 1
2                 1
3                 0
4                 0
```

2.5.2

We can now use the previous information about categorical variables and dummies to create sample code where we load our dataset and transfer all the necessary features using dummies.

```
import pandas
loans =
pandas.read_csv('http://priscilla.fitped.eu/data/machine_learn
ing/uvery.csv', sep=';')
print("----------------------------")
print(loans.head())

loans =
pandas.get_dummies(loans,columns=["Prijem"],drop_first=False)
loans=
pandas.get_dummies(loans,columns=["Konto"],drop_first=False)
```

```
loans=
pandas.get_dummies(loans,columns=["Pohlavie"],drop_first=False
)
loans=
pandas.get_dummies(loans,columns=["Nezamestnany"],drop_first=F
alse)
print("----------------------------")
print("Dataset po dummies:")
print("----------------------------")
print(loans.head())
```

Program output:

```
----------------------------
  Klient  Prijem   Konto Pohlavie Nezamestnany Uver
0     K1  Vysoky  Vysoke     Zena           Nie  Ano
1     K2  Vysoky  Vysoke      Muz           Nie  Ano
2     K3   Nizky   Nizke      Muz           Nie  Nie
3     K4   Nizky  Vysoke     Zena           Ano  Ano
4     K5   Nizky  Vysoke      Muz           Ano  Ano
----------------------------
Dataset po dummies:
----------------------------
  Klient Uver  Prijem_Nizky  Prijem_Vysoky  Konto_Nizke
Konto_Stredne  \
0     K1  Ano             0              1            0
0
1     K2  Ano             0              1            0
0
2     K3  Nie             1              0            1
0
3     K4  Ano             1              0            0
0
4     K5  Ano             1              0            0
0

   Konto_Vysoke  Pohlavie_Muz  Pohlavie_Zena  Nezamestnany_Ano
\
0             1             0              1                 0
1             1             1              0                 0
2             0             1              0                 0
3             1             0              1                 1
4             1             1              0                 1

   Nezamestnany_Nie
0                 1
```

```
1                   1
2                   1
3                   0
4                   0
```

Note, that after applying dummies, we not only created new features, but we also deleted the original features.

An interesting and often used method is the **Counters** method. This gives us a quick look at the distribution of values in each feature. For example, if we want to know the number of 0 and 1 values for the variable **Income_High**, we can use **Counters.**

```
from collections import Counter
print(Counter(loans.Prijem_Vysoky))
```

Program output:

```
Counter({0: 7, 1: 5})
```

Use the **Counters** function to see how many **Yes** and **No** values are in the **Loan** feature.

```
from collections import Counter
print(Counter(___))
```

Program output:

```
Counter()
```

- 8 values **Yes** and 4 values **No**
- 4 values **Yes** and 8 values **No**
- 0 values **Yes** and 1 values **No**

2.5.3

Example

```
import pandas
from collections import Counter
```

```
from sklearn.tree import DecisionTreeClassifier # Import
Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import
train_test_split function
from sklearn import metrics #Import scikit-learn metrics
module for accuracy calculation

golf =
pandas.read_csv('http://priscilla.fitped.eu/data/machine_learn
ing/golf_nominal.csv', sep=';')

golf=
pandas.get_dummies(golf,columns=["Outlook"],drop_first=False)
golf=
pandas.get_dummies(golf,columns=["Temperature"],drop_first=Fal
se)
golf=
pandas.get_dummies(golf,columns=["Humidity"],drop_first=False)
golf=
pandas.get_dummies(golf,columns=["Windy"],drop_first=False)

X = golf[golf.columns.difference(['Play'])]
y = golf.Play

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.5) # 70% training and 30% test

# Create Decision Tree classifer object

clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Grafika

```
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print(Counter(y_test))
cols = X_train.columns
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=False,
                special_characters=True,feature_names =
cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('graf.png')
Image(graph.create_png())
```

Program output:

```
Accuracy: 0.42857142857142855
Counter({'yes': 4, 'no': 3})
```

2.6 Practical Tasks

2.6.1

In these practical examples, we will guide you through the creation of a simple decision tree.

The example shows a decision on whether or not a person will survive on the Titanic, based on that person's features (characteristics).

We will use the same Titanic dataset as in the previous exercise.

A description of the individual columns is available at https://data.world/nrippner/titanic-disaster-dataset

```
import pandas as pd

data =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv')
print(data)
```

Program output:

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
```

```
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                    Name     Sex
Age  SibSp  \
0                                Braund, Mr. Owen Harris    male
22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female
38.0      1
2                                 Heikkinen, Miss. Laina  female
26.0      0
3          Futrelle, Mrs. Jacques Heath (Lily May Peel)  female
35.0      1
4                               Allen, Mr. William Henry    male
35.0      0
..                                                  ...     ...
...    ...
886                               Montvila, Rev. Juozas    male
27.0      0
887                        Graham, Miss. Margaret Edith  female
19.0      0
888            Johnston, Miss. Catherine Helen "Carrie"  female
NaN       1
889                               Behr, Mr. Karl Howell    male
26.0      0
890                                 Dooley, Mr. Patrick    male
32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
..     ...               ...      ...   ...      ...
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
```

```
890      0            370376   7.7500   NaN         Q

[891 rows x 12 columns]
```

We choose what data from the dataset we want to use to classify whether or not a given person would have survived on the Titanic.

After analyzing the available data, we choose the following:

1. class Pclass
2. Sex
3. Age
4. number of siblings/spouses of the person who are travelling with SibSp
5. number of parents/children of a person, who are travelling with Parch
6. place of boarding Embarkment

Our target value will be whether or not the person survived the Titanic, i.e. the Survived column.

```
data = data[['Survived', 'Pclass', 'Sex', 'Age', 'SibSp',
'Parch', 'Embarked']]
print(data)
```

Program output:

```
     Survived  Pclass     Sex   Age  SibSp  Parch Embarked
0           0       3    male  22.0      1      0        S
1           1       1  female  38.0      1      0        C
2           1       3  female  26.0      0      0        S
3           1       1  female  35.0      1      0        S
4           0       3    male  35.0      0      0        S
..        ...     ...     ...   ...    ...    ...      ...
886         0       2    male  27.0      0      0        S
887         1       1  female  19.0      0      0        S
888         0       3  female   NaN      1      2        S
889         1       1    male  26.0      0      0        C
890         0       3    male  32.0      0      0        Q

[891 rows x 7 columns]
```

This dataset has several null values (they are marked as NaN), which we first delete by removing those rows that contain such values.

```
data = data.dropna()
```

```
print(data)
```

Program output:

```
     Survived  Pclass     Sex   Age  SibSp  Parch Embarked
0           0       3    male  22.0      1      0        S
1           1       1  female  38.0      1      0        C
2           1       3  female  26.0      0      0        S
3           1       1  female  35.0      1      0        S
4           0       3    male  35.0      0      0        S
..        ...     ...     ...   ...    ...    ...      ...
885         0       3  female  39.0      0      5        Q
886         0       2    male  27.0      0      0        S
887         1       1  female  19.0      0      0        S
889         1       1    male  26.0      0      0        C
890         0       3    male  32.0      0      0        Q

[712 rows x 7 columns]
```

Note, that after removing the null values out of the original 891 records, only 712 records left.

If we want to preserve the number of records, we can apply other methods to deal with null values, e.g. replace them with (substitution).

The Embarked column should be binarized. This is done using the get_dummies function.

We also change the gender values male to 0 and female to 1.

```
data =
pd.get_dummies(data,columns=["Embarked"],drop_first=False)

data['Sex'] = data['Sex'].replace({'male': 0, 'female': 1})

print(data)
```

Program output:

```
     Survived  Pclass  Sex   Age  SibSp  Parch  Embarked_C
Embarked_Q  \
0           0       3    0  22.0      1      0           0
0
```

```
1            1       1    1  38.0       1       0            1
0
2            1       3    1  26.0       0       0            0
0
3            1       1    1  35.0       1       0            0
0
4            0       3    0  35.0       0       0            0
0
..         ...     ...  ...   ...     ...     ...          ...
...
885          0       3    1  39.0       0       5            0
1
886          0       2    0  27.0       0       0            0
0
887          1       1    1  19.0       0       0            0
0
889          1       1    0  26.0       0       0            1
0
890          0       3    0  32.0       0       0            0
1

     Embarked_S
0             1
1             0
2             1
3             1
4             1
..          ...
885           0
886           1
887           1
889           0
890           0

[712 rows x 9 columns]
```

Data are prepared, now we can proceed to split it into features and target, and also into training and testing in a ratio of 80:20.

We have added randomization to the train_test_split function, which will always guarantee a deterministic distribution.

```
X = data[data.columns.difference(['Survived'])]
```

```
y = data['Survived']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Program output:

```
SyntaxError
unmatched ')' (, line 5)
```

The data is prepared; we can create a decision tree model that will be trained.

We will also use randomization, to achieve always the same tree.

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=42)
clf = clf.fit(X_train, y_train)
```

We have a decision tree model stored in the variable clf. Using the following line of code, we obtain a prediction for the test data, which we also display.

```
y_pred = clf.predict(X_test)
print(y_pred)
```

Program output:

```
[0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 0 1
1 0 1 0 0 1
 0 0 1 1 0 1 1 1 1 1 0 1 0 0 0 0 1 1 1 0 0 1 0 1 1 0 0 0 0 0 1
1 0 1 0 1 0
 0 0 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0
 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 0 1 1 0 0 1 1 0 0 0 0 1 0 1 0 0
1]
```

It remains to find out the classification metrics. First, we find out the accuracy – i.e. we compare the predicted values and the actual survival values attributed to the test data.

```
from sklearn import metrics
acc = metrics.accuracy_score(y_test, y_pred)
```

```
print("Accuracy:", acc)
```

Program output:

```
Accuracy: 0.7202797202797203
```

Precision, recall a f1 score will be calculated similarly.

```
from sklearn import metrics
prec = metrics.precision_score(y_test, y_pred)
print("Precision:", prec)

from sklearn import metrics
rec = metrics.recall_score(y_test, y_pred)
print("Recall:", rec)

from sklearn import metrics
f1 = metrics.f1_score(y_test, y_pred)
print("F1 score:", f1)
```

Program output:

```
Precision: 0.7916666666666666
Recall: 0.6195652173913043
F1 score: 0.6951219512195123
```

2.6.2

What is the correct code to create and train a decision tree?

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTree()
clf = clf.train(X_train, y_train)
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTree()
clf = clf.fit(X_train, y_train)
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
```

```
clf = clf.train(X_train, y_train)
```

⌨ 2.6.3

Predicted values

Complete the code to find out what values the model returns for test data that represents 30% of the total available data.

file1.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

data =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv')

data = data[['Survived', 'Pclass', 'Sex', 'Age', 'SibSp',
'Parch', 'Embarked']]

data = data.dropna()

data =
pd.get_dummies(data,columns=["Embarked"],drop_first=False)

data['Sex'] = data['Sex'].replace({'male': 0, 'female': 1})

X = data[data.columns.difference(['Survived'])]
y = data['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

clf = clf.fit(X_train, y_train)
```

2.6.4

How do we calculate accuracy?

```
acc = accuracy(y_true, y_pred)
acc = accuracy_score(y_true, y_pred)
acc = acc(y_true, y_pred)
acc = acc_score(y_true, y_pred)
```

2.6.5

Complete the calculation and listing of recall.

```
_____ sklearn import _____
rec = metrics. _____ (y_test, _____ )
print("Recall:", _____ )
```

- y_pred
- recall_score
- from
- rec
- metrics

Tree-Based Learning II.

Chapter 3

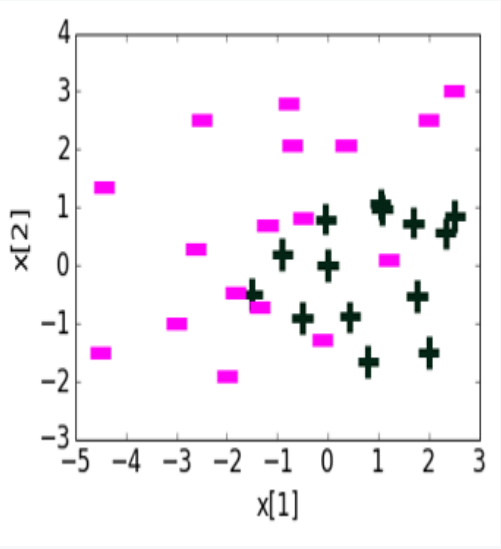
3.1 Relearning in a decision tree

🕮 3.1.1

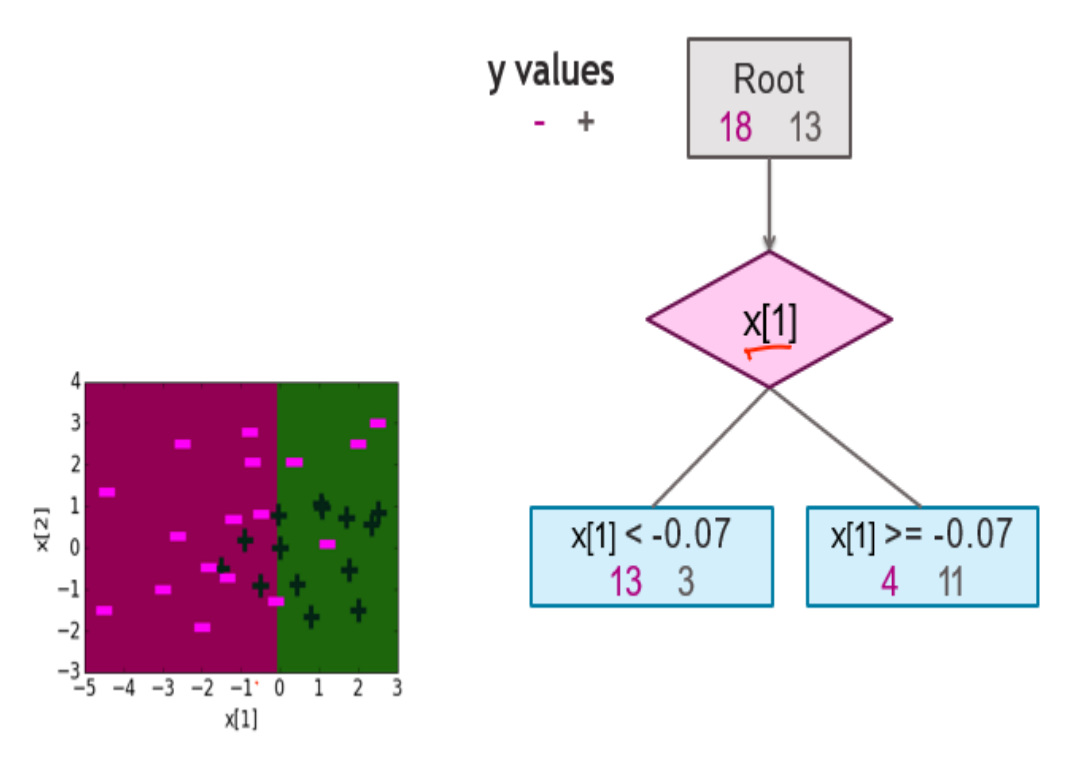
When creating the decision tree, we select individual features and sort the examples one by one according to the features. In this way, we reduce the so-called training error. Recall that the training error is computed as:

$$Classification\ Error = \frac{počet\ chybných\ klasifikácii}{počet\ všetkých\ príkladov}$$

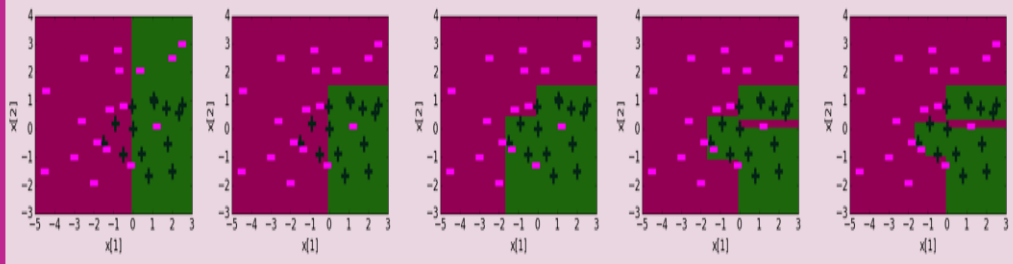
By selecting a feature for the distribution, we reduce this error. Consider only two possible features x[1] and x[2] , two classes + and - splitting as on the following picture.



By selecting the x[1] feature and choosing the threshold correctly, we can divide the examples as follows:

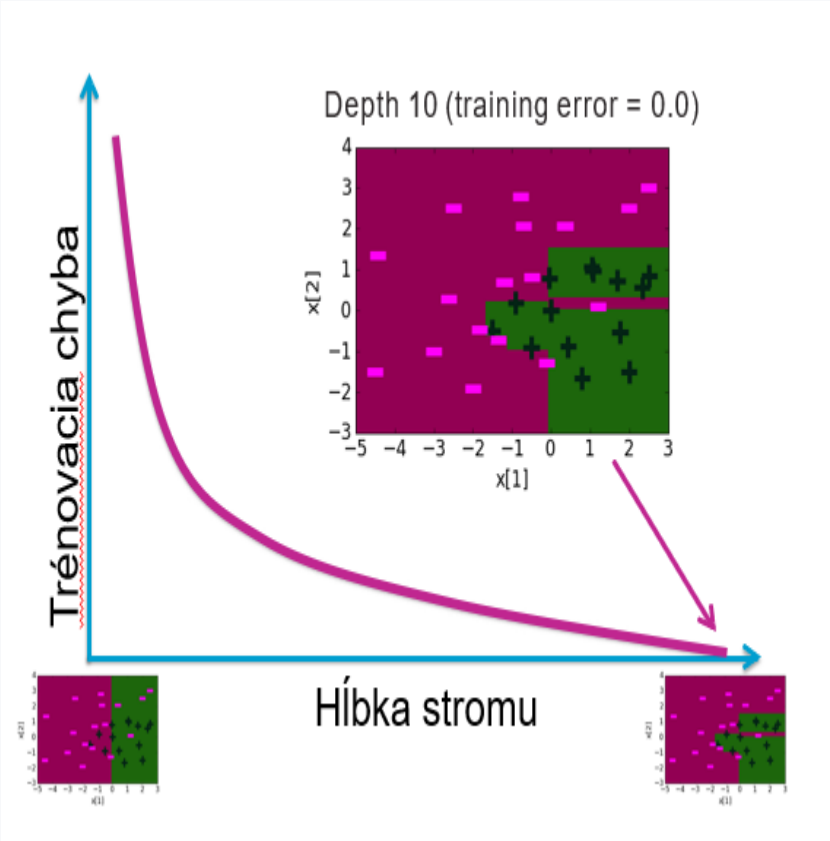


If we continued to build the tree further, we would actually increase its depth. For example, we would select x[2] feature for the next division. The question remains whether we could select x[1] and x[2] features again. In the case of numerical values, we can, but with a different threshold value. By adding more depths, we would be able to create a tree with an error equal to 0 in this example.

Tree depth	depth = 1	depth = 2	depth = 3	depth = 5	depth = 10
Training error	0.22	0.13	0.10	0.03	0.00
Decision boundary					

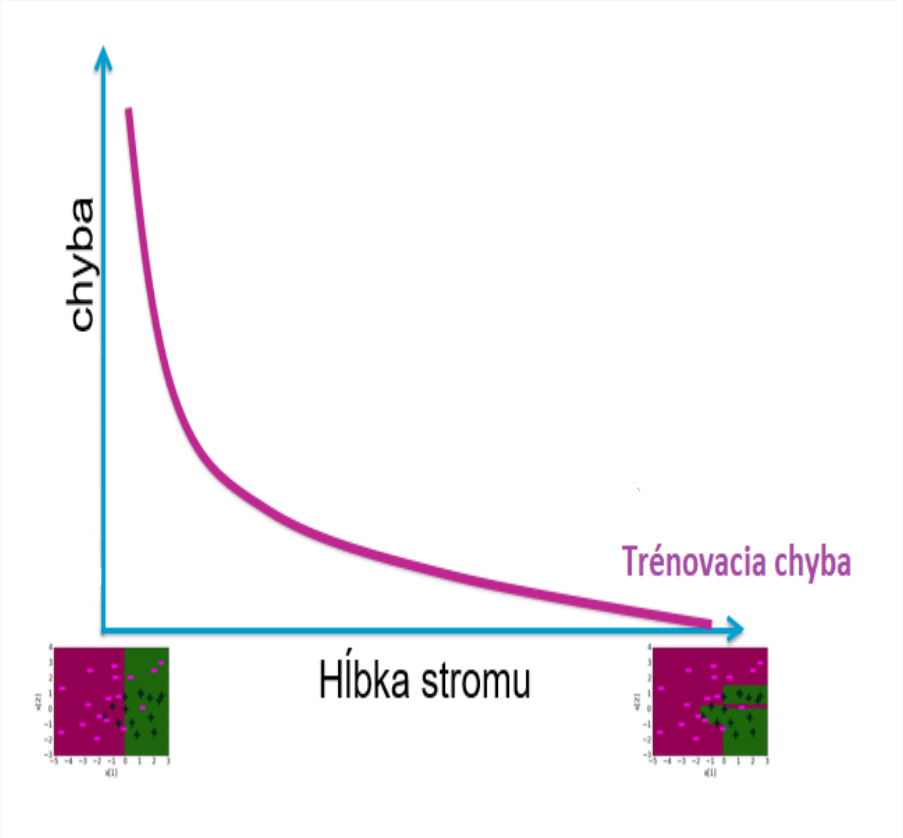
However, it is questionable whether such a deep tree, even with a classification error equal to 0, is really a good machine learning model.

From the above example, it is clear that the classification error, in our case the training error, decreases with the depth of the tree, i.e., deeper tree = lower training error.

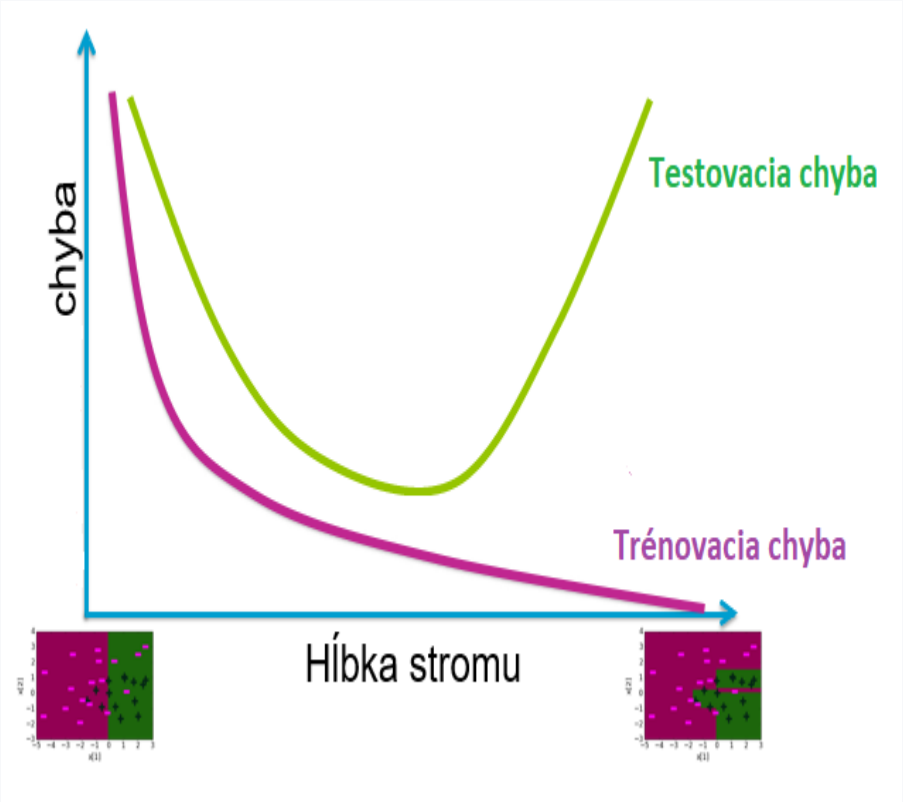


🕮 3.1.2

In the previous example, we found that the classification error decreases during training.



But we know that the model we have created always needs to be evaluated. This validation is done using examples that were not included in the creation of the model. In the previous chapters, we talked about the training set and the test set, whereby a model is built on the data from the training set and this model is validated using examples from the test set. If we validate the generated trees of available depths using test examples, we find that the error on the test examples will initially decrease, but at a certain model complexity this test error will increase.



We call this phenomenon overfitting. You will also find terms such as adaptation, exact copying or remodelling in the literature. Overfitting means that the model estimates the training examples perfectly, but it can no longer generalize and it estimates very poorly the examples other than the training ones.

🕮 3.1.3

If the training error of the model is equal to 0, the model is unlikely to be good and suitable. Therefore, the goal in creating a model is to find a model that will have the lowest testing error. At the same time, we probably intuitively suspect that a simpler decision tree model will be easier to understand and interpret.

The principle of Occam's Razor is applied in the creation of decision trees.

William Occam (Ockham) (1290-1349) was an English Franciscan philosopher and a prominent logician of the Middle Ages. He paid attention to particulars, and thus he is a forerunner of the inductive method.

The most famous is his claim that "Among competing hypotheses, the one with the fewest assumptions should be chosen."

The term Occam's razor first appeared in 1852 in the works of Sir William Rowan Hamilton (1805-1865), a long time after Occam's death.

Occam's razor can be interpreted in two slightly different ways:

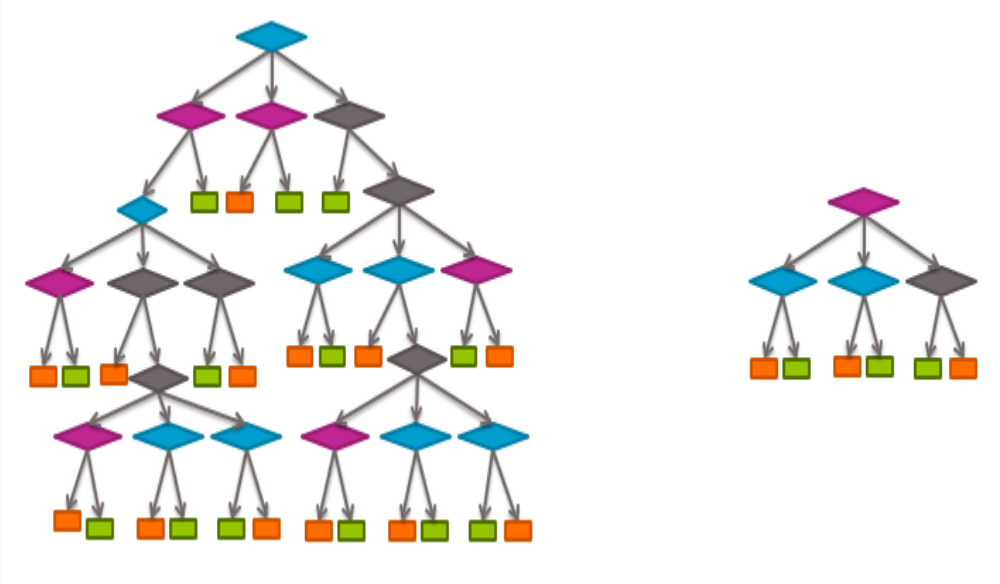
- If there are several explanations for a phenomenon, it is better to prefer the least complicated one
- If any part of the theory is not necessary to achieve the results, the theory does not include it

If we apply Occam's razor to the construction of decision trees, we can apply the following principle:

"If two trees have similar classification error in validation, the simpler one should be selected"

🕮 3.1.4

The question remains how to find out which tree is the simplest.



For the trees shown in the figure, the task of finding the simpler tree is relatively easy. However, it is important to note that such a visual evaluation is complicated for the algorithm. The algorithm needs a numerical representation of the complexity of the tree. Also, most of the time the trees we are comparing are visually very similar. In this case also the complexity of the tree needs to be expressed numerically.

📖 3.1.5

There are several methods available for constructing an ideal tree, i.e., a tree with an acceptable error that is simple enough. These can be categorized into one of the following two groups:

1. **Early Stopping - this is stopping the learning algorithm before it creates too complex tree**
2. **Pruning - this is the simplification of the tree after its creation**

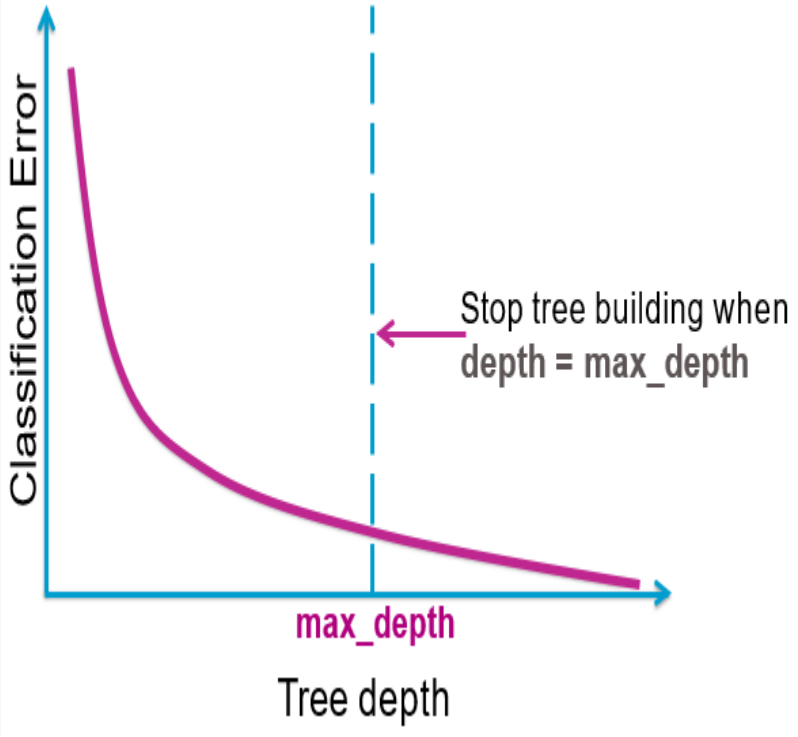
📝 3.1.6

If we want to simplify the tree after its creation, this method calls:

- Pruning
- Early Stopping

📖 3.1.7

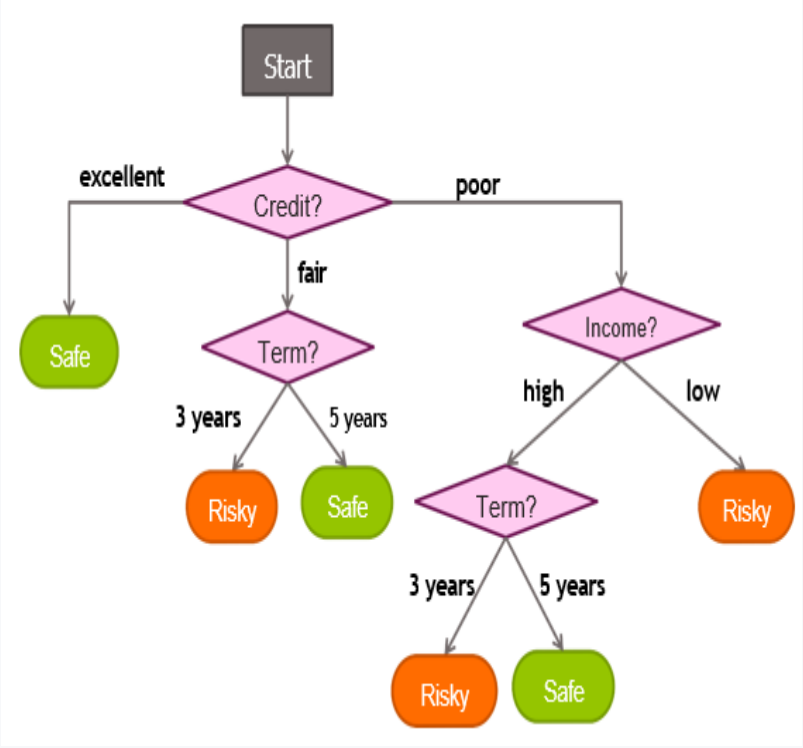
We can apply several early stopping conditions when stopping the tree building algorithm. The simplest is to set the maximum depth of the tree.



This simple condition is often used to build easily interpretable trees. Its problem, of course, is determining the appropriate depth.

3.1.8

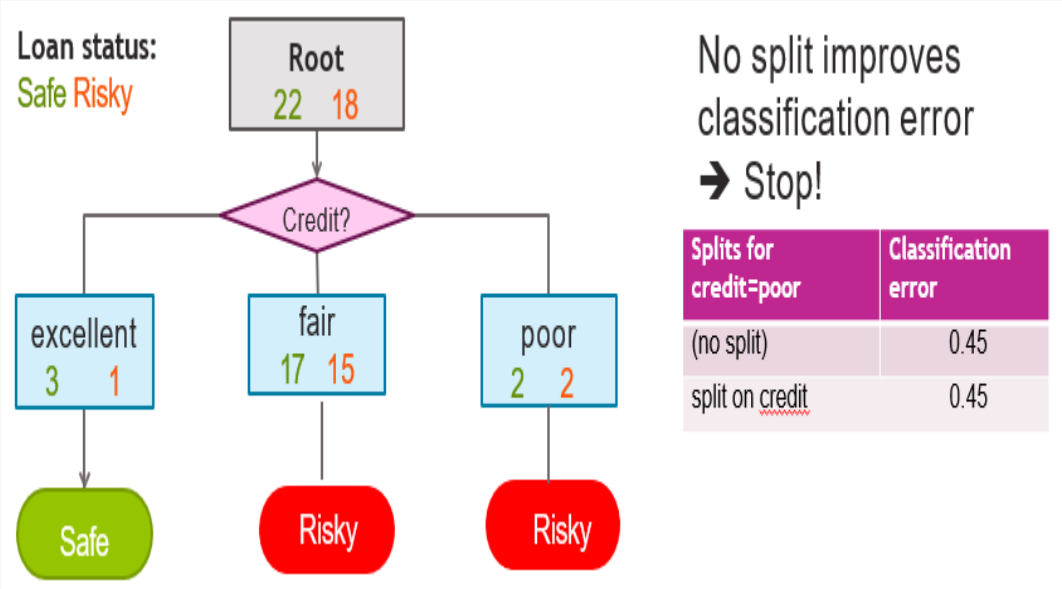
What is the depth of the tree in the picture?



- tree depth = 3
- tree depth = 6
- tree depth = 1

3.1.9

The problem of building a maximum depth tree is the depth setting itself. This is sometimes very difficult to estimate. Therefore, the second option is to detect the classification error of the generated decision tree at each possible node expansion. If the classification error does not decrease, or decreases very little, we may end up generating a decision tree.



In the example in the figure we have a tree with depth 1. Its classification error is (1 + 15 + 2) / 40 = 0.45

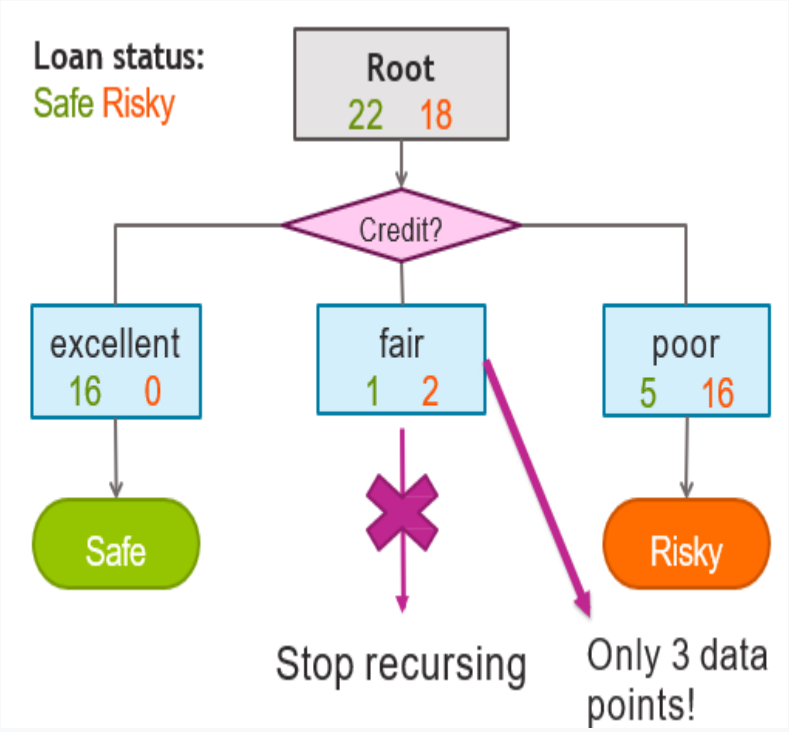
If we were to split the tree according to the **Credit** feature, the classification error would be 18 / (22 + 18) = 0.45

Under the early stopping condition, we would no longer continue generating the tree because the classification error would not decrease.

🕮 3.1.10

The third condition for stopping tree generation early is to stop the tree if the number of instances, contained in a node is very small.

For nodes with a small number of examples, their trustworthiness is questionable. However, it is important to note that what is and what is not a very small number of nodes depends on the task and the dataset used for training. Sometimes, 100 nodes is a small number, some other time it is not. Therefore, the minimum number of nodes can be set in several available programming methods for tree generation.



📖 3.1.11

To review, we present approaches to early termination of tree generation:

1. Setting the maximum tree depth
2. Stop tree generation if classification error is not reduced sufficiently (or at all)
3. Stop tree generation if the node contains too few examples

📝 3.1.12

What is overlearning or overfitting?

- A phenomenon where test error increases with model complexity.
- A phenomenon where test error decreases with model complexity.
- A phenomenon where the training error decreases with model complexity.
- A phenomenon where the training error increases with model complexity.

📝 3.1.13

"Among competing hypotheses, the one with the fewest assumptions should be selected",

This statement is also called:

- The principle of Occam's Razor
- Simple existence principle
- Regression rule
- The principle of selection of assumptions

3.1.14

Select the rules that can be applied to stop the generation of the decision tree, i.e. the stopping condition.

- If all the features from the dataset have already been used
- If the classification error is not reduced by further development
- If there are only leaves in the tree that contain a number of examples less than the set minimum of leaf for unfolding - min_samples_leaf
- If the GINI index of the whole tree is less than -1

3.2 Tree pruning

3.2.1

The early stopping conditions mentioned in the previous section belong among the quick and easy approaches to prevent overlearning in decision trees.

For example, stopping tree generation, if the classification error does not decrease appears to be a successful approach. Of course, this approach also has its limits. We can demonstrate one concerning the XOR problem.

Consider four training examples that classify a target variable y for features/attributes **x[1]** and **x[2].**

y= x[1] xor x[2]

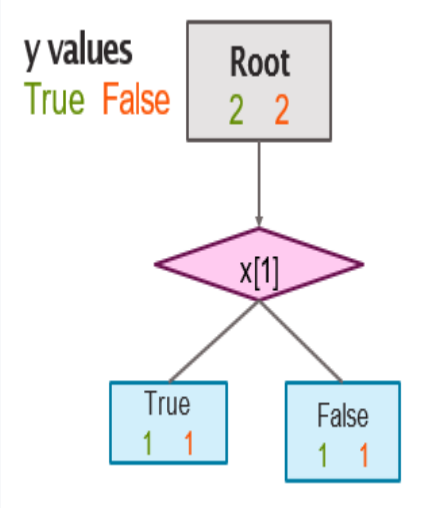
x[1]	x[2]	y
False	False	False
False	True	True
True	False	True
True	True	False

y values
True False

Root
2 2

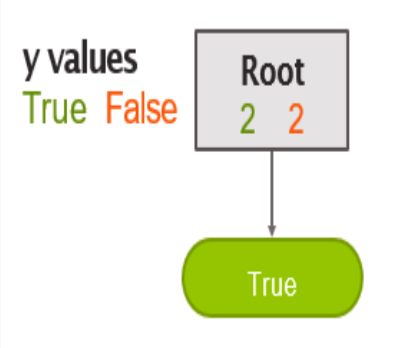
If we consider only these four training examples, the classification error of a classifier that classifies all examples as **True** (or even **False**) will be 2 / (2 +2) = 0.5

Generating the tree and dividing by the attribute **x[1]**, we get the following tree.



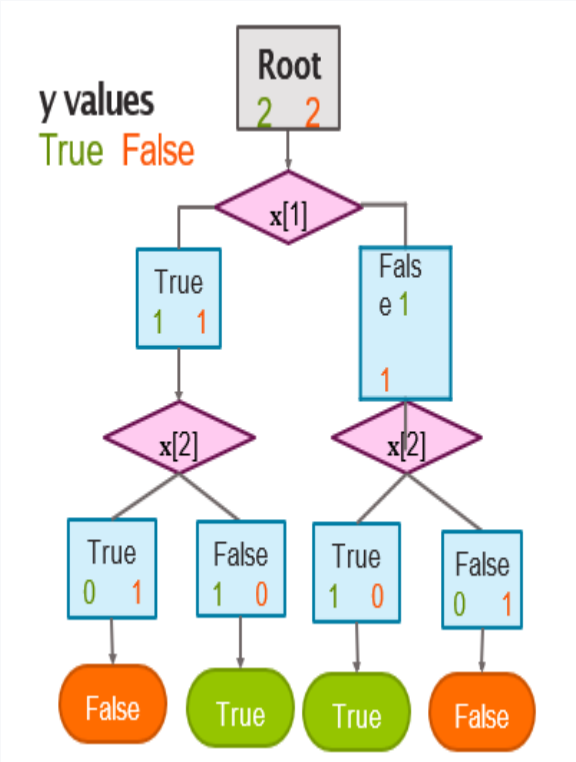
The classification error of this tree will be (1 + 1) / (2 +2) = 0.5

This means that the classification error has not decreased. If we stopped the tree generation early, we would not continue with the tree generation in this case. The final tree would look as follows:



Practically, we would not create any classifier.

However, if we further partition the tree according to the attribute **x[2]** we get the following tree with zero classification error.



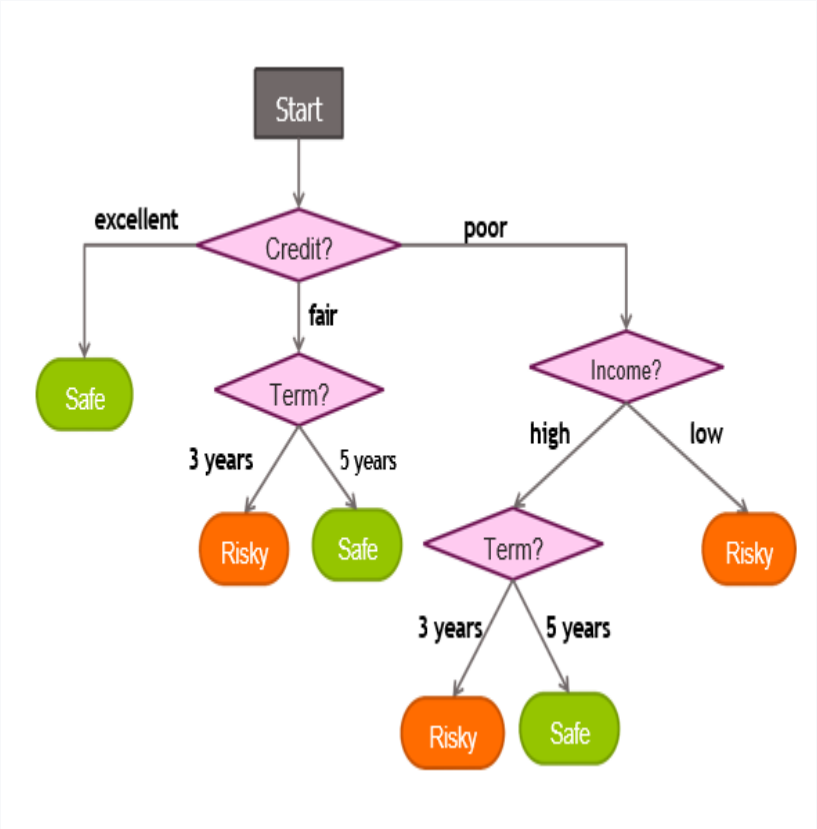
The example above illustrates the problem of the early stopping approach.

Another approach that eliminates this problem is pruning the tree.

3.2.2

Consider two decision trees

Tree A:



Tree B:

Which one of the above trees is simpler?

- Tree B
- Tree A

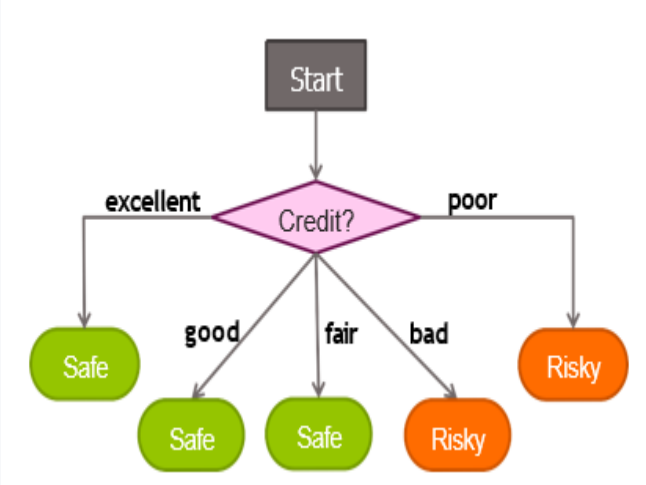
📖 3.2.3

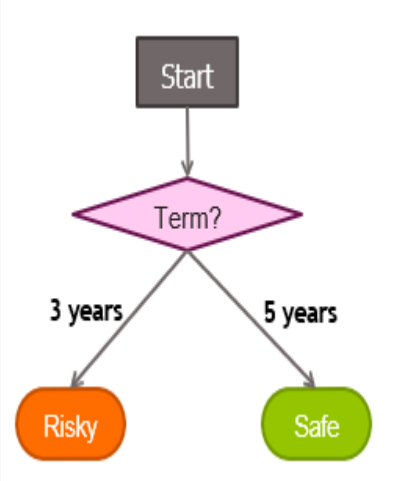
The main idea of pruning a tree is to train a complex tree that will be simplified later. An important fact in pruning is to express the simplicity of the tree - the so-called complexity.

In the case of different tree depths, it is not difficult to determine which of two trees is simpler. In the case of trees with the same depth, it is a more serious problem.

There are several metrics to determine the complexity of a tree. In our examples, we will consider the number of leaves.

For example, consider the following trees.





We express the number of nodes of the tree using the L variable.

The number of nodes of the first tree S1 will be L(S1) = 5

The number of nodes of the second tree S2 will be L(S2) = 2

By simple comparison, we find that the second tree is simpler, i.e., the first tree is more complex. Importantly, while it is obvious to a human observer which of the two trees is more complex, by expressing the node count metric, it is possible for the algorithm to detect the complexity of the tree as well.

🕮 3.2.4

In the tree pruning method, complexity alone is not enough.

A "good" tree must balance two perspectives:

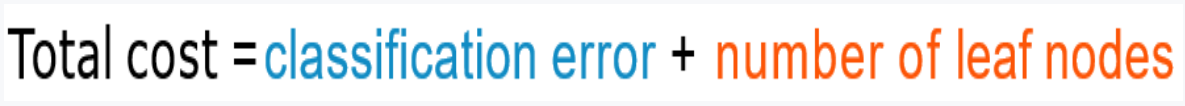
- How well it predicts data.
- The complexity of the tree.

Therefore, in the tree pruning method, we will consider the **total cost** function.

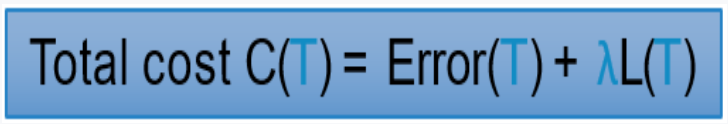
We calculate it as follows:

pas a **measure of fit,** we use a tree performance metric, e.g., classification error, and as a **measure of complexity,** the number of leaves in the tree.

Therefore, the formula can also be understood as follows:



or more appropriately, it can be expressed as follows:

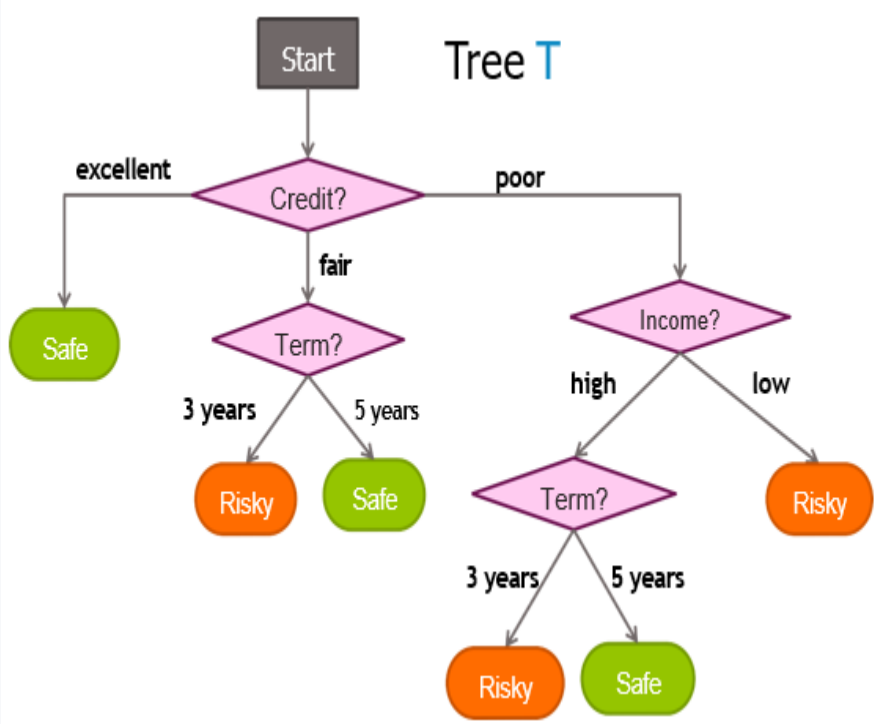


The constant **lambda** λ, which is found in the formula, is of great importance. It is important to note that while the classification error takes values with the interval <0, 1>, the number of leaves is a positive integer. The constant ensures that this number is transformed to <0, 1>, i.e. it ensures comparable values for the error as well as the number of leaves.

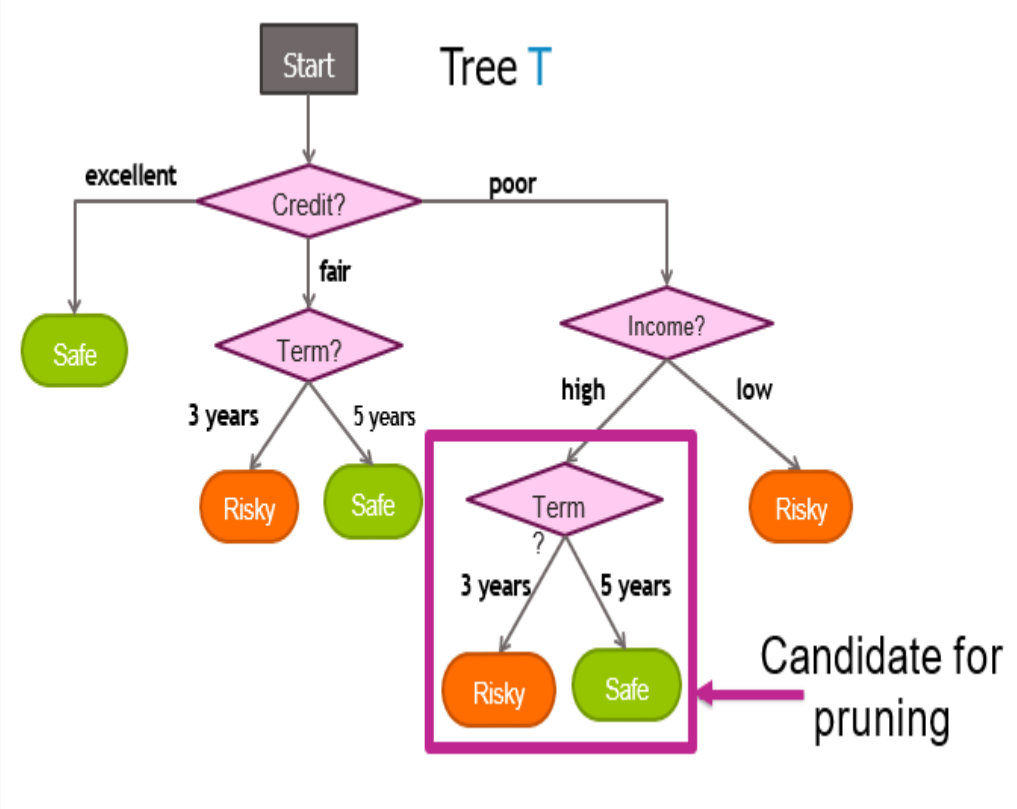
🕮 3.2.5

The tree pruning algorithm is as follows:

Step1 - Create a complete decision tree



Step 2 - Find a candidate node for removing and consider removing



Step 3 - Calculate the total cost - total cost tree with the candidate for removal T and without candidate for removal $T_{smaller}$ according to the formula:

$$C(T) = Error(T) + \lambda L(T)$$

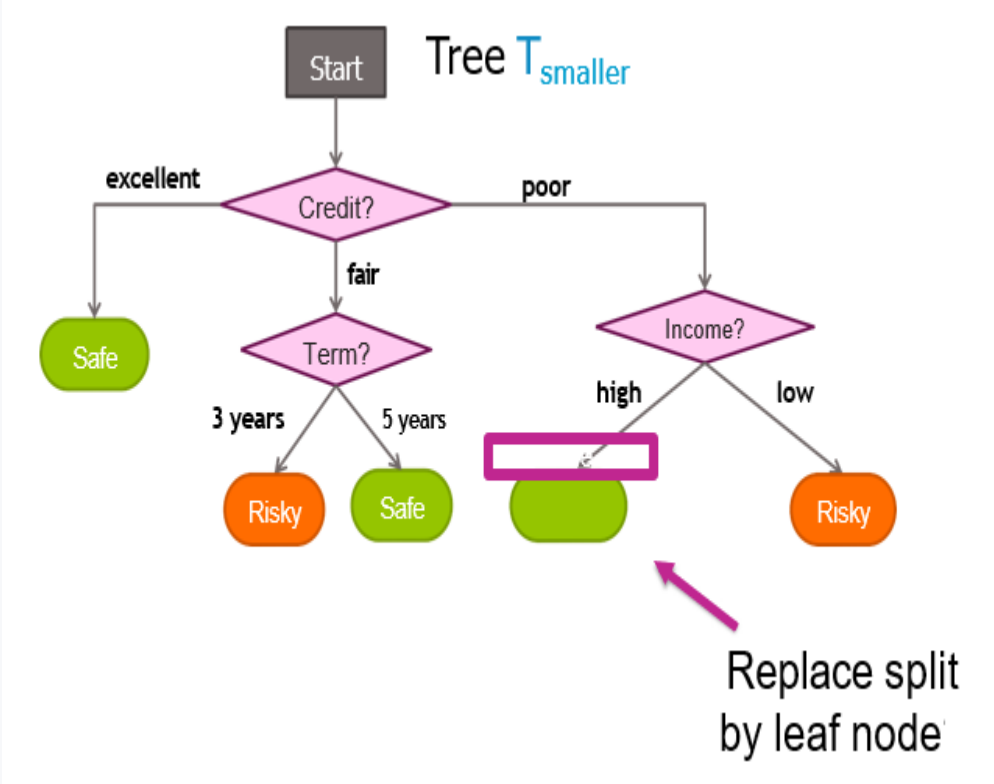
The total costs in our case look as follows:

$\lambda = 0.3$

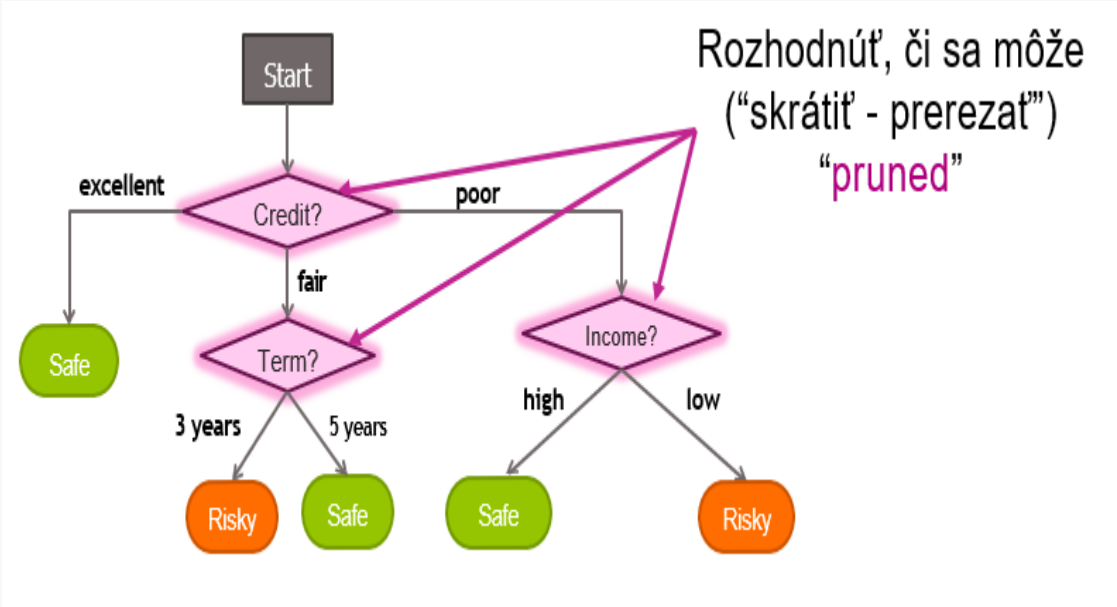
Tree	Error	#Leaves	Total
T	0.25	6	0.43
$T_{smaller}$	0.26	5	0,41

Step 4 - Compare the total cost and if it decreases after removing the candidate, then remove the candidate node.

Obviously, after removing the candidate, the value of the total cost will decrease in our example. Therefore, we can remove the candidate node.



Step 5 - Repeat steps 2 - 4 for each node in the tree



📖 3.2.6

At the end of this section we present a more elaborate Decision tree pruning algorithm

· Start at the bottom of the tree **T** and move up, apply the **prune_split** function to each decision node **M**

- **prune_split(T,M):**

1. Calculate the total cost for tree **T** using the formula **C(T) = error(T) + λ L(T)**
2. Let **Tsmaller** be the tree after pruning the subtree from the tree **M**
3. Calculate the complexity of the total cost of **Tsmaller** by the formula **C(Tsmaller) = error (Tsmaller) + λ L (Tsmaller)**
4. If **C(Tsmaller) < C(T),** then prune the tree **T** to the tree **Tsmaller**

📝 3.2.7

How the so-called Total cost is calculated for the decision tree

- as the sum of the classification accuracy metric and the model complexity metric
- as the difference of the classification accuracy metric and the model complexity metric
- as a proportion of the classification accuracy metric and the model complexity metric
- as the sum of the classification accuracy metric and the dataset balance metric
- as the difference of the classification accuracy metric and the dataset balance metric
- as a proportion of the classification accuracy metric and the dataset balance metric

3.3 Missing (incomplete) data

📖 3.3.1

In real-world machine learning tasks, incomplete datasets are a common problem.

For example, a bank's dataset of its customers. For example, the bank offered some customers a loan, while others maintained an account. Obviously, for

example, the maturity date column will not be registered for all clients. On the other hand, the bank also has information about clients who have not used credit services that it can use.

Credit	Term	Income	y
excellent	3 yrs	high	safe
fair	?	low	risky
fair	3 yrs	high	safe
poor	5 yrs	high	risky
excellent	3 yrs	low	risky
fair	5 yrs	high	safe
poor	?	high	risky
poor	5 yrs	low	safe
fair	?	high	safe

🕮 3.3.2

Approach 1:

The first approach to missing data is to disregard examples with missing data. This is possible if there are not very many such examples. But what if, for example, 50% of the records/examples do not have an attribute listed. Removing these examples would not only impoverish our dataset too much, but would even bias the machine learning model we have created. Therefore, the above approach needs to be considered responsibly.

The advantages of removing examples or attributes include:

- Easy to understand and implement
- It can be applied to any model (decision trees, logistic regression, linear regression,...)

Disadvantages include:

- Removing data points and features can remove important information from the data

- It is not clear when it is better to remove examples (rows) or when it is better to remove features/attributes (columns)
- It does not help if in the case of a model update, i.e. a model prediction, we are missing input data

🕮 3.3.3

Approach 2:

Another option is to fill in the missing data. This is mostly applied for missing data in features/columns.

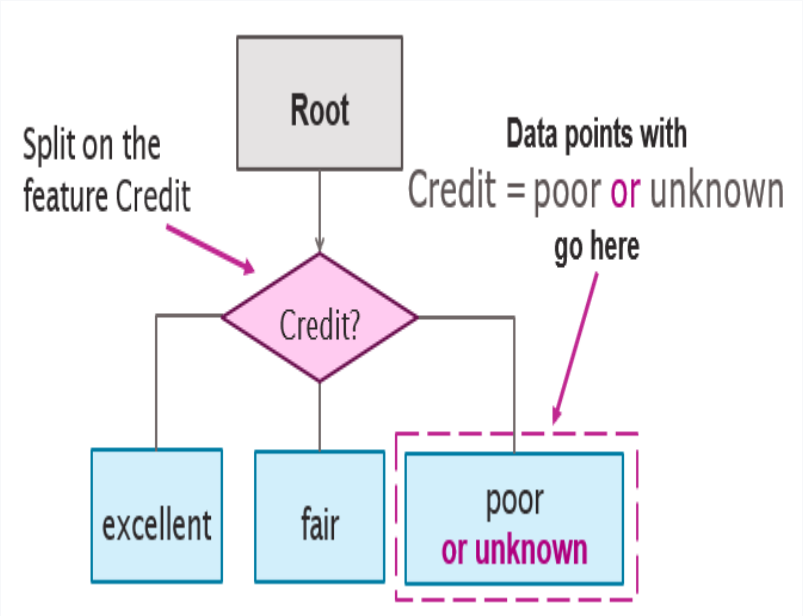
Credit	Term	Income	y
excellent	5 yrs	high	safe
fair	?	low	risky
fair	3 yrs	high	safe
poor	5 yrs	high	risky
excellent	3 yrs	low	risky
fair	5 yrs	high	safe
poor	3 yrs	high	risky
poor	?	low	safe
fair	?	high	safe

There are several approaches to data completion. These include, for example, supplementing with the most frequent value, supplementing with the average value, etc. The approaches depend on the distribution of values.

🕮 3.3.4

Approach 3:

A third approach is to reason with the missing data. The easiest one is to add, for example, the value "unknown".



This approach is particularly appropriate if we assume that even in the case of model application we will not know all the input data.

🕮 3.3.5

Literature used:

- Emily Fox, Carlos Guestrin: Machine Learning Specialization, University of Washington https://www.coursera.org/specializations/machine-learning
- Harikrishnan N B: Confusion Matrix, Accuracy, Precision, Recall, F1 Score - https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd
- Rotem Dror: Evaluation - https://www.seas.upenn.edu/~cis5190/fall2018/assets/lectures/lecture-3/03-eval.pptx

3.4 Practical tasks

📝 3.4.1

Create a decision tree using Titanic data without setting the depth of the tree.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

```
data =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv')

data = data[['Survived', 'Pclass', 'Sex', 'Age', 'SibSp',
'Parch', 'Embarked']]

data = data.dropna()

data =
pd.get_dummies(data,columns=["Embarked"],drop_first=False)

data['Sex'] = data['Sex'].replace({'male': 0, 'female': 1})

X = data[data.columns.difference(['Survived'])]
y = data['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

clf = DecisionTreeClassifier(random_state=42)
clf = clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

from sklearn import metrics
acc = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", acc)
```

Program output:

```
Accuracy: 0.7342657342657343
```

Find the depth of the generated tree.

```
print(clf.get_depth())
```

Program output:

```
17
```

The original tree has a depth of 17. Next, we find out how many leaves the tree has.

```
print(clf.get_n_leaves())
```

Program output:

```
151
```

The tree has up to 151 leaves. Based on the depth and number of leaves, we can conclude that this tree is complex given the number of data from which it is trained.

To confirm the reasoning, let us list all the accuracies of the tree using depths from 1 to 17.

```
for i in range(1,18):
    dtree = DecisionTreeClassifier(max_depth=i,
random_state=42)
    dtree.fit(X_train,y_train)
    y_pred = dtree.predict(X_test)
    print('Depth: ',i, ' accuracy:',
metrics.accuracy_score(y_test,y_pred))
```

Program output:

```
Hlbka:  1  presnost: 0.7482517482517482
Hlbka:  2  presnost: 0.7482517482517482
Hlbka:  3  presnost: 0.7272727272727273
Hlbka:  4  presnost: 0.7482517482517482
Hlbka:  5  presnost: 0.7832167832167832
Hlbka:  6  presnost: 0.7342657342657343
Hlbka:  7  presnost: 0.7762237762237763
Hlbka:  8  presnost: 0.7342657342657343
Hlbka:  9  presnost: 0.7412587412587412
Hlbka:  10  presnost: 0.7272727272727273
Hlbka:  11  presnost: 0.7132867132867133
Hlbka:  12  presnost: 0.7342657342657343
Hlbka:  13  presnost: 0.7412587412587412
Hlbka:  14  presnost: 0.7342657342657343
Hlbka:  15  presnost: 0.7482517482517482
Hlbka:  16  presnost: 0.7412587412587412
Hlbka:  17  presnost: 0.7342657342657343
```

At depth 5, the tree is 4% more accurate on the test data than at the original depth of 17. We pruned the tree, it is less complex and yet more accurate. We have prevented the tree from overtraining.

Such a tree also has a smaller number of leaves, only 22.

```
dtree = DecisionTreeClassifier(max_depth=5, random_state=42)
dtree.fit(X_train,y_train)

print(dtree.get_n_leaves())
```

Program output:

```
22
```

Another way to prevent overtraining is by using total impurity sheets and the so-called effective alpha tree (*https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html*).

So-called minimal cost complexity pruning recursively finds the weakest node. This is characterized by the effective alpha, and the nodes with the smallest effective alpha are pruned first.

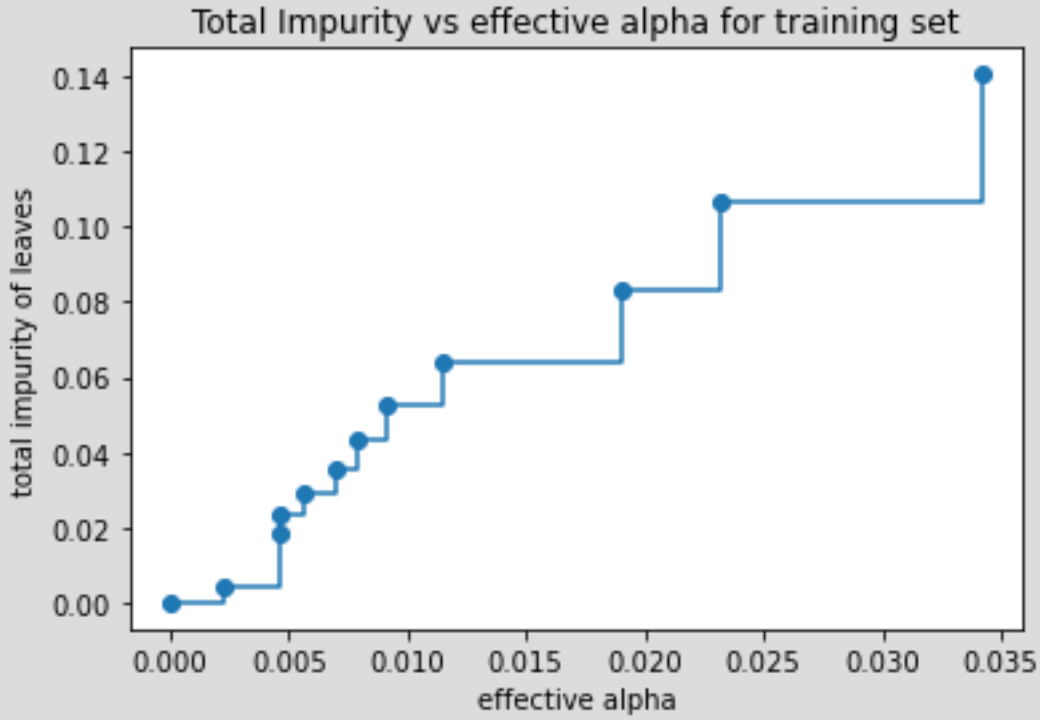
The sklearn library provides a cost_complexity_pruning_path function whose return value is the effective alpha and the corresponding total leaf impurity.

As the alpha value increases, more of the tree is pruned, increasing the total impurity of leaves.

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0)

clf = DecisionTreeClassifier(random_state=0)
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

fig, ax = plt.subplots()
ax.plot(ccp_alphas[:-1], impurities[:-1], marker="o",
drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
```

```
ax.set_title("Total Impurity vs effective alpha for training
set")
```

Program output:



We train a decision tree using the effective alpha. The last value in ccp_alpha is the value that prunes the whole tree, clfs[-1] is the tree with one node.

```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0,
ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
print(
    "Number of nodes in the last tree is: {} with ccp_alpha:
{}".format(
        clfs[-1].tree_.node_count, ccp_alphas[-1]
    )
)
```

Program output:

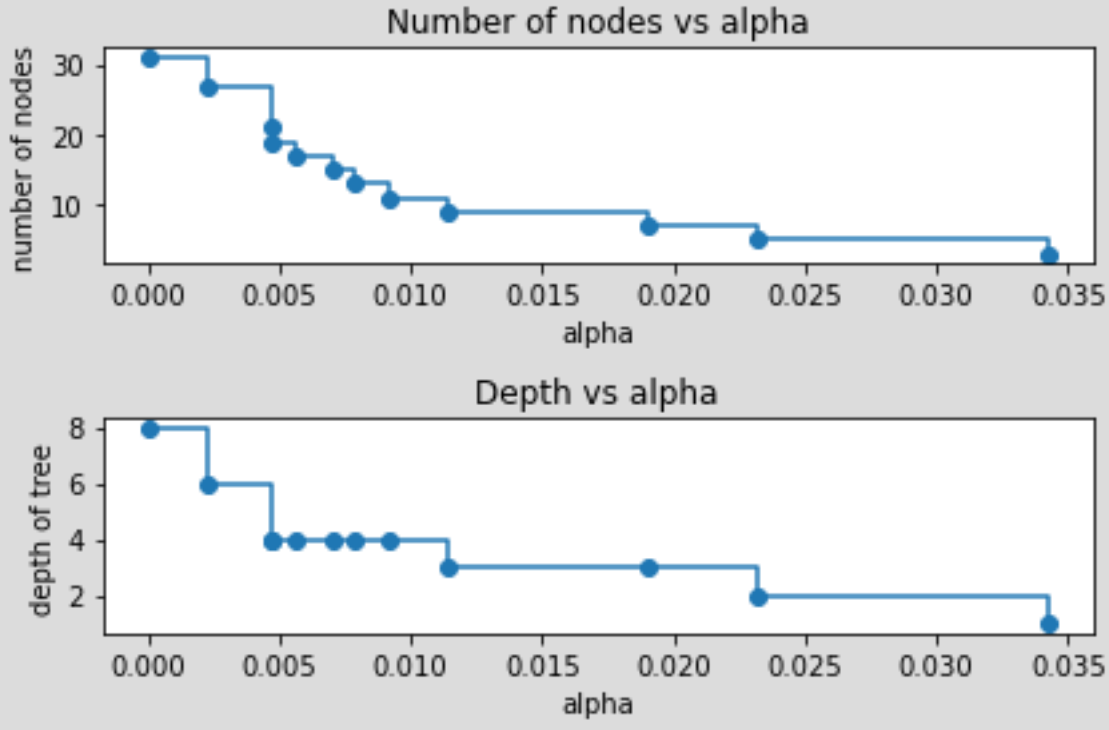
```
Number of nodes in the last tree is: 1 with ccp_alpha:
0.3272984419327777
```

Remove the last element clfs and ccp_alha.

The following graphs show how the number of nodes and the depth of the tree decreases with increasing alpha.

```
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

node_counts = [clf.tree_.node_count for clf in clfs]
depth = [clf.tree_.max_depth for clf in clfs]
fig, ax = plt.subplots(2, 1)
ax[0].plot(ccp_alphas, node_counts, marker="o",
drawstyle="steps-post")
ax[0].set_xlabel("alpha")
ax[0].set_ylabel("number of nodes")
ax[0].set_title("Number of nodes vs alpha")
ax[1].plot(ccp_alphas, depth, marker="o", drawstyle="steps-
post")
ax[1].set_xlabel("alpha")
ax[1].set_ylabel("depth of tree")
ax[1].set_title("Depth vs alpha")
fig.tight_layout()
```

Program output:

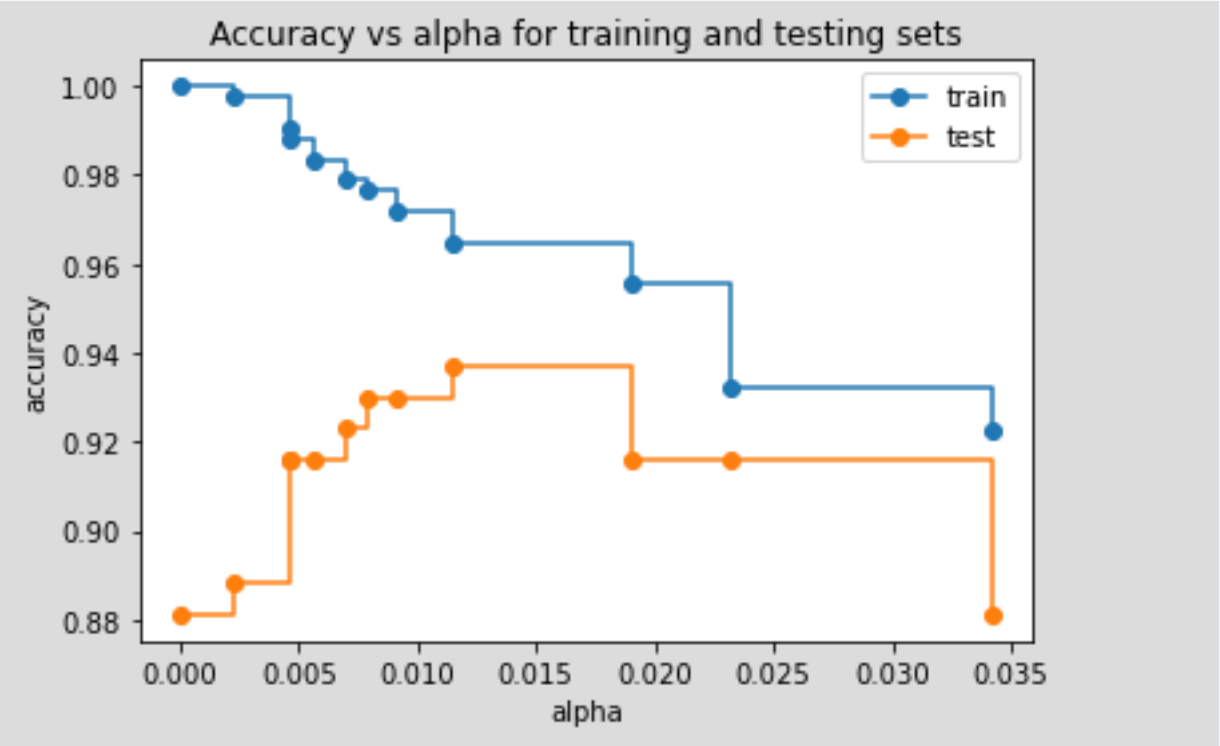


When ccp_alpha is set to zero and the other parameters are default, the tree is retrained, resulting in 100% training accuracy and 88% testing accuracy.

As alpha increases, more of the tree is pruned, leading to better generalization. In the following example, alpha is set to 0.015 to maximize testing accuracy.

```
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]

fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing
sets")
ax.plot(ccp_alphas, train_scores, marker="o", label="train",
drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker="o", label="test",
drawstyle="steps-post")
ax.legend()
plt.show()
```

Program output:



3.4.2

Complete the code so that it correctly calculates tree complexity and alpha.

```
_____ = DecisionTreeClassifier(random_state=0)
_____ = clf. _____ (X_train, y_train)
ccp_alphas, _____ = path. _____ , _____ .impurities
```

3.4.3

Assign the correct functions.

To obtain the depth of the decision tree, the following is used: _____

To obtain the number of leaves in the decision tree, the following is used: _____

To create a decision tree, the following is used: _____

To train the decision tree, the following is used: _____

- .fit()
- .get_depth()
- DecisionTreeClassifier()
- .get_n_leaves()

3.4.4

Maximum depth and number of leaves

Complete the code so that the tree has a maximum depth of 5 and list the number of leaves of the tree.

file1.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

```
data =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv')

data = data[['Survived', 'Pclass', 'Sex', 'Age', 'SibSp',
'Parch', 'Embarked']]

data = data.dropna()

data =
pd.get_dummies(data,columns=["Embarked"],drop_first=False)

data['Sex'] = data['Sex'].replace({'male': 0, 'female': 1})

X = data[data.columns.difference(['Survived'])]
y = data['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

clf = DecisionTreeClassifier(random_state=42)
```

Tree-Based Learning III.

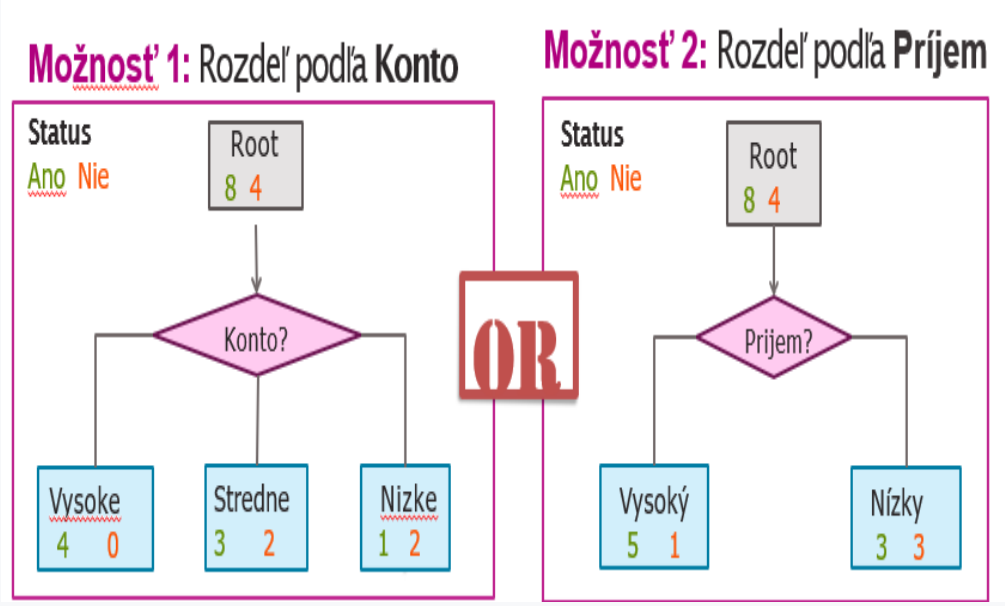
Chapter 4

4.1 GINI index

📖 4.1.1

An important step of the algorithm for decision tree formation is to select the best feature for the distribution. The algorithm considers all potential trees of depth 1.

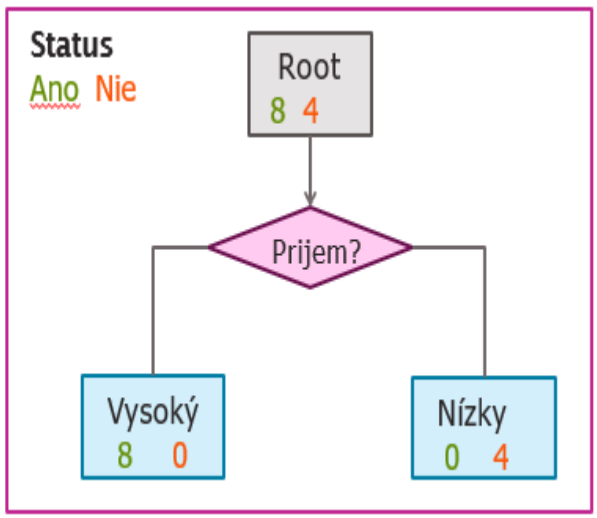
In the previous examples, we used classification error to find the most appropriate feature.



The algorithm selects the feature whose tree has the smallest classification error.

$$Classification\ Error = \frac{počet\ chybných\ klasifikácii}{počet\ všetkých\ príkladov}$$

In this case, the classification error tells us about the disorder of the subsets after distribution according to the selected feature. The ideal state is, if the subsets created contain only examples of one class, then the classification error is zero.



Thus, in decision trees, it is necessary to express the disorderedness of the subsets formed after distribution according to the selected feature.

🕮 4.1.2

There are several ways to quantify **impure** (disorder, impurity) in sets. The problem of classification error is that it is linear. For this reason, methods for constructing decision trees use the information gain or the GINI index.

The GINI index measures the rate or probability of misclassification of a particular variable when it is randomly selected. If all the elements belong to one class, it can be called pure.

The degree of the Gini index oscillates between 0 and 1, where **0** indicates that all elements **belong to a certain class** or if there is only one class, and **1** indicates that the elements are randomly distributed into different classes. A Gini index of 0.5 indicates equally distributed elements in some classes.

We calculate the Gini index according to the formula:

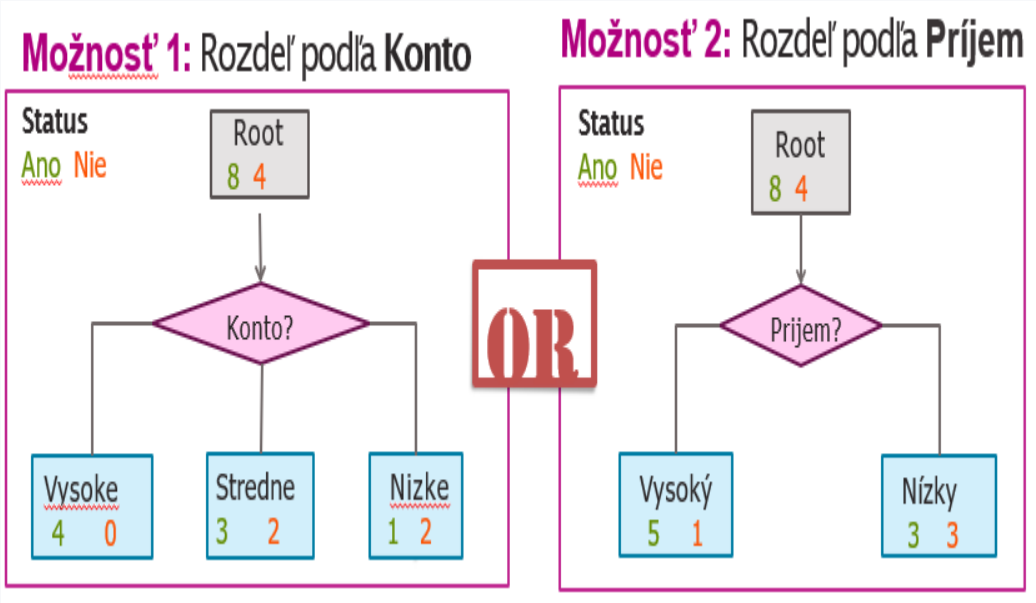
$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

- where p_i je is the probability of an object being assigned to a particular class.

📖 4.1.3

We will show the calculation of the Gini index with a practical example.

Consider 12 dataset examples and two properties, **Account** and **Income.**



We first compute the Gini index for the distribution under the **Account** features. The Gini index will be the weighted average of the Gini indexes of each subset.

$$Gini_{Konto} = weighted\ average(Gini)$$

For each subset, we will calculate the Gini index according to the formula above

$$\text{Gini} = 1 - \sum_{i=1}^{n} (p_i)^2$$

- where p_i je is the probability of an object being assigned to a particular class.

In the case of **High**, **Medium** and **Low** classes, the formulas will be as follows:

$$Gini_{Vysoke} = 1 - \left((propability\ of\ "Ano")^2 + (propability\ of\ "Nie")^2\right)$$
$$Gini_{Stredne} = 1 - \left((propability\ of\ "Ano")^2 + (propability\ of\ "Nie")^2\right)$$
$$Gini_{Nizke} = 1 - \left((propability\ of\ "Ano")^2 + (propability\ of\ "Nie")^2\right)$$

In the case of the Gini for the class **High**, where the examples are divided into **4** from the value of the target variable **Yes** and **0** from the value of **No**, we calculate the Gini as follows:

$$Gini_{Vysoke} = 1 - \left((propability\ of\ "Ano")^2 + (propability\ of\ "Nie")^2\right)$$
$$= 1 - \left(\left(\frac{4}{4+0}\right)^2 + \left(\frac{0}{4+0}\right)^2\right) = 1 - (1^2 + 0^2) = 0$$

Similarly, we calculate the Gini for the **Medium** and **Low** classes

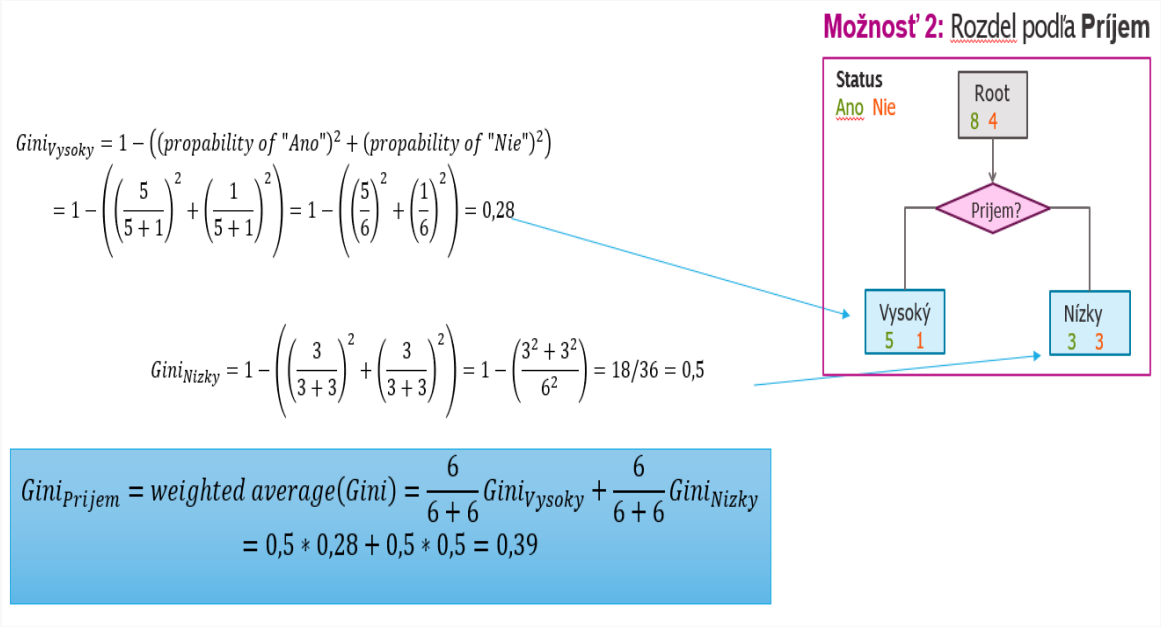
$$Gini_{Stredne} = 1 - \left(\left(\frac{3}{3+2}\right)^2 + \left(\frac{2}{3+2}\right)^2\right) = 1 - \left(\frac{3^2+2^2}{5^2}\right) = 1 - 13/25 = 0{,}48$$

$$Gini_{Nizke} = 1 - \left(\left(\frac{1}{1+2}\right)^2 + \left(\frac{2}{1+2}\right)^2\right) = 1 - \left(\frac{1^2+2^2}{3^2}\right) = 1 - 5/9 = 0{,}45$$

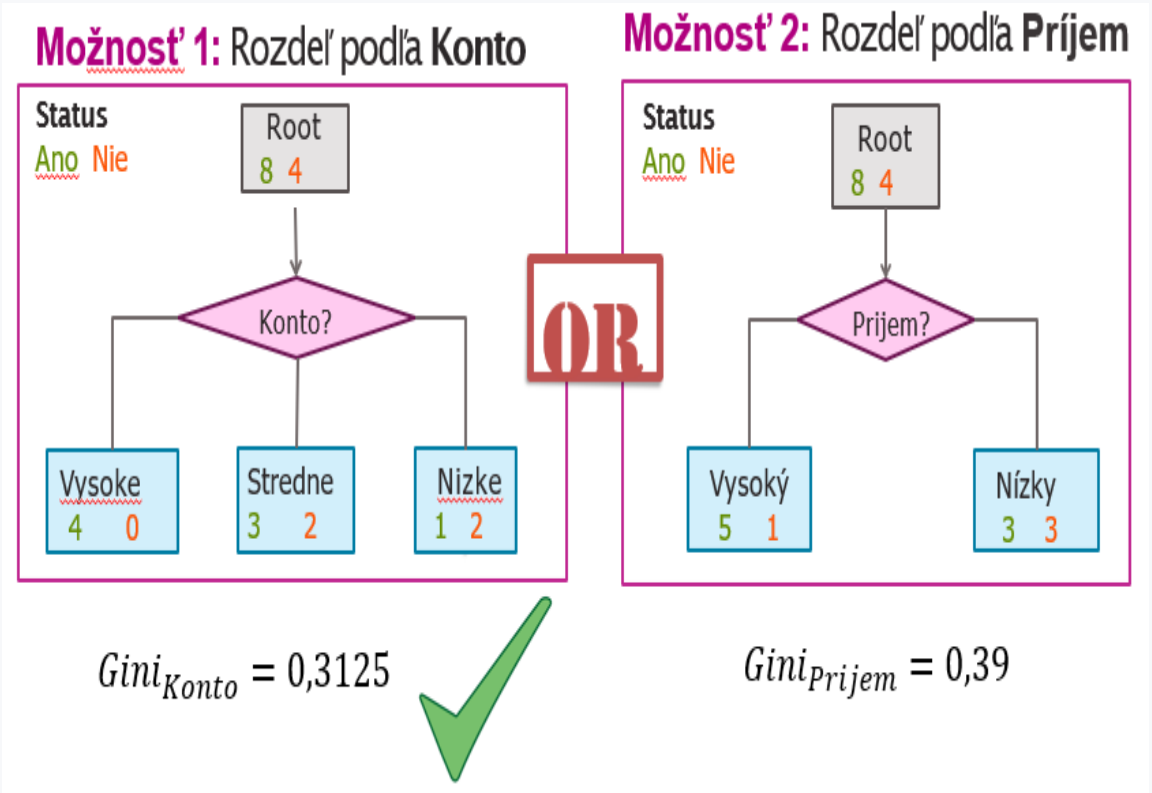
We calculate the final Gini for the Account trait as a weighted average

$$Gini_{Konto} = weighted\ average(Gini) = \frac{4}{4+5+3} Gini_{Vysoke} + \frac{5}{4+5+3} Gini_{Stredne} + \frac{3}{4+5+3} Gini_{Nizke}$$
$$= \frac{4}{12} * 0 + \frac{5}{12} * 0{,}48 + \frac{3}{12} * 0{,}45 = 0{,}3125$$

Similarly, we proceed for the **Income** feature



Finally, we need to compare the two calculated Gini values. After the comparison, we find that the best feature for the distribution will be the **Account** feature.



4.2 Entropy

📖 4.2.1

Another, often used way to quantify **impure** in sets is **entropy**. It is a measure taken from physics and it expresses the degree of disorder in a system, or otherwise, characterizes the (dis)purity in an arbitrary set of examples.

Given a set ***S***, containing only positive and negative examples of some target concept. Then the entropy of the set ***S***, corresponding to this simple example of binary classification, is defined as:

$$Entropy(S) = - p_p log_2 p_p - p_n log_2 p_n$$

where p_p is the proportion of positive examples in ***S*** and p_n is the proportion of negative examples in ***S***.

In all entropy calculations, we define 0log0 equal to 0.

Entropy is equal to **0** if all members of ***S*** belong to the same class.

For example, if all members are positive (p_p= 1),

then p_n is 0,

and Entropy(S) = $-1*\log_2(1) - 0*\log_2 0 = -1*0 - 0*\log_2 0 = 0$.

Entropy reaches its maximum value, i.e. 1, if the set of examples contains the same number of positive and negative examples.

If the set contains different numbers of positive and negative examples, the entropy ranges from 0 to 1.

📖 4.2.2

The entropy defined in the previous section was the so-called binary entropy.

$$Entropy(S) = - p_p log_2 p_p - p_n log_2 p_n$$

where p_p is the proportion of positive examples in ***S*** and p_n is the proportion of negative examples in ***S***.

In general, entropy is defined by this relationship:

$$Entropy(H) = -\sum_{t=1}^{T}(p_t \log_2 p_t)$$

where p_t is the proportion of examples of class **t** out of all classes **T** in set **H**.

We refer this entropy as non-binary entropy. The logarithm is still with base 2 because the entropy is a measure of the expected length of the encoding in bits.

🕮 4.2.3

Consider a binary entropy defined by the relation:

$$\text{Entropy(S)} = -\,p_p \log 2\, p_p \; - p_n \log_2 p_n$$

If we have a set R1 containing 6 elements "**a**" and 2 elements "**b**".

$$R1 = \{a, a, a, a, a, a, b, b\}$$

We calculate the entropy for this set as follows:

$$entropy(R1) = -\left[\frac{6}{8} log_2 \frac{6}{8} + \frac{2}{8} log_2 \frac{2}{8}\right] = -(-0{,}3112 + (-0{,}5)) = 0{,}8112$$

Next, consider the set R2, which has the same ratio of elements, but more elements of "**b**"

$$R2 = \{a, a, b, b, b, b, b, b\}$$

$$entropy(R2) = -\left[\frac{2}{8} log_2 \frac{2}{8} + \frac{6}{8} log_2 \frac{6}{8}\right] = -(-0{,}5 + (-0{,}3112)) = 0{,}8112$$

Note that the entropy rate is the same. Thus, for entropy, it is not important which elements are more, what is important is the ratio of the number of elements in the set.

Next, we can consider the sets R3 and R4 and their entropies.

$$R3 = \{a, b, b, b, b, b, b, b\}$$

$$R4 = \{a, a, a, b, b, b, b, b\}$$

$$entropy(R3) = -\left[\frac{1}{8} log_2 \frac{1}{8} + \frac{7}{8} log_2 \frac{7}{8}\right] = -(-0{,}375 + (-0{,}1686)) = 0{,}5436$$

$$entropy(R4) = -\left[\frac{3}{8} log_2 \frac{3}{8} + \frac{5}{8} log_2 \frac{5}{8}\right] = -(-0{,}5306 + (-0{,}4237)) = 0{,}9543$$

Note that the set R3 is more ordered, i.e. it contains most of the elements from "**b**" and only one element from "**a**". Its entropy is therefore lower. Also note that the sets R1 and R2 have two elements different, therefore their entropy is greater than the entropy of R2 set, but less than the entropy of R4 set.

For completeness, we still present the calculation of the ideally ordered R5 set, i.e., the set with no impurity.

$$R5 = \{a, a, a, a, a, a, a, a\}$$

$$entropy(R5) = -\left[\frac{8}{8} log_2 \frac{8}{8} + \frac{0}{8} log_2 \frac{0}{8}\right] = -(0 + 0) = 0$$

For comparison, we also present the most disordered set for 8 elements, the R6 set.

$$R6 = \{a, a, a, a, b, b, b, b\}$$

$$entropy(R6) = -\left[\frac{4}{8} log_2 \frac{4}{8} + \frac{4}{8} log_2 \frac{4}{8}\right] = -(0{,}5 * (-1) + 0{,}5 * (-1)) = 1$$

📖 4.2.4

Practical example:

Calculate the entropy rate for the target variable **Edible** from the given dataset.

Color	Size	Shape	Edible?
Yellow	Small	Round	+
Yellow	Small	Round	-
Green	Small	Irregular	+
Green	Large	Irregular	-
Yellow	Large	Round	+
Yellow	Small	Round	+
Yellow	Small	Round	+
Yellow	Small	Round	+
Green	Small	Round	-
Yellow	Large	Round	-
Yellow	Large	Round	+
Yellow	Large	Round	-
Yellow	Large	Round	-
Yellow	Large	Round	-
Yellow	Small	Irregular	+
Yellow	Large	Irregular	+

The target variable **Edible** contains 9 positive examples (+) and 7 negative examples (-). There are 16 examples in total.

We calculate the entropy for the **Edible** variable as follows:

$$entropy(Edible) = -\left[\frac{9}{16} log_2 \frac{9}{16} + \frac{7}{16} log_2 \frac{7}{16}\right] = 0{,}9836$$

The result of 0.9836 is a number very close to 1. Thus, it means that the set has a high degree of disorder.

4.3 Information Gain

📖 4.3.1

We use the entropy measure in the decision tree generation algorithm to calculate the appropriateness of an attribute. However, we do not use entropy there directly but as part of the so-called information gain measure.

Information gain is the **expected reduction** in entropy **caused** by the distribution of examples related to a given **attribute**.

The Information **Gain (S, A)** of attribute **A** corresponding to a set of examples **S** is defined as:

$$Gain(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where ***Values(A)*** is a set of all values for the attribute ***A***, S_v is a subset of ***S*** for which the attribute has the value of ***v***.

📖 4.3.2

We will show the calculation and use of information gain with a practical example.

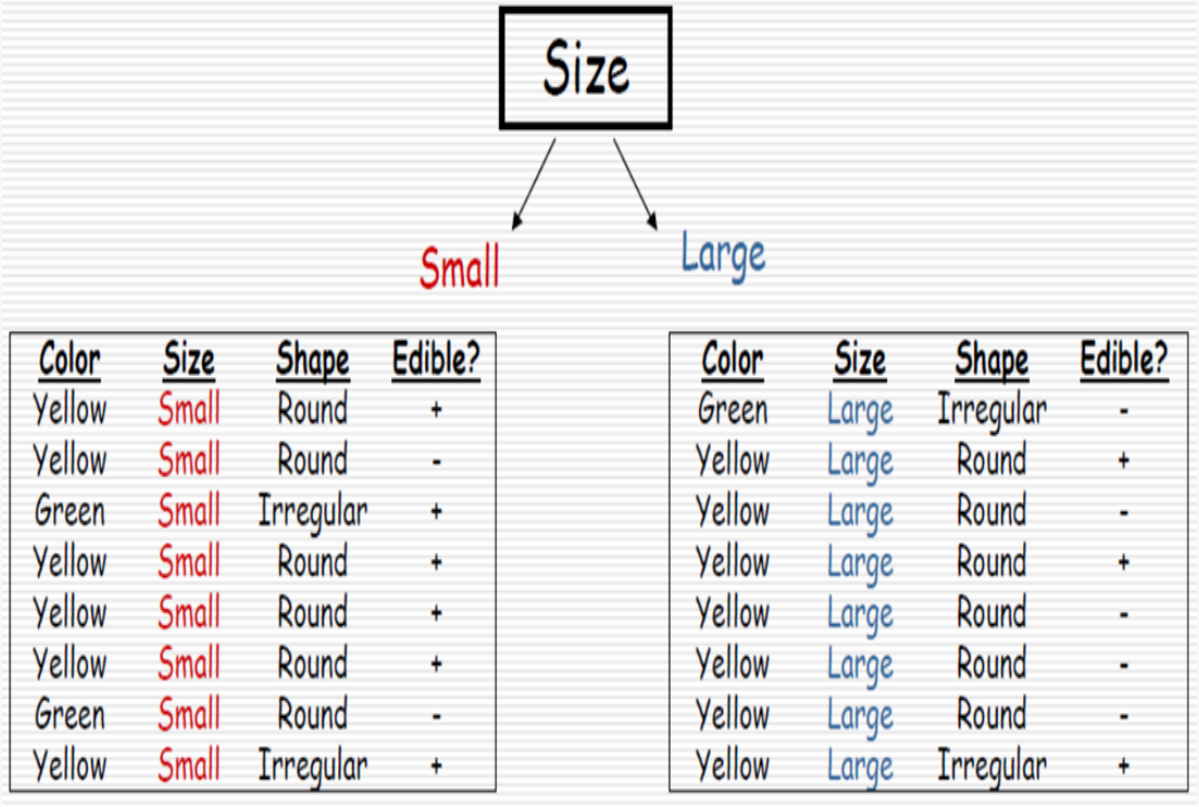
In the previous section, we worked with the following dataset. The task is to calculate the information gain for the attribute **Size**.

Color	Size	Shape	Edible?
Yellow	Small	Round	+
Yellow	Small	Round	-
Green	Small	Irregular	+
Green	Large	Irregular	-
Yellow	Large	Round	+
Yellow	Small	Round	+
Yellow	Small	Round	+
Yellow	Small	Round	+
Green	Small	Round	-
Yellow	Large	Round	-
Yellow	Large	Round	+
Yellow	Large	Round	-
Yellow	Large	Round	-
Yellow	Large	Round	-
Yellow	Small	Irregular	+
Yellow	Large	Irregular	+

In the previous section, we calculated the entropy for the target variable **Edible**:

$$entropy(Edible) = -\left[\frac{9}{16}log_2\frac{9}{16} + \frac{7}{16}log_2\frac{7}{16}\right] = 0{,}9836$$

If we divide the dataset by the **Size** attribute, we get the following subsets:



Color	Size	Shape	Edible?
Yellow	Small	Round	+
Yellow	Small	Round	-
Green	Small	Irregular	+
Yellow	Small	Round	+
Yellow	Small	Round	+
Yellow	Small	Round	+
Green	Small	Round	-
Yellow	Small	Irregular	+

Color	Size	Shape	Edible?
Green	Large	Irregular	-
Yellow	Large	Round	+
Yellow	Large	Round	-
Yellow	Large	Round	+
Yellow	Large	Round	-
Yellow	Large	Round	-
Yellow	Large	Round	-
Yellow	Large	Irregular	+

We calculate the entropy for each subset:

$$Entropy(size = small) = -\left[\left(\frac{6}{8}log_2\frac{6}{8}\right) + \left(\frac{2}{8}log_2\frac{2}{8}\right)\right] = -[(0{,}75 * log_2 0{,}75) + (0{,}25 * log_2 0{,}25)] = 0{,}8113$$

$$Entropy(size = large) = -\left[\left(\frac{3}{8}log_2\frac{3}{8}\right) + \left(\frac{5}{8}log_2\frac{5}{8}\right)\right] = 0{,}9544$$

then we calculate the entropy for the **Size** attribute by weighted averages.

$$Entropy(size) = \frac{8}{16} * 0{,}8113 + \frac{8}{16} * 0{,}9544 = 0{,}8828$$

Finally, we compute the information gain (or entropy reduction) of selecting the **Size** attribute, which we compute as the reduction of the entropy of the original dataset by the entropy of the **Size** attribute.

$$InfGain(attrib) = Entropy(parent) - Entropy(attrib)$$

$$InfGain(size) = Entropy(all\ dataset) - Entropy(size) = 0{,}9836 - 0{,}8828 = 0{,}1008$$

So, we obtained 0.1008 bits of information about the dataset by selecting "size" as the first branch of our decision tree.

📝 4.3.3

Let be the probability of an object being assigned to a particular class.

We calculate it using the following formula:

- Gini index
- Entropy
- Information gain
- Precision
- Accuracy

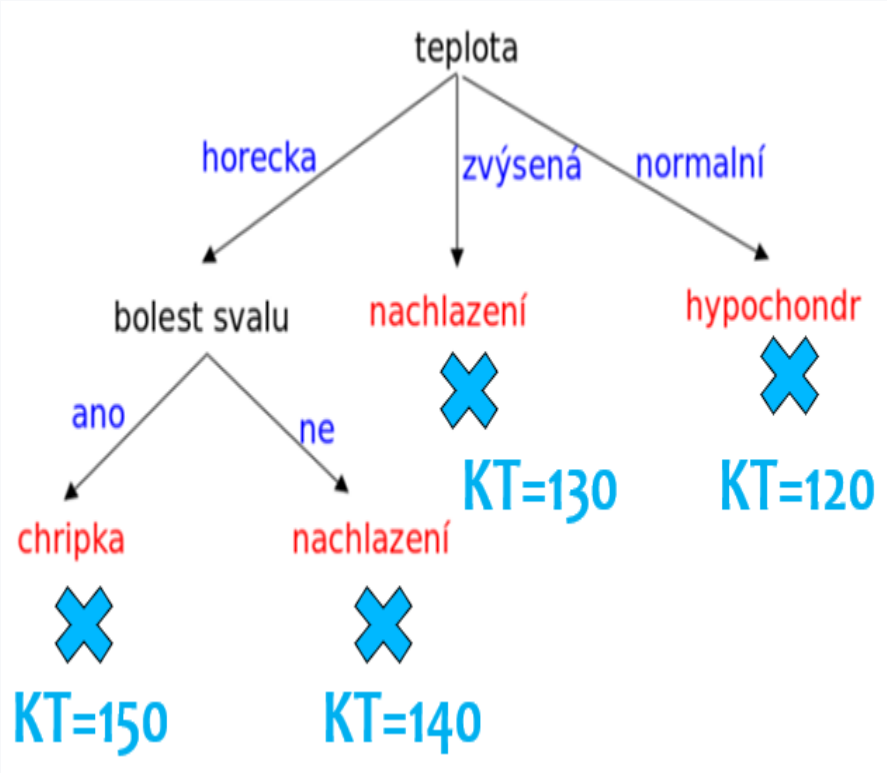
4.4 How to use numeral values?

🕮 4.4.1

All previous examples have focused on categorical data. Decision trees can also handle numeric data, i.e. continuous variables. Here we need to distinguish whether the numeral data is in the target variable or in the individual attributes.

In the case of the target variable, we speak of so-called **regression trees**. These do not model a nominal variable (flu, cold, hypochondria), but model a continuous variable, e.g. blood pressure (BP).

The value of the continuous variable is usually the average of the corresponding cases in the class. As node selection metric, e.g., standard deviation reduction is used.



4.4.2

What value do the regression trees model?

- continuous
- nominal
- categorical
- absent

4.4.3

We will show the use of continuous values in attributes by an example. Consider the following dataset of loan applicants.

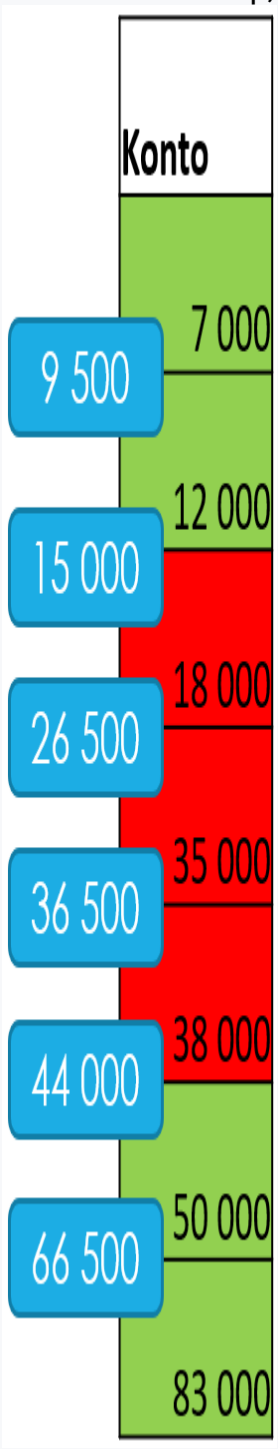
Prijem	Konto	Pohlavie	Neza-mestnany	Uver
Vysoky	12000	Zena	Nie	Ano
Vysoky	7000	Muz	Nie	Ano
Vysoky	35000	Muz	Nie	Nie
Nizky	50000	Zena	Ano	Ano
Vysoky	83000	Muz	Ano	Ano
Nizky	18000	Zena	Ano	Nie
Nizky	38000	Muz	Nie	Nie

The **Account** variable is continuous. It is possible to calculate the degree of disorder for this attribute. In this example, we will use the Gini index as the measure of disorder. Therefore, we will calculate the Gini index of the **Account** attribute.

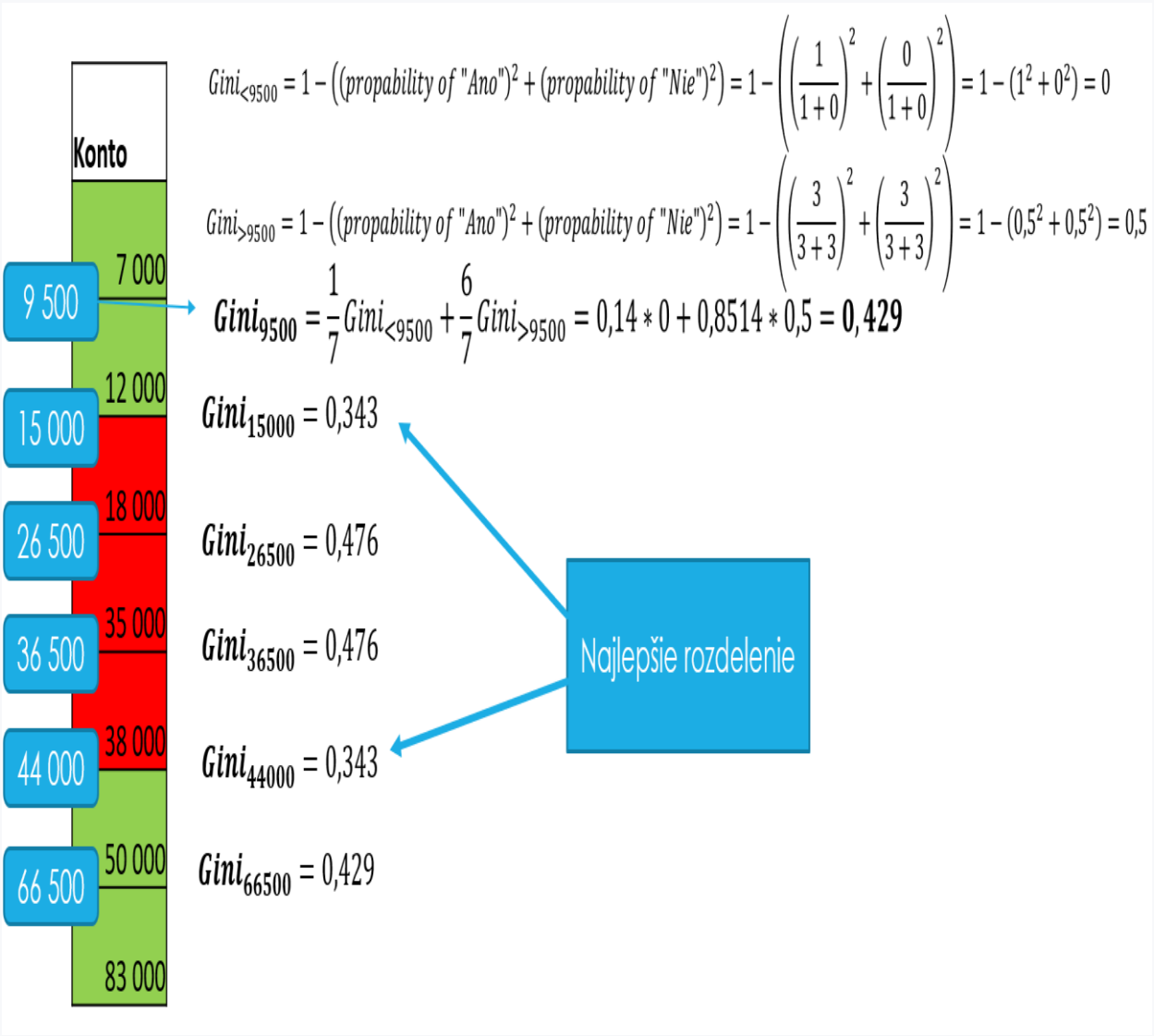
The first step will be to sort the dataset according to the values of the attribute **Account**.

Prijem	Konto	Pohlavie	Neza-mestnany	Uver
Vysoky	7000	Muz	Nie	Ano
Vysoky	12000	Zena	Nie	Ano
Nizky	18000	Zena	Ano	Nie
Vysoky	35000	Muz	Nie	Nie
Nizky	38000	Muz	Nie	Nie
Nizky	50000	Zena	Ano	Ano
Vysoky	83000	Muz	Ano	Ano

In the second step, we calculate the average values of the individual data



Next, we will calculate GINI for each distribution



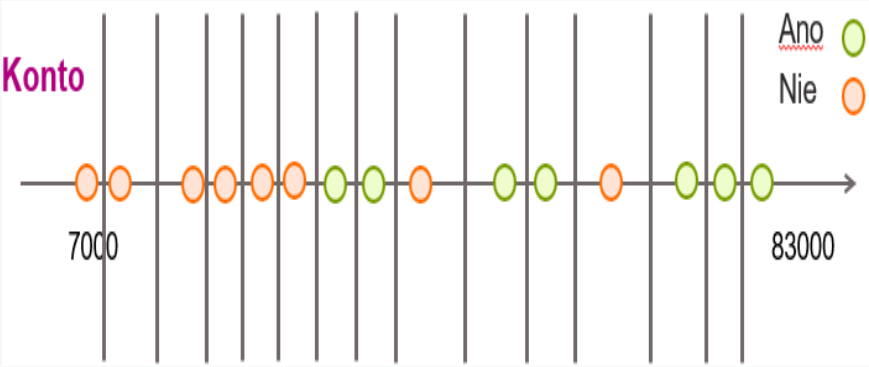
Final Gini index for the column **Account** = 0.343. It is the lower Gini index of all distributions of the dataset according to average values.

📖 4.4.4

We review the process of calculating Gini index for integer values of attributes:

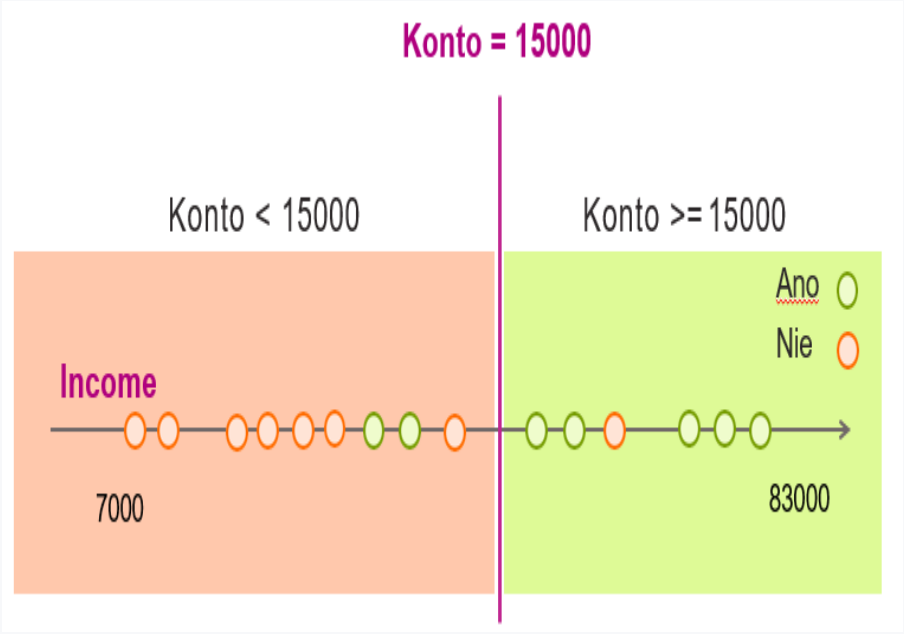
1. Ordering of the values

2. Calculation of average values for each pair of consecutive values



3. Next, only the calculated average values are considered

4. Calculate GINI index for every average value



🕮 4.4.5

At the end of the section on decision trees, we present their strengths and weaknesses.

Strengths:

- Decision trees can generate understandable rules.
- Decision trees achieve classification without the need for too much computing.
- Decision trees can work with both continual and categorical variables.
- Decision trees offer clear indication which areas are the most important for prediction or classification.

Weaknesses:

- Decision trees are prone to classification errors for problems with many classes and relatively few training examples.
- Decision trees can be computationally demanding to train. The process of growing a decision tree is computationally demanding. At each node, each split candidate must be sorted before its best distribution is found. In some algorithms, combinations of fields are used and a search must be made for the optimal combination of weights. The pruning algorithm can also be challenging because many candidate subtrees must be formed and compared.

🕮 4.4.6

References:

- Emily Fox, Carlos Guestrin: Machine Learning Specialization, University of Washington - https://www.coursera.org/specializations/machine-learning
- S. Tahsildar - Gini Index For Decision Trees - https://blog.quantinsti.com/gini-index/
- StatQuest: Decision and Classification Trees, Clearly Explained!! - https://www.youtube.com/watch?v=_L39rN6gz7Y

4.5 Practical tasks

4.5.1

We will create a decision tree that predicts whether or not a person would survive the Titanic just like in the previous sections of this course.

Let's calculate the accuracy of this model.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

data =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv')

data = data[['Survived', 'Pclass', 'Sex', 'Age', 'SibSp',
'Parch']]

data = data.dropna()

data['Sex'] = data['Sex'].replace({'male': 0, 'female': 1})

X = data[data.columns.difference(['Survived'])]
y = data['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

clf = DecisionTreeClassifier(random_state=42)
clf = clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

Program output:

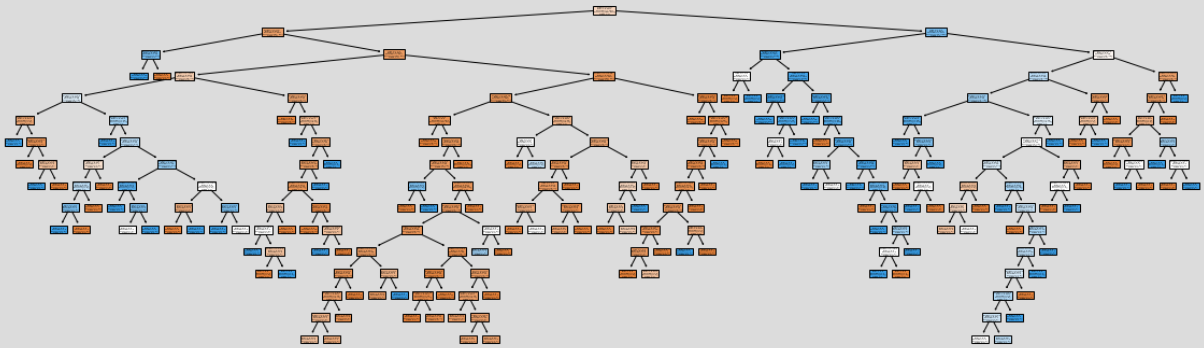
```
0.7767441860465116
```

We found out that 77.67 % of cases are classified correctly.

With the following code, we can visualize the tree.

```
import matplotlib.pyplot as plt
from sklearn import tree
fig = plt.figure(figsize=(20,6))
_ = tree.plot_tree(clf,
                   feature_names = ['Pclass', 'Sex', 'Age',
'SibSp', 'Parch'],
                   class_names=['0','1'],
                   filled=True)
```

Program output:



We can see that the tree is complex. Let's change the default criterion for building the tree from *gini* to *entropy* and show the accuracy of the model.

```
dtree = DecisionTreeClassifier(criterion='gini',
random_state=42)
dtree.fit(X_train, y_train)
pred = dtree.predict(X_test)
print('Criterion=gini', accuracy_score(y_test, pred))

dtree = DecisionTreeClassifier(criterion='entropy',
random_state=42)
dtree.fit(X_train, y_train)
pred = dtree.predict(X_test)
print('Criterion=entropy', accuracy_score(y_test, pred))
```

Program output:

```
Criterion=gini 0.7767441860465116
Criterion=entropy 0.7767441860465116
```

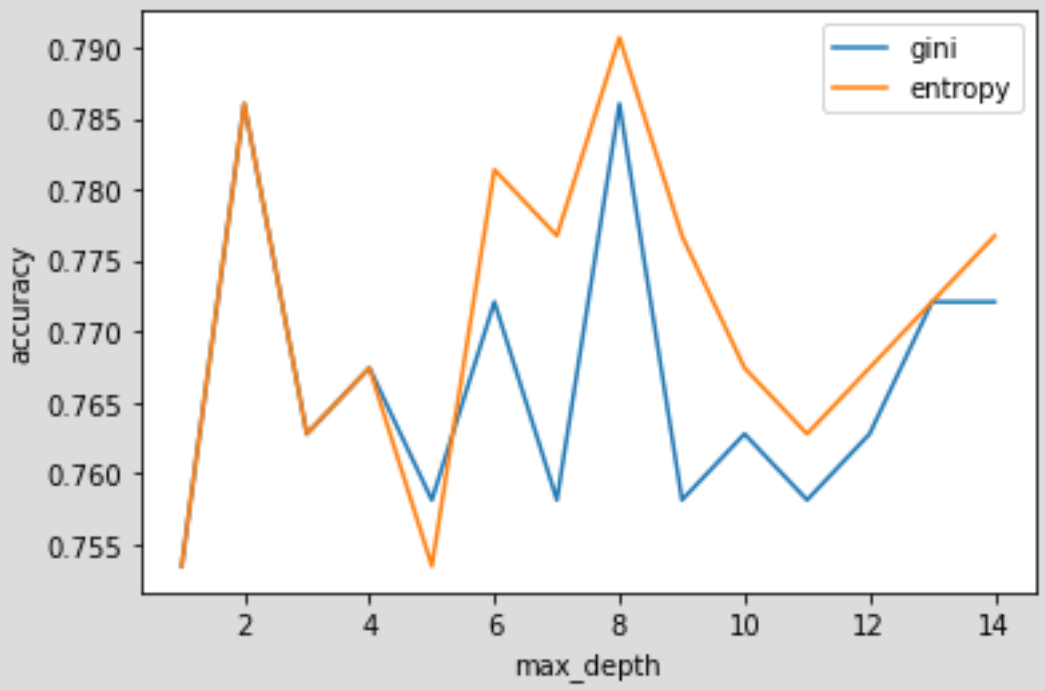
With gini (the default criterion for generating the tree) the accuracy is 77.67 %, and with entropy the accuracy is also 77.67 %.

Let's see whether pruning the tree by changing the maximum depth of the tree gives us better results in any scenario.

Let's create trees sequentially from depth 1 to 15 and visualize the results.

```
max_depth = []
acc_gini = []
acc_entropy = []
for i in range(1,15):
  dtree = DecisionTreeClassifier(criterion='gini',
max_depth=i, random_state = 42)
  dtree.fit(X_train, y_train)
  pred = dtree.predict(X_test)
  acc_gini.append(accuracy_score(y_test, pred))
 ####
  dtree = DecisionTreeClassifier(criterion='entropy',
max_depth=i, random_state = 42)
  dtree.fit(X_train, y_train)
  pred = dtree.predict(X_test)
  acc_entropy.append(accuracy_score(y_test, pred))
 ####
  max_depth.append(i)
d = pd.DataFrame({'acc_gini':pd.Series(acc_gini),
  'acc_entropy':pd.Series(acc_entropy),
  'max_depth':pd.Series(max_depth)})
# visualizing changes in parameters
plt.plot('max_depth','acc_gini', data=d, label='gini')
plt.plot('max_depth','acc_entropy', data=d, label='entropy')
plt.xlabel('max_depth')
plt.ylabel('accuracy')
plt.legend()
```

Program output:



On this graph we can see that by choosing entropy and tree depth of 7, we get the best accuracy of the model. Let's calculate the accuracy of the mentioned model.

```
clf = DecisionTreeClassifier(criterion = 'entropy', max_depth
= 8, random_state=42)
clf = clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

Program output:

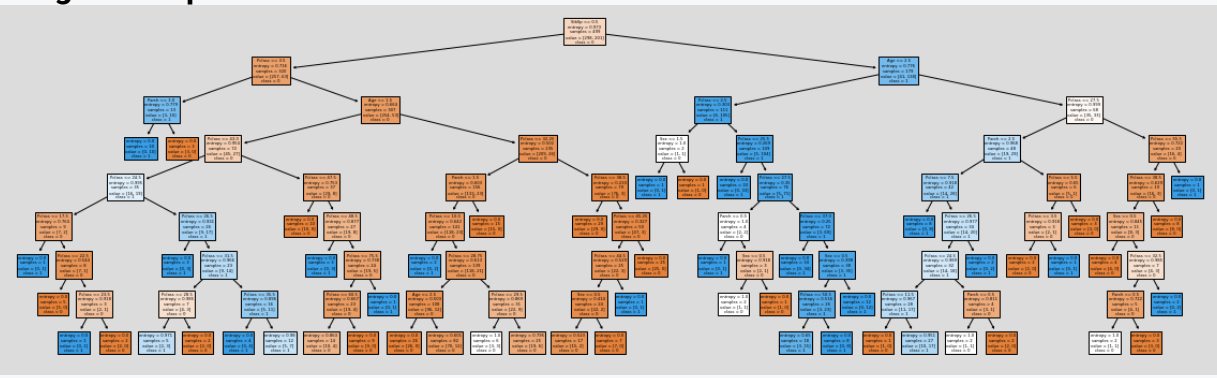
```
0.7906976744186046
```

The accuracy of a decision tree with a depth of 8 and built by entropy has reached 79.07%. We increased the accuracy of the model by 1.4 % just by setting the function to create tree branches and limiting the depth of the tree. The built tree is less complex.

```
import matplotlib.pyplot as plt
from sklearn import tree
fig = plt.figure(figsize=(20,6))
```

```
_ = tree.plot_tree(clf,
                   feature_names = ['Pclass', 'Sex', 'Age',
'SibSp', 'Parch'],
                   class_names=['0','1'],
                   filled=True)
```

Program output:



4.5.2

Number of leaves after pruning

Find the number of leaves before and after pruning the tree and print it. Use the creation of a decision tree:

```
DecisionTreeClassifier(criterion = 'entropy', max_depth = 8,
random_state=42)
```

Print 2 values divided with a space

Create the trees with parameter of random_state=42

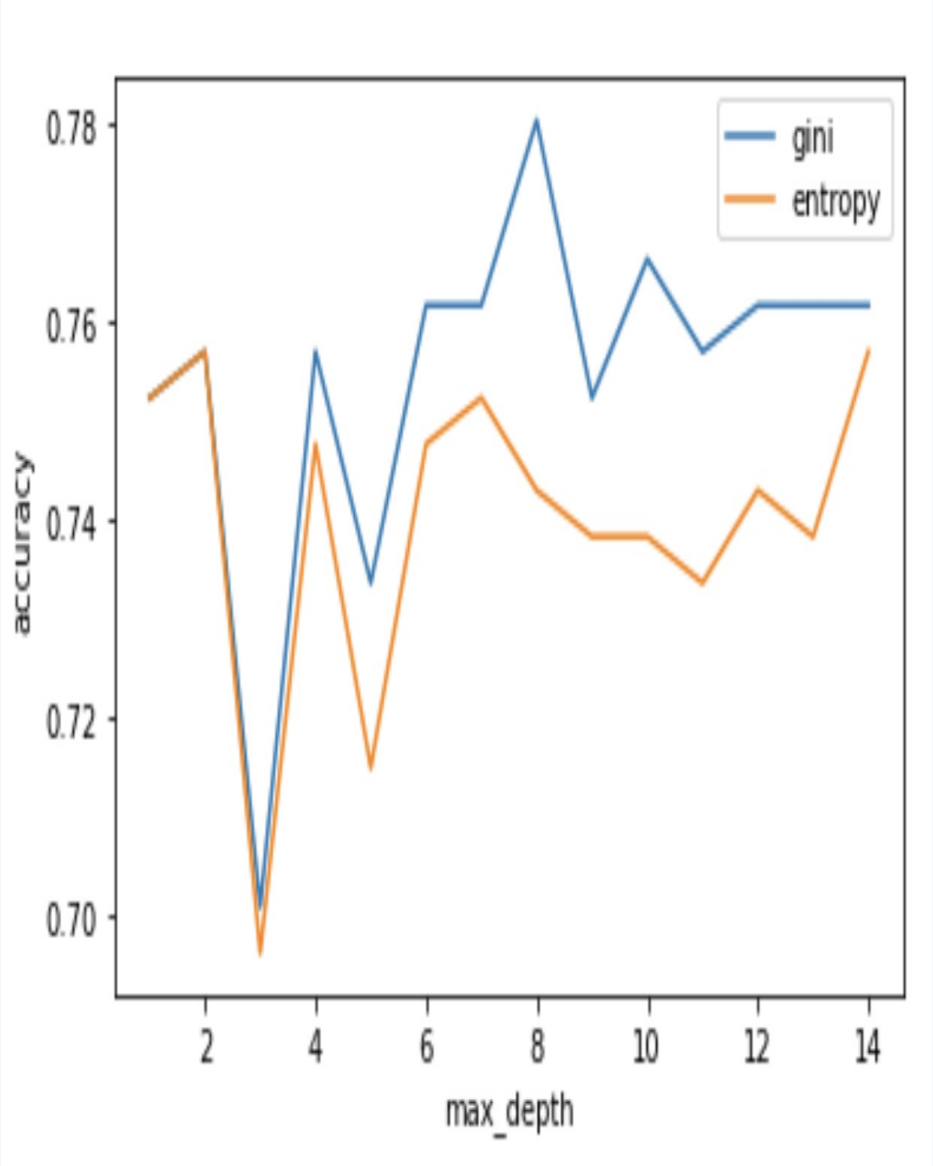
file1.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

```
data =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv')

data = data[['Survived', 'Pclass', 'Sex', 'Age', 'SibSp',
'Parch']]

data = data.dropna()

data['Sex'] = data['Sex'].replace({'male': 0, 'female': 1})

X = data[data.columns.difference(['Survived'])]
y = data['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

4.5.3

What setting of the decision tree would you use based on the following graph?




```
clf = DecisionTreeClassifier(criterion='gini', max_depth=10,
random_state = 42)
clf = DecisionTreeClassifier(criterion='gini', max_depth=8,
random_state = 42)
clf = DecisionTreeClassifier(criterion='entropy',
max_depth=10, random_state = 42)
clf = DecisionTreeClassifier(criterion='entropy', max_depth=8,
random_state = 42)
```

4.5.4

Regression decision tree

We will create a decision tree that predicts whether or not a person would survive the Titanic just like in the previous sections of this course.

Let's calculate the accuracy of this model.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

data =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv')

data = data[['Survived', 'Pclass', 'Sex', 'Age', 'SibSp',
'Parch']]

data = data.dropna()

data['Sex'] = data['Sex'].replace({'male': 0, 'female': 1})

X = data[data.columns.difference(['Survived'])]
y = data['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

clf = DecisionTreeClassifier(random_state=42)
clf = clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

Program output:

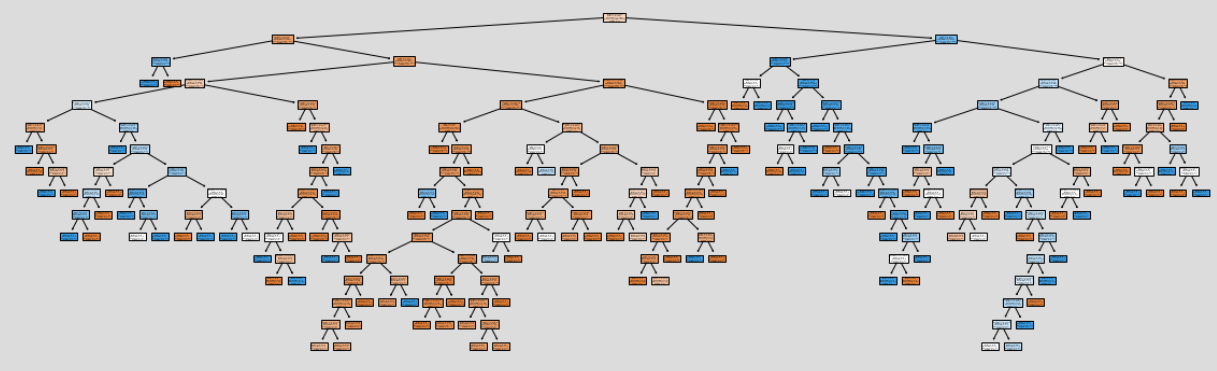
```
0.7767441860465116
```

We found out that 77.67 % of cases are classified correctly.

With the following code, we can visualize the tree.

```
import matplotlib.pyplot as plt
from sklearn import tree
fig = plt.figure(figsize=(20,6))
_ = tree.plot_tree(clf,
                   feature_names = ['Pclass', 'Sex', 'Age',
'SibSp', 'Parch'],
                   class_names=['0','1'],
                   filled=True)
```

Program output:



We can see that the tree is complex. Let's change the default criterion for building the tree from *gini* to *entropy* and show the accuracy of the model.

```
dtree = DecisionTreeClassifier(criterion='gini',
random_state=42)
dtree.fit(X_train, y_train)
pred = dtree.predict(X_test)
print('Criterion=gini', accuracy_score(y_test, pred))

dtree = DecisionTreeClassifier(criterion='entropy',
random_state=42)
dtree.fit(X_train, y_train)
pred = dtree.predict(X_test)
print('Criterion=entropy', accuracy_score(y_test, pred))
```

Program output:

```
Criterion=gini 0.7767441860465116
Criterion=entropy 0.7767441860465116
```

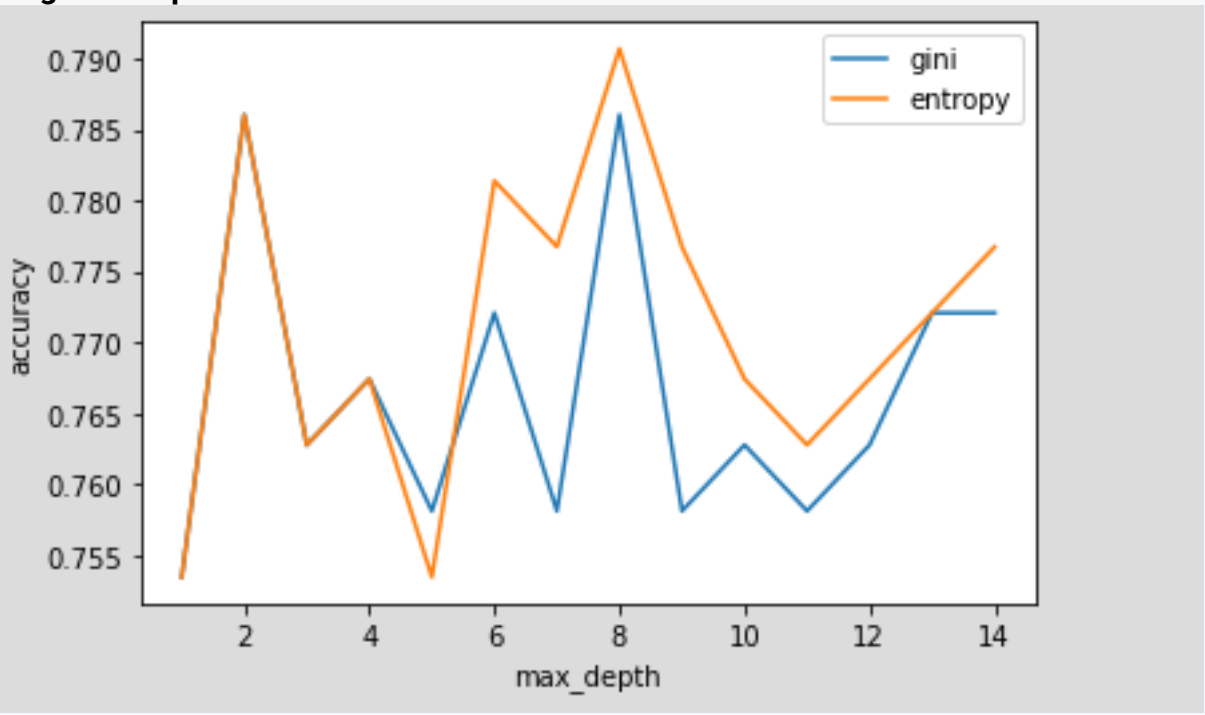
With gini (the default criterion for generating the tree) the accuracy is 77.67 %, and with entropy the accuracy is also 77.67 %.

Let's see whether pruning the tree by changing the maximum depth of the tree gives us better results in any scenario.

Let's create trees sequentially from depth 1 to 15 and visualize the results.

```
max_depth = []
acc_gini = []
acc_entropy = []
for i in range(1,15):
  dtree = DecisionTreeClassifier(criterion='gini',
max_depth=i, random_state = 42)
  dtree.fit(X_train, y_train)
  pred = dtree.predict(X_test)
  acc_gini.append(accuracy_score(y_test, pred))
 ####
  dtree = DecisionTreeClassifier(criterion='entropy',
max_depth=i, random_state = 42)
  dtree.fit(X_train, y_train)
  pred = dtree.predict(X_test)
  acc_entropy.append(accuracy_score(y_test, pred))
 ####
  max_depth.append(i)
d = pd.DataFrame({'acc_gini':pd.Series(acc_gini),
  'acc_entropy':pd.Series(acc_entropy),
  'max_depth':pd.Series(max_depth)})
# visualizing changes in parameters
plt.plot('max_depth','acc_gini', data=d, label='gini')
plt.plot('max_depth','acc_entropy', data=d, label='entropy')
plt.xlabel('max_depth')
plt.ylabel('accuracy')
plt.legend()
```

Program output:



On this graph we can see that by choosing entropy and tree depth of 7, we get the best accuracy of the model. Let's calculate the accuracy of the mentioned model.

```
clf = DecisionTreeClassifier(criterion = 'entropy', max_depth
= 8, random_state=42)
clf = clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

Program output:

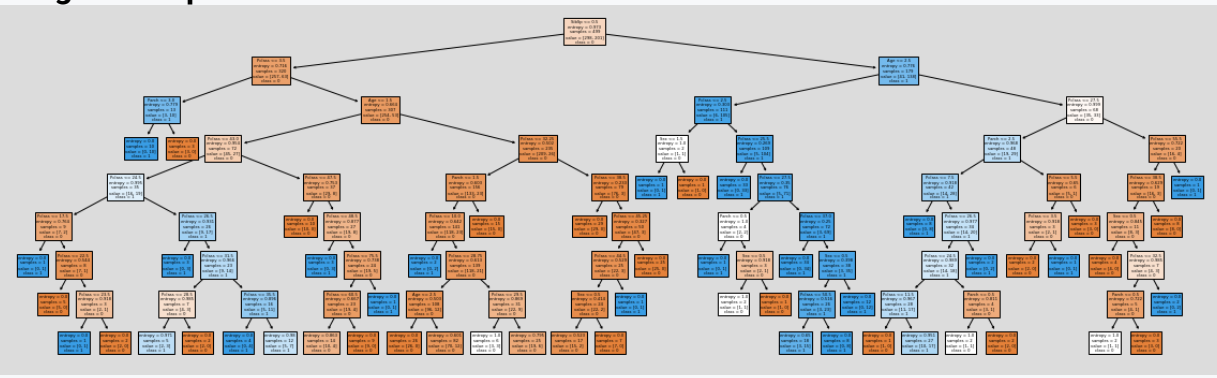
```
0.7906976744186046
```

The accuracy of a decision tree with a depth of 8 and built by entropy has reached 79.07%. We increased the accuracy of the model by 1.4 % just by setting the function to create tree branches and limiting the depth of the tree. The built tree is less complex.

```
import matplotlib.pyplot as plt
from sklearn import tree
fig = plt.figure(figsize=(20,6))
```

```
_ = tree.plot_tree(clf,
                   feature_names = ['Pclass', 'Sex', 'Age',
'SibSp', 'Parch'],
                   class_names=['0','1'],
                   filled=True)
```

Program output:



A regression decision tree predicts a numeral value.

In the following example, we will predict the age of an opossum based on the features (characteristics) of the animal.

Import the required libraries and load the data file from:
https://priscilla.fitped.eu/data/machine_learning/possum.csv

Data file from: *Lindenmayer, D. B., Viggers, K. L., Cunningham, R. B., and Donnelly, C. F. 1995. Morphological variation among columns of the mountain brushtail possum, Trichosurus caninus Ogilby (Phalangeridae: Marsupialia). Australian Journal of Zoology 43: 449-458."*

```
import pandas as pd
import matplotlib.pyplot as plt

df =
pd.read_csv('https://priscilla.fitped.eu/data/machine_learning
/possum.csv')
print(df)
print(df.info())
```

Program output:

```
     case  site     Pop sex  age  hdlngth  skullw  totlngth
taill  footlgth  \
```

```
0       1     1    Vic   m  8.0     94.1    60.4      89.0
36.0      74.5
1       2     1    Vic   f  6.0     92.5    57.6      91.5
36.5      72.5
2       3     1    Vic   f  6.0     94.0    60.0      95.5
39.0      75.4
3       4     1    Vic   f  6.0     93.2    57.1      92.0
38.0      76.1
4       5     1    Vic   f  2.0     91.5    56.3      85.5
36.0      71.0
..    ...   ...    ...  ..  ...      ...     ...       ...
...       ...
99    100     7  other   m  1.0     89.5    56.0      81.5
36.5      66.0
100   101     7  other   m  1.0     88.6    54.7      82.5
39.0      64.4
101   102     7  other   f  6.0     92.4    55.0      89.0
38.0      63.5
102   103     7  other   m  4.0     91.5    55.2      82.5
36.5      62.9
103   104     7  other   f  3.0     93.6    59.9      89.0
40.0      67.6

     earconch   eye  chest  belly
0        54.5  15.2   28.0   36.0
1        51.2  16.0   28.5   33.0
2        51.9  15.5   30.0   34.0
3        52.2  15.2   28.0   34.0
4        53.2  15.1   28.5   33.0
..        ...   ...    ...    ...
99       46.8  14.8   23.0   27.0
100      48.0  14.0   25.0   33.0
101      45.4  13.0   25.0   30.0
102      45.9  15.4   25.0   29.0
103      46.0  14.8   28.5   33.5

[104 rows x 14 columns]

RangeIndex: 104 entries, 0 to 103
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   case      104 non-null    int64
 1   site      104 non-null    int64
```

```
 2   Pop        104 non-null    object
 3   sex        104 non-null    object
 4   age        102 non-null    float64
 5   hdlngth    104 non-null    float64
 6   skullw     104 non-null    float64
 7   totlngth   104 non-null    float64
 8   taill      104 non-null    float64
 9   footlgth   103 non-null    float64
 10  earconch   104 non-null    float64
 11  eye        104 non-null    float64
 12  chest      104 non-null    float64
 13  belly      104 non-null    float64
dtypes: float64(10), int64(2), object(2)
memory usage: 11.5+ KB
None
```

The data file contains records about 104 opossums. The records contain the age of the opossum, sex, length of the head, legs, etc.

For age feature, 2 data are missing and for footlgth feature one data is missing, we will remove these records.

```
df = df.dropna()
print(df.info())
```

Program output:

```
Int64Index: 101 entries, 0 to 103
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   case       101 non-null    int64
 1   site       101 non-null    int64
 2   Pop        101 non-null    object
 3   sex        101 non-null    object
 4   age        101 non-null    float64
 5   hdlngth    101 non-null    float64
 6   skullw     101 non-null    float64
 7   totlngth   101 non-null    float64
 8   taill      101 non-null    float64
 9   footlgth   101 non-null    float64
 10  earconch   101 non-null    float64
 11  eye        101 non-null    float64
```

```
 12  chest     101 non-null    float64
 13  belly     101 non-null    float64
dtypes: float64(10), int64(2), object(2)
memory usage: 11.8+ KB
None
```

We have 101 records left to work with. We will prepare our features and target value.

Features are all numeral characteristics of the animal; the target value is the age of the opossum.

```
X = df.drop(["case", "site", "Pop", "sex", "age"], axis=1)
y = df["age"]

print(X)
print(y)
```

Program output:

```
     hdlngth  skullw  totlngth  taill  footlgth  earconch
eye  chest  belly
0       94.1    60.4      89.0   36.0      74.5      54.5
15.2   28.0   36.0
1       92.5    57.6      91.5   36.5      72.5      51.2
16.0   28.5   33.0
2       94.0    60.0      95.5   39.0      75.4      51.9
15.5   30.0   34.0
3       93.2    57.1      92.0   38.0      76.1      52.2
15.2   28.0   34.0
4       91.5    56.3      85.5   36.0      71.0      53.2
15.1   28.5   33.0
..       ...     ...       ...    ...       ...       ...
...    ...    ...
99      89.5    56.0      81.5   36.5      66.0      46.8
14.8   23.0   27.0
100     88.6    54.7      82.5   39.0      64.4      48.0
14.0   25.0   33.0
101     92.4    55.0      89.0   38.0      63.5      45.4
13.0   25.0   30.0
102     91.5    55.2      82.5   36.5      62.9      45.9
15.4   25.0   29.0
```

```
103     93.6    59.9      89.0   40.0      67.6      46.0
14.8   28.5   33.5

[101 rows x 9 columns]
0      8.0
1      6.0
2      6.0
3      6.0
4      2.0
      ...
99     1.0
100    1.0
101    6.0
102    4.0
103    3.0
Name: age, Length: 101, dtype: float64
```

We will divide the data into training and testing, similar to the standard decision tree.

We will use an 80:20 distribution

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

All that remains is to build a regression decision tree model

```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print('realny vek')
print(y_test.to_numpy())
print('-----')
print('predikovany vek')
print(y_pred)
```

Program output:

```
realny vek
```

```
[2. 2. 7. 6. 4. 3. 4. 5. 9. 8. 5. 3. 1. 2. 3. 2. 3. 4. 5. 4.
1.]
-----
predikovany vek
[3. 3. 6. 6. 4. 2. 7. 2. 6. 6. 2. 3. 2. 3. 3. 4. 3. 1. 3. 4.
1.]
```

With the regression tree, we calculate the prediction accuracy using *Root mean square error (RMSE).*

The value represents the standard deviation of the residuals (prediction error).

```
from sklearn.metrics import mean_squared_error

rmse = mean_squared_error(y_test, y_pred, squared=False)
print(rmse)
```

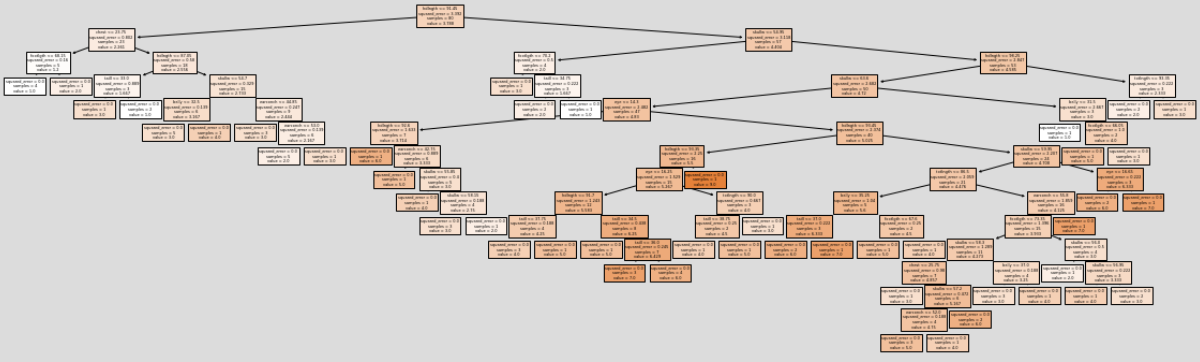
Program output:

```
1.7698260500276368
```

Similar to standard decision trees, we can plot the generated tree.

```
import matplotlib.pyplot as plt
from sklearn import tree
fig = plt.figure(figsize=(20,6))
_ = tree.plot_tree(model,
                   feature_names = ['hdlngth', 'skullw',
'totlngth', 'taill', 'footlgth', 'earconch', 'eye', 'chest',
'belly' ],
                   filled=True)
```

Program output:



4.5.5

Choose the correct statement.

- A regression decision tree predicts a numeral value.
- A regression decision tree predicts a categorical value.

4.5.6

Complete the correct code to create a regression decision tree model with a generation depth of maximum 4.

```
from sklearn.tree import _____

model = _____(random_state=42, _____=4)
_____._____(X_train, y_train)
```

Ensemble Learning Methods - Random Forest

Chapter 5

5.1 Random Forest

📖 5.1.1

Decision trees are easy to build, easy to use and easy to interpret. Despite their many advantages, they are not very successful in practice.

In practice, a combination of a large number of decision trees or other classification methods is quite successful. These methods are referred to as ensemble machine learning methods.

Why the combination of several methods is successful can be easily illustrated by the following example.

Assume a game where we roll a die:

- If number 1 or 2 is rolled, our opponent wins,

- if 3, 4, 5, or 6 is rolled, we win.

It is obvious that the chance of our winning is 4:2 or 2:1

Consider the following options:

- Game 1 - let's play 100 times, with a bet per game of 1 EURO

- Game 2 - let's play 10 times with a bet of 10 EUR

- Game 3 - let's play once, the bet is EUR 100.

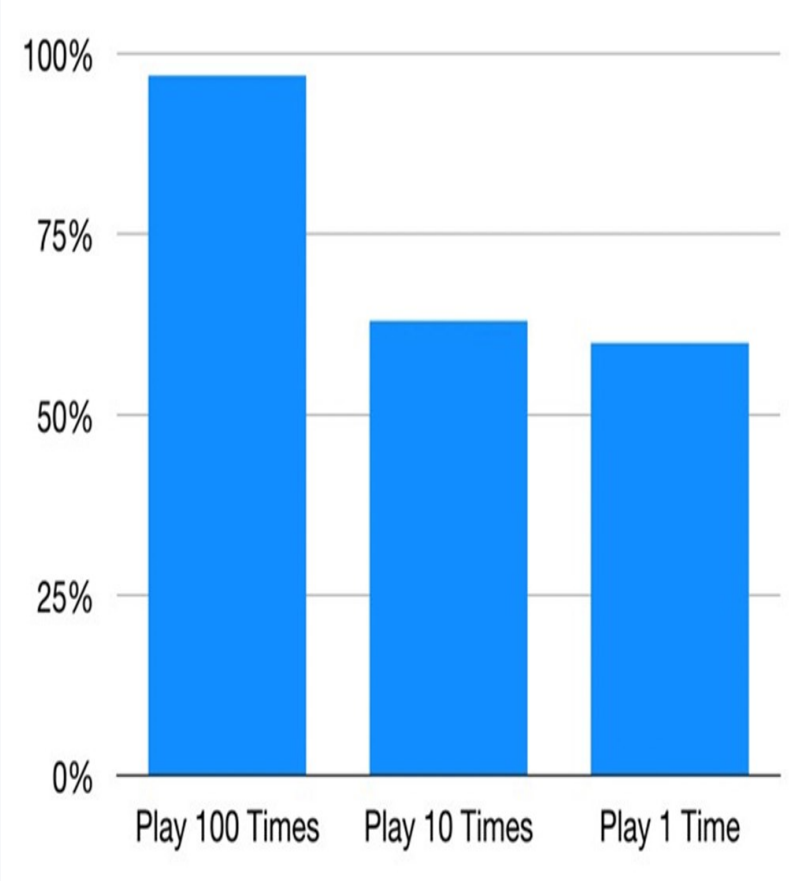
The expected value of the win is the same for all three games

- **Game1** = (P(4/6) * 1)*100 = 0.6666 * 1 * 100 = EUR 66.66
- **Game2** = (P(4/6) * 10)*10 = 0.6666 * 10 * 10 = EUR 66.66
- **Game3** = (P(4/6) * 100)*1 = 0.6666 * 100 * 1 = EUR 66.66

The difference lies in the independent measurements. Although the expected values are the same, the distributions of the results are significantly different!

When are we the most sure that we will definitely earn money?

These are still random rolls of the dice. In the graph we show the observed results (% of our winnings) when simulating the game.



From this simple example, it is clear that a single tree, with a large weight of its classification, can be successful, but also very unsuccessful in classification. We minimize the risk (i.e. we increase the success), if we use multiple trees with small classification weight.

🕮 5.1.2

Approach when we learn and create multiple models and combine them:

· Can lead to higher accuracy

· The variance of the results can be reduced by averaging, if the predicted measurements are mutually independent

· Individual models may be re-learned, but the combination may be resistant to relearning

Ensemble learning methods are also referred to as **ensemble** methods, i.e. connection or harmony of several models. Thus, to make predictions an ensemble of models as an individual model is used.

There are three basic approaches:

- bagging,
- boosting,
- stacking.

📖 5.1.3

Bagging - in this approach we create different subsets of the training set, the final output is based on majority voting.

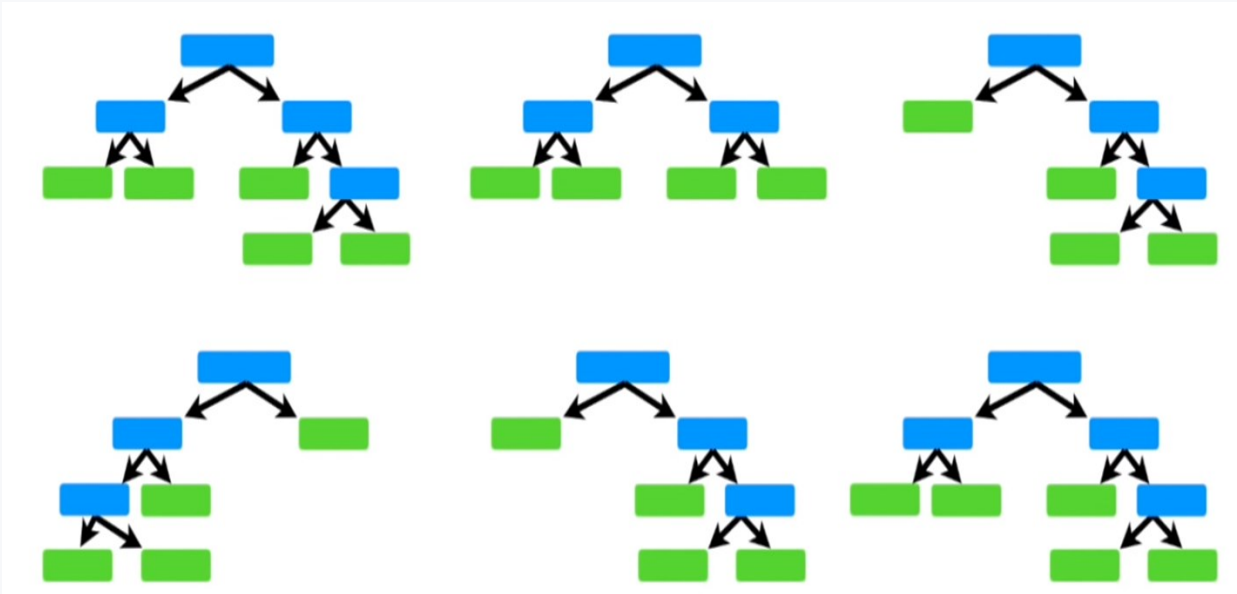
Bagging is also referred to as Bootstrap Aggregation. This ensemble technique is also used by **Random forest**.

In Random forest, a number of decision trees are generated. The question remains how to create different decision tree models when one dataset is used. Bagging selects a random sample from the dataset. Thus, each model is generated from different samples from the original dataset. Subsequently, each model is trained independently.

Thus, in Bagging, a sample of the original dataset is created by randomly duplicating its rows, or by randomly removing some rows. Similarly, randomness is applied not only to examples/rows, but also to features/attributes i.e. columns. This technique leads to reduce the influence of specific data (variance reduction).

🕮 5.1.4

Using Bagging in the Random Forest method, we create a number of decision trees.

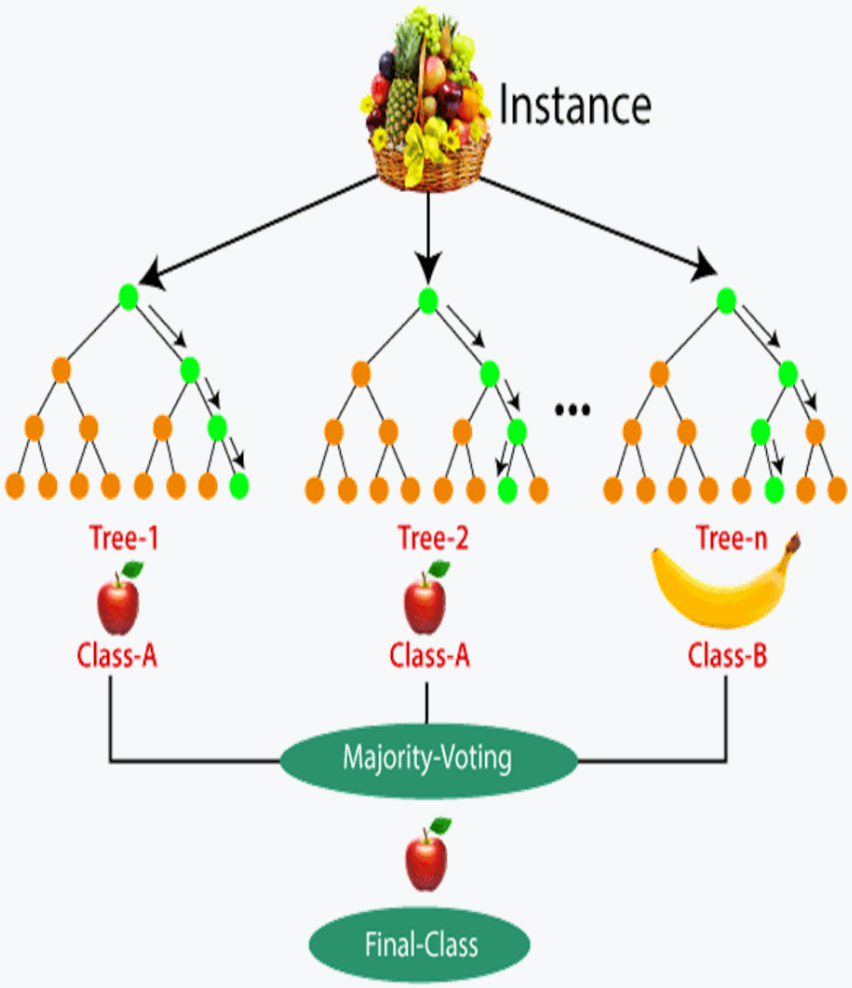


The output of the classification is then based on the majority voting after linking the results of all models. This step, which involves combining all the results and generating an output based on the majority voting, is known as **aggregation**.

🕮 5.1.5

The bootstrap sample is taken from the real training dataset data. There is a high probability that each sample will not contain a unique data.

Each model is obtained from a different bootstrap sample and trained independently. Each model generates results. At the end, a majority voting takes place.



📖 5.1.6

Random Forest Algorithm:

· **Step 1**: In random forest, ***n*** number of random records are selected from a dataset that has a ***k*** number of records.

· **Step 2**: Individual decision trees are built for each sample.

· **Step 3**: Each decision tree generates an output.

· **Step 4**: The final output is produced by majority voting or averaging for regression.

Important features of Random forest include:

· **Diversity**- not all attributes/variables/features are considered when creating an individual tree, each tree is different.

· **Immune to multidimensionality**- since each tree does not take all features into account, the space is reduced.

· **Parallelization** - each tree is created independently from different data and attributes. Thus, we can fully used the CPU to create trees.

· **Split-Train-Test** - in the random forest, in principle, we do not need to separate the data for training and testing, because there will always be data that the tree canot see.

- · **Stability** -occurs because the result is based on majority voting/averaging.

📖 5.1.7

If we compare Random Forest to decision trees so,

Decision Trees:

· have a problem with overfitting,

· build faster,

· takes all dataset examples as an input.

Random Forest:

- · Trees are built from subsets of the data and the final output is based on average or majority ordering, thus no overfitting occurs.
- · They are much slower to build.
- · It randomly selects observations, builds a decision tree, and takes the average result.

🕮 5.1.8

If we use the **scikit-learn** library to create a Random Forest, we can set the following parameters:

- · **n_estimators** - the number of trees that the algorithm creates before averaging the predictions.
- · **max_features**- the maximum number of elements that the Random Forest considers to be a node distribution.
- · **mini_sample_leaf**- specifies the minimum number of leaves needed to distribute the inner node.
- · **n_jobs** - information about the number of processors that can be used. If the value is 1, only one processor can be used, but if the value is -1, there is no limit.
- · **random_state** - checks the randomness of the sample. The model will always produce the same results if it has a certain value of random state
- · **oob_score** - OOB stand for Out Of the Bag. This is a random forest cross-validation method. A portion of the sample is not used to train the data, but it is used to evaluate its performance

🕮 5.1.9

As in all methods, we conclude this section with the advantages and disadvantages of Random Forest.

Advantages:

· It can be used in classification and regression problems.

· It solves the problem of overfitting.

· It works fine even if the data contains null/missing values.

· It exhibits the property of parallelization.

· It is highly stable because it takes the average responses provided by a large number of trees.

· It preserves diversity because not all attributes are considered during the building of each decision tree.

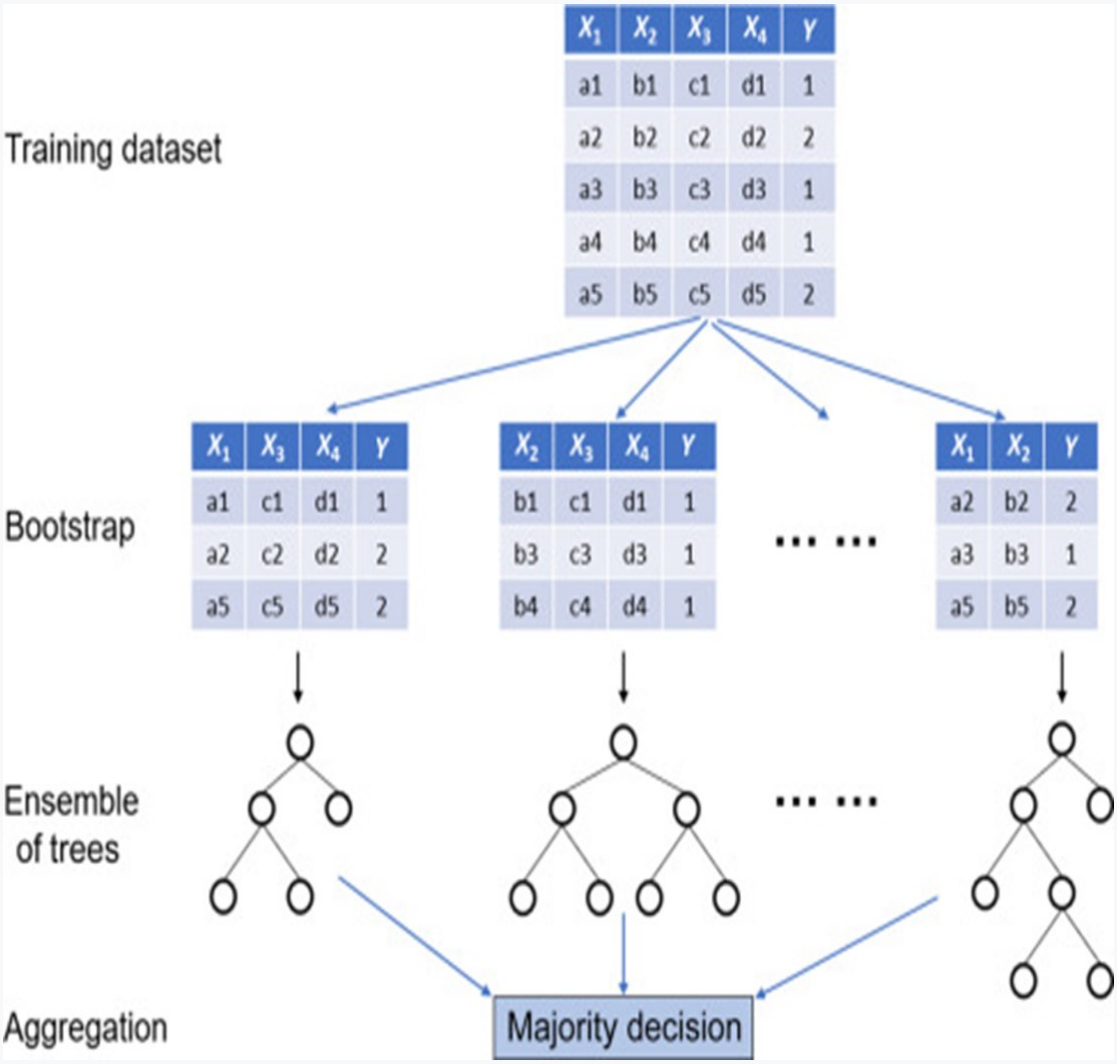
Disadvantages

· It is very complex compared to decision trees where decisions can be made by following the path of the tree.

· Training time is the most demanding compared to other models.

· Whenever it has to make a prediction, each decision tree has to generate an output for the given input data.

· **You cannot see into the forest!**



📝 5.1.10

What are the benefits of Random forest?

- It solves the problem of overfitting
- It works well even if the data contains null/missing values
- It exhibits the features of parallelization
- It combines GINI index and Entropy within a single tree
- It also works at higher degrees of polynomial

5.2 Other ensemble learning methods

🕮 5.2.1

There are three basic techniques in ensemble machine learning methods:

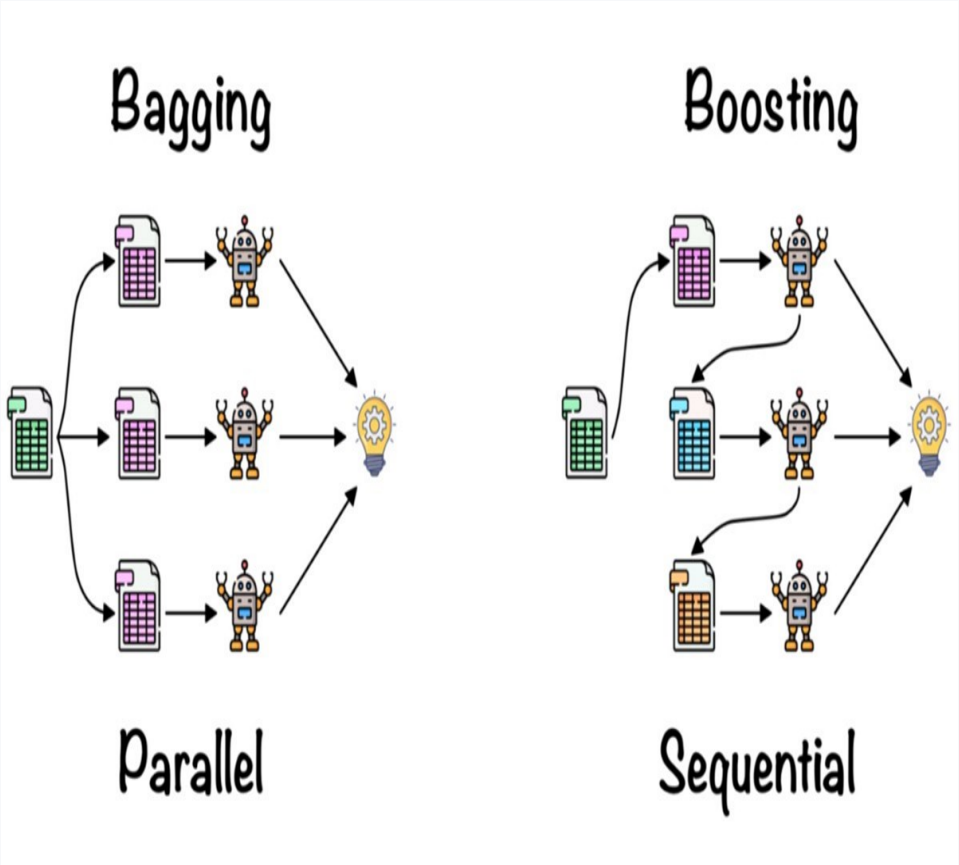
· bagging,

· boosting,

· stacking.

We introduced the bagging technique within Random Forest. The second technique is **boosting**. This technique combines "weak learners" into a strong sequential model for the highest possible accuracy.

A comparison of the this technique with bagging is shown in the picture.



Bagging runs parallel, boosting runs sequential. In Boosting, each input pattern has its own weight. At the beginning all have the same, for misclassified ones the weight has increased. Patterns are combined by weighted voting.

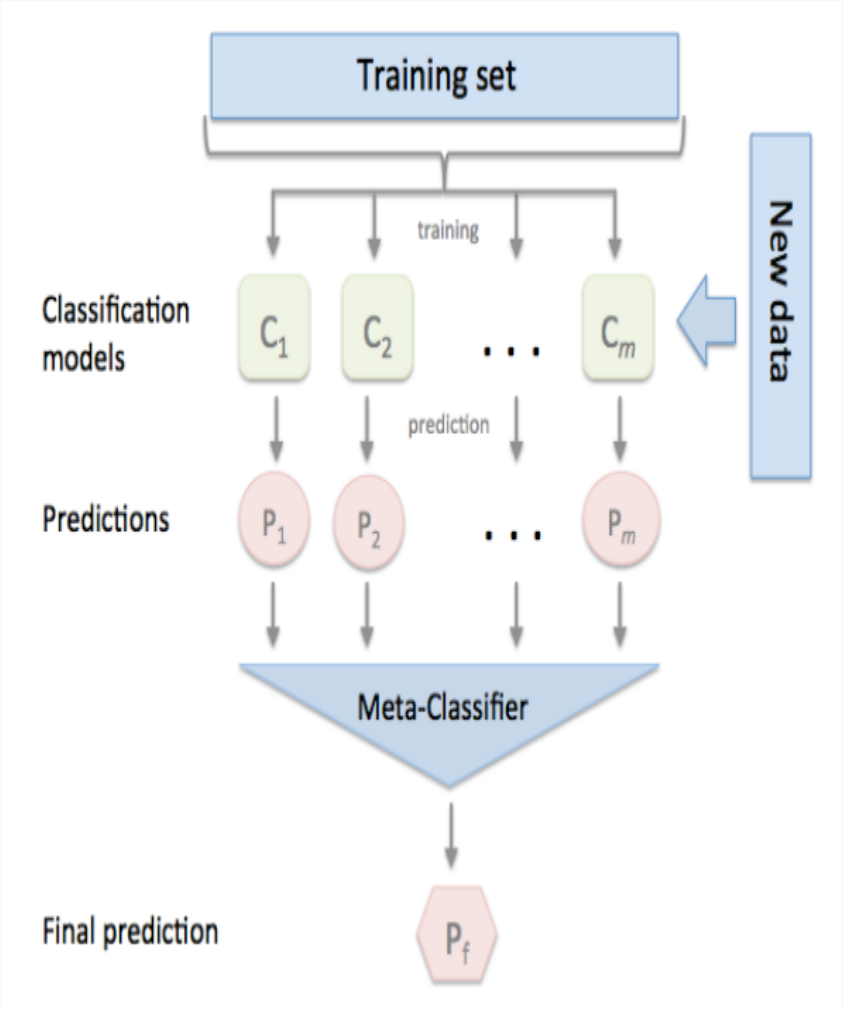
Within the Boosting technique, it is possible to solve the hyperparameter configuration problem.

🕮 5.2.2

The last technique is **Voting**.

In this technique, by combining different methods, we select several basic classifiers. In prediction, we do not use only the average voting or the most frequent voting, but the classifiers voting is the input to the final classifier.

The estimates (reliability) of the classes are used at the next level of the meta-classifier, which determines the final prediction.



🕮 5.2.3

Literature used:

· **Emily Fox, Carlos Guestrin: Machine Learning Specialization, University of Washington** https://www.coursera.org/specializations/machine-learning

· **Sruthi E R: Understanding Random Forest -** https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

· **StatQuest: Random Forests Part 1 - Building, Using and Evaluating - https://www.youtube.com/watch?v=J4Wdy0Wc_xQ**

· **StatQuest: Random Forests Part 2: Missing data and clustering - https://www.youtube.com/watch?v=sQ870aTKqiM**

5.2.4

What is Bagging?

- a technique that creates different subsets of the training set
- a technique that combines "weak learners" into a strong sequential model for the highest possible accuracy
- a technique that compares all the metrics of success of a single decision tree
- a metric of the equilibrium representation of classes in the dataset

5.3 Practical tasks

5.3.1

Building a random forest is a very similar process to building a decision tree. A random forest is actually a model that consists of several individual decision trees. The final predicted value of the random forest is most often the most frequent resulting value from the individual decision trees.

The following example shows a solution to the Titanic survival prediction using a random forest.

We load the dataset and prepare a suitable training and test set as in the previous sections.

```
import pandas as pd
from sklearn.model_selection import train_test_split

data =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv')

data = data[['Survived', 'Pclass', 'Sex', 'Age', 'SibSp',
'Parch']]

data = data.dropna()

data['Sex'] = data['Sex'].replace({'male': 0, 'female': 1})

X = data[data.columns.difference(['Survived'])]
y = data['Survived']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

We will build a Random Forest model using the sklearn library:

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(random_state=42)
```

We will train the created model and make predictions using the test set.

```
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
print(y_pred)
```

Program output:

```
[1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1 1 1 0 0 0 0 1 0 0
0 1 0 0 0 0
 0 0 0 0 0 1 0 0 1 1 0 1 1 1 1 0 0 0 1 1 0 0 0 1 0 1 0 0 0 1 1
0 1 0 1 0 0
 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 1 1 1 1 0 0 1 1 0 0 1
0 0 1 0 1 0
 0 0 1 0 0 0 1 1 0 1 0 0 0 1 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 1
1 0 0 0 0 0
 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 1 0 1 0 0 1 0 1 1 1 0 0 0
0 1 0 1 1 0
 0 1 0 1 0 0 1 1 0 1 0 0 0 1 0 0 1 0 1 0 0 1 1 0 1 0 0 1 0 0]
```

We compare the values with the real y_test values.

```
print(y_test.to_numpy())
```

Program output:

```
[0 1 1 1 0 1 1 1 0 0 1 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0
0 1 0 0 1 0
 1 0 0 1 0 1 0 0 1 1 0 0 1 0 0 1 0 0 0 0 1 0 1 1 0 1 0 1 0 1 1
1 0 0 1 0 0
```

```
 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0 0 0 1 1 0 0 1
0 0 0 1 1 0
 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 0 0 1
1 1 0 0 0 0
 0 0 0 1 0 1 0 0 1 0 1 1 0 0 0 1 1 1 1 1 1 0 0 1 0 1 1 1 0 0 0
0 1 0 1 1 0
 0 1 1 1 0 1 0 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 1 0 0 1 0 0]
```

```
We compute the prediction accuracy in the same way as for
decision trees.
```

```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Program output:

```
Accuracy: 0.786046511627907
```

```
We compare the results with the default decision tree model.
```

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred_DT = clf.predict(X_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred_DT))
```

Program output:

```
Accuracy: 0.7767441860465116
```

The accuracy of the random forest model compared to the decision tree is 0.93% higher.

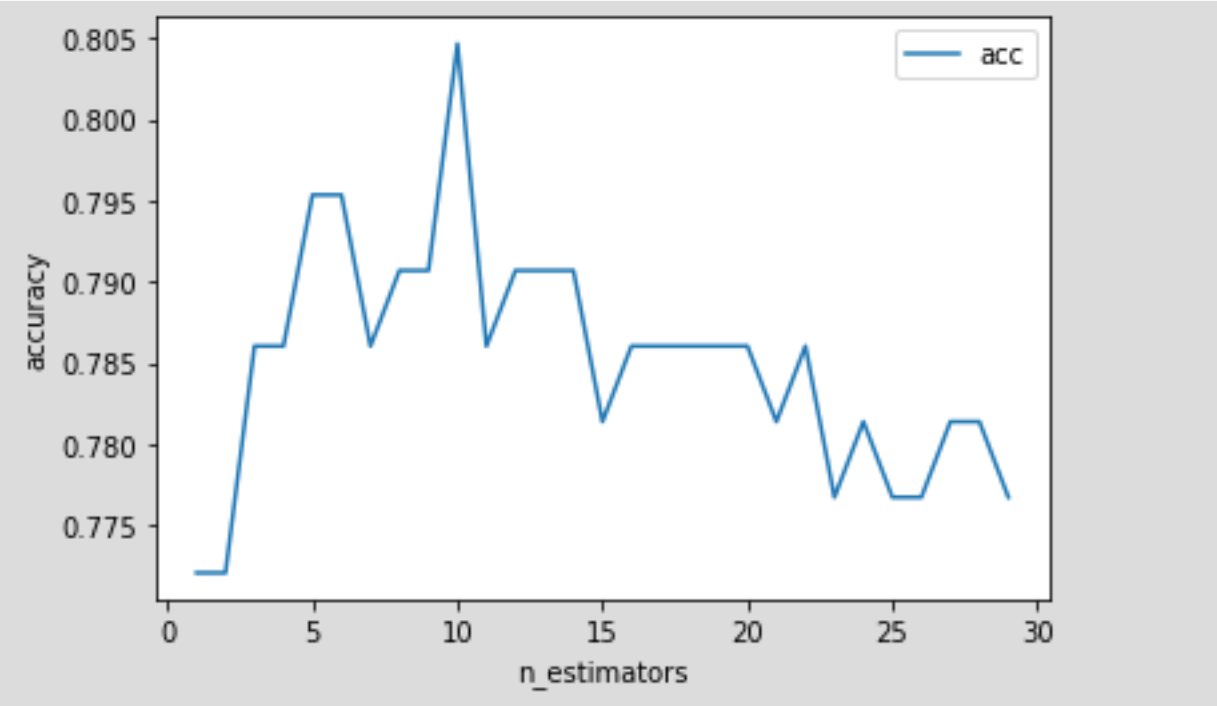
The random forest, like the decision tree, can be fitted with parameters.

The following example shows how the prediction accuracy of a random forest varies based on the number of trees built of.

We create the forest sequentially from 1 to 30 trees and wefind the prediction accuracy.

```
import matplotlib.pyplot as plt
n_estimators = []
acc = []

for i in range(1, 30):
  rf = RandomForestClassifier(n_estimators=i, random_state =
42)
  rf.fit(X_train, y_train)
  pred = rf.predict(X_test)
  acc.append(metrics.accuracy_score(y_test, pred))
 ####
  n_estimators.append(i)


d = pd.DataFrame({'acc':pd.Series(acc),
  'n_estimators':pd.Series(n_estimators)})
# visualizing changes in parameters
plt.plot('n_estimators','acc', data=d, label='acc')

plt.xlabel('n_estimators')
plt.ylabel('accuracy')
plt.legend()
```

Program output:



The forest with 10 trees has the highest accuracy.

```
rf_model = RandomForestClassifier(random_state=42,
n_estimators = 10)
rf_model.fit(X_train, y_train)
pred = rf_model.predict(X_test)
print(metrics.accuracy_score(y_test, pred))
```

Program output:

```
0.8046511627906977
```

A forest with 10 trees has an accuracy of 80.47%, which is 1.87% higher than the accuracy of the random forest model with default settings.

5.3.2

A random forest is a model, which consists of several individual decision trees.

- True
- False

5.3.3

RandomForest with six trees

Complete the code to build a random forest with six trees. Use the parameter random_state=42 to preserve the randomization.

Set the size of the test set to 20%.

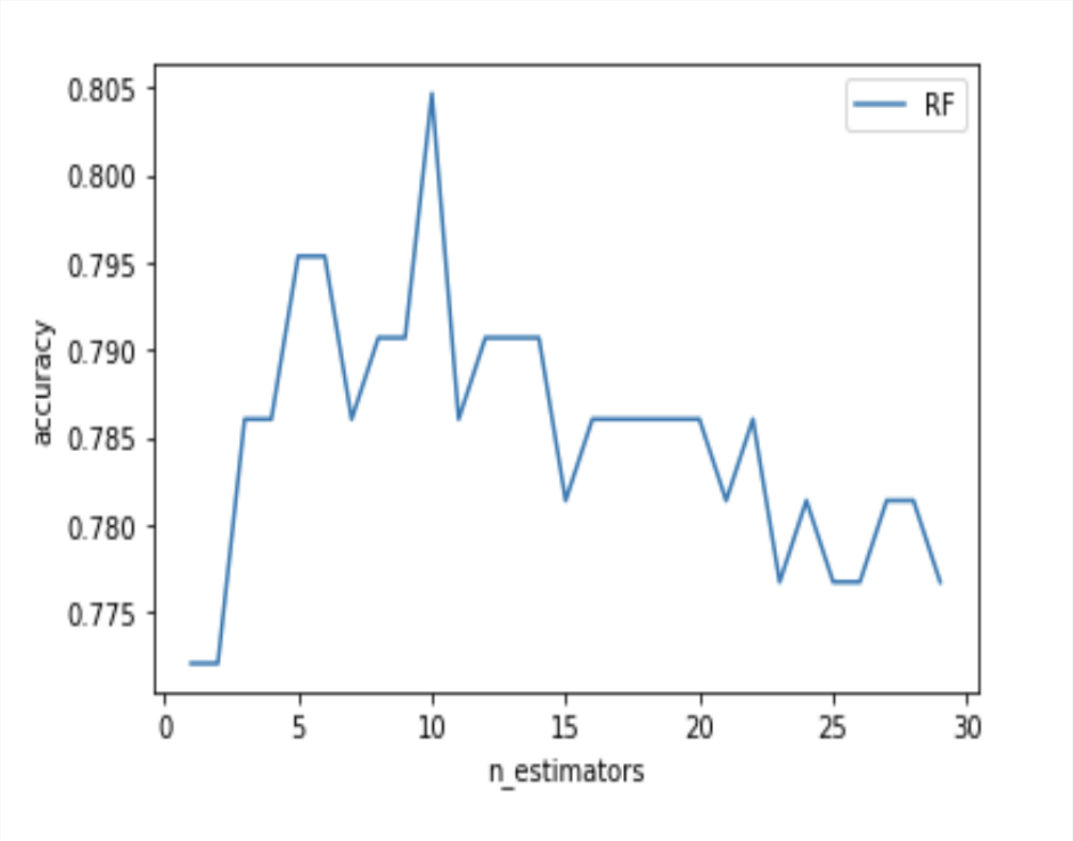
As a result, write the model accuracy.

file1.py

```
import pandas as pd
from sklearn.model_selection import train_test_split

data =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv')

data = data[['Survived', 'Pclass', 'Sex', 'Age', 'SibSp',
'Parch']]

data = data.dropna()

data['Sex'] = data['Sex'].replace({'male': 0, 'female': 1})

X = data[data.columns.difference(['Survived'])]
y = data['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

5.3.4

How many decision trees would you use in a random forest model based on the following graph?


RF
accuracy
n_estimators
0.805
0.800
0.795
0.790
0.785
0.780
0.775
0
5
10
15
20
25
30

PRISCILLA

priscilla.fitped.eu