Machine learning with Python

Johan Decorte

Sabine De Vreese

Contents

1 Artificial Intelligence, Data Mining and Machine Learning

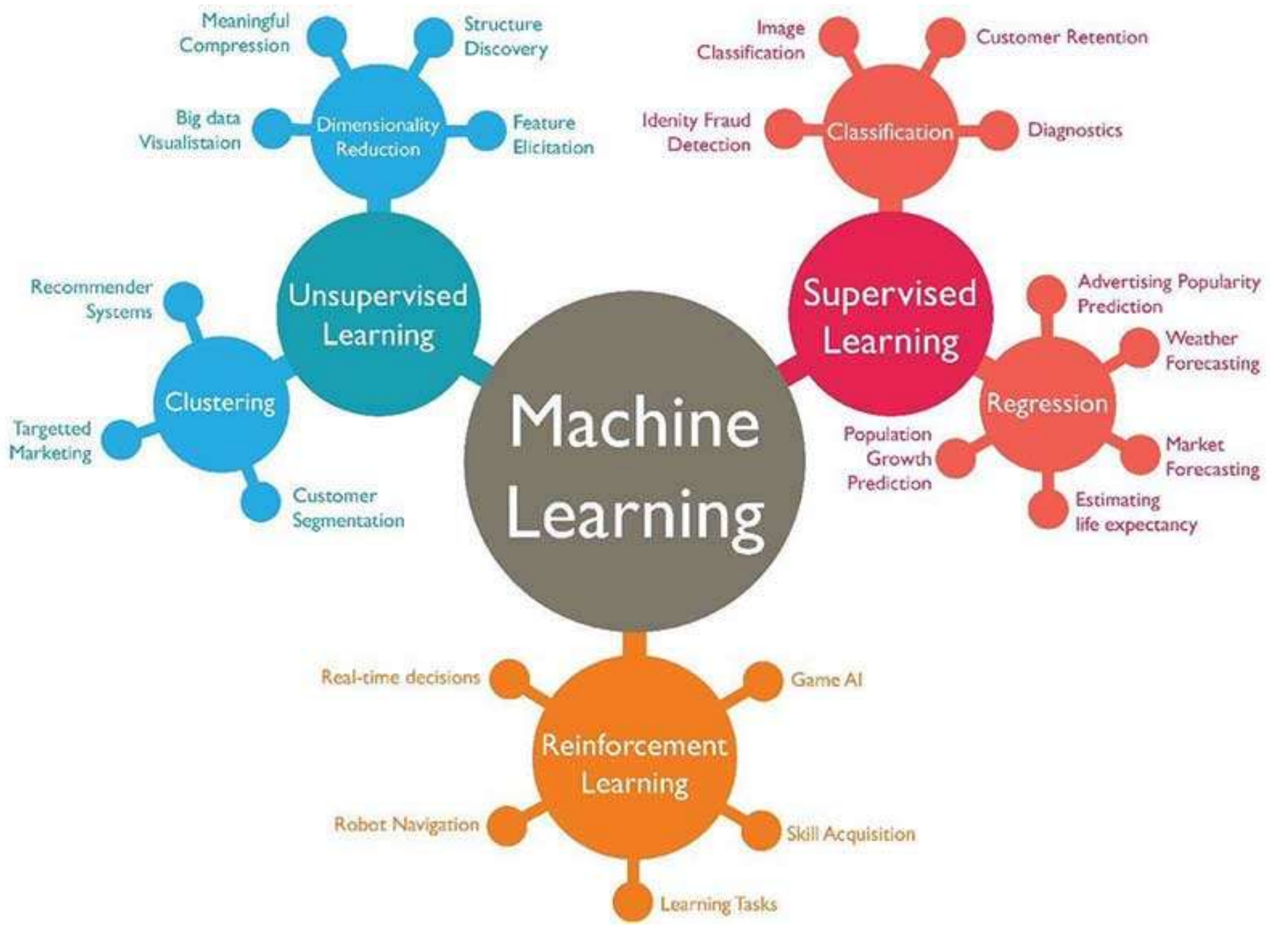
Artificial intelligence, machine learning and data mining are three related techniques, that can be defined as follows:

Artificial intelligence: the automation of activities that we associate with human thinking, activities such as decision-making, problem solving and learning.

Machine learning: the study, design and development of algorithms that give computers the capability to learn without being explicitly programmed.

Data mining: the process of extracting valid, previously unknown, comprehensible, and actionable information from large databases and using it to make crucial business decisions. Often, machine learning techniques are used in data mining.

Machine learning can again be subdivided into three categories, as shown in the figure below. The smallest bullets show some application fields for each subcategory.



In supervised and unsupervised learning we start from a data set of examples (called *instances*), each of which comprises the values of a number of variables, which are often called *attributes* or *features*.

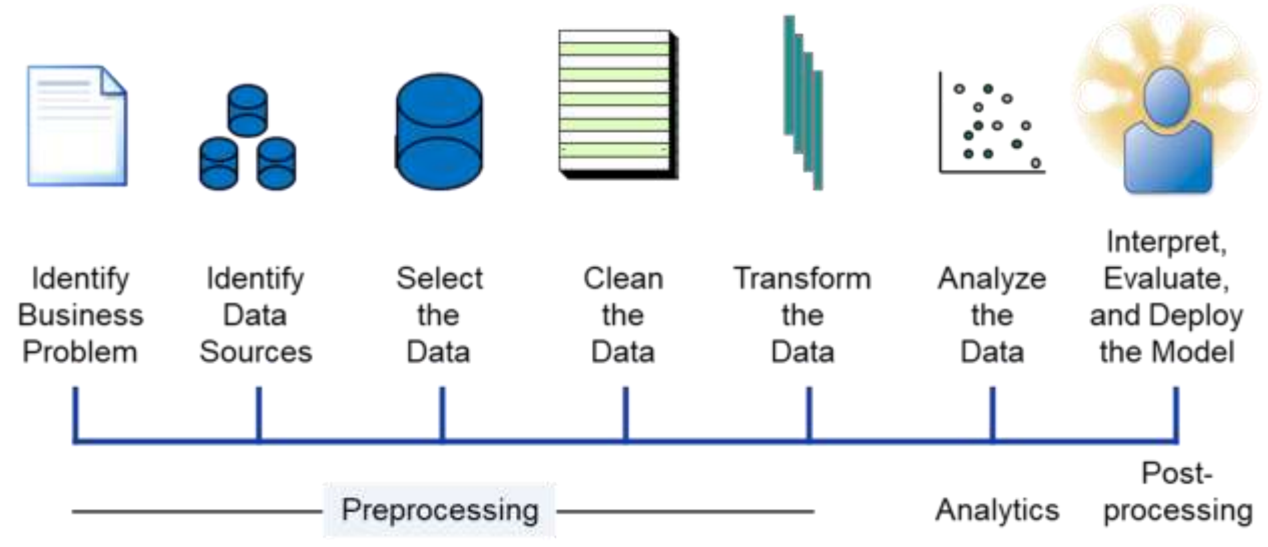
There are two types of data, which are treated in radically different ways. For the first type there is a specifically designated attribute and the aim is to use the data given to predict the value of that attribute for instances that have not yet been seen. Data of this kind is called labelled. Data mining using labelled data is known as ***supervised learning***. If the designated attribute is categorical, i.e. it must take one of a number of distinct values such as 'very good', 'good' or 'poor', or (in an object recognition application) 'car', 'bicycle', 'person', 'bus' or 'taxi' the task is called **classification**. If the designated attribute is numerical, e.g. the expected sale price of a house or the opening price of a share on tomorrow's stock market, the task is called

regression (see figure above). Data that does not have any specifically designated attribute is called unlabeled. Data mining of unlabeled data is known as ***unsupervised learning***.

We do not consider reinforcement learning in this lecture and we further explore only supervised learning classification and regression techniques.

2 The analytics process model

Each data analytics project consists of several steps (Lemahieu, Vanden Broucke, & Baesens, 2018):



It is assumed that ca. 80-90 % of the overall project time is spent of preprocessing because understanding the business and the data and cleaning the data can consume a lot of time. After all, most data analytics algorithms are available off-the-shelf in software libraries or even complete applications.

3 Basics of supervised learning

3.1 Classification

3.1.1 Data exploration

Classification is one of the most common applications for data mining. It corresponds to a task that occurs frequently in everyday life.

For example the table below is a (very small) subset of the passenger list of the Titanic, with some attributes per passenger and whether they survived or not. The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg. A disaster in which 1502 out of 2224 (= 68%) passengers and crew died. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

The meaning of the attributes or features is as follows.

Variable	**Definition**	**Key**
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Gender	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Variable Notes

pclass: A proxy for socio-economic status (SES)

1st = Upper

2nd = Middle

3rd = Lower

age: Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5

sibsp: The dataset defines family relations in this way...

Sibling = brother, sister, stepbrother, stepsister

Spouse = husband, wife (mistresses and fiancés were ignored)

parch: The dataset defines family relations in this way...

Parent = mother, father

Child = daughter, son, stepdaughter, stepson

Some children travelled only with a nanny, therefore parch=0 for them.

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25		S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2.	7.925		S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S
6	0	3	Moran, Mr. James	male		0	0	330877	8.4583		Q
7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	male	2	3	1	349909	21.075		S
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27	0	2	347742	11.1333		S
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14	1	0	237736	30.0708		C

Categorical and continuous attributes

Many practical data mining systems divide attributes into just two types:

– Categorical: corresponding to values that fall into a distinct number of classes, like male/female or ticket class.
– Continuous: corresponding to integer or double values, like age.

3.2 Some classification algorithms

There are numerous classification algorithms in use. We consider three of the most commonly used.

3.2.1 Naïve Bayes Classification

This algorithm uses the branch of Mathematics known as probability theory to find the most likely of the possible classifications.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

To understand the naive Bayes classifier we need to understand the Bayes theorem. So let's first discuss the Bayes Theorem.

Bayes theorem is named after Rev. Thomas Bayes. It works on conditional probability. **Conditional probability is the probability that something will happen, given that something else has already occurred.** Using the conditional probability, we can calculate the probability of an event using its prior knowledge.

Below is the formula for calculating the conditional probability.

$$P(H|E) = \frac{P(E|H)*P(H)}{P(E)}$$

where

P(H) is the probability of hypothesis H being true. This is known as the prior probability.

P(E) is the probability of the evidence (regardless of the hypothesis).

P(E|H) is the probability of the evidence given that hypothesis is true.

P(H|E) is the probability of the hypothesis given that the evidence is there.

Naive Bayes is a kind of classifier which uses the Bayes Theorem. It predicts membership probabilities for each class such as the probability that a given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class. This is also known as Maximum A Posteriori (MAP).

Naive Bayes classifier assumes that all the features are unrelated to each other. Presence or absence of a feature does not influence the presence or absence of any other feature.

Fruit Example

Let's say that we have data on 1000 pieces of fruit. They happen to be Banana, Orange or some Other Fruit. We know 3 characteristics about each fruit:

- Whether it is Long
- Whether it is Sweet and
- If its color is Yellow.

This is our 'training set', which exists of a table of 1000 lines (one per piece of fruit) and 3 yes/no input columns and 1 output column with the value banana, orange or "other fruit". The statistics of this table can be summarized as follows.

```
Type           Long | Not Long || Sweet | Not Sweet || Yellow |Not Yellow|Total
             ___________________________________________________________________
Banana      |  400  |     100  || 350   |     150   ||  450   |  50      |  500
Orange      |    0  |     300  || 150   |     150   ||  300   |   0      |  300
Other Fruit |  100  |     100  || 150   |      50   ||   50   | 150      |  200
            ____________________________________________________________________
Total       |  500  |     500  || 650   |     350   ||  800   | 200      | 1000
             ___________________________________________________________________
```

We will use this table to predict the type of any new fruit we encounter.

We can pre-compute a lot of things about our fruit collection.

The so-called "Prior" probabilities. (If we didn't know any of the fruit attributes, this would be our guess.) These are our **base rates**.

```
 P(Banana)      = 0.5 (500/1000)
 P(Orange)      = 0.3
 P(Other Fruit) = 0.2
```

Probability of "Evidence"

```
p(Long)   = 0.5
P(Sweet)  = 0.65
P(Yellow) = 0.8
```

Probability of "Likelihood"

```
P(Long|Banana) = 0.8
P(Long|Orange) = 0  [Oranges are never long in all the fruit we have seen.]
 ....

P(Yellow|Other Fruit)     =  50/200 = 0.25
P(Not Yellow|Other Fruit) = 0.75
```

Given a Fruit, how to classify it?

Let's say that we are given the properties of an unknown fruit, and asked to classify it. We are told that the fruit is Long, Sweet and Yellow. Is it a Banana? Is it an Orange? Or Is it some Other Fruit?

We can simply run the numbers for each of the 3 outcomes, one by one. Then we choose the highest probability and 'classify' our unknown fruit as belonging to the class that had the highest probability based on our prior evidence (our 1000 fruit training set):

```
P(Banana|Long, Sweet and Yellow)
      P(Long|Banana) * P(Sweet|Banana) * P(Yellow|Banana) * P(banana)
    = _______________________________________________________________
                     P(Long) * P(Sweet) * P(Yellow)

    = 0.8 * 0.7 * 0.9 * 0.5 / P(evidence)

    = 0.252 / P(evidence)


P(Orange|Long, Sweet and Yellow) = 0


P(Other Fruit|Long, Sweet and Yellow)
      P(Long|Other fruit) * P(Sweet|Other fruit) * P(Yellow|Other fruit) * P(Other Fruit)
    = ____________________________________________________________________________________
                                    P(evidence)

    = (100/200 * 150/200 * 50/200 * 200/1000) / P(evidence)

    = 0.01875 / P(evidence)
```

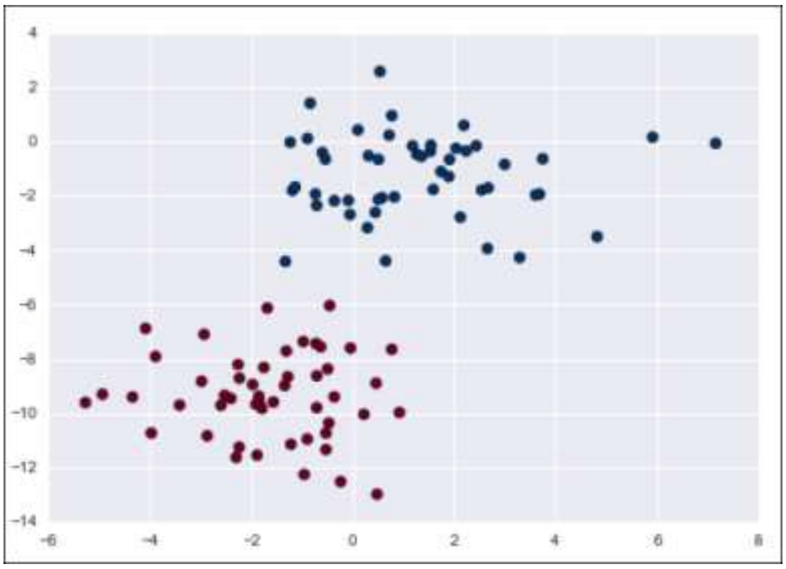
By an overwhelming margin (0.252 >> 0.01875), we classify this Sweet/Long/Yellow fruit as likely to be a Banana.

Gaussian Naïve Bayes

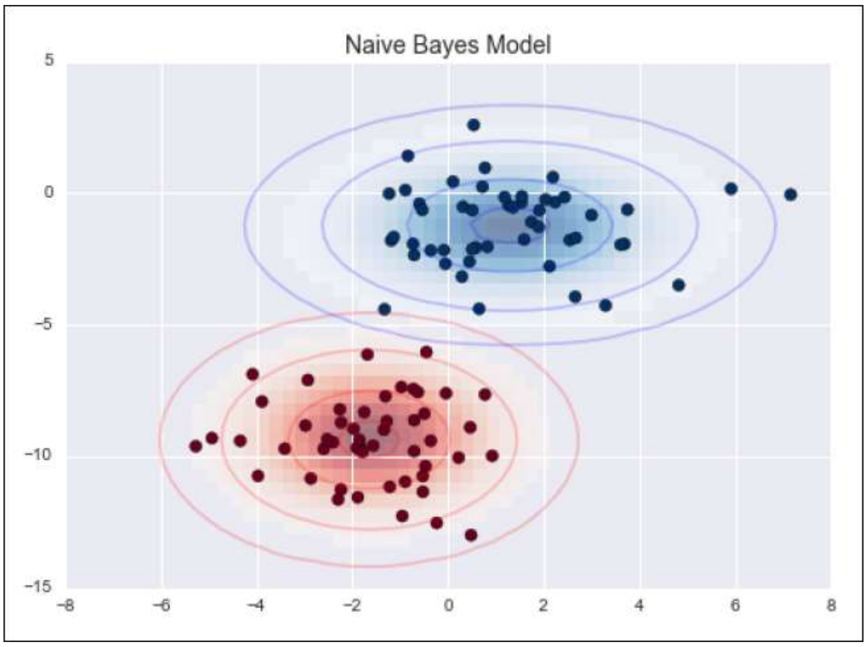
In the previous example it’s easy to calculate the probabilities because for each fruit we have three attributes (long, sweet, yellow) that can each have two values (yes/no) and we can count the number of fruit samples for each combination. However, in case your attributes can take lots of possible values (like shoe size) or are even continuous (like a person’s height or weight) it’s not longer that easy to calculate the probabilities for each combination in your dataset. In such a case you have to assume a probability distribution for your data samples, for instance assume that the height of a person is normally (or Gaussion) distributed.

When configuring the Naïve Bayes classifier in your machine learning program you have to indicate the kind of distribution of your data.

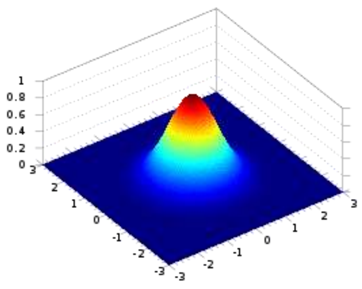
Obviously most Naïve Bayes implementation support the normal distribution, which is then called Gaussian Naive Bayes. In this classifier, the assumption is that data from each label is drawn from a simple Gaussian distribution. Imagine that you have the following data, where the x and y values represent features and each dot has to be classified (there are two labels: blue and red):



One extremely fast way to create a simple model is to assume that the data is described by a Gaussian distribution with no covariance between dimensions. We can fit this model by simply finding the mean and standard deviation of the points within each label, which is all you need to define such a distribution. The result of this naïve Gaussian assumption is shown in the figure below:



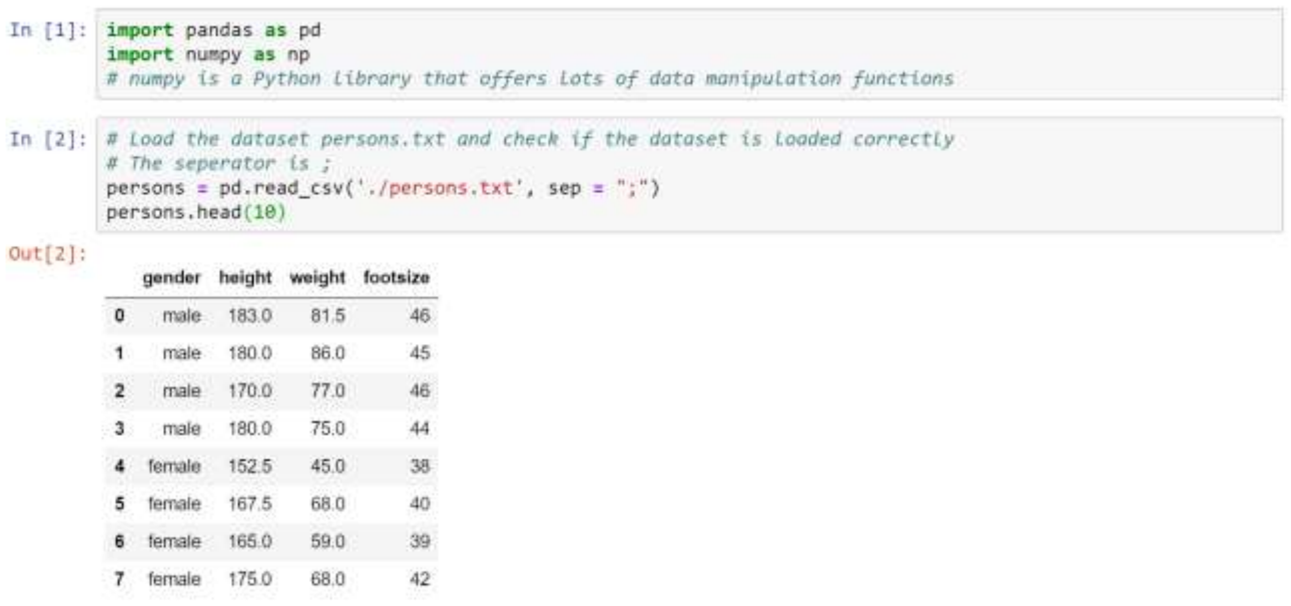
Each cloud of dots has to be interpreted as a three dimensional gauss curve, where the color saturation represents the z value or height: the darker the higher. A three-dimensional gauss curve is illustrated below.



Source: wikipedia

Example: classifying persons as male or female based on their height (in cm), weight (in kg) and foot size, see *1 Gaussian Naive Bayes.ipynb*.

Load the dataset persons.txt and check if the dataset is loaded correctly. The separator is ;



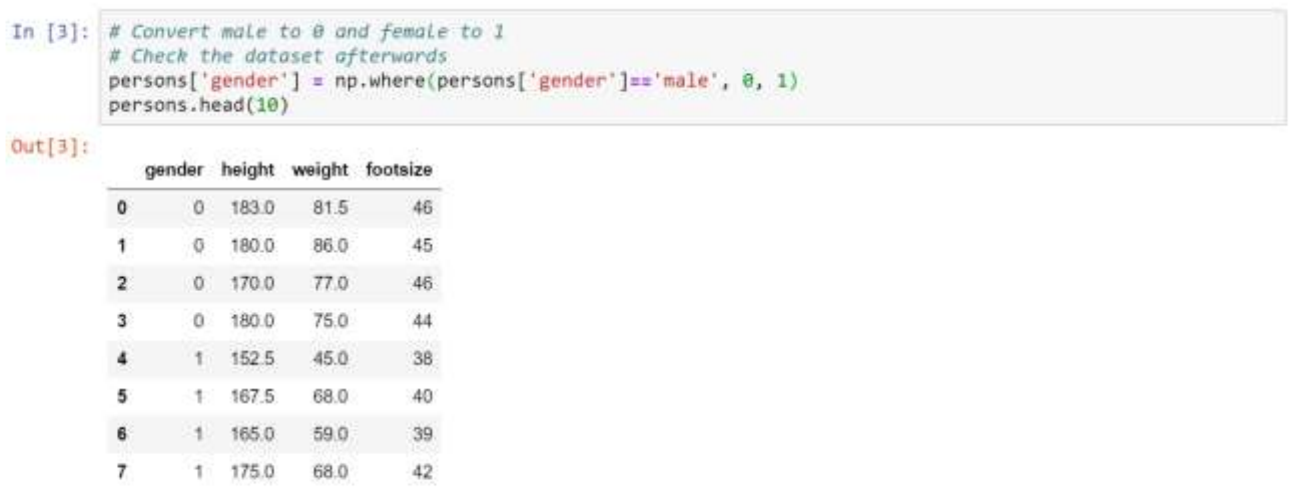
In [1]:
```python
import pandas as pd
import numpy as np
# numpy is a Python library that offers lots of data manipulation functions
```

In [2]:
```python
# Load the dataset persons.txt and check if the dataset is loaded correctly
# The seperator is ;
persons = pd.read_csv('./persons.txt', sep = ";")
persons.head(10)
```

Out[2]:

	gender	height	weight	footsize
0	male	183.0	81.5	46
1	male	180.0	86.0	45
2	male	170.0	77.0	46
3	male	180.0	75.0	44
4	female	152.5	45.0	38
5	female	167.5	68.0	40
6	female	165.0	59.0	39
7	female	175.0	68.0	42

Convert male to 0 and female to 1

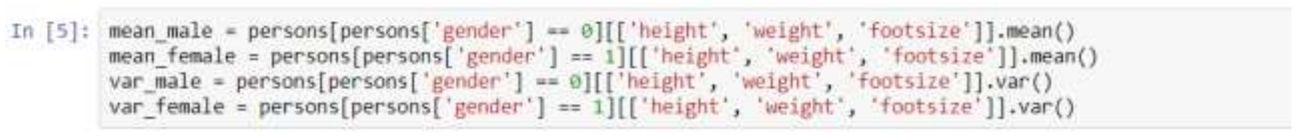


In [3]:
```python
# Convert male to 0 and female to 1
# Check the dataset afterwards
persons['gender'] = np.where(persons['gender']=='male', 0, 1)
persons.head(10)
```

Out[3]:

	gender	height	weight	footsize
0	0	183.0	81.5	46
1	0	180.0	86.0	45
2	0	170.0	77.0	46
3	0	180.0	75.0	44
4	1	152.5	45.0	38
5	1	167.5	68.0	40
6	1	165.0	59.0	39
7	1	175.0	68.0	42

Before we can compute the probability distribution for features x, we must first compute the mean μ and variance σ^2 values of x_i for each k class.



In [5]:
```python
mean_male = persons[persons['gender'] == 0][['height', 'weight', 'footsize']].mean()
mean_female = persons[persons['gender'] == 1][['height', 'weight', 'footsize']].mean()
var_male = persons[persons['gender'] == 0][['height', 'weight', 'footsize']].var()
var_female = persons[persons['gender'] == 1][['height', 'weight', 'footsize']].var()
```

```
In [6]: # We take a look at the computed values.

        print('Mean values for male features: ')
        print(mean_male)
        print()
        print('Mean values for female features: ')
        print(mean_female)
        print()
        print('Variance values for male features: ')
        print(var_male)
        print()
        print('Variance values for female features: ')
        print(var_female)
        print()
```

```
Mean values for male features:
height      178.250
weight       79.875
footsize     45.250
dtype: float64

Mean values for female features:
height      165.00
weight       60.00
footsize     39.75
dtype: float64

Variance values for male features:
height      32.250000
weight      24.062500
footsize     0.916667
dtype: float64

Variance values for female features:
height       87.500000
weight      118.000000
footsize      2.916667
dtype: float64
```

Now that we have the μ and $\sigma 2$ values for each feature x_i per k -class, let us now write a function for the likelihood computation, i.e. $p(x_i|C_k)$. We are going to plugin the likelihood computation into the Gaussian probability density function:

$$p(x = x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \cdot exp\left(\frac{-(x_i - \mu_k)^2}{2\sigma_k^2}\right)$$

Hence, we implement the likelihood function as follows:

```
In [7]: def likelihood(feature, mean, variance):
            return (1 / np.sqrt(2 * np.pi * variance)) * np.exp((-(feature - mean) ** 2) / (2 * variance))
```

Now we want to calculate the probability that a person with height = 175 and weight = 59 and footsize = 40 is male or female. Applying the formula of Bayes we have to calculate 2 things:

- P(male | height = 175 , weight = 59, footsize = 40) = P(height = 175 | male) P(weight = 59 | male) P(footsize = 40 | male) P(male) / (P(height = 175) P(weight = 59) * P(footsize = 40))
- P(female | height = 175 , weight = 59, footsize = 40) = P(height = 175 | female) P(weight = 59 | female) P(footsize = 40 | female) P(female) / (P(height = 175) P(weight = 59) * P(footsize = 40))

Because P(male) = P(female) = 0.5 and (P(height = 175) P (weight = 59) P(footsize = 40)) is for both formulas the same, these values don't matter.

So we only need to calculate

- P(height = 175 | male) P(weight = 59 | male) P(footsize = 40 | male)
- P(height = 175 | female) P(weight = 59 | female) P(footsize = 40 | female)

```
In [13]: p_height = likelihood(feature=175,
                          mean=np.array([mean_male['height'], mean_female['height']]),
                          variance=np.array([var_male['height'], var_female['height']]))
         print(p_height)

         [ 0.0596383   0.02408451]
```

```
In [14]: p_weight = likelihood(feature=59,
                          mean=np.array([mean_male['weight'], mean_female['weight']]),
                          variance=np.array([var_male['weight'], var_female['weight']]))
         print(p_weight)

         [  9.50078654e-06   3.65703260e-02]
```

```
In [15]: p_footsize = likelihood(feature=40,
                          mean=np.array([mean_male['footsize'], mean_female['footsize']]),
                          variance=np.array([var_male['footsize'], var_female['footsize']]))
         print(p_footsize)

         [  1.23191750e-07   2.31107219e-01]
```

```
In [16]: result = p_height * p_weight * p_footsize
         print(result)

         [  6.98017742e-14   2.03554218e-04]
```

Conclusion: this person is most likely to be a female because the second (female) result is the highest.

➔ This example is just to illustrate how Gaussian Naïve Bayes works. The calculations above are implemented in the Python libraries as we will see in a few moments.

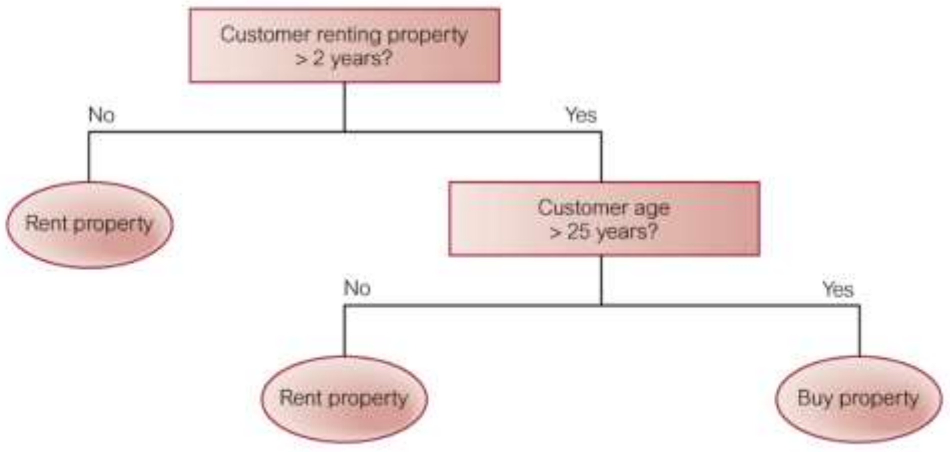
Multinomial Naïve Bayes

The Gaussian assumption just described is by no means the only simple assumption that could be used to specify the generative distribution for each label. Another useful example is multinomial naive Bayes, where the features are assumed to be generated from a simple multinomial distribution. The multinomial distribution describes the probability of observing counts among a number of categories, and thus multinomial naive Bayes is most appropriate for features that represent counts or count rates from a simple multinomial distribution. We will use Multinomial Naïve Bayes in the chapter about Natural Language Processing (NLP) for instance when predicting spam based on count rates of words.

3.2.2 Tree Classification

This algorithm is a widely-used method of constructing a model from a dataset in the form of a decision tree. It is often claimed that this representation of the data has the advantage compared with other approaches of being meaningful and easy to interpret. As an example you can consider a database in use at a real-estate company. This database contains customer data (including age) and rental history. The company wants to predict how likely it is that a renter will buy a house, based on historical data about buyers and renters.

A tree classification algorithm will create a tree classifier as in the figure below.

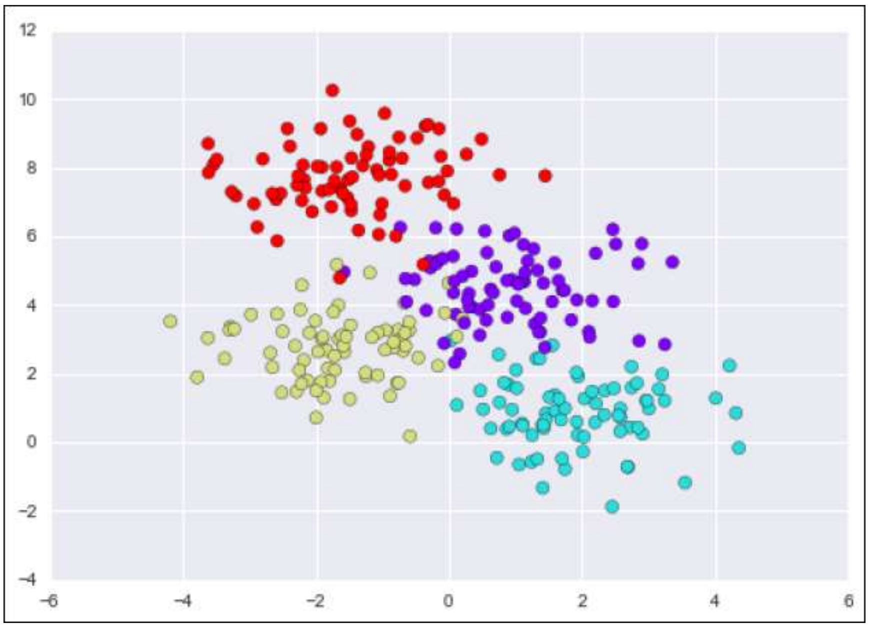


We can conclude from this tree classification that only customers that are renting for over two years and that are older than 25 years are considering to buy a property. The real-estate company can use this information to advertise properties for sale to this customer segment.

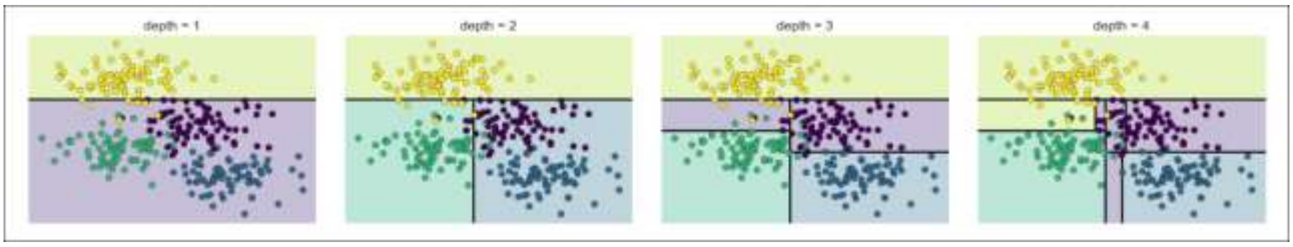
The binary splitting makes this extremely efficient: in a well-constructed tree, each question will cut the number of options by approximately half, very quickly narrowing the options even among a large number of classes. The trick, of course, comes in deciding which questions to ask at each step. In machine learning implementations of decision trees, the questions generally take the form of axis-aligned splits in the data; that is, each node in the tree splits the data into two groups using a cutoff value within one of the features. Let’s now take a look at an example.

Creating a decision tree

Consider the following two-dimensional data, which has one of four class labels (VanderPlas, 2016).



A simple decision tree built on this data will iteratively split the data along one or the other axis according to some quantitative criterion, and at each level assign the label of the new region according to a majority vote of points within it. The next image presents a visualization of the first four levels of a decision tree classifier for this data.



Notice that after the first split, every point in the upper branch remains unchanged, so there is no need to further subdivide this branch. Except for nodes that contain all of one color, at each level every region is again split along one of the two features.

Notice that as the depth increases, we tend to get very strangely shaped classification regions; for example, at a depth of four, there is a tall and skinny purple region between the yellow and blue regions. It's clear that this is less a result of the true, intrinsic data distribution, and more a result of the particular sampling or noise properties of the data. That is, this decision tree, even at only five levels deep, is clearly overfitting our data.

Decision trees and overfitting

Such overfitting turns out to be a general property of decision trees; it is very easy to go too deep in the tree, and thus to fit details of the particular data rather than the overall properties of the distributions they are drawn from.

Ensembles of Estimators: Random Forests

We can also see that if we train models on different subsets of the data – for example, we train two different trees, each on half of the original data, the two trees produce consistent results for some combinations of the features, while in other places, the two trees give very different. The key observation is that the inconsistencies tend to happen where the classification is less certain, and thus by using information from both of these trees, we might come up with a better result! This notion—that multiple overfitting estimators can be combined to reduce the effect of this overfitting—is what underlies an ensemble method called bagging. Bagging makes use of an ensemble (a grab bag, perhaps) of parallel estimators, each of which overfits the data, and averages the results to find a better classification. An ensemble of randomized decision trees is known as a random forest.

Because random forests train a set of decision trees separately, the training can be done in parallel. The algorithm injects randomness into the training process so that each decision tree is a bit different. Combining the predictions from each tree reduces the variance of the predictions, improving the performance on test data.

The randomness injected into the training process includes:

- Subsampling the original dataset on each iteration to get a different training set (a.k.a. bootstrapping).
- Considering different random subsets of features to split on at each tree node.

Apart from these randomizations, decision tree training is done in the same way as for individual decision trees.

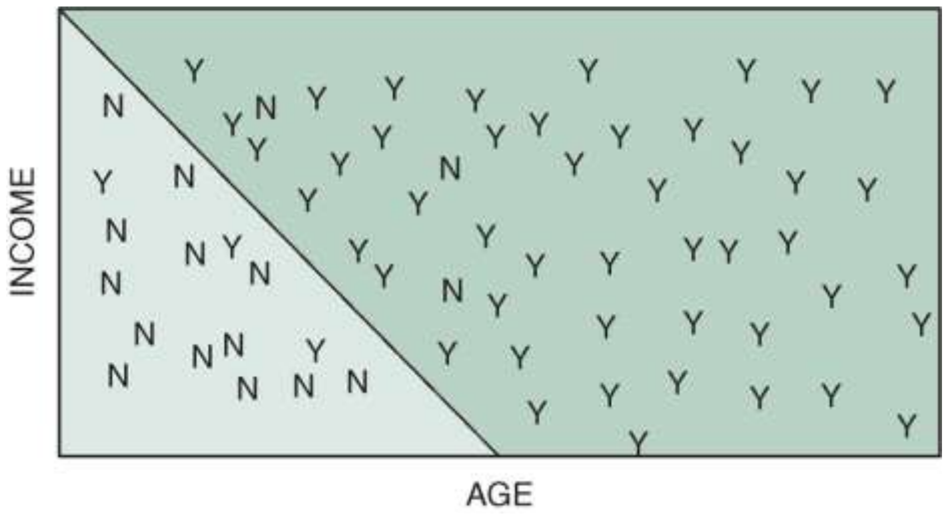
To make a prediction on a new instance, a random forest must aggregate the predictions from its set of decision trees. This aggregation is done differently for classification and regression (see §3.9).

- Classification: Majority vote. Each tree's prediction is counted as a vote for one class. The label is predicted to be the class which receives the most votes.

- Regression: Averaging. Each tree predicts a real value. The label is predicted to be the average of the tree predictions.

3.2.3 Logistic Regression

In (Lievens, 2018) it is explained that logistic regression is (another) classification technique in which essentially a (linear or nonlinear) decision boundary is determined. Suppose we have two attributes Income and Age and want to predict whether or not a customer will respond to an advertisement. This is a binary classification problem with two input variables. Logistic Regression will determine a decision boundary as in the figure below (Lemahieu, Vanden Broucke, & Baesens, 2018). (Income, Age) combinations below the line are then estimated as N (does not respond) and vice versa.



In the above example we have to determine a line with the equation

$$\theta_0 + \theta_1 \text{Income} + \theta_2 \text{Age} = 0$$

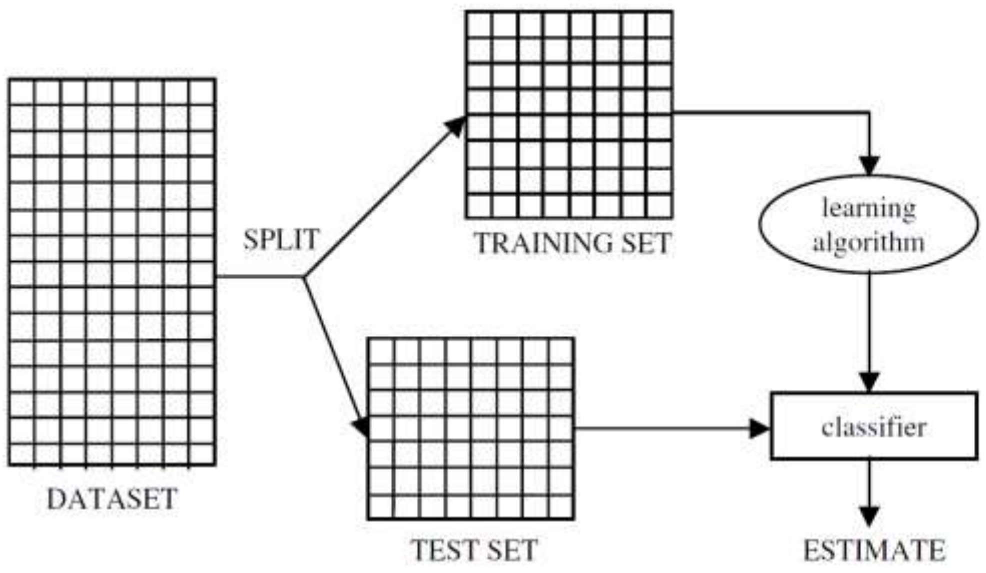
which separates " as much as possible" the points with response = N (below the line) from the points with response = Y (above the line).

3.3 Ensemble methods

Random Forest is an example of an ensemble method in which several estimators of the same type (i.e. decision tree) are combined and a voting mechanism is used to determine the estimated class. The same principle can be used when combining estimators of different types (for instance Naïve Bayes, Random Forest and Logistic Regression) and predict the label based on a voting mechanism. It can be proven that, in general, the predictive accuracy (see next paragraph) of an ensemble classifier is better than that of the individual classifiers.

3.4 Estimating the predictive accuracy of a classifier

The most obvious criterion to use for estimating the performance of a classifier is predictive accuracy, i.e. the proportion of a set of unseen instances that it correctly classifies. It is usual to estimate the predictive accuracy of a classifier by measuring its accuracy for a sample of data not used when it was generated. For this method the available data is split into two parts called a training set and a test set (see figure below, (Bramer, 2016)). First, the training set is used to construct a classifier (decision tree, naïve bayes, etc.). The classifier is then used to predict the classification for the instances in the test set. If the test set contains N instances of which C are correctly classified the predictive accuracy of the classifier for the test set is p = C/N. This can be used as an estimate of its performance on any unseen dataset.

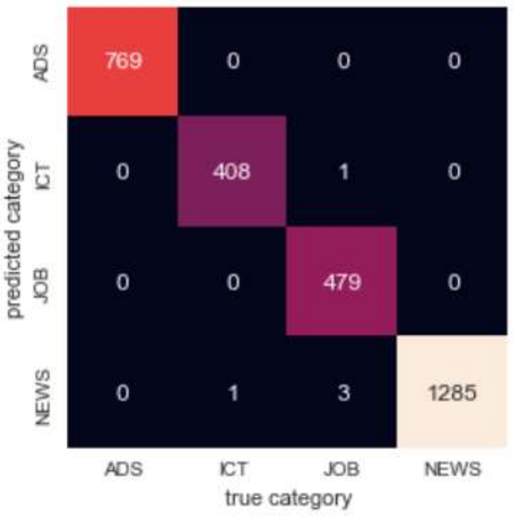


A random division into two parts in proportions such as 1:1, 2:1, 70:30 or 60:40 is customary (the largest part is the training set). Obviously, the larger the proportion of the training set, the better the model, but the worse the correctness of the calculated accuracy and vice versa.

3.5 Other estimators for the predictive accuracy

Calculating the accuracy of a classifier merely as a percentage of the correctly classified samples in your test set is not always sufficient. Suppose you want to predict if a person has cancer based on some features like tumor size, tumor shape and age. If your test set is a subset of the complete population we know that only a very small subset of the people (fortunately) really have cancer. So if you always predict “no cancer” you will probably have an accuracy of over 95%, but are still missing all the persons who have cancer. Your model should at least be better than this “guessing” method.

In general, it is often helpful to see a breakdown of the classifier’s performance, i.e. how frequently instances of class X were correctly classified as class X or misclassified as some other class. This information is given in a *confusion matrix*. Below is a confusion matrix for a (very efficient) e-mail classification model which classifies incoming e-mails (from mailing lists) into four categories: ADS, ICT, JOB and NEWS.

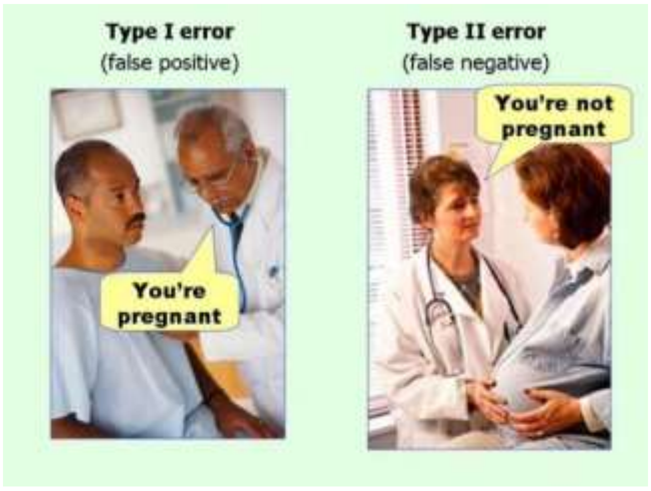


predicted category \ true category	ADS	ICT	JOB	NEWS
ADS	769	0	0	0
ICT	0	408	1	0
JOB	0	0	479	0
NEWS	0	1	3	1285

For binary classifiers (like cancer/no cancer) the *confusion matrix* looks like this:

	TRUE: yes	TRUE: no
PREDICTION: yes	True positive (TP)	False positive (FP)
PREDICTION: no	False negative (FN)	True Negative (TN)

In medicine false positives are considered as Type I errors and false negatives as Type II errors:



The false positive rate (FP rate) is the proportion of all negatives that still yield positive test outcomes:

$$\frac{FP}{FP + TN}$$

Complementarily, the false negative rate (FN rate) is the proportion of positives which yield negative test outcomes with the test:

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

Obviously, in the cancer case we have to avoid false negatives, because in that case we are missing people really having cancer. See (Lievens, 2018) for more measures to determine the accuracy of binary classifiers, like precision, recall (or true positive rate) and f-score.

One of the strengths of characterizing a classifier by its TP Rate and FP Rate values is that they do not depend on the relative sizes of positives values (P) and negatives values (N). The same applies to using the FN Rate and TN Rate values or any other combination of two 'rate' values calculated from different rows of the confusion matrix. In contrast, the Predictive Accuracy is derived from values in both rows of the table and so is affected by the relative sizes of P and N, which can be a serious weakness (Bramer, 2016).

3.6 Machine Learning with Python Scikit-Learn

There are several Python libraries that provide solid implementations of a range of machine learning algorithms. One of the best known is Scikit-Learn (scikit-learn Machine Learning in Python, n.d.), a package that provides efficient versions of a large number of commonly used algorithms. Scikit-Learn is characterized by a clean, uniform, and streamlined API, as well as by very useful and complete online documentation. A benefit of this uniformity is that once you understand the basic use and syntax of Scikit-Learn for one type of model, switching to a new model or algorithm is very straightforward (VanderPlas, 2016).

3.6.1 Data Representation in Scikit-Learn

Machine learning is about creating models from data: for that reason, we'll start by discussing how data can be represented in order to be understood by the computer. The best way to think about data within Scikit-Learn is in terms of tables of data.

Data as table

A basic table is a two-dimensional grid of data, in which the rows represent individual elements of the dataset, and the columns represent quantities related to each of these elements. For example, in the titanic table each row refers to a single passenger, and the number of rows is the total number of passengers in the dataset. In general, we will refer to the rows of the matrix as samples, and the number of rows as n_samples.

Likewise, each column of the data refers to a particular quantitative piece of information that describes each sample. In general, we will refer to the columns of the matrix as features, and the number of columns as n_features.

Features matrix

This table layout makes clear that the information can be thought of as a two dimensional numerical array or matrix, which we will call the features matrix. By convention, this features matrix is often stored in a variable named X. The features matrix is assumed to be two-dimensional, with shape [n_samples, n_features], and is most often contained in a NumPy array or a Pandas DataFrame.

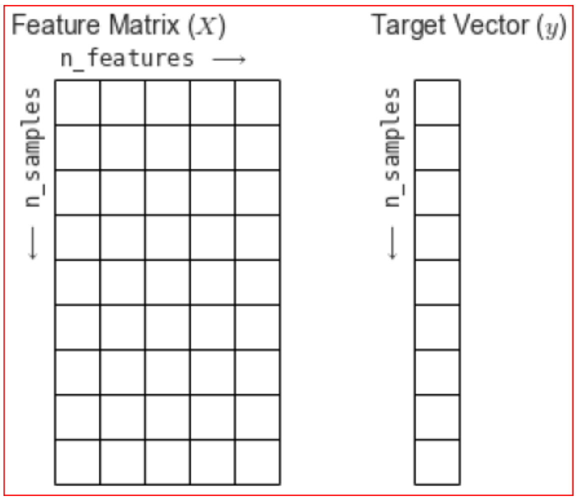
The samples (i.e., rows) always refer to the individual objects described by the dataset. For example, the sample might be a flower, a person, a document, an image, a sound file, a video, an astronomical object, or anything else you can describe with a set of quantitative measurements.

The features (i.e., columns) always refer to the distinct observations that describe each sample in a quantitative manner. Features are generally real-valued, but may be Boolean or discrete-valued in some cases.

Target array

In addition to the feature matrix X, we also generally work with a label or target array, which by convention we will usually call y. The target array is usually one dimensional, with length n_samples, and is generally contained in a NumPy array or Pandas Series. The target array may have continuous numerical values, or discrete classes/labels. While some Scikit-Learn estimators do handle multiple target values in the form of a two-dimensional [n_samples, n_targets] target array, we will primarily be working with the common case of a one-dimensional target array.

Often one point of confusion is how the target array differs from the other features columns. The distinguishing feature of the target array is that it is usually the quantity we want to predict from the data: in statistical terms, it is the dependent variable. For example, in the titanic case we may wish to construct a model that can predict if a passenger has survived or not based on the other features; in this case, the Survived column would be considered the target. To summarize, the expected layout of features and target values can be visualized as follows:



3.6.2 Scikit-Learn's Estimator API

The Scikit-Learn API is designed with the following guiding principles in mind.

Consistency

All objects share a common interface drawn from a limited set of methods, with consistent documentation.

Inspection

All specified parameter values are exposed as public attributes.

Limited object hierarchy

Only algorithms are represented by Python classes; datasets are represented in standard formats (NumPy arrays, Pandas DataFrames, SciPy sparse matrices) and parameter names use standard Python strings.

Composition

Many machine learning tasks can be expressed as sequences of more fundamental algorithms, and Scikit-Learn makes use of this wherever possible.

Sensible defaults

When models require user-specified parameters, the library defines an appropriate default value.

Basics of the API

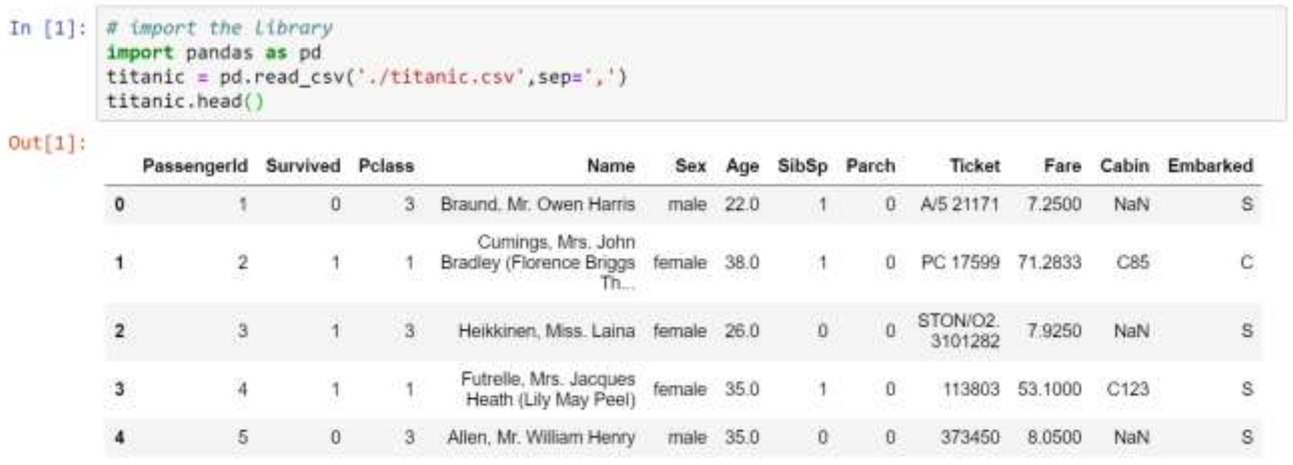
Most commonly, the steps in using the Scikit-Learn estimator API are as follows (we will step through a handful of detailed examples in the sections that follow):

1. Choose a class of model by importing the appropriate estimator class from Scikit-Learn.
2. Choose model hyperparameters by instantiating this class with desired values.
3. Arrange data into a features matrix and target vector following the discussion from before.
4. Fit the model to your data by calling the fit() method of the model instance.
5. Apply the model to new data:
   - For supervised learning, often we predict labels for unknown data using the predict() method.
   - For unsupervised learning, we often transform or infer properties of the data using the transform() or predict() method.

3.6.3 Naïve Bayes classification with Scikit-Learn

In this example (see *2. Titanic naive bayes.ipynb*), we start by analyzing what kind of people were likely to survive the Titanic. In particular, we will apply the tools of machine learning to predict which passengers survived the tragedy.

We can import the csv file that represents the data and turn it into a dataframe.



```
In [1]: # import the library
        import pandas as pd
        titanic = pd.read_csv('./titanic.csv',sep=',')
        titanic.head()
```

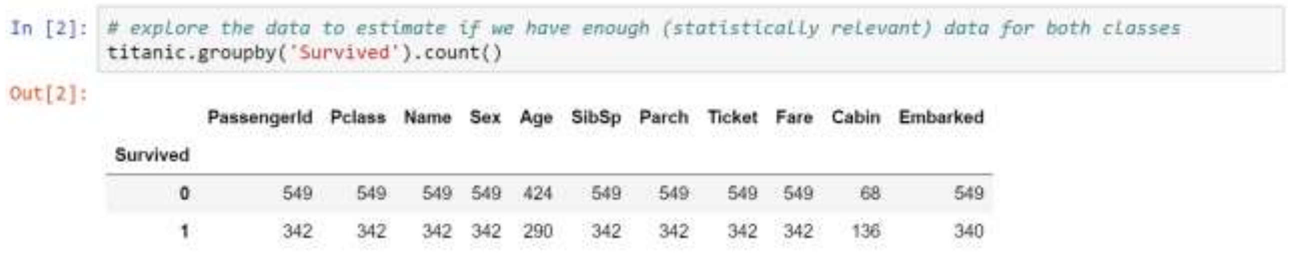
Out[1]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Data from all kinds of sources can be turned into a Pandas DataFrame, even from a connection to a SQL database.

The column "Survived" is considered as the label or the class. Supervised learning classification techniques create a mathematical model, also called a *classifier*, from a large set of labelled data and try to predict the label for unclassified data. In the case of the Titanic we could try to guess, based on what happened to a large number of passengers, whether female passenger X, who travelled in third class, was 32 years old, had no parents, two children and no siblings on board and who embarked in Southampton survived the disaster (supposing this information is unknown).

Let's check how many passengers in our dataset survived and how many didn't.



```
In [2]: # explore the data to estimate if we have enough (statistically relevant) data for both classes
        titanic.groupby('Survived').count()
```

Out[2]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
Survived											
0	549	549	549	549	424	549	549	549	549	68	549
1	342	342	342	342	290	342	342	342	342	136	340

The process of calculating a classifier based on a labelled data set that can be used for predicting the label of unlabeled data is called *training* or *fitting*.

3.6.4 Data preparation

In each data mining case the raw data that comes in is not ready to process, but has to be prepared before it can be fed to the algorithm.

Reducing the number of attributes

Some attributes are clearly irrelevant for prediction of the label. In our case the name, the tickets and cabin numbers and the port of embarkation seem irrelevant. The ticket fare is directly related to the ticket class. We decide to ignore all these attributes. This process is called feature selection. It is often by far not so obvious as in this case. Also pay attention for bias! Don't let you own opinion play.

```
In [3]: # We drop clearly irrelevant attributes. Pay attention for bias! Don't let your own opinion play.
        titanic = titanic.drop(['PassengerId','Name','Ticket','Fare','Cabin','Embarked'],axis=1)
        titanic.head()
```

Out[3]:

	Survived	Pclass	Sex	Age	SibSp	Parch
0	0	3	male	22.0	1	0
1	1	1	female	38.0	1	0
2	1	3	female	26.0	0	0
3	1	1	female	35.0	1	0
4	0	3	male	35.0	0	0

Data cleaning

In the dataset some necessary data might be missing or can be inconsistent with other data. In our example, as we see in the result of the groupby and count above, for some passengers the age is missing. Since we consider this feature as possibly important for survival we choose to delete these passengers from the dataset we use for training the model.

Using Pandas dataframes we can easily drop those lines from the dataframe (dropna = drop if not available):

```
In [4]: print('Before')
        print(titanic.count())
        print()

        # drop all lines that contain empty (null or NaN) values
        titanic = titanic.dropna()

        print('After')
        print(titanic.count())
```

```
Before
Survived    891
Pclass      891
Sex         891
Age         714
SibSp       891
Parch       891
dtype: int64

After
Survived    714
Pclass      714
Sex         714
Age         714
SibSp       714
Parch       714
dtype: int64
```

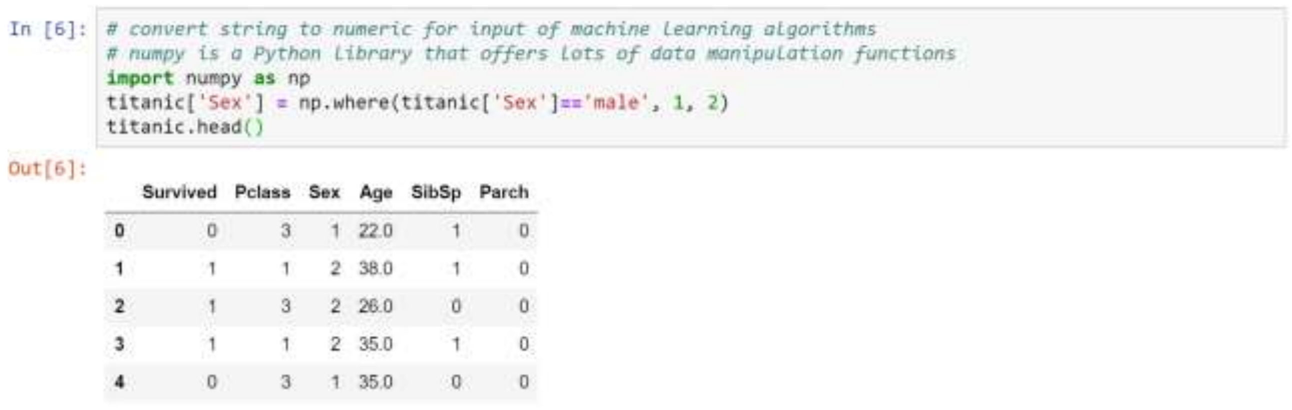
```
In [5]: # see what remains
        titanic.groupby('Survived').count()
```

Out[5]:

	Pclass	Sex	Age	SibSp	Parch
Survived					
0	424	424	424	424	424
1	290	290	290	290	290

Data type conversion

Some algorithms only work with numeric data. This means we have to convert data like male/female to 1/2.

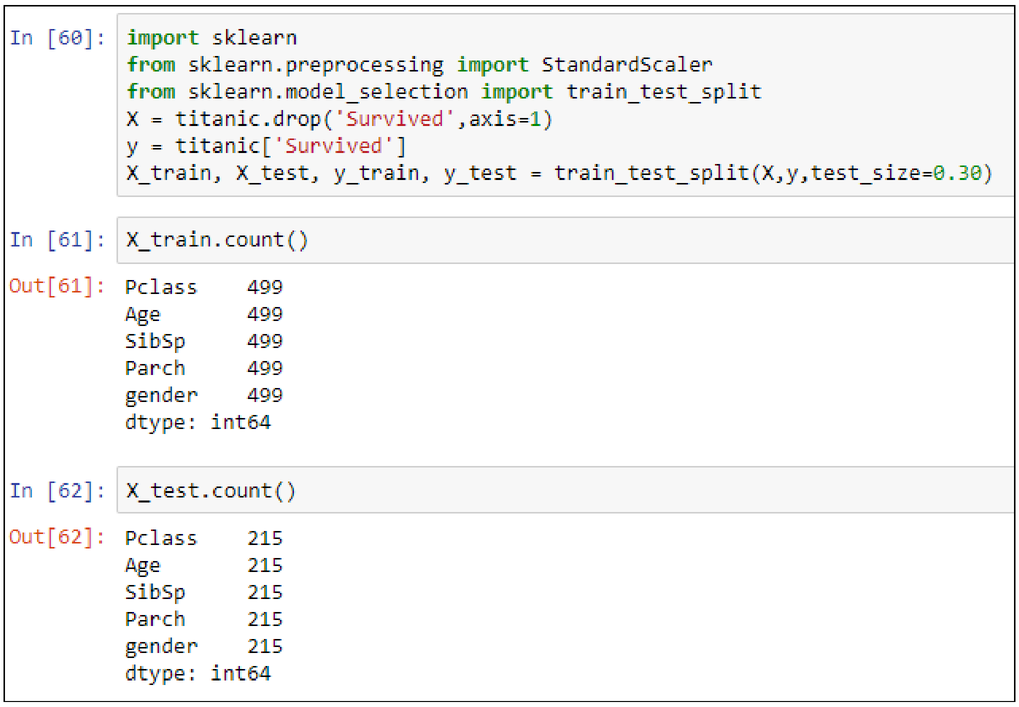


```
In [6]: # convert string to numeric for input of machine learning algorithms
        # numpy is a Python library that offers lots of data manipulation functions
        import numpy as np
        titanic['Sex'] = np.where(titanic['Sex']=='male', 1, 2)
        titanic.head()
```

Out[6]:

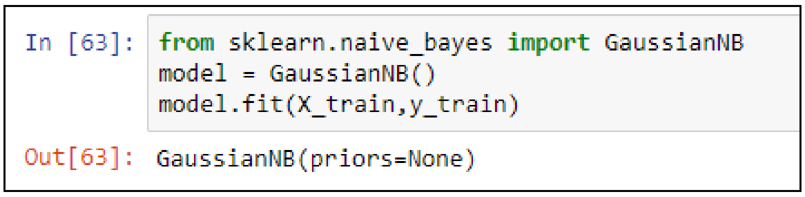
	Survived	Pclass	Sex	Age	SibSp	Parch
0	0	3	1	22.0	1	0
1	1	1	2	38.0	1	0
2	1	3	2	26.0	0	0
3	1	1	2	35.0	1	0
4	0	3	1	35.0	0	0

Split the data set in a training set and a test set



```
In [60]: import sklearn
         from sklearn.preprocessing import StandardScaler
         from sklearn.model_selection import train_test_split
         X = titanic.drop('Survived',axis=1)
         y = titanic['Survived']
         X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30)

In [61]: X_train.count()

Out[61]: Pclass     499
         Age        499
         SibSp      499
         Parch      499
         gender     499
         dtype: int64

In [62]: X_test.count()

Out[62]: Pclass     215
         Age        215
         SibSp      215
         Parch      215
         gender     215
         dtype: int64
```

Choose the model and fit the data to the model

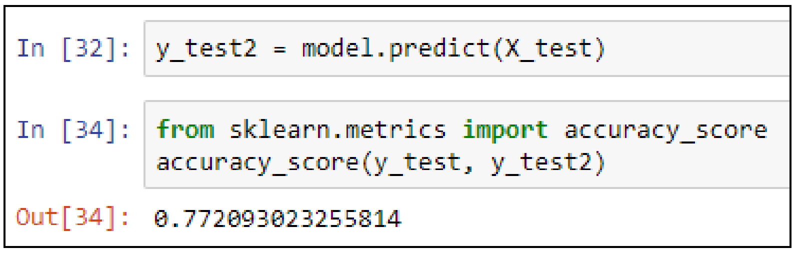


```
In [63]: from sklearn.naive_bayes import GaussianNB
         model = GaussianNB()
         model.fit(X_train,y_train)

Out[63]: GaussianNB(priors=None)
```

No the model is calculated and we are ready to use it to predict the labels (Survived/Not survived) of the test set and thus calculate the predictive accuracy.

Estimate the predictive accuracy of the classifier

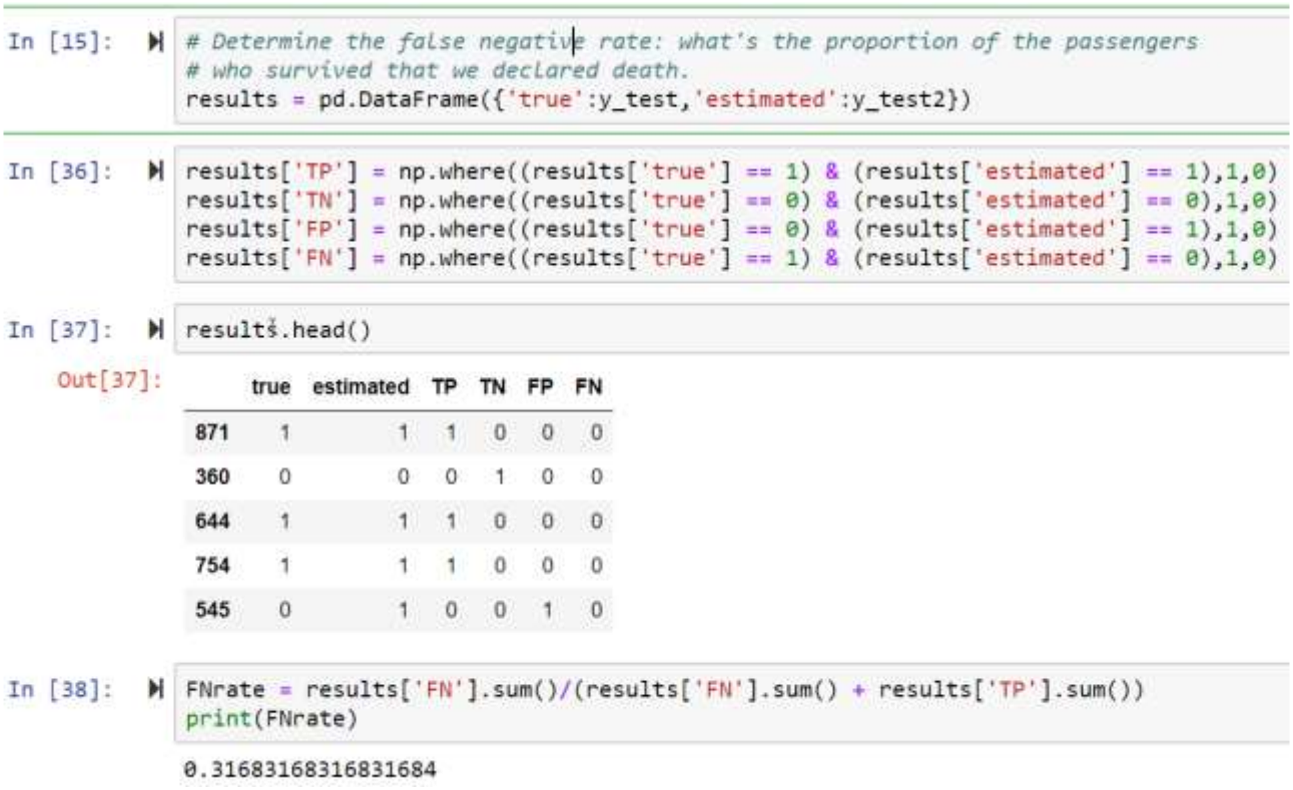
To estimate the accuracy of the model we simply apply the model to the test data we have kept apart and compare the labels we have kept apart with the predicted labels using the function accuracy_score from the sklearn.cross_validation library.



```
In [32]: y_test2 = model.predict(X_test)

In [34]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test, y_test2)

Out[34]: 0.772093023255814
```

We see that the accuracy is about 77%. Is this better than simply predicting “Not survived” for everyone? Yes, because “only” 424/(424+290) = 59% of the passengers didn’t survive.

Determine the false negative rate



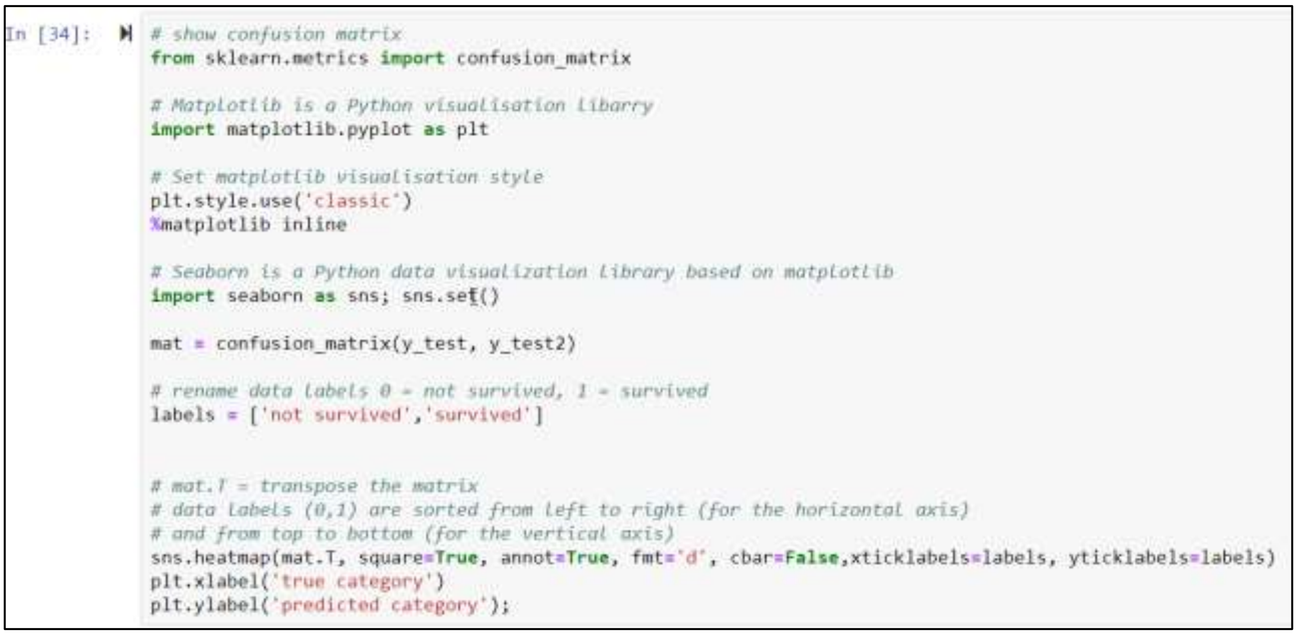
```
In [15]: # Determine the false negative rate: what's the proportion of the passengers
         # who survived that we declared death.
         results = pd.DataFrame({'true':y_test,'estimated':y_test2})

In [36]: results['TP'] = np.where((results['true'] == 1) & (results['estimated'] == 1),1,0)
         results['TN'] = np.where((results['true'] == 0) & (results['estimated'] == 0),1,0)
         results['FP'] = np.where((results['true'] == 0) & (results['estimated'] == 1),1,0)
         results['FN'] = np.where((results['true'] == 1) & (results['estimated'] == 0),1,0)

In [37]: results.head()

Out[37]:
```

	true	estimated	TP	TN	FP	FN
871	1	1	1	0	0	0
360	0	0	0	1	0	0
644	1	1	1	0	0	0
754	1	1	1	0	0	0
545	0	1	0	0	1	0

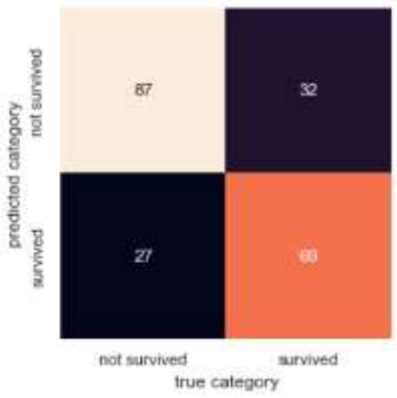
```
In [38]: FNrate = results['FN'].sum()/(results['FN'].sum() + results['TP'].sum())
         print(FNrate)

         0.31683168316831684
```

This means for about 32 % of all passengers who did survive we predicted they didn’t.

Show the confusion matrix



```
In [34]: # show confusion matrix
         from sklearn.metrics import confusion_matrix

         # Matplotlib is a Python visualisation library
         import matplotlib.pyplot as plt

         # Set matplotlib visualisation style
         plt.style.use('classic')
         %matplotlib inline

         # Seaborn is a Python data visualization library based on matplotlib
         import seaborn as sns; sns.set()

         mat = confusion_matrix(y_test, y_test2)

         # rename data labels 0 = not survived, 1 = survived
         labels = ['not survived','survived']


         # mat.T = transpose the matrix
         # data labels (0,1) are sorted from left to right (for the horizontal axis)
         # and from top to bottom (for the vertical axis)
         sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,xticklabels=labels, yticklabels=labels)
         plt.xlabel('true category')
         plt.ylabel('predicted category');
```

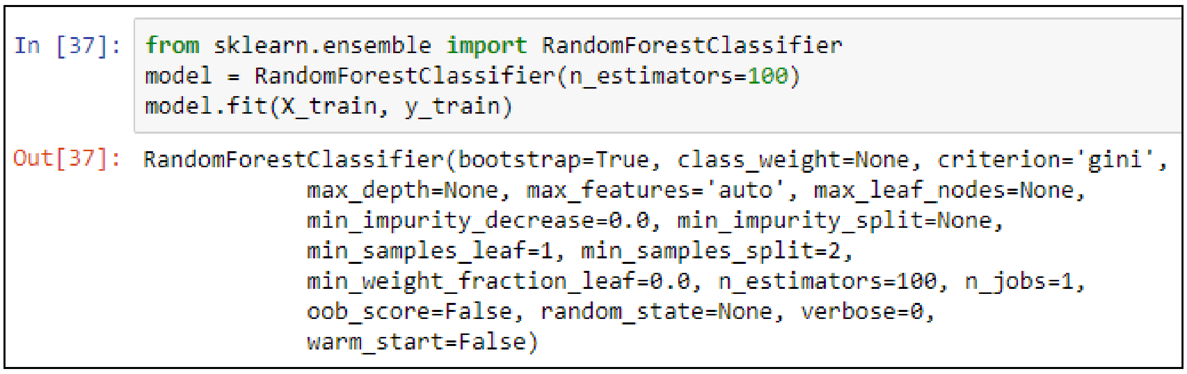


3.6.5 Random Forest Classification with Scikit-Learn

The process of fitting a decision tree to our data can be done in Scikit-Learn with the DecisionTreeClassifier estimator (see *3. Titanic random forest.ipynb*):

```
In[63]: from sklearn.tree import DecisionTreeClassifier

    tree = DecisionTreeClassifier().fit(X, y)
```

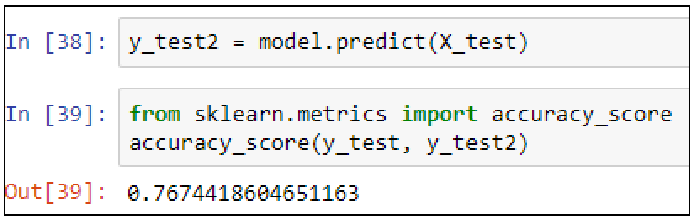
However, as we have seen earlier, it’s better to use Random Forest to reduce overfitting. Applying the Random Forest classifier to the titanic data is very similar to Naïve Bayes.



```
In [37]: from sklearn.ensemble import RandomForestClassifier
         model = RandomForestClassifier(n_estimators=100)
         model.fit(X_train, y_train)

Out[37]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False)
```

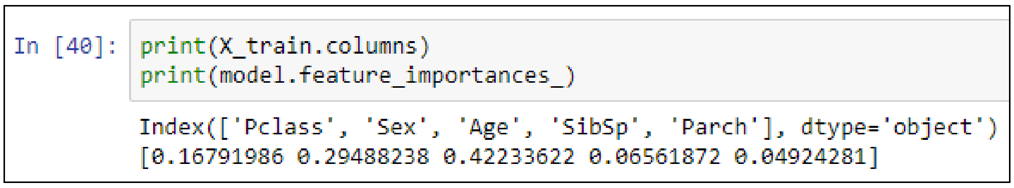
Parameter n_estimators is the number of trees in the forest. Default = 10

It turns out that the accuracy of the Random Forest classifier is approximately 76,7 %, which is, in this case, very close to Naïve Bayes.



```
In [38]: y_test2 = model.predict(X_test)

In [39]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test, y_test2)

Out[39]: 0.7674418604651163
```

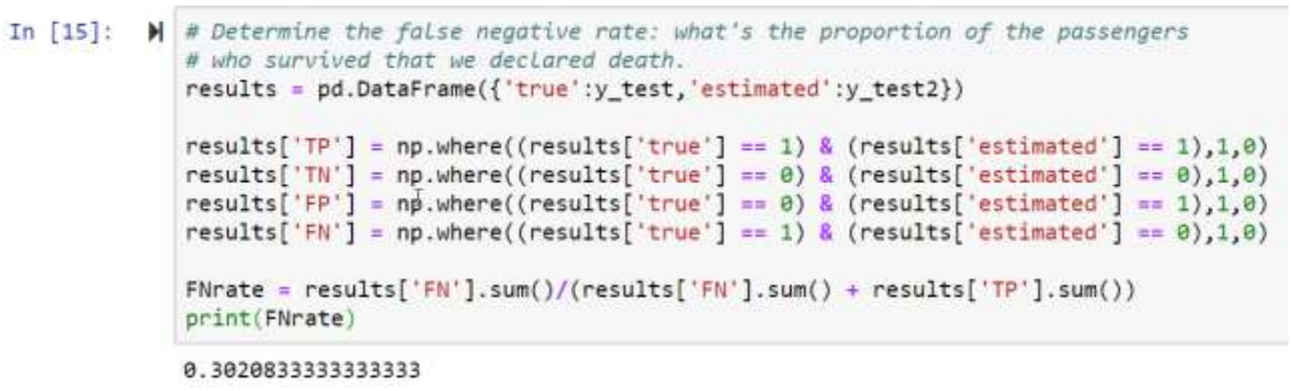
However, Decision Tree and Random Forest classifiers have one major advantage over Naïve Bayes: you can determine the relative importance of each feature.



```
In [40]: print(X_train.columns)
         print(model.feature_importances_)

         Index(['Pclass', 'Sex', 'Age', 'SibSp', 'Parch'], dtype='object')
         [0.16791986 0.29488238 0.42233622 0.06561872 0.04924281]
```

We learn from this that by far the three most important criteria to survive the Titanic disaster were: (1) Age, (2) Sex and (3) Ticket class.

Again, determine the false negative rate:



```
In [15]:  # Determine the false negative rate: what's the proportion of the passengers
          # who survived that we declared death.
          results = pd.DataFrame({'true':y_test,'estimated':y_test2})

          results['TP'] = np.where((results['true'] == 1) & (results['estimated'] == 1),1,0)
          results['TN'] = np.where((results['true'] == 0) & (results['estimated'] == 0),1,0)
          results['FP'] = np.where((results['true'] == 0) & (results['estimated'] == 1),1,0)
          results['FN'] = np.where((results['true'] == 1) & (results['estimated'] == 0),1,0)

          FNrate = results['FN'].sum()/(results['FN'].sum() + results['TP'].sum())
          print(FNrate)

          0.3020833333333333
```

The results, both in terms of predictive accuracy and FN rate are approximately the same as for Naïve Bayes. Also note that for consecutive runs of the algorithms you get slightly different results. This is because the data set is randomly split into a training and test set and (in the case of random forest) the trees are randomly chosen at each run.

3.6.6 One hot encoding for categorical features

One common type of non-numerical data is categorical data. For example, in the previous example, Sex = male or female. We encode male as 1 and female as 2 because the algorithms we used only work with numerical features. However, Scikit-Learn models make the fundamental assumption that numerical features reflect algebraic quantities, so in our example they would assume that a female is twice a male, which does not make much sense. In this case, one proven technique is to use one-hot encoding, which effectively creates extra columns indicating the presence or absence of a category with a value of 1 or 0 respectively. This has the benefit of not weighting a value improperly but does have the downside of adding more columns to the data set. Pandas supports this feature using get_dummies. This function is named this way because it creates dummy/indicator variables (aka 1 or 0).

In our example we can then replace the line

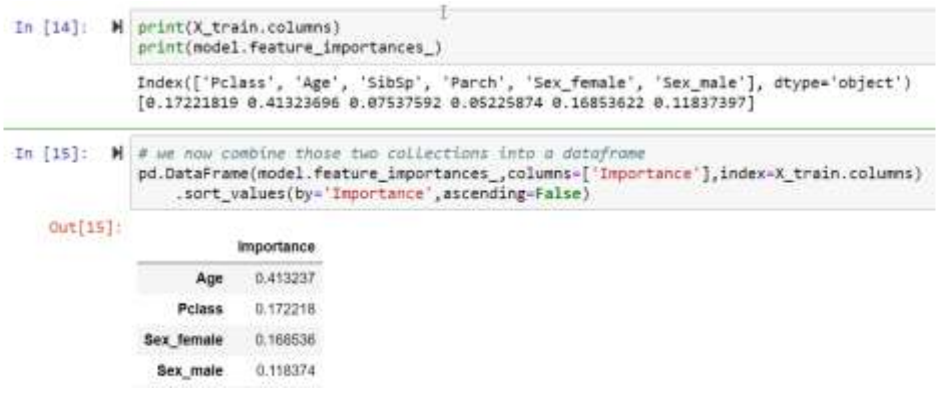
```
    titanic['Sex'] = np.where(titanic['Sex']>='male', 1, 2)
```

by (see *4. Titanic random forest-onehotencoded.ipynb*)

```
    titanic = pd.get_dummies(titanic, columns=["Sex"], prefix=["Sex"])
```

Of course, you can also use one hot encoding for 'numerical' features that are categorical by nature. For example, if in the original titanic data the gender would have been encoded as 1 or 2.

A disadvantage of splitting the columns is that the relative importances are also split:



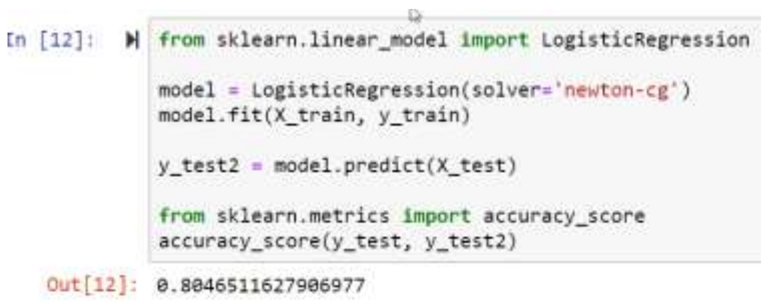
```
In [14]:  print(X_train.columns)
          print(model.feature_importances_)

          Index(['Pclass', 'Age', 'SibSp', 'Parch', 'Sex_female', 'Sex_male'], dtype='object')
          [0.17221819 0.41323696 0.07537592 0.05225874 0.16853622 0.11837397]

In [15]:  # we now combine those two collections into a dataframe
          pd.DataFrame(model.feature_importances_,columns=['Importance'],index=X_train.columns)
              .sort_values(by='Importance',ascending=False)

Out[15]:
```

	Importance
Age	0.413237
Pclass	0.172218
Sex_female	0.168536
Sex_male	0.118374

To get the "complete" importance of the attribute Sex in the above example we have to sum Sex_female and Sex_male, which shows that age and gender were the two most important factors in determining a passenger's survival chances in the Titanic disaster ("women and children first").

3.6.7 Logistic Regression with Scikit-Learn

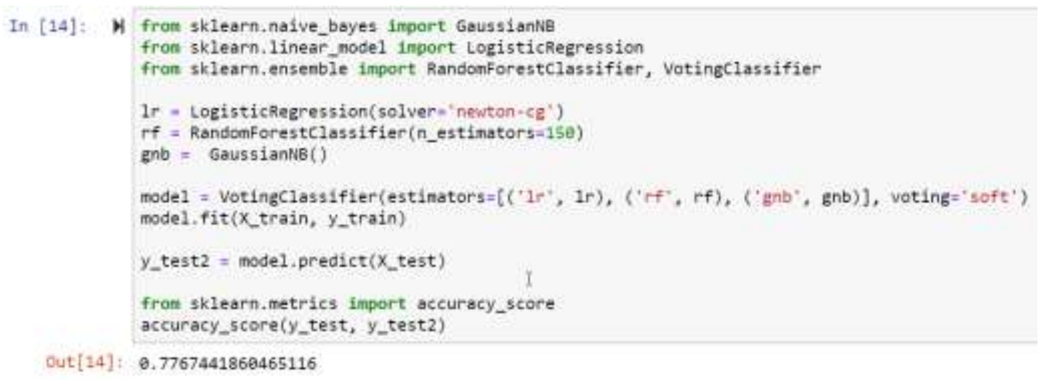
Due to the uniform API in Scikit-Learn we only have to change the Naïve Bayes or Random Forest classifier into Logistic Regression. We can avoid a warning by specifying a solver (= algorithm) to use in the optimization problem for solving the gradient descent problem (Lievens, 2018). We choose for instance solver='newton-cg' (see *5. Titanic logistic regression - onehotencoded.ipynb*).



```
In [12]:  from sklearn.linear_model import LogisticRegression

          model = LogisticRegression(solver='newton-cg')
          model.fit(X_train, y_train)

          y_test2 = model.predict(X_test)

          from sklearn.metrics import accuracy_score
          accuracy_score(y_test, y_test2)
Out[12]:  0.8046511627906977
```

3.6.8 Ensemble Classification with Scikit-Learn

It's also quite easy to use an ensemble (or bagging) classifier (see *6. Titanic Ensemble - OneHotEncoded.ipynb*).



```
In [14]:  from sklearn.naive_bayes import GaussianNB
          from sklearn.linear_model import LogisticRegression
          from sklearn.ensemble import RandomForestClassifier, VotingClassifier

          lr = LogisticRegression(solver='newton-cg')
          rf = RandomForestClassifier(n_estimators=150)
          gnb =  GaussianNB()

          model = VotingClassifier(estimators=[('lr', lr), ('rf', rf), ('gnb', gnb)], voting='soft')
          model.fit(X_train, y_train)

          y_test2 = model.predict(X_test)

          from sklearn.metrics import accuracy_score
          accuracy_score(y_test, y_test2)
Out[14]:  0.7767441860465116
```

The voting parameter can be either hard of soft (scikit-learn Machine Learning in Python, sd). If 'hard' it just counts for each instance the number of times each predicted class appears and chooses the class which appears most. If 'soft' it predicts the class label based on the sum of the predicted probabilities of each class. Indeed, the model can also determine probability of the predicted class.

Example:

Suppose part of the test set looks like below and we use four classifiers in the bag: Gaussian Naïve Bayes (GNB), Random Forest with 100 trees (RF100), Random Forest with 150 trees (RF150) and Logistic Regression (LR).

```
   Pclass     Sex   Age  SibSp  Parch
0       3    male  22.0      1      0
1       1  female  38.0      1      0
2       3  female  26.0      0      0
3       1  female  35.0      1      0
4       3    male  35.0      0      0
```

Assume for line 3 GNB predicts 1 (survived), RNF100 and RNF150 both predict 0 and LR predicts 1. In case of voting='hard', there is no winner in the majority voting and a class (0 or 1) will be chosen at random.

In case of voting='soft' we look at the probability of each prediction (which can also be determined by Scikit-Learn). So suppose GNB predicts one with a probability of 0.6, RNF100 predicts 0 with a probability of 0.8 and RNF with a probability of 0.7. LR predicts 1 with a probability of 0.6. The sum of the probabilities for class 1 = 0.6 + 0.6 = 1.2. For class 0 these sum = 0.8 + 0.7 = 1.5. Now we have a clear winner: class 0. In general, soft voting is recommended.

3.7 Exercises

3.7.1 Exercise 1

The Iris data set is one of the best known classification datasets, which is widely referenced in the technical literature. The aim is to classify iris plants into one of three classes on the basis of the values of four categorical attributes.

Information about the dataset:

Source: UCI Repository https://archive.ics.uci.edu/ml/datasets/Iris

Classes

Iris-setosa, Iris-versicolor, Iris-virginica (there are 50 instances in the dataset for each classification)

Attributes and Attribute Values

Four continuous attributes: sepal length, sepal width, petal length and petal width.

Number of instances: 150

Develop a model to predict the class of an iris plant based on the four attributes. Use both Naïve Bayes and Random Forest. Predict the accuracy of both models and draw the confusion matrix. For random forest also find the two most important features.

3.7.2 Exercise 2

Mammography is the most effective method for breast cancer screening available today. However, the low positive predictive value of breast biopsy resulting from mammogram interpretation leads to approximately 70% unnecessary biopsies with benign outcomes. To reduce the high number of unnecessary breast biopsies, several computer-aided diagnosis (CAD) systems have been proposed in the last years. These systems help physicians in their decision to perform a breast biopsy on a suspicious lesion seen in a mammogram or to perform a short term follow-up examination instead. This data set can be used to predict the severity (benign or malignant) of a mammographic mass lesion from BI-RADS attributes and the patient's age. It contains a BI-RADS assessment, the patient's age and three BI-RADS attributes together with the ground truth (the severity field) for 516 benign and 445 malignant masses that have been identified on full field digital mammograms collected at the Institute of Radiology of the University Erlangen-Nuremberg between 2003 and 2006. Each instance has an associated BI-RADS assessment ranging from 1 (definitely benign) to 5 (highly suggestive of malignancy) assigned in a double-review process by physicians. Assuming that all cases with BI-RADS assessments greater or equal a given value (varying from 1 to 5), are malignant and the other cases benign, sensitivities and associated specificities can be calculated. These can be an indication of how well a CAD system performs compared to the radiologists.

Number of Instances: 961

Number of Attributes: 6 (1 goal field, 1 non-predictive, 4 predictive attributes)

Attribute Information:

1. BI-RADS assessment: 1 to 5 (ordinal) = assessment by radiologist:

0- incomplete, 1-negative, 2-benign findings, 3-probably benign, 4-suspicious abnormality, 5-highly suspicious of malignancy

2. Age: patient's age in years (integer)
3. Shape: mass shape: round=1 oval=2 lobular=3 irregular=4 (nominal)
4. Margin: mass margin:

   circumscribed=1 microlobulated=2 obscured=3 ill-defined=4 spiculated=5 (nominal)
5. Density: mass density high=1 iso=2 low=3 fat-containing=4 (ordinal)
6. Severity: benign=0 or malignant=1 (binominal)

Missing Attribute Values: Yes

- BI-RADS assessment:	2
- Age:	5
- Shape:	31
- Margin:	48
- Density:	76
- Severity:	0

Class Distribution: benign: 516; malignant: 445

See https://archive.ics.uci.edu/ml/datasets/Mammographic+Mass for the source of the data set.

1. Calculate how often breast cancer occurs on average in this data set. (0,46…)
2. Calculate how many of the samples are benign and how many are malignant (benign: 516, malign: 445)
3. Missing attributes are indicated as "?". Remove all lines that contain some "?".
4. Calculate the percentage of the BI-RADS assessments that were correct, assuming a BI-RADS assessment of 4 or higher is malign. (51 %)
5. Use random forest classification to determine a model for predicting the severity (benign/malign) of new mammography results. Determine the accuracy of the classifier (+/- 79%)
6. Which two factors are the most determining for malign breast tumors. (Age and margin)
7. Determine the false negative rate and the false positive rate. Which proportion of the real cancer cases are we missing with this test? (+/- 25%)

3.7.3 Exercise 3

The data provided in Demographic Studentscore contains information about individual study results for mathematics, together with demographic information about the student. Solve each subquest in one cell.

3.7.4 Exercise 4

Each row in the file "wifi.csv" contains signal strengths of 7 wifi signals in an office building (columns s1-s7), each measured with a smartphone on a different location (one line per location). The column floor contains the floor of the location.

Create an ensemble model to predict the floor from an unseen set of 7 wifi signal strengths. Combine 4 models: Naïve Bayes, Random Forest with 100 trees, Random Forest with 300 trees and Logistic Regression. Determine the predictive accuracy.

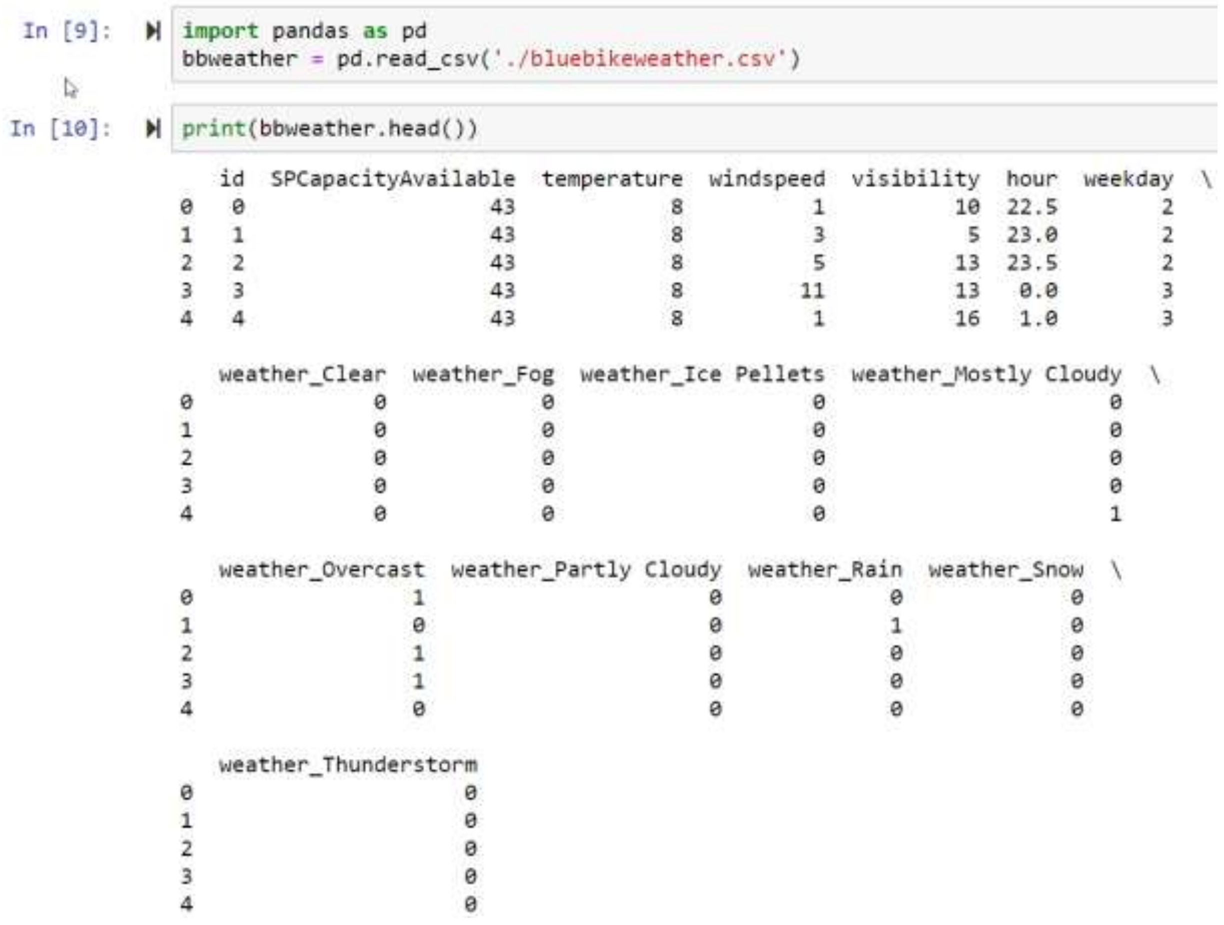
3.8 Regression

3.8.1 Random Forest Regression

If, in supervised learning, the attribute to be predicted is numerical, e.g. the expected sale price of a house or the opening price of a share on tomorrow's stock market, the task is called regression.

In the previous section we considered random forests within the context of classification. Random forests can also be made to work in the case of regression (that is, with continuous rather than categorical variables). The estimator to use for this is the RandomForestRegressor, and the syntax is very similar to what we saw earlier. Under the hood a process known as *discretization* is used to convert a continuous attribute to a categorical one.

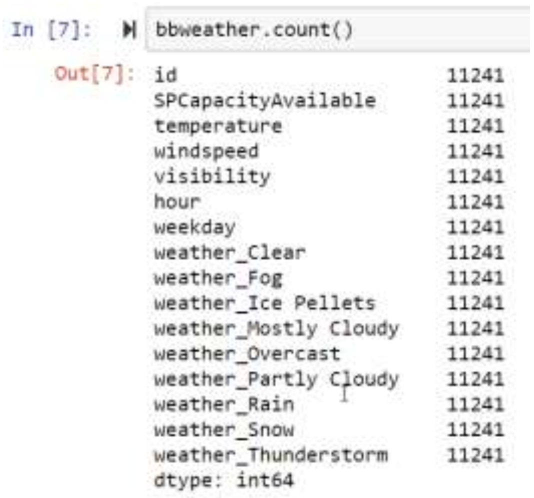
In every major Belgian railway station you can rent a blue-bike. Through a website, a mobile app and an API you can see the real time availability of blue-bikes in the railway station of your choice. The data for the station of Gent Sint-Pieters has been collected every 30 minutes from January till October 2018. The file bluebike.csv contains a (heavily) prepared set of observations of the available blue-bikes in the station of Gent-Sint-Pieters, together with the weather observations at the same moment. Since we want to make a prediction per half hour we keep for each observation only "the hour", for instance for 10.30 pm the field column contains the value 22.5. Weekday coding is as follows: 0 = Monday, 1 = Tuesday, etc. As you can see the "weather" attribute is already hot encoded. See *7. Bluebike - regression - prepared data.ipynb.*



```
In [9]:  import pandas as pd
         bbweather = pd.read_csv('./bluebikeweather.csv')

In [10]: print(bbweather.head())
```

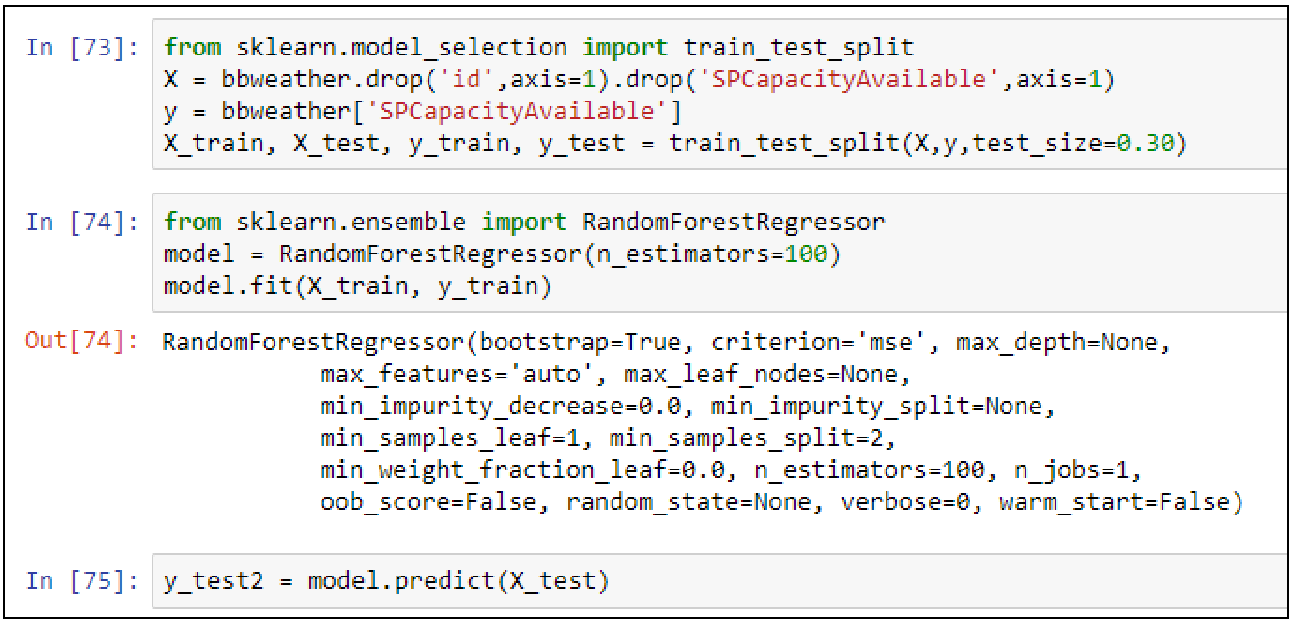
```
   id  SPCapacityAvailable  temperature  windspeed  visibility  hour  weekday  \
0   0                   43            8          1          10  22.5        2
1   1                   43            8          3           5  23.0        2
2   2                   43            8          5          13  23.5        2
3   3                   43            8         11          13   0.0        3
4   4                   43            8          1          16   1.0        3

   weather_Clear  weather_Fog  weather_Ice Pellets  weather_Mostly Cloudy  \
0              0            0                    0                      0
1              0            0                    0                      0
2              0            0                    0                      0
3              0            0                    0                      0
4              0            0                    0                      1

   weather_Overcast  weather_Partly Cloudy  weather_Rain  weather_Snow  \
0                 1                      0             0             0
1                 0                      0             1             0
2                 1                      0             0             0
3                 1                      0             0             0
4                 0                      0             0             0

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

We have more than 11.200 observations:



```
In [7]:    bbweather.count()

Out[7]: id                          11241
        SPCapacityAvailable         11241
        temperature                 11241
        windspeed                   11241
        visibility                  11241
        hour                        11241
        weekday                     11241
        weather_Clear               11241
        weather_Fog                 11241
        weather_Ice Pellets         11241
        weather_Mostly Cloudy       11241
        weather_Overcast            11241
        weather_Partly Cloudy       11241
        weather_Rain                11241
        weather_Snow                11241
        weather_Thunderstorm        11241
        dtype: int64
```

We will now build a model to predict the number of available blue-bikes (column SPCapacityAvailable) based on hour of the day, day of the week and weather predictions. Building and using the model is very similar to the RandomForestClassifier:



```
In [73]: from sklearn.model_selection import train_test_split
         X = bbweather.drop('id',axis=1).drop('SPCapacityAvailable',axis=1)
         y = bbweather['SPCapacityAvailable']
         X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30)

In [74]: from sklearn.ensemble import RandomForestRegressor
         model = RandomForestRegressor(n_estimators=100)
         model.fit(X_train, y_train)

Out[74]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
                    oob_score=False, random_state=None, verbose=0, warm_start=False)

In [75]: y_test2 = model.predict(X_test)
```

It does not make sense to use the `accuracy_score` function when using regression techniques because it checks if a classification is correct or not (Lievens, 2018). Instead we can use some metrics that are also available from the `sklearn`-metrics library. The most commonly used metrics to compare two vectors with continuous values are:

1. Root Mean Square Deviation (RMSD) or Error (RMSE). This is the square root of the mean of the squares of the deviations:

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}}.$$

2. Mean Absolute Error (MAE). This is the average absolute value of the error:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

3. R^2 or R squared. This is a statistic that gives some information about the goodness of fit of a model. In regression, the R^2 coefficient of determination is a statistical measure of how well the regression predictions approximate the real data points. An R^2 of 1 indicates that the regression predictions perfectly fit the data. Values of R^2 outside the range 0 to 1 can occur when the model fits the data worse than a horizontal hyperplane. This means: the closer R^2 is to 1 (but lower than 1) the better our model.

When looking for the best model (by trying different algorithms and different parameter combinations) we have to choose a metric and find the model with the best result for our metric. In our blue-bike case we prefer MAE over RMSD since with RMSD larger absolute deviations (that tend to happen more with larger absolute numbers) get a higher weight (because of the square). However, we are more interested in the lower range: people want an estimate of the available bikes at busy hours, so we choose MAE for comparing classifiers. It is part of the job of a data engineer to reflect about the most appropriate metric for determining the accuracy of the model in the actual application .

```
In [15]: # create dictionary of models
         modeldict = {'modelForest25':RandomForestRegressor(n_estimators=25),\
                      'modelForest50':RandomForestRegressor(n_estimators=50),\
                      'modelForest100':RandomForestRegressor(n_estimators=100),\
                      'modelForest150':RandomForestRegressor(n_estimators=150),\
                      'modelForest200':RandomForestRegressor(n_estimators=200),\
                      'modelForest250':RandomForestRegressor(n_estimators=250)}
         # initialize MAE by choosing a high value
         MAE = 100000
         # initialize bestmodel
         bestmodel = 'modelForest25'

         for modelkey in modeldict:
             model = modeldict[modelkey]

             model.fit(X_train,y_train)

             y_predict = model.predict(X_test)

             NEWMAE = mean_absolute_error(y_test,y_predict)
             if newmae < MAE:
                 MAE = NEWMAE
                 bestmodel = modelkey

         print('Bestmodel: ' + modelkey)
         print('Mean Absolute Error: '+ str(MAE))
         r2 = r2_score(y_test,y_predict)
         print('R square: ' + str(r2))
```

```
Bestmodel: modelForest250
Mean Absolute Error: 4.857218706911562
R square: 0.6833659645846072
```

We learn that the more trees the better the prediction (at least till 250 trees). Our predictions from the best model have an error (mean absolute value of deviation) of less than 5 bikes. We could try to further improve the model by e.g. hot encoding the weekday or adding extra features, for example “holiday (y/n)”.

RandomForestRegression is only one example of regression. Many more algorithms exist in Python for estimating and predicting continuous variables.

3.8.2 Exercise

De file parking.csv contains the available parking spots for six public parkings in Gent between October 24, 2018 and November 10, 2018.

Create a model, based on day of the week, time of the day (rounded to half hours: 0, 0.5, 1, 1.5, …) and school holiday (y/n) for the parking “P01 Vrijdagmarkt”.

a) Some LastModified dates are duplicate. How should you handle that?
b) You can find the list of school holidays in file holiday.csv. Merge it with the parking file to obtain an integrated dataframe.
c) Use one-hot encoding for the day of the week.
d) Use RandomForest and determine the optimal number of trees (25, 50, 75, 100, 125 or 150) according to the MAE mesure.

4 Sources

Attuchirayil, S. (2016, 2 10). *Learn Pandas by comparing with SQL*. Retrieved from https://medium.com/@sajuas/learn-pandas-by-comparing-with-sql-29ad113aed75

Bramer, M. (2016). *Principles of Data Mining.* Springer.

Lemahieu, W., Vanden Broucke, S., & Baesens, B. (2018). *Principles of Database Management.* Cambridge: Cambridge University Press.

Lievens, S. (2018). *Artificiële Intelligentie, lesnota's.* HOGENT.

scikit-learn Machine Learning in Python. (n.d.). Retrieved from http://scikit-learn.org/stable/

VanderPlas, J. (2016). *Python Data Science Handbook.* Sebastopol: O'Reilly Media.