Management of Multimedia Information in Internet

Module 5. Natural Language Processing (NLP)

Unit 3. Statistical NLP

Boni García

http://bonigarcia.github.io/
boni.garcia@uc3m.es

Telematic Engineering Department
School of Engineering

2020/2021

uc3m | Universidad Carlos III de Madrid

Table of contents

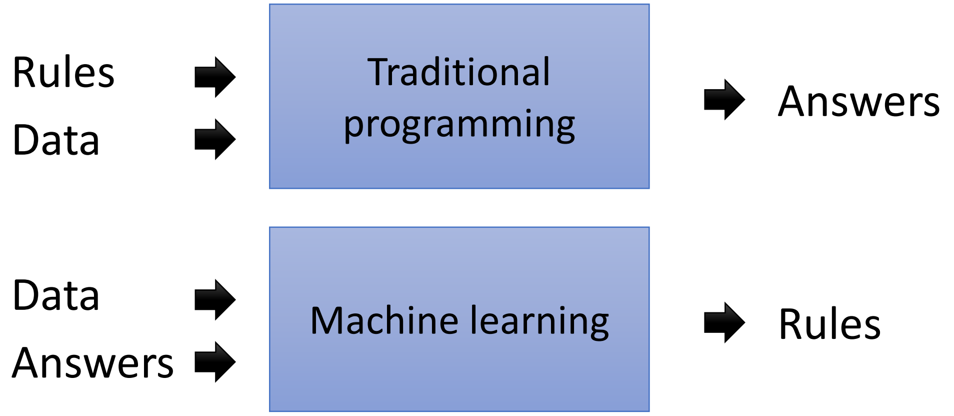
1. Introduction

- In the late 1980s and mid 1990s, there was a revolution in NLP with the introduction of **Machine Learning** (ML) algorithms for language processing (this is called **statistical NLP**)
- By analysing large samples of real-world texts (**corpora**), a computer system can learn and develop its own linguistic rules that it will use to analyse future input
- To execute ML algorithms in NLP applications, first we need to convert the raw text to some convenient mathematical **text representation** (e.g. bags of words, vector space model)
- In NLP, its is very relevant the case of **text classification**, i.e. choosing the correct label for a given text document (e.g. spam detection, sentiment analysis, etc.)

Table of contents

2. Machine learning

- **Machine learning** (ML) is a subset of Artificial Intelligence (AI) which aims to build systems that are capable of improving automatically (without being explicitly programmed to do so) through **experience**
  - ML arises from this question: *Could a computer automatically learn rules by looking at data rather than programmers create rules by hand?*

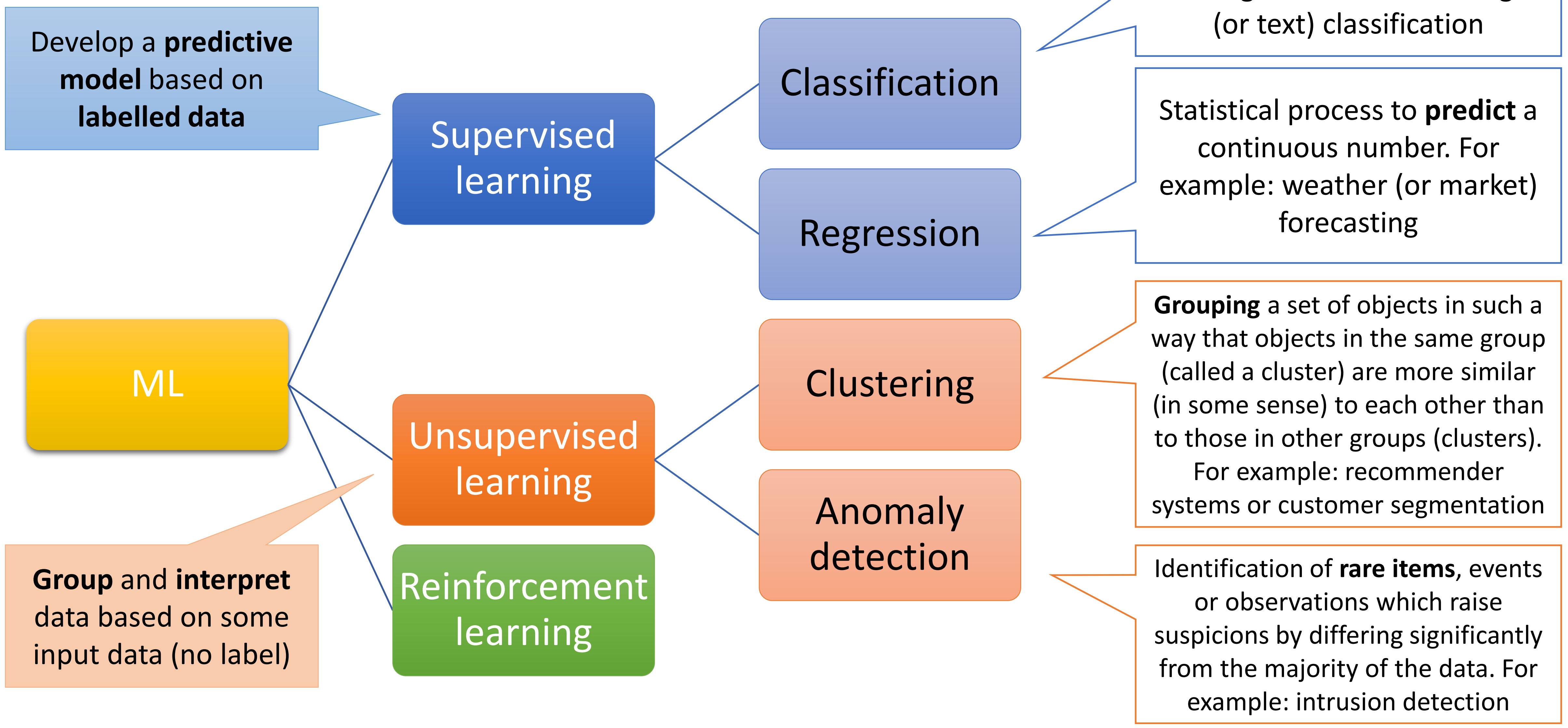


2. Machine learning

- The main categories for ML **algorithms** are:

- **Supervised learning**: It is a technique to deduce a function (called predictive model) from **training data**. These data are pairs of objects (**label**-value). In these algorithms, an expert labels the data (usually called teacher)
- **Unsupervised learning**: The goal can be discovering hidden patterns in data. No labels are given to the learning algorithm, leaving it on its own to find structure in its input. No teacher to instruct the learning algorithm
- **Reinforcement learning**: The algorithm interacts with a dynamic environment a number of times and receives feedback depending on the action. Consequently, the algorithm updates its strategy

2. Machine learning



Develop a **predictive model** based on **labelled data**

Group and **interpret** data based on some input data (no label)

To identify to which of a set of **categories** a observation belongs. For instance: image (or text) classification

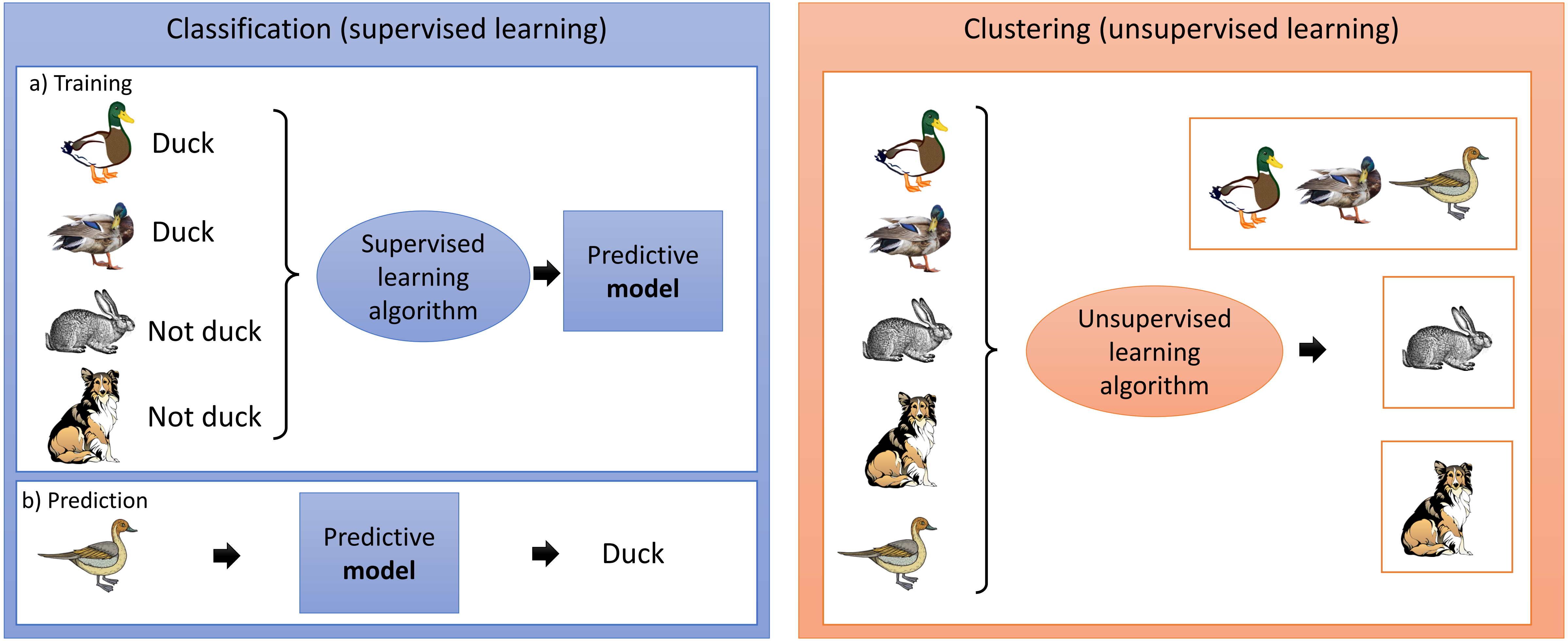
Statistical process to **predict** a continuous number. For example: weather (or market) forecasting

Grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). For example: recommender systems or customer segmentation

Identification of **rare items**, events or observations which raise suspicions by differing significantly from the majority of the data. For example: intrusion detection

2. Machine learning

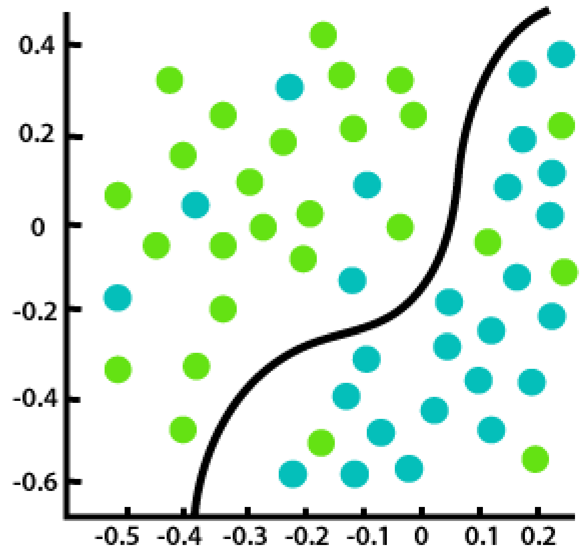
- Classification vs clustering (supervised vs unsupervised learning):


Classification (supervised learning)
a) Training
Duck
Duck
Not duck
Not duck
Supervised learning algorithm
Predictive model
b) Prediction
Predictive model
Duck
Clustering (unsupervised learning)
Unsupervised learning algorithm

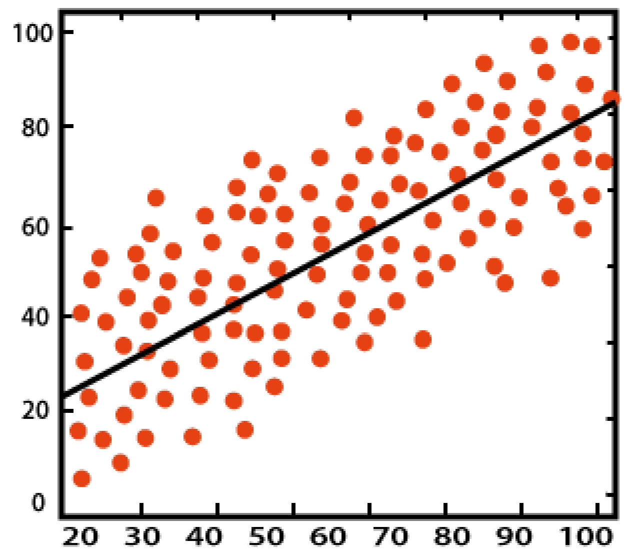
2. Machine learning

- Classification vs regression (supervised learning):

Classification: process of finding a function which helps in dividing the dataset into classes based on different parameters



Classification



Regression

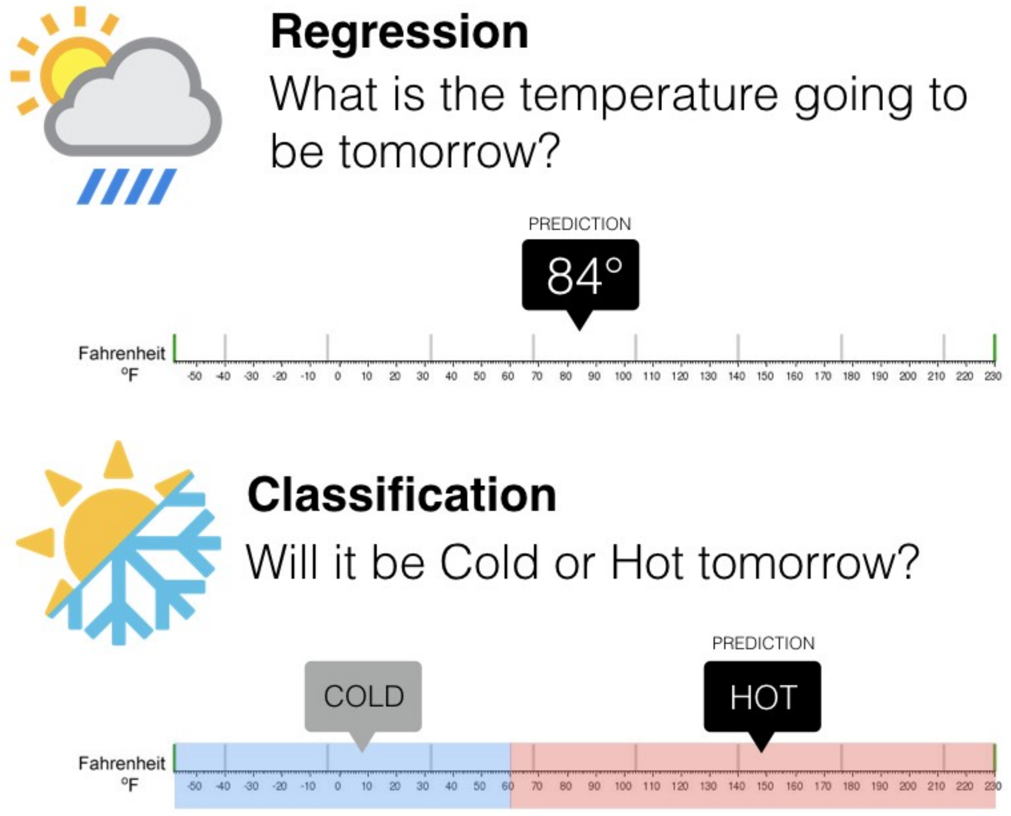
Regression: process of finding a function most closely approximates the input data. There are different regression approaches, e.g.:
- Simple linear regression: using lines (i.e., traditional slope-intercept functions, y=mx+b) to produce predictions
- Multiple linear regression: using multi-variable linear equations (e.g. planes)

Source: https://www.javatpoint.com/regression-vs-classification-in-machine-learning

2. Machine learning

- Classification vs regression (supervised learning):

Input variables in ML algorithms are known as **features** (individual measurable property or characteristic of a phenomenon being observed)



The main difference between them is that the output variable in regression is numerical (**continuous**) while that for classification is categorical (**discrete**)

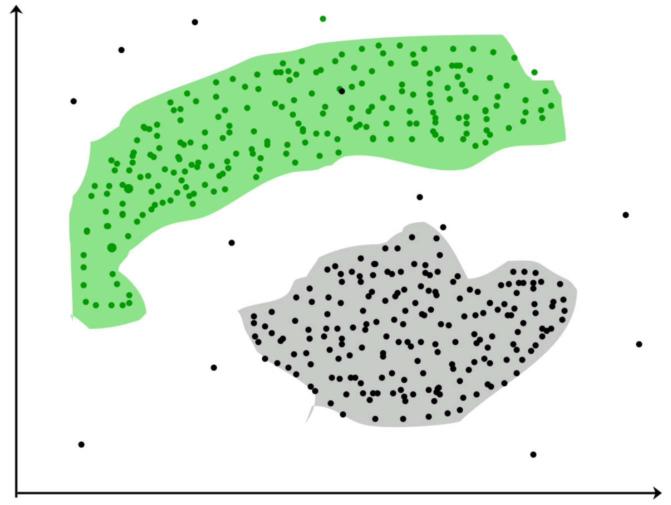
Source: https://towardsdatascience.com/regression-or-classification-linear-or-logistic-f093e8757b9c

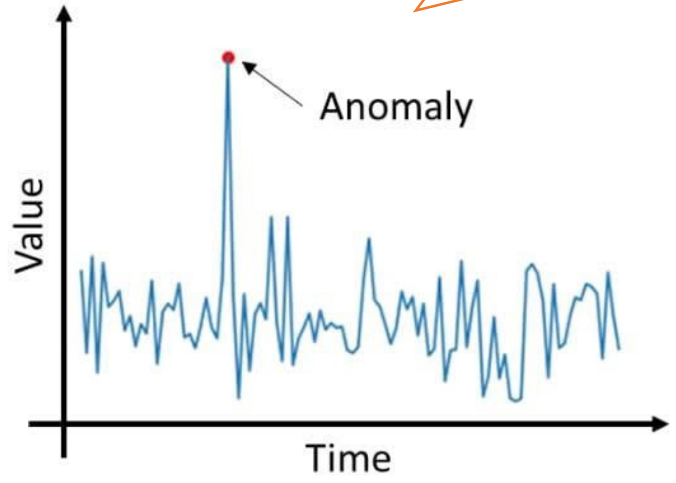
2. Machine learning

- Clustering vs anomaly detection (unsupervised learning):

Clustering: dividing the input data points into a number of groups such that data points in the same groups are more similar

Anomaly detection: identifying data in a observation that differs majorly from the rest of the data





Source: https://www.geeksforgeeks.org/clustering-in-machine-learning/

Source: https://medium.com/datadriveninvestor/how-machine-learning-can-enable-anomaly-detection-eed9286c5306

Table of contents

3. Classification

- Classification is a type of supervised learning approach in which we try to identify to which of a set of categories a observation belongs
- The goal is to predict a class **label**, which is a choice from a predefined list of possibilities
  - In NLP, it is very relevant the case of **text classification**, i.e. choosing the correct label for a given text document
- Classification is sometimes separated into:
  - Binary classification, which is the special case of distinguishing between exactly two classes (for instance, in NLP, decide whether an email is spam or not)
  - Multiclass classification, which is classification between more than two classes (for instance, in NLP, deciding what the topic of a news article is from a fixed list of topics such as “sports”, “technology”, or “politics”)

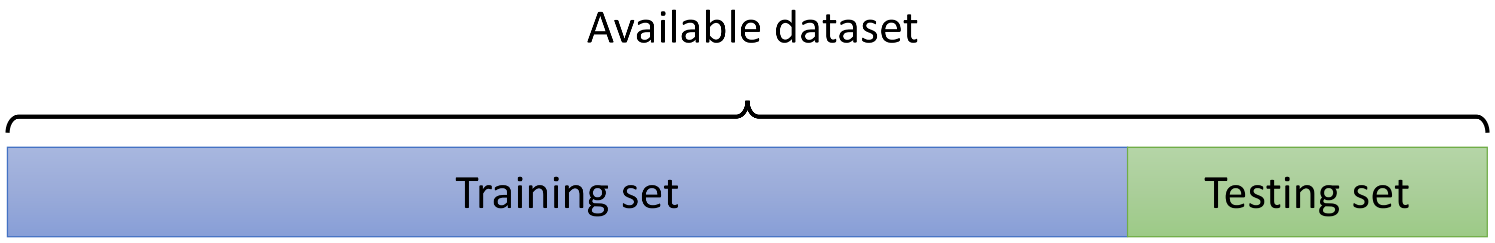
3. Classification - Prediction model

- In classification, we want to build a **prediction model** using some **training data** and then be able to make accurate predictions on new, **unseen data** that has the same characteristics as the training set that we used
  - The best analogy is to think of the prediction model (or simply, model) as a program (or class)
- We want to build a model that is able to generalize as accurately as possible. For that, we will need to **evaluate** our model
  - If a model is able to make accurate predictions on unseen data, we say it is able to generalize from the **training set** to the **test set**

3. Classification - Model evaluation

- To measure if a prediction model is good enough (i.e. its accuracy), we can use a method called **train/test**
- The procedure to carry out train/test is:

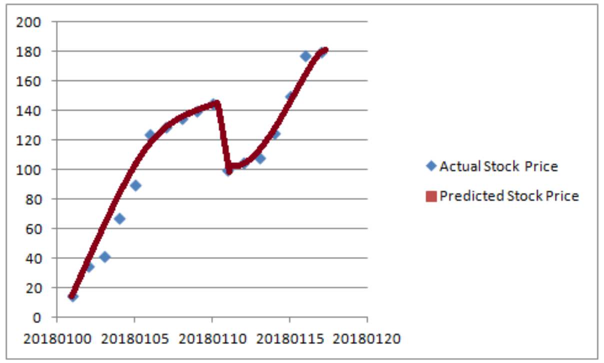
1. We split our dataset into two parts: the **training set** (e.g. 80% of the dataset) and the **testing set** (e.g. the remaining 20% of the dataset)
2. We **fit the model** (i.e., create the model) using the **training set**
3. We **test the model** (i.e., check accuracy of the model) using the **testing set**



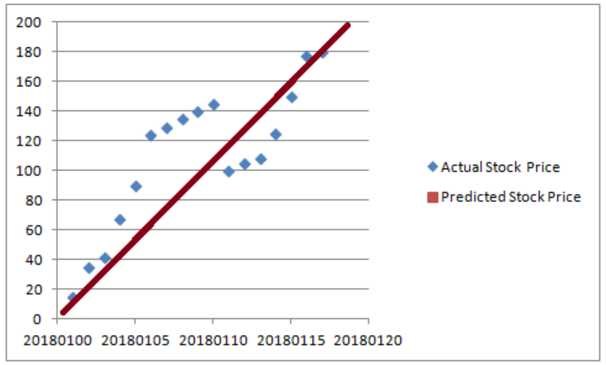
3. Classification - Model evaluation

- Two typical problems can happen in our predictive model:

1. **Overfitting**. The model has learned the training data so well that it cannot generalize well to make predictions on new and unseen data. These models are not good for predicting new data



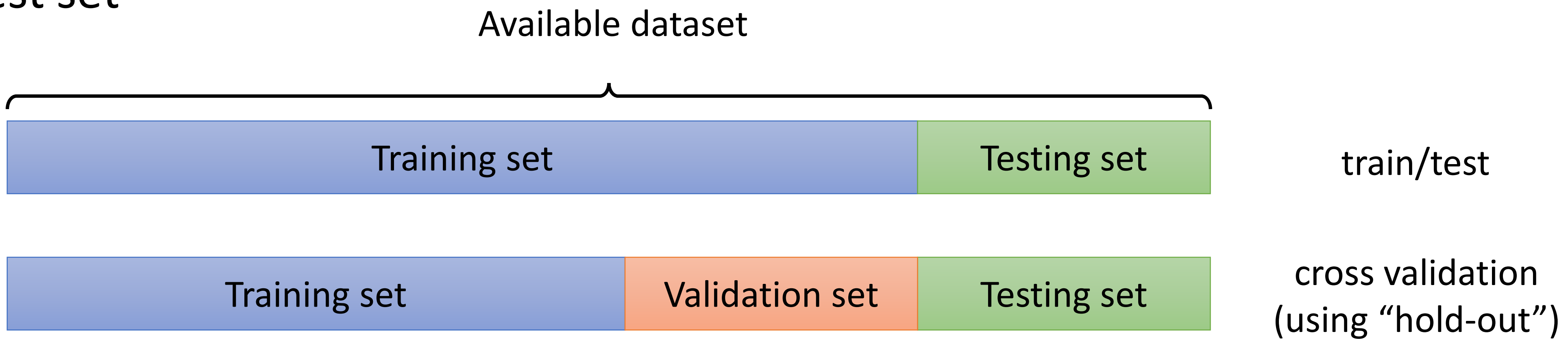
2. **Underfitting**. The model does not fit the training data and therefore misses the trends in the data. It implies that it has high-bias, and therefore means that the model cannot be generalized to new data



Source: https://medium.com/fintechexplained/the-problem-of-overfitting-and-how-to-resolve-it-1eb9456b1dfd

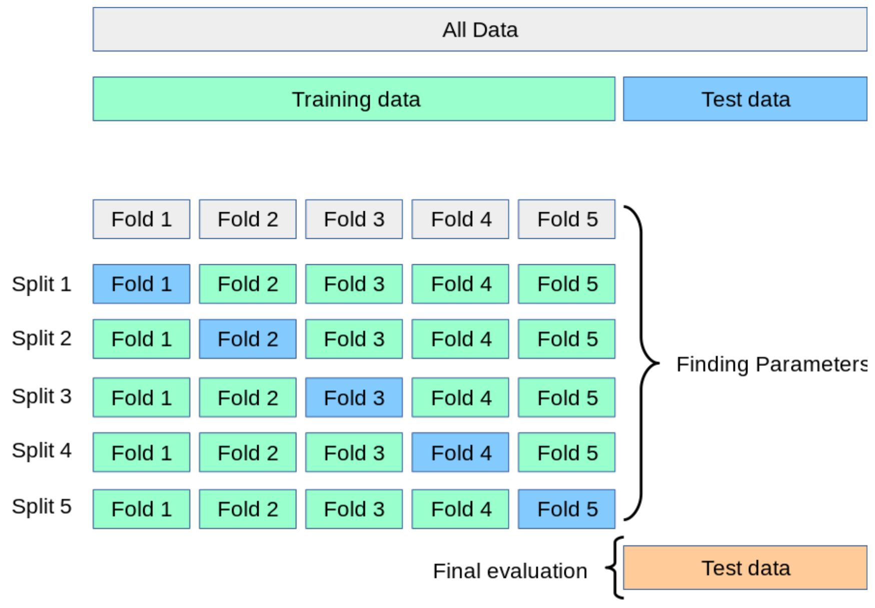
3. Classification - Model evaluation

- To achieve a better evaluation of our model and try to avoid these problems, we can made **cross validation**
- In cross validation, yet another part of the dataset can be held out as a so-called **validation set**:
  - Training proceeds on the training set
  - Evaluation is done on the validation set
  - When the experiment seems to be successful, final evaluation is done on the test set



3. Classification - Model evaluation

- If out dataset is not large enough to carry out hold-out, we can make cross-validation using the a popular technique called **K-fold**, which consists of:
  - We split the training data into k equally sized sets (called **folds**). For instance, k=5 represents 80% or the data is used for training and the rest 20% for is used for validation
  - The model is trained using k−1 of the folds as training data
  - Then, the model is validated on the remaining data (i.e., it is used as a test set to compute a performance measure such as accuracy)
  - The performance measure reported by *k*-fold cross-validation is then the average of the values computed in the loop



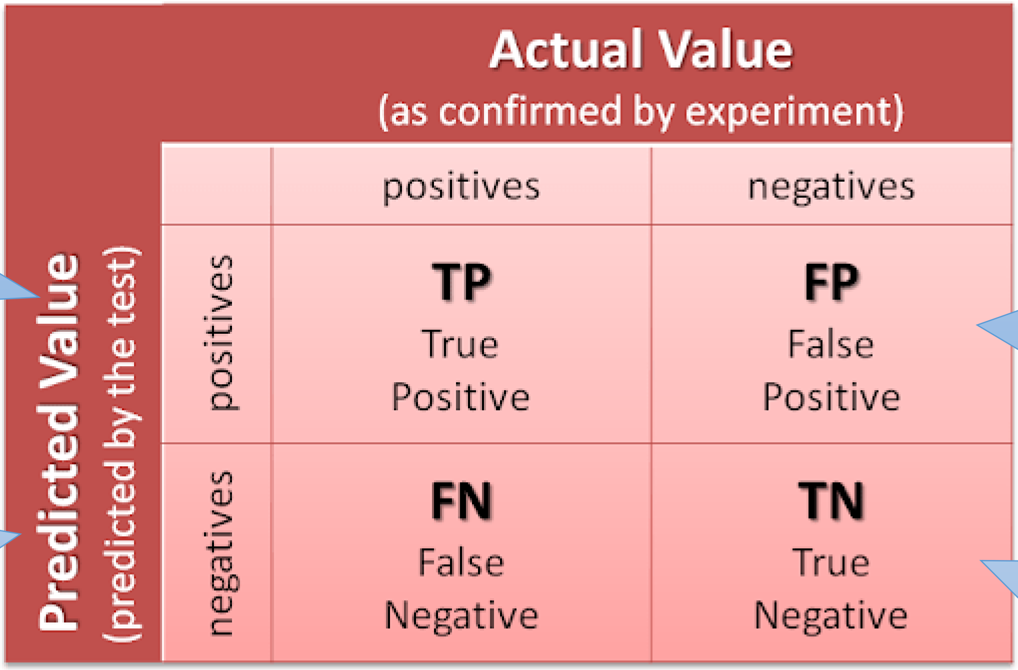
Source:
https://scikit-learn.org/stable/modules/cross_validation.html

3. Classification - Metrics

- A **confusion matrix** is a table that we use to understand the performance of a classification model

TP (**True Positive**): The classifier predicted positive and it is (e.g. an email is predicted as ham and it is)

FN (**False Negative**): The classifier predicted negative but it is not (e.g. an email is predicted as ham but it is not)



		Actual Value (as confirmed by experiment)	
		positives	negatives
Predicted Value (predicted by the test)	positives	**TP** True Positive	**FP** False Positive
	negatives	**FN** False Negative	**TN** True Negative

FP (**False Positive**): The classifier predicted positive but it is not (e.g. an email is predicted as spam but it is not)

TN (**True Negative**): The classifier predicted negative and it is (e.g. an email is predicted as spam and it is)

Source: https://alearningaday.blog/2016/09/14/confusion-matrix/

3. Classification - Metrics

- Some of the metrics that can be derived from the confusion matrix are:

- **Precision**: ratio of the correct positive predictions (TP) divided by the total number of positive experiments (TP + FP)

precision = TP / (TP + FP)

- **Recall** (also know as "sensitivity"): ratio of the correct positive predictions (TP) divided by the total of positive predictions (TP + FN)

recall = TP / (TP + FN)

- **F1 score**: harmonic mean of precision and recall

*F1 score = 2 * precision * recall / (precision + recall)*

3. Classification - Algorithms

- An **algorithm** in ML is a procedure that is run on data to create a prediction model
  - The model represents what was learned by a ML algorithm
- Some supervised learning algorithms used to create classifiers are:
  - Naive Bayes
  - Support Vector Machines (SVMs)
  - Decision Trees
  - K-nearest neighbors (kNN)

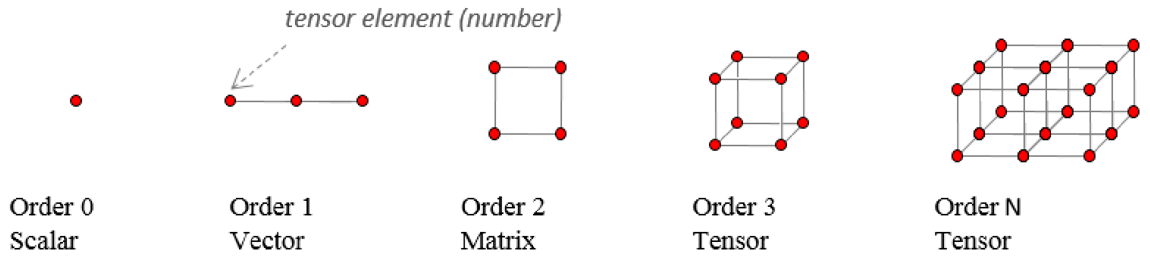
Table of contents

4. Text representation

- ML algorithms cannot work with raw text directly, and therefore, the text must be converted into some mathematical data structures
- To transform text into numbers, we typically use the following mathematical constructs:
  - **Scalars**: Single numbers
  - **Vectors**: One-dimensional array of numbers
  - **Matrices**: Two-dimensional arrays of numbers
  - **Tensors**: N-dimensional arrays of numbers

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$A = \begin{bmatrix} x_{11} & z_{12} \\ x_{21} & z_{22} \\ x_{31} & z_{32} \end{bmatrix}$$



Source: Thanaki, J. (2017) ***Python Natural Language Processing***. Packt Publishing.

4. Text representation

- From a given **corpus** (i.e. a list of **documents**), we can extract a **vocabulary** (i.e. unique set of words)
  - We can carry out a text processing pipeline to refine the vocabulary (or at least, remove the stop words)
- Each word in the vocabulary are uses as **features** (individual measurable properties) in our ML model
- Our objective is to represent each document as a **feature vector** (i.e. one dimensional array of numbers). The whole corpus is represented as **feature matrix**
- We are going to use two approaches for vectorize documents:
  - Bag of words (**BoW**): Counting the **occurrence** of each word in a corpus
  - **TF-IDF**: Counting the **importance** of each word in a corpus

4. Text representation - Bag of words

- The **Bag-of-words** (**BoW**) model is the simplest form of text representation in numbers
- It is a vector representation of text which captures the word occurrence **frequencies** in a text
  - Each of the vector dimensions captures frequency
- It is called a “*bag*” of words, because any information about the order or structure of words in the document is discarded
  - The BoW model is commonly used in classification methods where the occurrence of each word is used as a feature for training a classifier

4. Text representation - Bag of words

- Example of BoW. Supposing a corpus composed by 2 documents:
  - **Doc A:** *Mercury is the planet that is closest to the Sun. Mercury is the smallest planet.*
  - **Doc B:** *The warmest planet in the solar system is Mercury. Mercury is very close to the Sun.*
  - **Vocabulary**: mercury, is, planet, close, sun, small, warm, solar, system

	mercury	is	planet	close	sun	small	warm	solar	system
Doc A	2	3	2	1	1	1	0	0	0
Doc B	2	2	1	1	1	0	1	1	1

Doc A: (2, 3, 2, 1, 1, 1, 0, 0, 0)
Doc B: (2, 2, 1, 1, 1, 0, 1, 1, 1)

4. Text representation - Bag of words

- **`CountVectorizer`** is a class provided by **scikit-learn** that allows us to calculate the BoW matrix of a given corpus
  - It can be instantiated using a **stop word list** in a given language (to remove these stop words from the vocabulary)
- The main methods in `CountVectorizer` are:
  - `fit(raw_documents)` : learn vocabulary from the raw documents
  - `transform(raw_documents)` : calculate BoW matrix
  - `fit_transform(raw_documents)` : learn vocabulary and calculate BoW matrix

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

4. Text representation - Bag of words

- Basic example using BoW:

```python
from sklearn.feature_extraction.text import CountVectorizer

corpus = ["Computers can analyze text", "They do it using vectors and matrices",
 "Computers can process massive amounts of text data"]

vectorizer = CountVectorizer(stop_words="english")
X = vectorizer.fit_transform(corpus)

print("Vocabulary", vectorizer.vocabulary_)
print("Feature names", vectorizer.get_feature_names())
print("BoW matrix", X.toarray())
print("BoW matrix shape", X.shape)
```

```
Vocabulary {'computers': 2, 'analyze': 1, 'text': 7, 'using': 8, 'vectors': 9,
'matrices': 5, 'process': 6, 'massive': 4, 'amounts': 0, 'data': 3}
Feature names ['amounts', 'analyze', 'computers', 'data', 'massive', 'matrices',
'process', 'text', 'using', 'vectors']
BoW matrix [[0 1 1 0 0 0 0 1 0 0]
 [0 0 0 0 0 1 0 0 1 1]
 [1 0 1 1 1 0 1 1 0 0]]
BoW matrix shape (3, 10)
```

We typically use the variable name X for features

Each entry in the BoW matrix corresponds to the count of a particular term (feature) in the documents

Terms with a higher score in the BoW matrix, are more frequent in the corpus

4. Text representation - TF-IDF

- The main drawback of the BoW model is that we do retain information on the grammar of the sentences nor on the words ordering
- As a solution, the **TF-IDF** (Term Frequency-Inverse Document Frequency) model has been proposed
- TF-IDF is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus
  - The TF-IDF model allows to convert the textual representation of information into a **Vector Space Model** (VSM)
  - VSM is an algebraic model representing textual information as a vector (each document within a corpus is represented as a point in space)

4. Text representation - TF-IDF

- TF-IDF is calculates as:
  - Term Frequency (TF) is a measure of how frequently a term (t) appears in a document (d)
  - Inverse Document Frequency (IDF) is a measure of how important a term is
  - TF-IDF is calculated as the matrix multiplication of TF and IDF

$$tf_{t,d} = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$idf_{t,d} = log\left(\frac{total\ number\ of\ documents\ in\ corpus}{number\ of\ documents\ which\ contains\ t}\right)$$

$$tf\_idf_{t,d} = tf_{t,d} \times idf_{t,d}$$

Fork me on GitHub

4. Text representation - TF-IDF

- **`TfidfVectorizer`** is a class provided by scikit-learn that allows us to calculate the TF-IDF matrix for a given corpus
  - The methods `fit()`, `transform()`, and `fit_transform()` are also available in this class (just like in `CountVectorizer`)

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words="english")
X = vectorizer.fit_transform(corpus)

print("Corpus", corpus)
print("Feature names", vectorizer.get_feature_names())
print("TF-IDF matrix", X.toarray())
print("TF-IDF matrix shape", X.shape)
```

```
Corpus ['Computers can analyze text', 'They do it using vectors and matrices', 'Computers can process massive amounts of
text data']
Feature names ['amounts', 'analyze', 'computers', 'data', 'massive', 'matrices', 'process', 'text', 'using', 'vectors']
TF-IDF matrix [[0.         0.68091856 0.51785612 0.         0.         0.
  0.         0.51785612 0.         0.        ]
 [0.         0.         0.         0.         0.         0.57735027
  0.         0.         0.57735027 0.57735027]
 [0.44036207 0.         0.3349067  0.44036207 0.44036207 0.
  0.44036207 0.3349067  0.         0.        ]]
TF-IDF matrix shape (3, 10)
```

Terms with a higher score in the TF-IDF matrix (normalized from 0 to 1), are considered more important

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html

4. Text representation - Cosine similarity

- **Cosine similarity** is a metric used to measure how similar the documents are
  - Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space
  - Cosine similarity is a metric irrespective of documents size (the cosine of the angle of these 2 vectors compare the angle, not the modules)
- The higher cosine similarity, the higher similarity between documents. Or what is the same: the smaller the separation angle, the higher the similarity
  - **Maximal similarity** if they are parallel (angle 0°, cosine similarity = 1)
  - **Maximal difference** if they are perpendicular (angle 90°, cosine similarity = 0

$$\cos\theta = \frac{v_1 \cdot v_2}{\|v_1\|\,\|v_2\|}$$

Fork me on GitHub

4. Text representation - Cosine similarity

- The function **`cosine_similarity`** in scikit-learn allows to calculate the cosine similarity
- We can use **TF-IDF** matrices as input of the cosine similarity function

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

vectorizer = TfidfVectorizer(stop_words="english")
X = vectorizer.fit_transform(corpus)

similarity_matrix = cosine_similarity(X)

print(corpus)
print(similarity_matrix)
```

```
['Computers can analyze text', 'They do it using vectors and matrices',
'Computers can process massive amounts of text data']
[[1.         0.         0.34686697]
 [0.         1.         0.        ]
 [0.34686697 0.         1.        ]]
```

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html

4. Text representation - Cosine similarity

- Example which calculate the most similar sentence in two corpora:

```
import numpy as np

corpus_2 = ["Computers can store data"]

X_2 = vectorizer.transform(corpus_2)
similarity_matrix = cosine_similarity(X_2, X)

print("Comparing", corpus_2, "and", corpus)
print("The similaryty matrix is", similarity_matrix)

max_similary_value = np.amax(similarity_matrix)
max_similary_angle = np.rad2deg(np.arccos(max_similary_value))
max_similary_index = np.argmax(similarity_matrix)

print(f"The most similar sentence in both corpora are (angle {max_similary_angle}°)")
print(corpus[max_similary_index])
print(corpus_2[0])
```

```
Comparing ['Computers can store data'] and ['Computers can analyze text', 'They do it using
vectors and matrices', 'Computers can process massive amounts of text data']
The similaryty matrix is [[0.31348343 0.         0.5532461 ]]
The most similar sentence in both corpora are (angle 56.410004560487295°)
Computers can process massive amounts of text data
Computers can store data
```

We reuse the features (vocabulary) from the first corpus

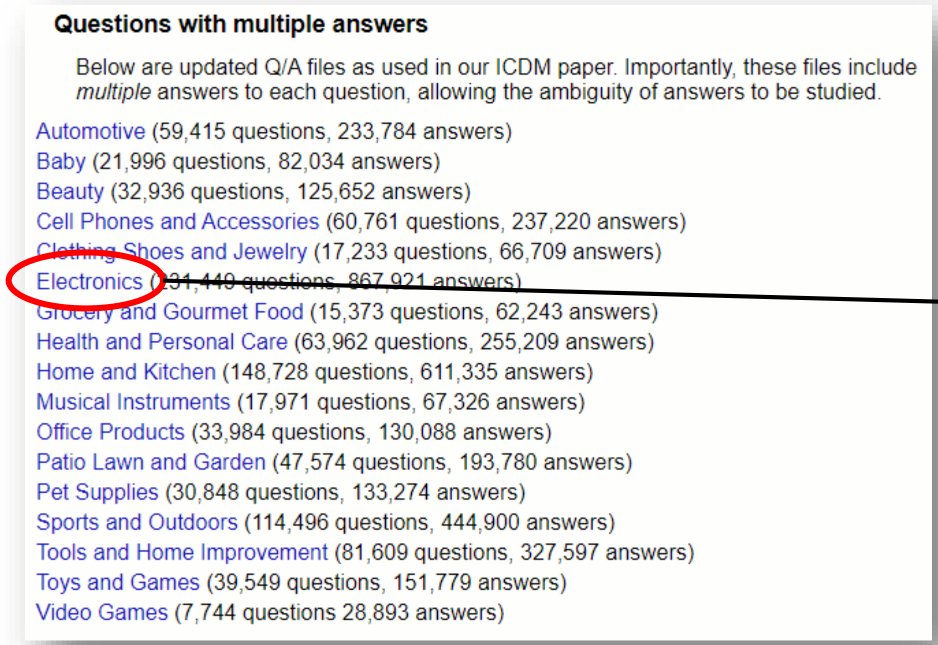
Internally, scikit-learn uses **Numpy** arrays to store vectors and matrices

The function `amax` of Numpy returns the maximum value along an array

The function `argmax` of Numpy returns the indices of the maximum values along an array

4. Text representation - Example: chatbot

- The following example is basic example of **chatbot**
- As corpus, we use we real Amazon's Q&A data, public available at http://jmcauley.ucsd.edu/data/amazon/qa/
- In particular, we use Q&A about electronics (data in JSON)



Questions with multiple answers

Below are updated Q/A files as used in our ICDM paper. Importantly, these files include *multiple* answers to each question, allowing the ambiguity of answers to be studied.

Automotive (59,415 questions, 233,784 answers)
Baby (21,996 questions, 82,034 answers)
Beauty (32,936 questions, 125,652 answers)
Cell Phones and Accessories (60,761 questions, 237,220 answers)
Clothing Shoes and Jewelry (17,233 questions, 66,709 answers)
Electronics (231,449 questions, 867,921 answers)
Grocery and Gourmet Food (15,373 questions, 62,243 answers)
Health and Personal Care (63,962 questions, 255,209 answers)
Home and Kitchen (148,728 questions, 611,335 answers)
Musical Instruments (17,971 questions, 67,326 answers)
Office Products (33,984 questions, 130,088 answers)
Patio Lawn and Garden (47,574 questions, 193,780 answers)
Pet Supplies (30,848 questions, 133,274 answers)
Sports and Outdoors (114,496 questions, 444,900 answers)
Tools and Home Improvement (81,609 questions, 327,597 answers)
Toys and Games (39,549 questions, 151,779 answers)
Video Games (7,744 questions 28,893 answers)

```
{
   "questionType":"yes/no",
   "asin":"0594033926",
   "answerTime":"Dec 27, 2013",
   "unixTime":1388131200,
   "question":"Is this cover the one that fits the old nook
color? Which I believe is 8x5.",
   "answerType":"Y",
   "answer":"Yes this fits both the nook color and the same-
shaped nook tablet"
}
{
   "questionType":"yes/no",
   "asin":"0594033926",
   "answerTime":"Jan 5, 2015",
   "unixTime":1420444800,
   "question":"Does it fit Nook GlowLight?",
   "answerType":"N",
   "answer":"No. The nook color or color tablet"
}
```

Fork me on GitHub

4. Text representation - Example: chatbot

- First, we need to download the JSON data and store it in our Google Drive (to be loaded in Colab)

```
import ast
from google.colab import drive

questions = []
answers = []
drive.mount('/content/drive')

with open('/content/drive/My Drive/qa_Electronics.json') as f:
   for line in f:
     data = ast.literal_eval(line)
     questions.append(data['question'].lower())
     answers.append(data['answer'].lower())
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

4. Text representation - Example: chatbot

```python
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

vectorizer = TfidfVectorizer(stop_words="english")
X_questions = vectorizer.fit_transform(questions)

def conversation(user_input):
  global vectorizer, answers, X_questions

  X_user_input = vectorizer.transform(user_input)
  similarity_matrix = cosine_similarity(X_user_input, X_questions)
  max_similarity = np.amax(similarity_matrix)
  angle = np.rad2deg(np.arccos(max_similarity))

  if angle > 60:
    return "sorry, I did not quite understand that"
  else:
    index_max_similarity = np.argmax(similarity_matrix)
    return answers[index_max_similarity]

def main():
  usr = input("Please enter your username: ")
  print("Q&A support: Hi, welcome to Q&A support. How can I help you?")
  while True:
    user_input = input("{}: ".format(usr))
    if user_input.lower() == "bye":
      print("Q&A support: bye!")
      break
    else:
      print("Q&A support: " + conversation([user_input]))
```

The logic of our chatbot is defined in two Python functions

Table of contents

5. Text classification

- **Text classification** (supervised learning) is a relevant use case in NLP
- Classification problems can be divided into different types according to the cardinality of the labels per document
  - Binary: only two categories exist and they are mutually exclusive. For example, spam detection (a message is considered “spam” or “ham”)
  - Multi-class: multiple categories which are mutually exclusive. For example, news that are considered as “sports”, “economics”, “technology”, etc.
  - Multi-label: multiple categories with the possibility of multiple (or none) assignments. For example, news that are considered “sports” and “economics” at the same time
- There are many different algorithms used in text classification. We are going to study two of them: **Naive Bayes** and **Random Forest**

5. Text classification - Naive Bayes

- **Naive Bayes** is a popular classification algorithm for based on the Bayes' theorem
- The Bayes' theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event. It can be represented as follows:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- Here, A and B are events:
  - *P(A|B)* is the probability of *A* given *B*
  - *P(B|A)* is the probability of *B* given *A*
  - *P(A)* is the independent probability of *A*
  - *P(B)* is the independent probability of B

5. Text classification - Naive Bayes

- Let's suppose we have a Deck of Cards, we wish to find out the "Probability of the Card we picked at random to be a King given that it is a Face Card"
- We can solve this problem using the Bayes' Theorem, as follows:

$$P(King|Face) = \frac{P(Face|King)P(King)}{P(Face)} = \frac{1 \cdot 4/52}{12/52} = 1/3$$

- Where:
  - P(King) = 4/52 (4 Kings out of the 52 total cards)
  - P(Face/King) = 1 (all Kings are face cards)
  - P(Face) = 12/52 (12 cards: Kings, Queens, and Jacks in ♠ ♣ ♥ ♦)

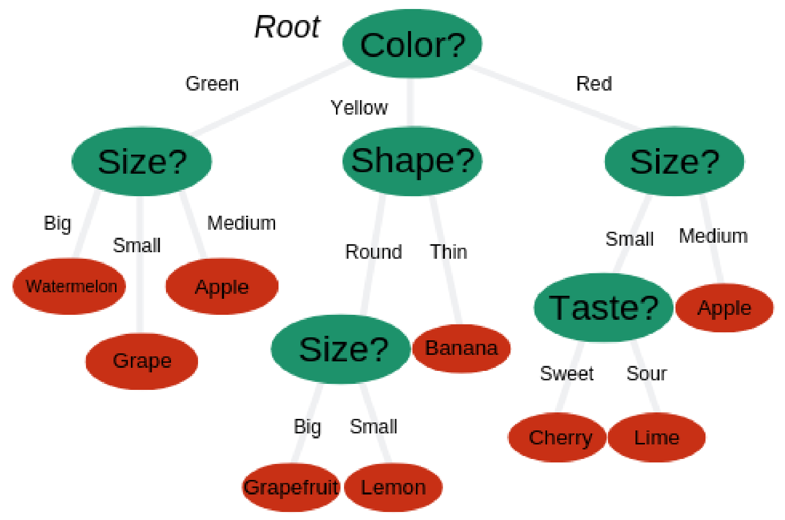
5. Text classification - Naive Bayes

- Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem
- These classifiers are the commonly applied to text classification
- For a text classification, the Naive Bayes theorem can be interpreted as the probability of predicting a target with class **k** given feature matrix **X**, and is given by the probability of predicting feature matrix **X** given a certain class of **y** times the probability of belonging to a certain class **k**

$$P(y = k|X) = \frac{P(X|y = k)P(y = k)}{P(X)}$$

5. Text classification - Random Forest

- **Random Forest** (RF) is a supervised learning ML algorithm that produces, even without hyper-parameter tuning, a great result for **text classification**
  - The RF algorithm is composed of different **decision trees**, each with the same nodes, but using different data that leads to different leaves
  - It merges the decisions of multiple decision trees in order to find an answer, which represents the average of all these decision trees



Source: https://medium.com/capital-one-tech/random-forest-algorithm-for-machine-learning-c4b2c8cc9feb

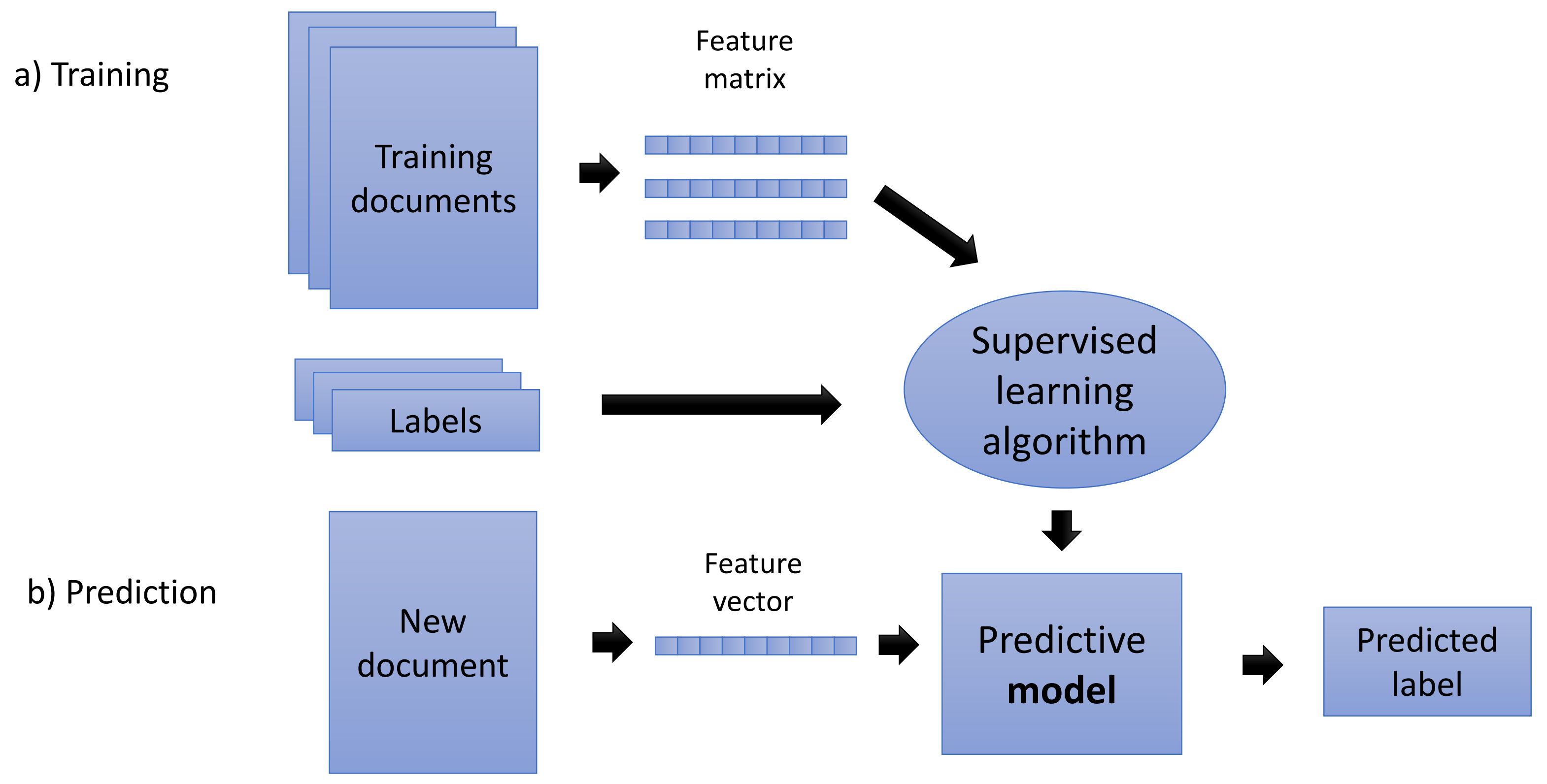
5. Text classification - General workflow

- The general workflow of text classification is the following:

1. Define the problem (e.g. binary or multiclass classification)
2. Select (or create) a **dataset** (or corpus), i.e. a collection of text documents
   - In Python it will be stored a as list of string
3. Generate **vocabulary** and select **labels**
   - Word in the vocabulary are uses as **features**
4. Convert **features** in vectors (e.g. using BoW or TF-IDF matrices)
5. Divide dataset in **training** and **test** sets (also validation set in cross evaluation)
6. Select **algorithm** (e.g. Naive Bayes, Random Forest)
7. Train model (using the training set)
8. **Evaluate** the model (using the test-set if plenty of data, or K-Fold if few samples)
   - Calculate metrics such as: precision, recall, F1-score, accuracy, confusion matrix
9. Use **unseen data** to generate new predictions
10. Serialize model (to store it persistently)

5. Text classification - General workflow

- A simplify version of that workflow is the following:



5. Text classification - scikit-learn

- We use the library **scikit-learn** to implement the previous workflow to implement text classifiers
- In scikit-learn, an estimator for classification is a Python object that implements the methods `fit(X_train, y)` and `predict(X_test)`
  - Here, **X** and **y** contain the features and labels of our classification dataset
- We are going to see the following examples:
  1. Binary classifier: to predict SMS spam (or ham) messages
  2. Multiclass classifier: to predict news categories (business, entertainment, politics, sport, technology)
  3. Sentiment analysis: to predict positive or negative texts

Fork me on GitHub

5. Text classification - scikit-learn

- Binary classifier (spam detector)

We get the dataset for this example from the Department of Telematics, School of Electrical and Computer Engineering at University of Campinas, Brazil
http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/

```
# Dataset

from google.colab import drive
import pandas as pd

drive.mount("/content/drive")

y_names = ["label", "message"]

dataset = pd.read_csv("/content/drive/My Drive/data/sms_spam/SMSSpamCollection.txt", sep="\t", names=y_names)

y = dataset.get(y_names[0]).tolist()
raw_dataset = dataset.get(y_names[1]).tolist()

dataset.head()
```

```
Drive mounted at /content/drive.
 label                                              message
0    ham  Go until jurong point, crazy.. Available only ...
1    ham                        Ok lar... Joking wif u oni...
2   spam  Free entry in 2 a wkly comp to win FA Cup fina...
3    ham  U dun say so early hor... U c already then say...
4    ham  Nah I don't think he goes to usf, he lives aro...
```

The dataset is stored on Google Drive. Then, we use **Pandas** to parse the dataset content

5. Text classification - scikit-learn

- Binary classifier (spam detector)

To create the vocabulary, we implement a simple text processing pipeline composed by: tokenization, stemming, and removing stop words

```python
# Text preprocessing

import nltk
from nltk.tokenize import regexp_tokenize
from nltk.stem.snowball import SnowballStemmer
from nltk.corpus import stopwords
nltk.download("stopwords")

dataset = []
stemmer = SnowballStemmer("english")
stopwords_en = stopwords.words("english")

for i in range(0, len(raw_dataset)):
  tokens = regexp_tokenize(str(raw_dataset[i]), r"\w+")
  stems = [stemmer.stem(token) for token in tokens]
  words_no_stopwords = [word for word in stems if word not in stopwords_en]
  document = ' '.join(words_no_stopwords)
  dataset.append(document)
```

```
Drive mounted at /content/drive.
```

Fork me on GitHub

5. Text classification - scikit-learn

- Binary classifier (spam detector)

```
# Feature extraction (converting text to vectors)

from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(dataset).toarray()
```

In this example, we use **TF-IDF** to create the features matrix

Then, we split the dataset in training set (80%) and test set (20%), shuffling the data before splitting (the seed 0 is used the random split algorithm)

```
# Split training and testing sets

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=0)
```

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

5. Text classification - scikit-learn

- Binary classifier (spam detector)

We selected the **Naive Bayes for multinomial models** algorithm

```
# Train model

from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB()

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

5. Text classification - scikit-learn

- Binary classifier (spam detector)

```
# Model evaluation

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```
[[955   0]
 [ 33 127]]
              precision    recall  f1-score   support

         ham       0.97      1.00      0.98       955
        spam       1.00      0.79      0.89       160

    accuracy                           0.97      1115
   macro avg       0.98      0.90      0.93      1115
weighted avg       0.97      0.97      0.97      1115

0.9704035874439462
```

In this case, we obtain a 97% accuracy score

5. Text classification - scikit-learn

- Binary classifier (spam detector)

Now, we can use the model to predict new unseen data

```
# Predict unseen data

unseen_sentence = input("Enter a message: ")
X_unseen = vectorizer.transform([unseen_sentence]).toarray()
y_unseen = classifier.predict(X_unseen)

print("The pedicted class for that message is:", y_unseen)
```

```
Enter a message: You win the first prize.
The pedicted class for that message is: ['spam']
```

Optionally, we can serialize/deserialize the resulting model (using the Python's module **pickle**)

```
# Model serialization

import pickle

with open("binary_classifier.pickle", "wb") as pickle_file:
  pickle.dump(classifier, pickle_file)

with open("binary_classifier.pickle", "rb") as serialized_model:
  loaded_model = pickle.load(serialized_model)
```

Fork me on GitHub

5. Text classification - scikit-learn

- Multiclass classifier (types of news):

```
# Dataset

from google.colab import drive
from sklearn.datasets import load_files

drive.mount("/content/drive")

# Raw data (BCC article datasets) obtained from the Insight Project
# http://mlg.ucd.ie/datasets/bbc.html
loaded_data = load_files("/content/drive/My Drive/data/bbc")

raw_dataset, y, y_names = loaded_data.data, loaded_data.target, loaded_data.target_names

print("Number of documents in the dataset:", len(raw_dataset))
print("Labels:")
for label in y_names:
  print("\t", label)
```

```
Mounted at /content/drive
Number of documents in the dataset: 2225
Labels (automatically generated from subfolder names):
           business
           entertainment
           politics
           sport
           tech
```

For this example, we got the dataset from the BBC article dataset from the Insight Project: http://mlg.ucd.ie/datasets/bbc.html

We load the data using the function of scikit-learn `load_files`, which individual samples are assumed to be files stored a two levels folder structure

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_files.htm

5. Text classification - scikit-learn

- Multiclass classifier (types of news):

For this example, we use a different algorithm: **Random Forest**

```
# Train model

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

Fork me on GitHub

5. Text classification - scikit-learn

- Sentiment analysis (positive or negative):

Simply changing the dataset, we can convert the example before in a sentiment analysis (using text for positive or negative reviews)

```python
# Dataset

from google.colab import drive
from sklearn.datasets import load_files

drive.mount("/content/drive")

# Raw data (movie reviews) obtained from the Cornell Natural Language Processing Group
# http://www.cs.cornell.edu/people/pabo/movie-review-data/
loaded_data = load_files("/content/drive/My Drive/data/txt_sentoken")

raw_dataset, y, y_names = loaded_data.data, loaded_data.target, loaded_data.target_names

print("Number of documents in the dataset:", len(raw_dataset))
print("Labels (automatically generated from subfolder names):")
for label in y_names:
  print("\t", label)
```

```
Mounted at /content/drive
Number of documents in the dataset: 2000
Labels (automatically generated from subfolder names):
           neg
           pos
```

5. Text classification - Final remarks

- Many other kinds of text classifiers can be done with scikit-learn
- There are many **hyperparameters** that can be set to tweak the performance of the model, but we are choosing the default ones in the previous examples
- We can choose a different vectorizer (e.g. BoW) to try to improve the accuracy of our model
- We can choose alternative algorithms, such as Decision Trees or Support Vector Machine (SVM)
- If our dataset is small, we can use cross-validation (e.g. using K-folds) for the model evaluation

Table of contents

6. Takeaways

- **Statistical NLP** is based on **Machine Learning** (ML)
- ML is about computer algorithms that improve automatically through experience
- **Text classification** is used to implement different NLP applications types, such as binary classifiers (e.g. spam detector), among others
- To implement a text classifier we need a **training set** to fit a model (using a ML algorithm, such Naive Bayes or Random Forest). Then, we use a **test set** to validate the model, calculating metrics such as the accuracy, precision, recall, or F1-score
- Two of the most used ML algorithms for text classification are **Random Forest** and **Multinomial Naive Bayes**