

SIAD M2 DS – 2024/2025

# Machine Learning for Data Science

## Introduction and classification

---

Nadarajen Veerapen

# Syllabus

Introduction and general concepts

Supervised Learning

Classification

K-nearest neighbours, logistic regression, decision trees, neural networks, random forests, gradient boosting

Regression

Applying above algorithms in a regression context

Recommender systems

Machine Learning Engineering

Feature engineering

Evaluation

Hyper-parameter Tuning

Pipelines

- 26 contact hours
- 4 e-learning hours

# Course Evaluation

Continuous assessment (20%)

2 Multiple Choice Questionnaires (2 x 10%)

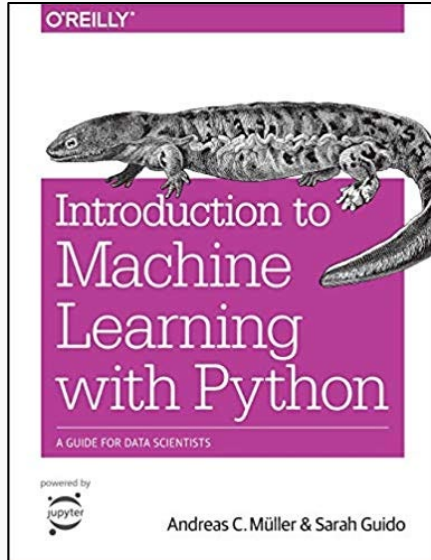
Project work (40%)

In pairs

Exam (40%)

Written exam, no documents allowed (except dictionary)

# Bibliography



Introduction to Machine Learning with Python.  
Andreas C. Müller & Sarah Guido, O'Reilly, 2019.



Python pour le Data Scientist.  
Emmanuel Jakobowicz, Dunod, 2018.

# What is Machine Learning ?

---

# Machine Learning

The definition by Tom M. Mitchell, Machine Learning, 1997, is often cited :

"A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$  if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ ."

Depending on the person you talk to, machine learning will either be considered a **subset of Artificial Intelligence** or at the **intersection of statistics, AI and computer science**. Some will prefer to call it **predictive analysis** or **statistical learning**.

In fact, ML *simply* amounts to designing the function

$$f(X) = y$$

where  $f$  is the model,  $X$  the data and  $y$  the prediction.

It is helpful to remember that machine learning and other related terms are often used as buzzwords for marketing purposes.

# Learning by example

## Supervised Learning

From a series of **input-output examples**, the machine learns a model that matches outputs to inputs. A function is inferred from **labeled data**. The learning success is measured based on the percentage of correct predictions made by the model. Tasks: regression, classification.

## Unsupervised Learning

From a series of **inputs only**, the machine learns how these are organised. We look for structure within the data but **without feedback**. Success can be difficult to evaluate. Tasks : clustering/segmentation, dimensionality reduction, ...

## Reinforcement Learning

The machine learns how to find the ideal behavior based on **feedback from the environment**, rewards or punishment. There is interaction with the environment towards a certain goal. Success is determined by the maximum reward. Examples : video game AI, online control of a system, ...

# Types of algorithms according to their output

Training data are used to build a model that is then used to predict what happens to new data where...

Type of Output	Algorithm Category
Output is one or more discrete classes	Classification (supervised)
Output is continuous	Regression (supervised)
Output is membership of a similar group	Clustering (unsupervised)
Output is a simplification of dimensionality	Dimensionality reduction (unsupervised)



# A non-exhaustive timeline

**1805** – Legendre describes the least squares method, used in regression

**1812** – Laplace proposes the Bayes theorem (conditional probability of A knowing B)

**<1950** – The majority of commonly used statistical methods have already been invented

**1950** – Turing proposes the concept of a *learning machine*

**1951** – First neural network

**1967** – Nearest Neighbor algorithm

**1970s** – "Artificial Intelligence Winter" pessimism towards AI because of over-promising (also again around 1990)

**1989** – Reinforcement Learning

**1995** – Random Forests

**1990s/2000s** – New work on neural networks

**2010s** – Deep Learning

# Two cultures

Two ways of saying more or less the same thing.

## Machine Learning

- A field within Artificial Intelligence.
- Emphasis is often on **algorithms** and applications.
- Objective is to make a **prediction**.

## Statistical Learning

- A field within statistics.
- Emphasis is often on **models** and their interpretability.
- Objective is on **inference**.

# Python Tools

---

# Tools

In this course we will use:

- The **Python** programming language ([tutorial](#) and [documentation](#))
- The **scikit-learn (sklearn)** library for the Machine Learning algorithms ([documentation](#)), as well as **numpy**, **pandas** and **matplotlib** (amongst others) to represent, manipulate and visualize data
- **Anaconda (Conda)** to manage the Python environment ([documentation](#))
- **Jupyter Notebook** to structure, document and share our analyses ([documentation](#))
- **Google Colab** and **Kaggle Notebooks** are also an option

# Anaconda

**Anaconda** is a *Python distribution* composed of a set of packages that are useful for data science and machine learning.



*Packages* are libraries composed of code meant to perform some specific tasks. For instance *matplotlib* is used to generate plots.

**Conda** is the package, dependency and environment manager within Anaconda. When a package is downloaded, Conda will ensure that the other packages required are also downloaded (same goes for updates).

*Environments* are package configurations. They allow us to have different packages or different versions of some package on a single machine. If you install Anaconda on your machine, you can thus configure it to have exactly the same environment as the one on the PCs in the computer lab.

# Cloning the environments from the computer labs

Launch Anaconda Prompt. It will open with the base environment. Then run:

```
conda env export > C104environment.yml
```

If there are multiple environments, it is possible to switch to another one:

```
activate otherEnvironment
```

The YML contains the list of packages within the environment as well as their version. It is possible to import the environment into your own installation using Anaconda Navigator (Environments tab, Import button) or via Anaconda Prompt :

```
conda env create -f C104environment.yml
```

It is possible to change the environment name directly within the YML file.

# Jupyter Notebook/Lab

**Jupyter Notebook** and **Jupyter Lab** are web applications that allow for the creation and sharing of documents that contain (executable) code, equations, visualizations and documentation.

They are often used for : data preparation, numerical simulations, statistical modeling, data visualisation, and machine learning.

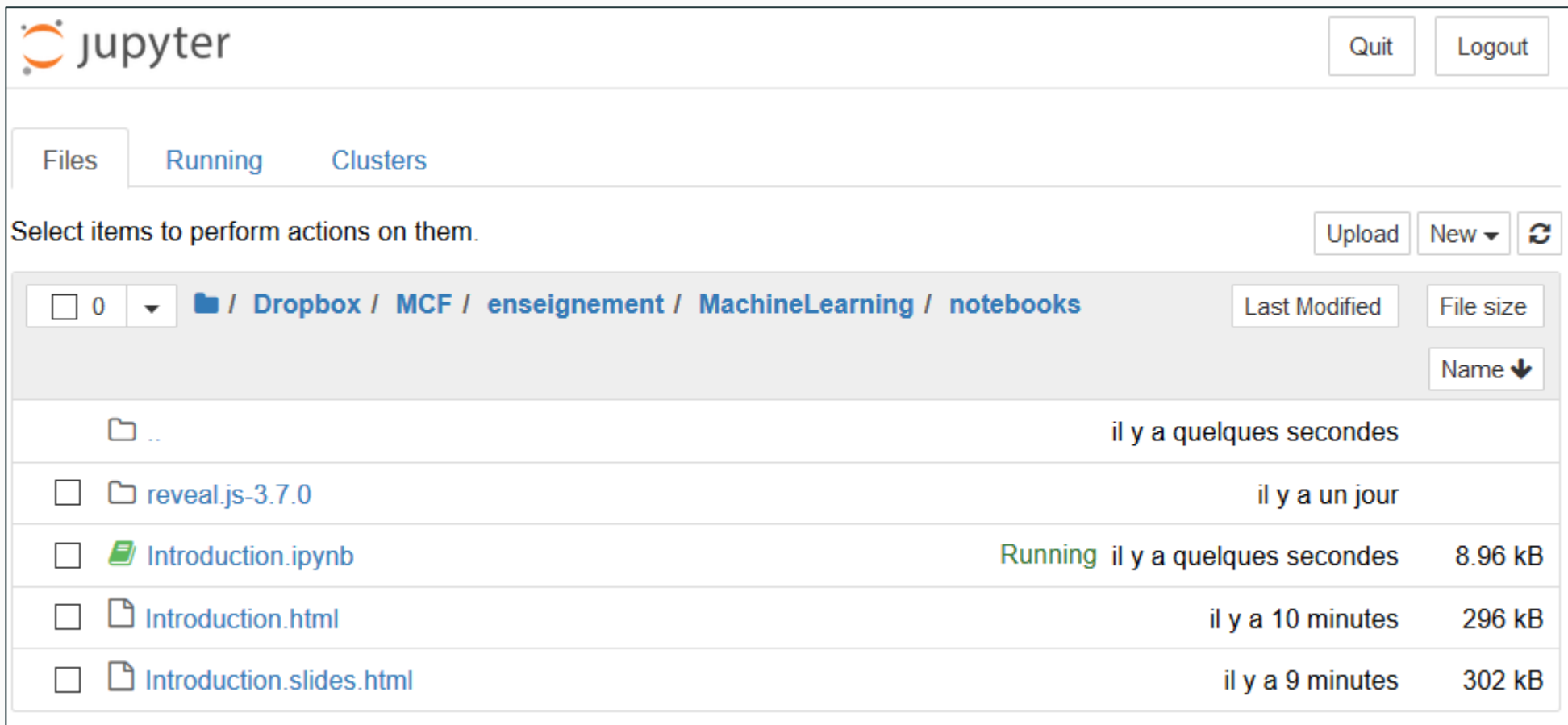
They can be launched from the Windows Start menu (with the default environment), or via Anaconda Navigator (with any of the environments), or from Anaconda Prompt :

```
jupyter notebook  
jupyter lab
```



# Jupyter Notebook

A browser window containing the web application will launch automatically (or use the localhost:8888 address in your browser). Browse to the desired folder and click on *New* to create a new *Python 3* notebook.

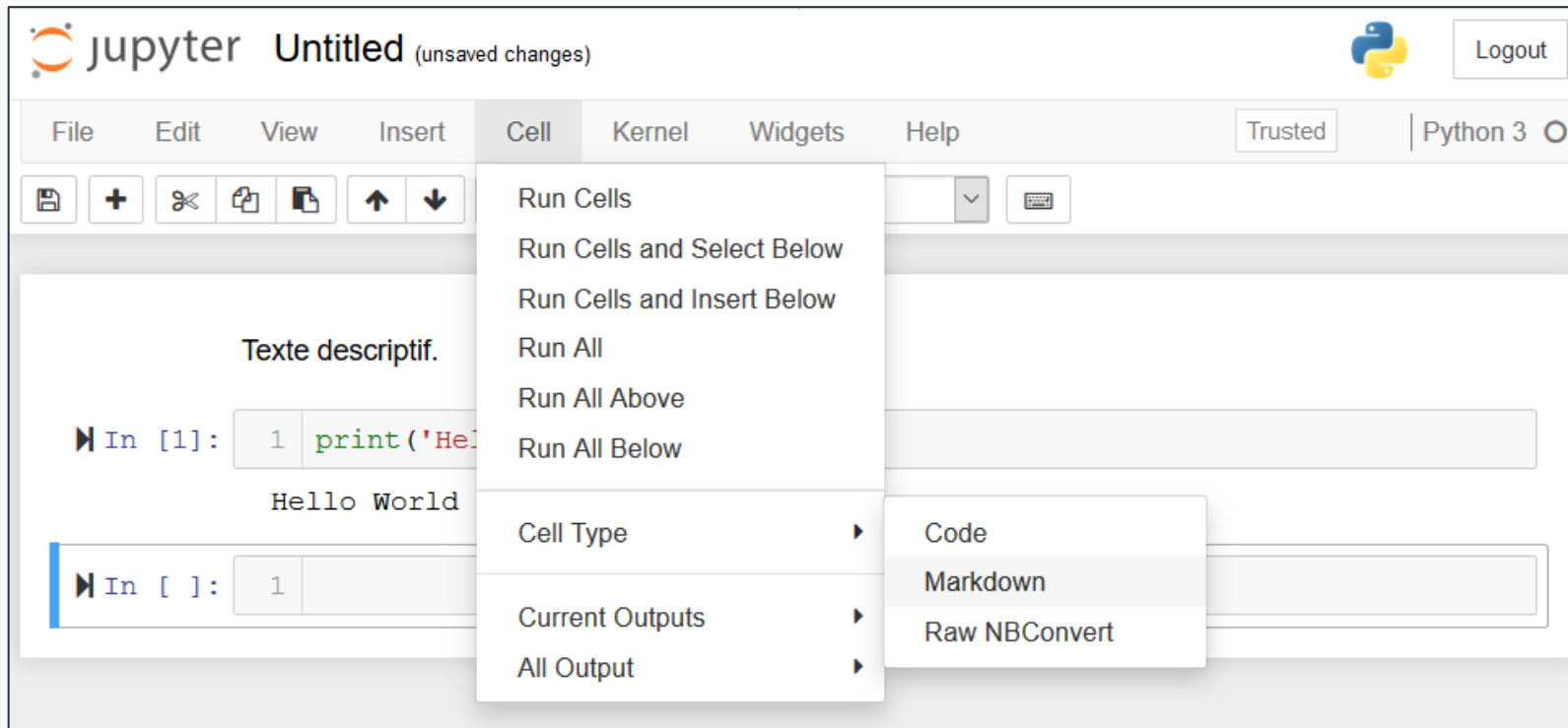
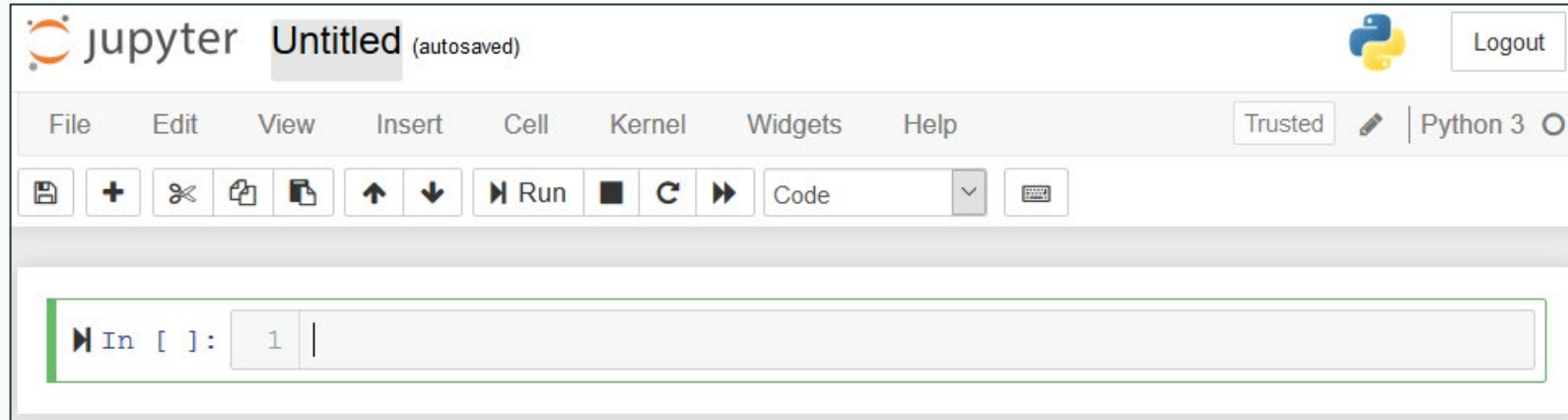


The screenshot shows the Jupyter web interface. At the top, there's a header with the Jupyter logo and the word "jupyter". To the right are "Quit" and "Logout" buttons. Below the header, there are tabs for "Files", "Running", and "Clusters". The "Files" tab is active. Below the tabs, there's a message "Select items to perform actions on them." and buttons for "Upload", "New" (with a dropdown arrow), and a refresh icon. Below this is a breadcrumb path: `/ Dropbox / MCF / enseignement / MachineLearning / notebooks`. To the right of the path are buttons for "Last Modified" and "File size". Below the path is a table of files and folders. The table has columns for a checkbox, a file/folder icon, the name, the status (if running), the last modified time, and the file size (if a file).

<input type="checkbox"/>	0				
		<b>/ Dropbox / MCF / enseignement / MachineLearning / notebooks</b>			
				Last Modified	File size
				Name	
		..		il y a quelques secondes	
<input type="checkbox"/>		reveal.js-3.7.0		il y a un jour	
<input type="checkbox"/>		Introduction.ipynb	Running	il y a quelques secondes	8.96 kB
<input type="checkbox"/>		Introduction.html		il y a 10 minutes	296 kB
<input type="checkbox"/>		Introduction.slides.html		il y a 9 minutes	302 kB



# Jupyter Notebook



# Using Python libraries

## Importing libraries

```
import random # random number library in Python
import sklearn # scikit-learn library
sklearn.__version__ # gives the version number of the library
```

Once a library is imported, a function is called using the [library].[function] syntax:

```
random.randint(0,10) # generate a random number between 0 et 10
```

## Importing an element from a library

```
from random import randint
value = randint(0,10)
```

## Library alias

```
import numpy as np
value = np.arange(5) # first 5 integers starting from 0
```

# NumPy (Numerical Python)

The central element of numpy is the **array**. It stores values within a data structure supporting all types of advanced computation.



Numpy data structures are used within SciPy, the Python library for scientific computation.

Numpy features the **ndarray**, an  $n$  dimensional data structure, used to store data :

- There is a single type of data within an ndarray;
- As many dimensions as required (1D – a vector; 2D – un matrix, etc.);
- Specific methods optimized for fast computation.

# Building a numpy array

## Load the library

```
import numpy as np
np.__version__ #check library version if necessary
```

## Create an array based on a Python list

```
a = np.array([1,3,5,7])
> array([1, 3, 5, 7])
```

## Create an array based on a sequence of numbers

```
a = np.arange(5) #first 5 integers beginning from 0
> array([0, 1, 2, 3, 4])
a = np.arange(2,15,3) #integers 2 to 15 (excluded) with a step of 3
> array([ 2,  5,  8, 11, 14])
a = np.linspace(0,9,10) #divide interval [0,9] into 10 values
> array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

# Building a numpy array

## Filling an array with values

```
a = np.ones(5)
> array([1., 1., 1., 1., 1.])
a = np.zeros(6)
> array([0., 0., 0., 0., 0., 0.])
a = np.full(7,3.0)
> array([3., 3., 3., 3., 3., 3., 3.])
```

## Specifying the data type when creating an object

```
a = np.zeros(7,dtype="int") # also float, bool, str, etc.
> array([0, 0, 0, 0, 0, 0, 0])
```

## Deleting elements

```
a = np.delete([1,2,3,4,5], [0,1,3])
> array([3, 5])
```

# Properties of an array

```
a = np.array([[1,0],[1,2],[5,3]])
```

Dimensions

```
a.shape  
> (3, 2)
```

Number of dimensions

```
a.ndim  
> 2
```

Number of elements

```
a.size  
> 6
```

Data type

```
a.dtype  
> int32
```

# Operations on arrays

In Python, the + sign concatenates lists. For arrays, the + sign means summing individual array elements.

```
np.array([1,2,3]) + np.array([1,1,1])  
> array([2, 3, 4])
```

## Accessing elements

```
a[5:] # elements in a starting from the 5th, a[start:end:step]  
a[-1] # last element of a
```

## Changing dimensions

```
a = np.random.randn(1000) # array of 1000 elements  
b = a.reshape(100,10) # transformed into a 100x10 matrix  
c = a.reshape(-1,1) # transform rows into columns  
> (1000,) # a.shape, 1 dimension, as a row  
> (100, 10) # b.shape, 2 dimensions, 100 rows, 10 columns  
> (1000, 1) # c.shape, 2 dimensions, 1000 rows, 1 column
```

# Functions

Function	Usage	Example
all	Tests whether all elements are True	<code>np.all([True, False, True])</code> > False
any	Tests whether at least one element is True	<code>np.any([True, False, True])</code> > True
argmax/argmin	Index of the max/min value	<code>np.argmax([1,3,3,5,2])</code> > 3
argsort	Array of the indices of values in the array after sorting	<code>np.argsort([3,1,2])</code> > [1, 2, 0]
average/mean	Mean (weighted mean for <i>average</i> )	<code>np.average([1,2], weights=[0.75,0.25])</code> > 1.25
clip	Data outside the clip interval are changed to the closest clip value	<code>np.clip([1,2,3,4,5], 2, 3)</code> > [2, 2, 3, 3, 3]



# Functions

Function	Usage	Example
dot	Dot product between 2 arrays	
floor	Round to the lower integer	<code>np.floor([-1.7, -0.2, 1.5, 2.])</code> > <code>[-2., -1., 1., 2.]</code>
round	Round to a number of decimal places	<code>np.round([1.234, 2.76, 3.1], decimals=1)</code> > <code>[1.2, 2.8, 3.1]</code>
sort	Sort in ascending order	<code>np.sort([3.2, 5.3, 3.1, 1.1])</code> > <code>[1.1, 3.1, 3.2, 5.3]</code>
sum	Sum the elements	<code>np.sum([[1, 2], [3, 7]], axis=1) # axis=0</code> > <code>[3, 10]</code>
transpose	Transpose an array	<code>np.transpose([[1, 2], [3, 7]])</code> > <code>[[1, 3], [2, 7]]</code>

# Generating random numbers

Scikit-learn and ML algorithms often use random numbers. Pseudo-random number generators are used to create sequences of numbers that have the properties of random numbers.

```
np.random.rand(3, 2) # 6 random values in a (3, 2) matrix
np.random.randn(2, 5) # 10 values from a normal distribution
np.random.randint(10, size=3) # 3 integers in the interval [0,10[ ([0,9])
np.random.randint(5,10, size=(3,2)) # Matrix of size (3,2) in [5,10[
np.random.poisson() # value from a Poisson distribution
```

In order to get replicable results and analyses, it is recommended to fix the "**seed**" of the generator. This is the number that initialises the generator. Reusing a generator with the same seed will produce the same sequence of numbers.

```
np.random.seed(0) # no seed, date/time is used as seed
```

Alternatively (recommended way):

```
rnd = np.random.RandomState(0)
rnd.rand(3,2)
```

# Base libraries – pandas

**pandas** is a Python library to manipulate and analyse data. It is built around a data structure known as the **DataFrame** inspired from dataframes in R. A dataframe is a table similar to an Excel sheet.

As opposed to Numpy, where all cells within an array have the same type, in a dataframe each column can have a different type. Pandas allows for easy input of different file types.

```
import pandas as pd
# creating a simple dataset
data = {'Nom' : ["Jean", "Anita", "Aslan", "Nélia"],
        'Lieu' : ["Lille", "Paris", "Berlin", "Nice"],
        'Age' : [24, 13, 53, 33]}
data_pandas = pd.DataFrame(data)
display(data_pandas)
# selecting a subset of the data
display(data_pandas[data_pandas.Age > 30])
```

# Matplotlib

**matplotlib** is the base Python library for visualization. It allows for the creation of histograms, scatter plots, etc.

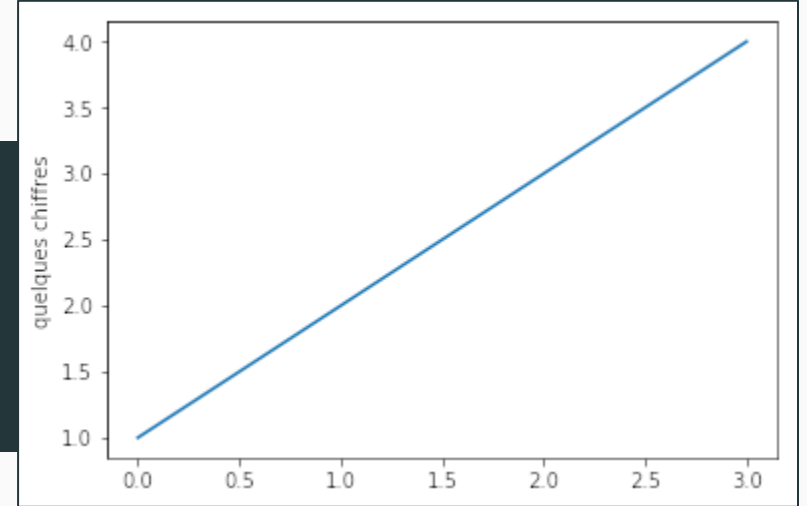
`matplotlib.pyplot` is a collection of functions. Each `pyplot` function corresponds to a change in the plot: creating a plot, creating a drawing area within the plot, drawing lines, adding labels, etc.

**Seaborn** is a visualisation library based on matplotlib. It offers high level functionality to create pleasing and informative visualizations. When using seaborn, it is always possible (and sometimes required) to use matplotlib functions.

# Matplotlib.pyplot

A simple plot:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('quelques chiffres')
plt.show()
```

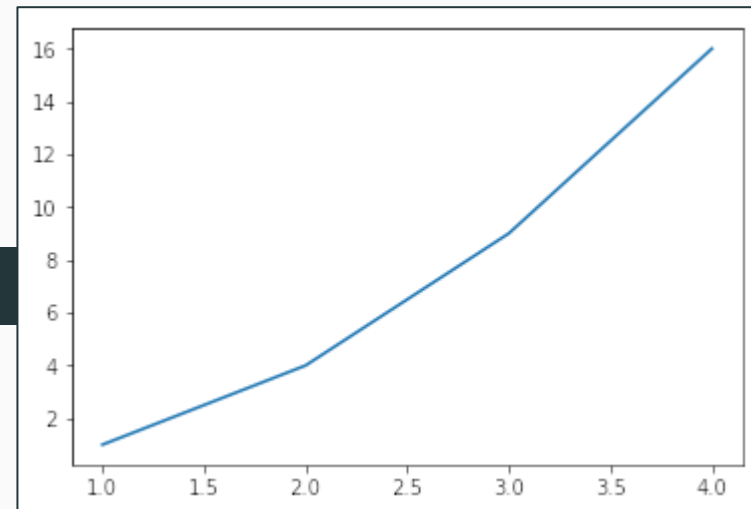


*Why does the x-axis go from 0 to 3?*

If a single list or array is passed to `plot()`, it will be interpreted as the y-axis and the x-axis will be generated automatically starting from 0 because Python sequences start from 0. In the example we have `[0, 1, 2, 3]`.

2 parameters can be passed to `plot()` :

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
```



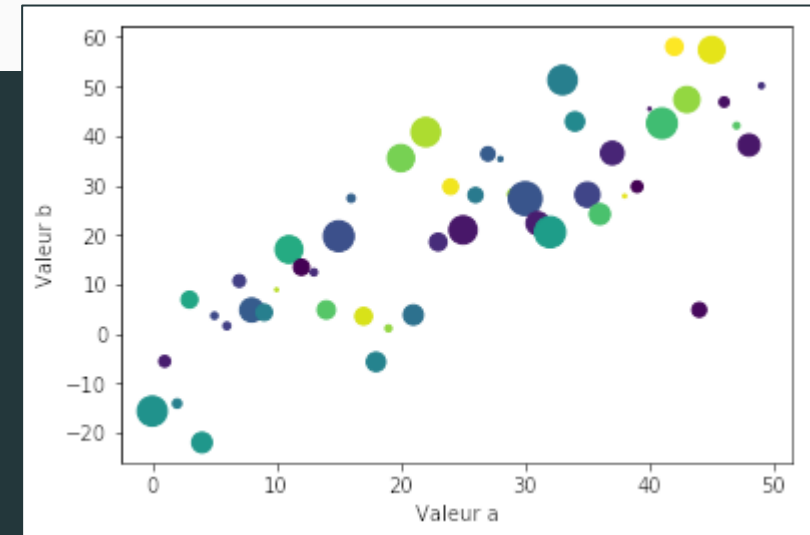
# Formating a pyplot plot – column keys

```
import pandas as pd

data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50), # low=0, high=50, size=50
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

plt.scatter('a', 'b', c='c', s='d', data=data) # from a dictionary
plt.xlabel('Valeur a')
plt.ylabel('Valeur b')
plt.show()

dataframe = pd.DataFrame(data=data)
plt.scatter('a', 'b', c='c', s='d', data=dataframe) # from a dataframe
```



Manipulating data with dictionary keys and a dataframe.

**What do parameters c and s correspond to in the scatter() function?**

# Scikit-learn

---

# Scikit-Learn

Extensions to SciPy (Scientific Python) are called *SciKits*.

SciKit-Learn contains Machine Learning algorithms.

- Algorithms for supervised et unsupervised learning
- Built on SciPy and NumPy
- A standardized Python API (application program interface)
- Based on C language libraries: LAPACK, LibSVM, Cython, ...
- Open source : BSD license

Potentially the best general library for ML at the moment.

Development started in 2007 (as Google Summer of Code project); publicly launched in 2010 by INRIA (*Institut national de recherche en informatique et en automatique*) ; currently supported by a consortium of sponsors (Microsoft, BCG, AXA, BNP Paribas, Fujitsu, Intel, nVidia, Dataiku).



# Introduction to scikit-learn

Scikit-learn contains a number of datasets, including the classic *Iris* dataset.

```
from sklearn.datasets import load_iris
iris = load_iris()
type(iris) # sklearn.utils.Bunch
```

The `iris` object of the `Bunch` type (similar to a dictionary).

```
print("Keys: \n{}".format(iris.keys()))
> Keys:
> dict_keys(['data', 'target', 'target_names', 'DESCR',
  'feature_names', 'filename'])
print(iris['DESCR']) # dataset description
print("Classes: {}".format(iris['target_names'])) # target classes
> Classes: ['setosa' 'versicolor' 'virginica']
print("Features: \n{}".format(iris['feature_names']))
> Features:
> ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
  'petal width (cm)']
```



# Introduction to scikit-learn

```
type(iris['data']) # numpy.ndarray
iris['data'].shape # (150, 4)
```

We thus have 150 rows/observations and 4 columns/features for the data.

```
type(iris['target']) # numpy.ndarray
iris['target'].shape # (150,)
```

And an array of size 150 for the class of each observation.

*Print the first 5 rows of the data.*

[illegible]

Classes are encoded by integers that correspond to the `target_names`.

# Introduction to scikit-learn

## Classification – Measuring performance : training and test sets

*What would happen if a model was trained on all available data?*

We want to be able to assess the **generalization** capacity of a model.

The minimum is to split the dataset into a **training set** and a **test set**. Scikit-learn has a function called `train_test_split` that performs random draws without replacement in order to build the training set with 75% (by default) of the observations and the test set with the remaining 25% (these percentages can be changed). The test set is used to measure the performance of the model.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris['data'], iris['target'], random_state=0)
X_train.shape # (112, 4)
y_train.shape # (112,)
X_test.shape # (38, 4)
y_test.shape # (38,)
```

The feature data are generally referred to by X and target data referred to by y.

# Introduction to scikit-learn

Let us use the k-nearest neighbors (k-NN) classifier for our irises.

```
from sklearn.neighbors import KNeighborsClassifier  
knn = KNeighborsClassifier(n_neighbors=1)
```

The model here is directly composed of the observations. In order to make a prediction for a new observation, the algorithm finds the  $k$  points in the training set that are closest to the new previously unseen point.

The knn object encapsulates the algorithm used to build the model from the training set data, as well as the algorithm to make predictions on new unseen data. For the kNN algorithm, the object stores the training set data. This is not usually the case for other algorithms.

# Introduction to scikit-learn

To build the model on the training set, the `fit` method of the `knn` object is called, together with the training set.

```
knn.fit(X_train, y_train)  
> KNeighborsClassifier(algorithm='auto', leaf_size=30,  
metric='minkowski', metric_params=None, n_jobs=None,  
n_neighbors=1, p=2, weights='uniform')
```

`Fit` returns the `knn` object (and also modifies it), which is why we get the above print out (with its parameters).

# The scikit-learn API

An object-oriented interface centered on the concept of an **Estimator** object:

"An *estimator* is an object that fits a model based on some training data and is capable of inferring some properties on new data. It can be, for instance, a classifier or a regressor." scikit-learn documentation

```
class Estimator(object):  
  
    def fit(self, X, y=None):  
        """Train the estimator on the data. """  
        # gives the value to 'self'  
        self  
  
    def predict(self, X):  
        """Predict the response from 'X'. """  
        # compute the predictions 'pred'  
        return pred
```

# Les objets estimators

- `fit(X,y)` defines the state of the estimator.
- `X` is generally a numpy array (in 2D) of the form (number\_observations, number\_features) (rows, columns).
- `y` is a numpy array (1D) of the form (number\_observations, ).
- `predict(X)` returns the class or value.
- `predict_proba()` returns a numpy array (2D) of the form (number\_observations, number\_classes).

# Introduction à scikit-learn

Let us make a prediction with our model.

```
import numpy as np
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape : {}".format(X_new.shape) # (1, 4)
```

X\_new contains a new observation. Note that even if there is a single observation, we still create a 2D array in order to match the format that scikit-learn expects.

```
prediction = knn.predict(X_new)
print("Prediction: {}".format(prediction))
print("Predicted class: {}".format(iris['target_names'][prediction]))
> Prediction: [0]
> Predicted class: ['setosa']
```

***But how can we be confident in the model and its prediction?***



# Introduction to scikit-learn

By using the **test set**, we can measure the **accuracy** of the model (the fraction of observations that are correctly classified).

```
y_pred = knn.predict(X_test)
print("Predictions on the test set : \n{}".format(y_pred))
> Predictions on the test set :
> [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0 2]
print("Test score: {:.2f}".format(np.mean(y_pred == y_test)))
> Test score: 0.97
```

The Estimator object (knn) possesses a `score()` method that performs this computation.

```
print("Test score: {:.2f}".format(knn.score(X_test, y_test)))
> Test score: 0.97
```

The fraction of the correct predictions  $((\text{true positives} + \text{true negatives})/n)$ , or *accuracy*, is 0.97. This means that the model has correctly predicted 97% of the irises in our test set. Such a high accuracy can *potentially* mean the we can be confident in our model.

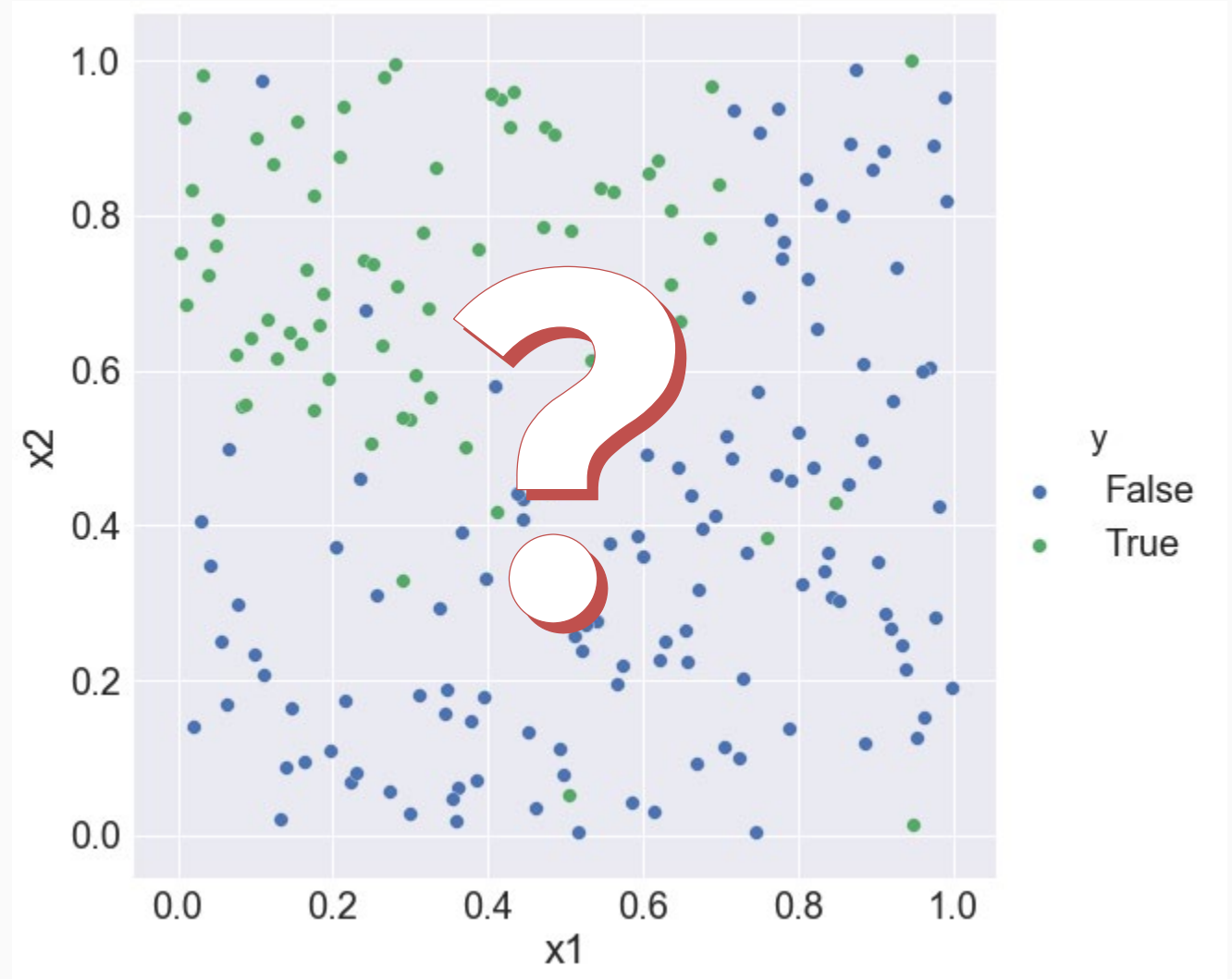
# Classification

---

# Classification

For classification, the objective is to predict a **class label** (the **target variable**). This is a choice among a finite set of possibilities.

Classification is sometimes split into **binary classification**, with exactly 2 classes, and **multiclass classification**, with more than 2 classes. Some methods only work on 2 classes.



# Evaluation methods

The accuracy (percentage of accurate predictions) – we will only consider this today.

Later we will see:

- Confusion Matrix
- Measures derived from the confusion matrix
- ROC curve
- Area under the ROC curve

# Generalization – error prediction

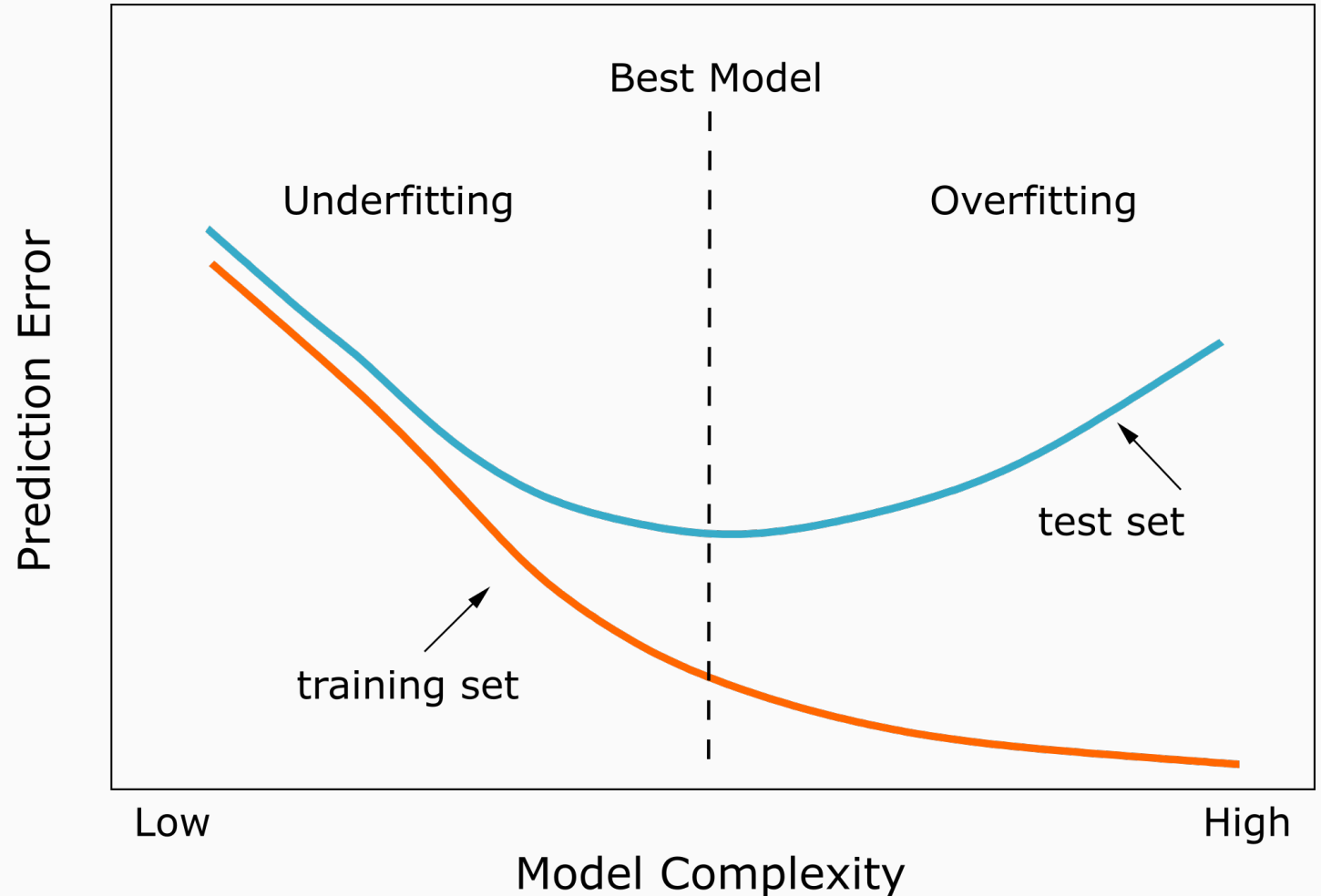
## Training Error vs Generalization Error

**Training error:** mean error when considering the data used to build the model

**Generalization error (test error):** mean error when considering predictions on new unseen data not used to build the model

## Underfitting and Overfitting

A small training error and a large generalization error are signs of **overfitting**, i.e. the model learned the noise rather than the signal



# K-Nearest Neighbors

---

# K-Nearest Neighbors

The k-Nearest Neighbors (k-NN) is a simple but intuitive method that classifies unlabelled observations according to their similarity to observations in the labelled training set.

**Idea:** For a given unlabelled observation  $x$ , find the  $k$  nearest labelled observations within the training set and assign to  $x$  the class label that is the most common (majority vote).

k-NN requires:

- an integer  $k$
- a training set
- a distance metric to measure similarity

k-NN classification is a type of learning without built-in generalization: we don't try to build a model, but simply store all the training data. All the computation is left to the moment when a prediction is required. k-NN is nonparametric.

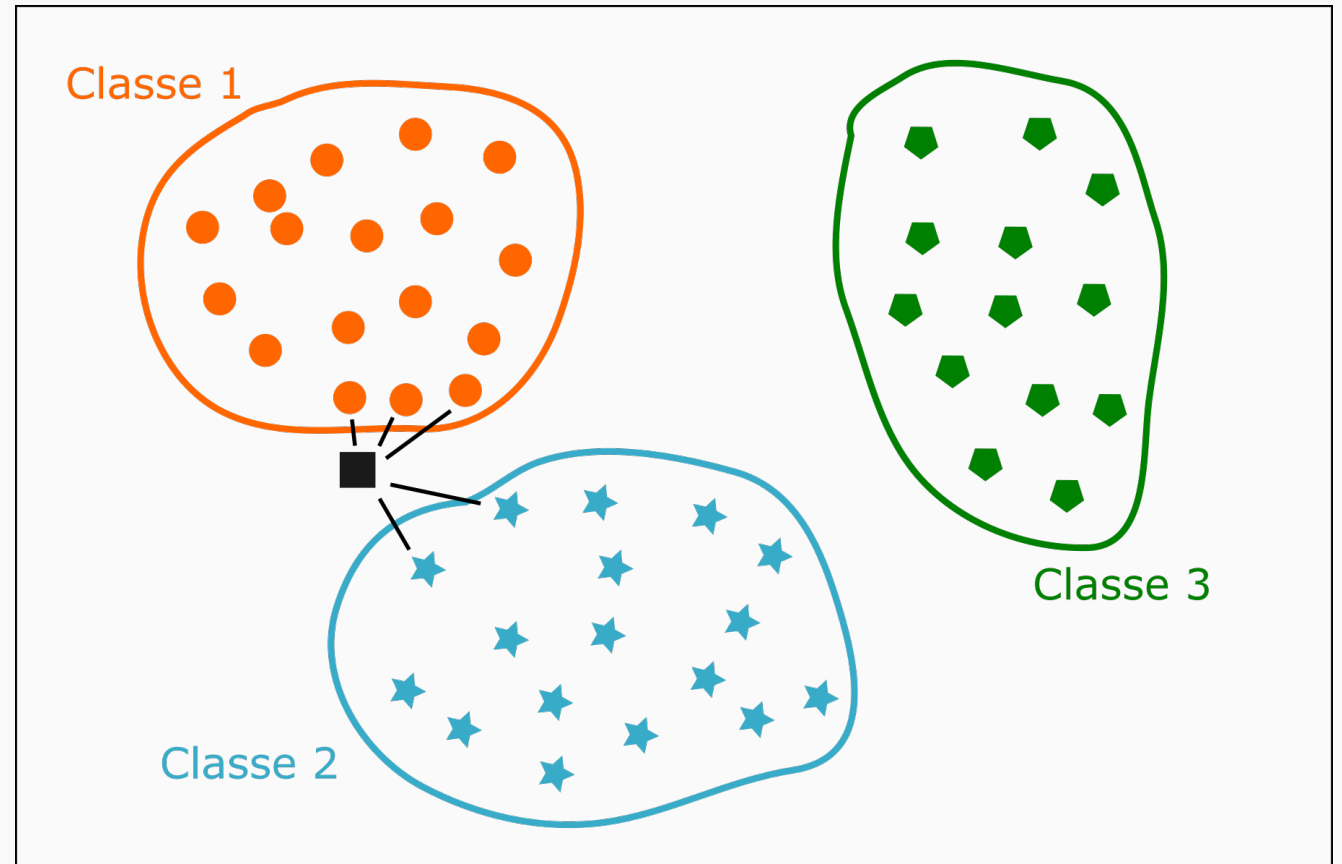
# K-NN – example

There are 3 classes and the objective is to find the value of the class of the unlabelled observation.

Let us consider the Euclidian distance and  $k=5$  neighbors.

Among the 5 nearest neighbors, 3 belong to Class 1 and 2 belong to Class 2.

The unlabelled observation is labelled with Class 1 because it is the majority class.





# K-NN – distance

The distance used will depend on the type of data, for example :

- Generally for continuous variables, the Euclidian distance is used,

$$d(a, b) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}$$

- For integer variables, we can use the Manhattan distance,

$$d(a, b) = \sum_{i=1}^n |a_i - b_i|$$

- For discrete variables, we can use the Hamming distance.

The Minkowski distance is the default option in scikit-learn. It is a generalisation of the Euclidian and the Manhattan distance.

$$d(a, b) = \left( \sum_{i=1}^n |a_i - b_i|^p \right)^{\frac{1}{p}}$$

For  $p=1$  we have the Manhattan distance and for  $p=2$  the Euclidian distance.

# Choosing the value of parameter k

## 1-NN

Works well if there is not much noise in the data or labels

## k-NN

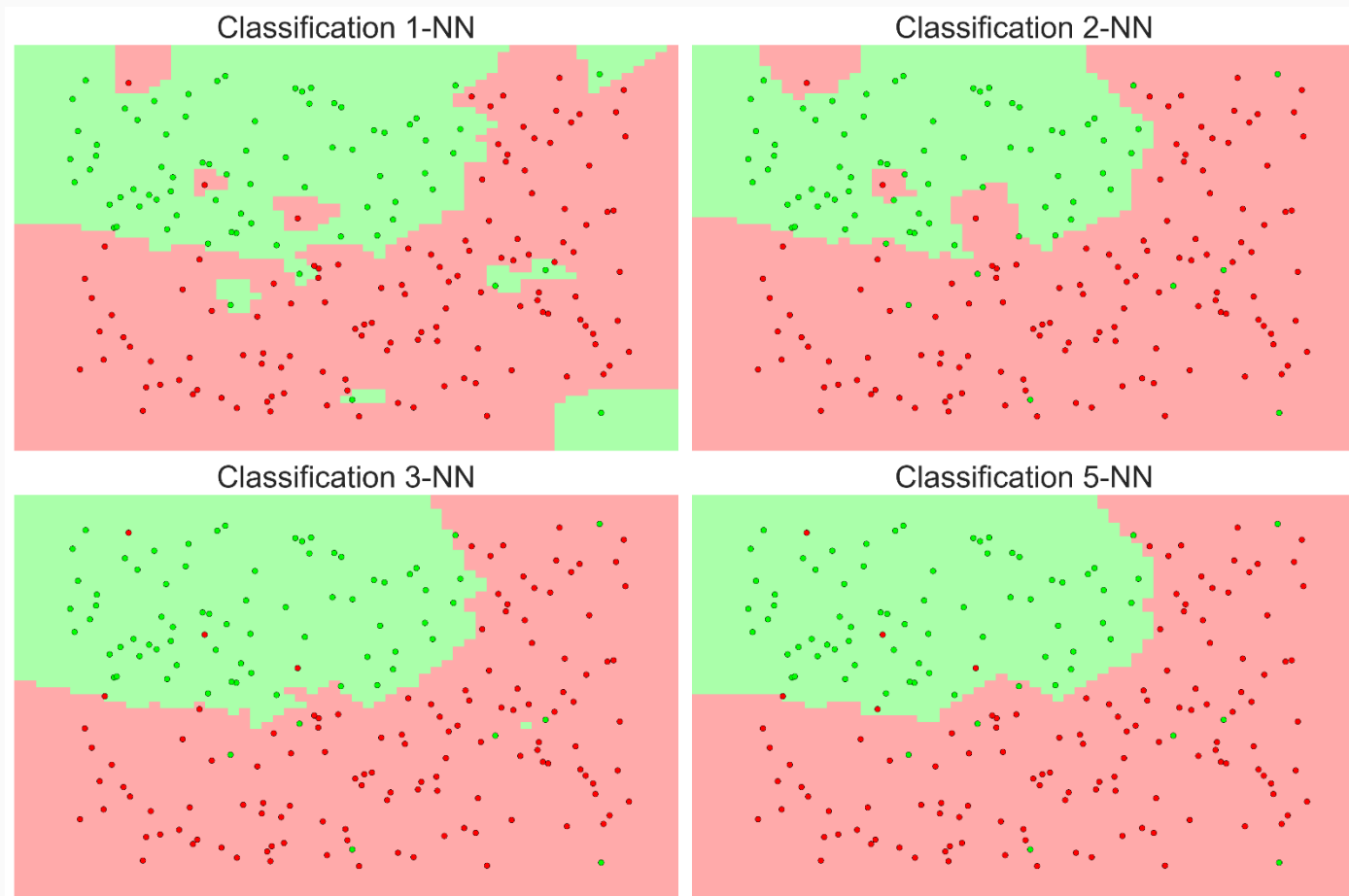
Majority vote among k works better:

- When there is noise in the data or labels;
- Classes partially overlap.

## Choosing k

**Large k**: less sensitive to noise; a larger training set allows for a larger value of k.

**Small k**: more accurately matches fine-grained structures ; necessary for smaller training data.



Decision boundaries according to the value of k

# K-NN in scikit-learn

Two types of classifiers based on proximity/distance:

1. Traditional k-NN ([`sklearn.neighbors.KNeighborsClassifier`](#)) where we consider the  $k$  closest points to a new observation.

This is the most common choice but finding the optimal value for  $k$  heavily depends on the data: in general a larger  $k$  value smoothes the effects of noise but produces less distinct decision boundaries.

2. Neighbors within a certain radius ([`sklearn.neighbors.RadiusNeighborsClassifier`](#)) where we consider points within predetermined radius  $r$  around the new observation.

May be a better choice if the observations are not uniformly sampled, in that case less dense regions have fewer neighbors.

By default, a uniform weight is associated to the neighbors (i.e. majority vote is used). In some circumstances, it may be better to weight the neighbors such that closer neighbors are more important. The k-NN in Scikit-learn can use the *inverse of the distance* between points as weights.

# Considerations for k-NN

There are 2 important parameters:  $k$  and the way distance is computed.

In practice,  $k=3$  or  $5$  works well but it is good practice to tune this parameter nonetheless.

The model is very interpretable and offers acceptable performance without major tuning. However, the larger the training set, the longer the prediction time.

It is important to preprocess the data before using k-NN. Ideally all the variables should have the same domain.

The approach does not work well if there are many features/columns in the data.

**Often worth trying as a first approach.**

# Linear Classifiers

---

# Linear Classifiers

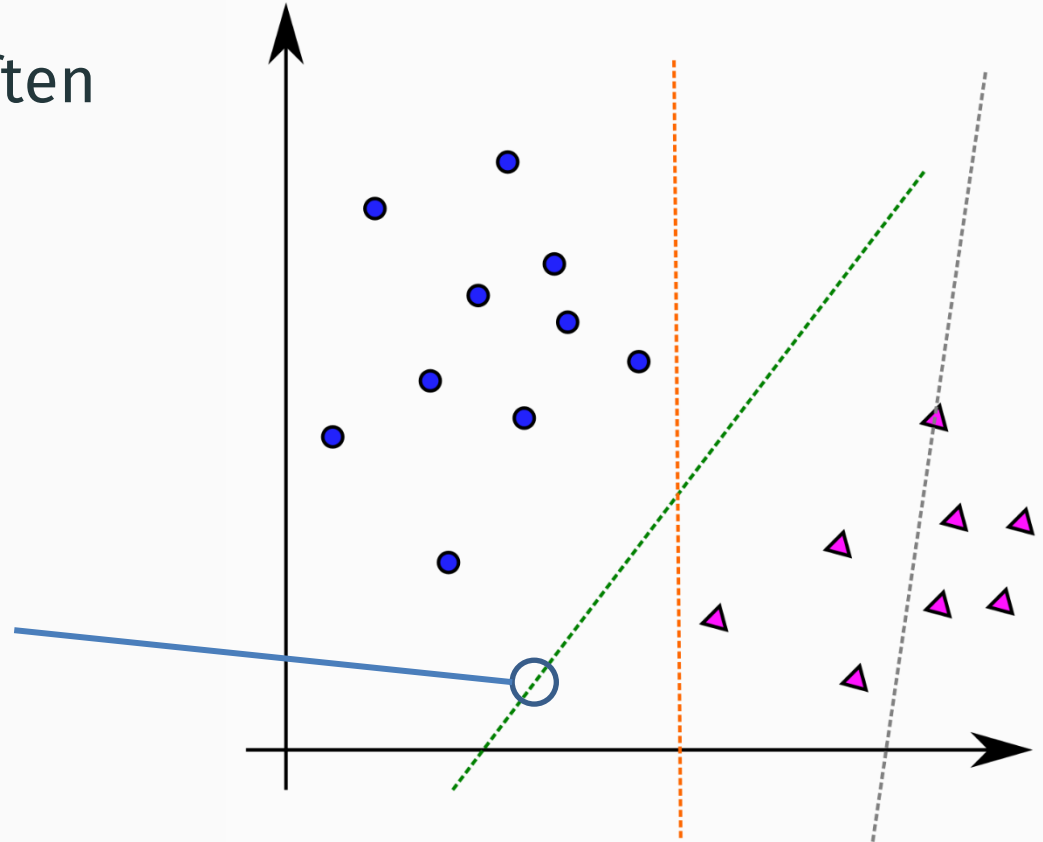
Linear models are frequently used in classification even if it may seem more intuitive to use them in regression.

$$y = f(\vec{w} \cdot \vec{x}) = f\left(\sum_{i=1}^n w_i x_i\right) = f(w_1 x_1 + \dots + w_n x_n)$$

Where  $\vec{w}$  is a weight vector and  $f$  is often a threshold function:

$$f(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{w} \cdot \mathbf{x} > T \\ 0, & \text{otherwise} \end{cases}$$

Straight line in 2 dimensions  
Plane in 3 dimensions  
Hyperplane in >3 dimensions



# Discriminative Training of Linear Classifiers

Discriminative training is a form of supervised learning.

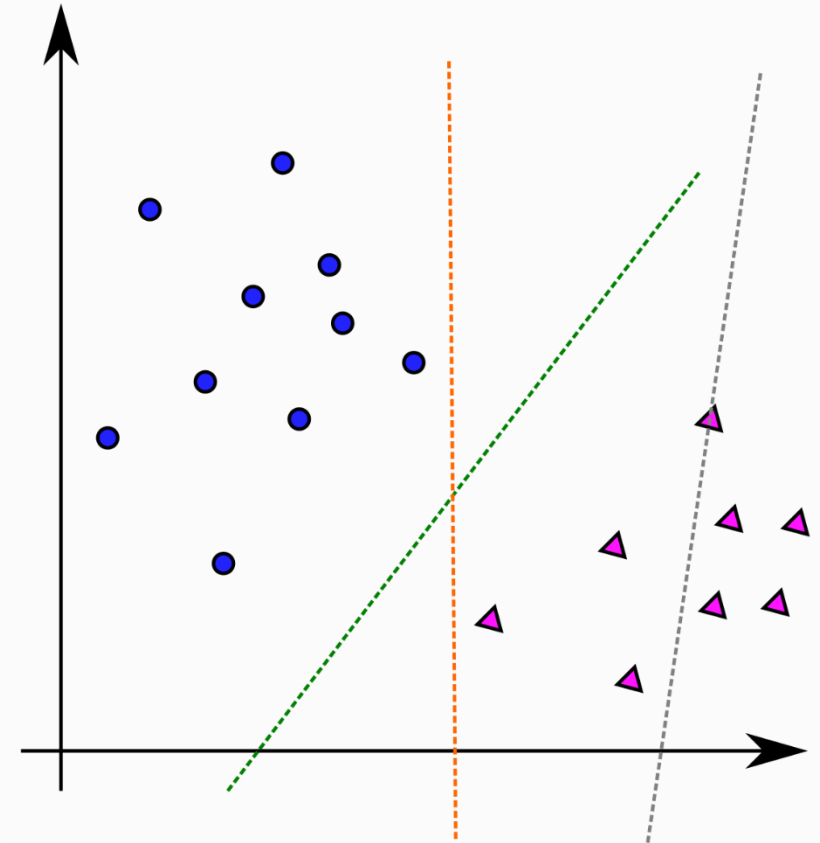
It uses an **optimization algorithm** that is given a training set and its expected outputs, and a **loss function**

$$V(f(\vec{x}), y)$$

that measures the difference between the classifier's predictions ( $f(\vec{x})$ ) and the expected outputs ( $y$ ).

The loss function usually depends on the linear classifier used. An example of a loss function is the mean squared error (MSE).

The optimization algorithms (or solvers) are usually generic but may only support some types of regularizations.



# Regularization

**Regularization** is a constraint on the choice of coefficients ( $\vec{w}$ ) in order to explicitly restrict the model. This is done to **prevent overfitting**. In practice, this involves minimizing the values of the coefficients. It can be used for both classification and regression.

$$\min_f \sum_{i=1}^n V(f(x_i), y_i) + \lambda R(f)$$

$V$  – loss function  
 $R$  – regularization function

2 common forms of regularization are:

$L_2$  or **Ridge** regularization penalizes the squared  $L_2$  norm of the coefficients,

$$\min_w \sum_{i=1}^n V(x_i \cdot w_i, y_i) + \lambda \sum_{i=1}^n w_i^2$$

$L_1$  or **Lasso** regularization penalizes the  $L_1$  norm of the coefficients,

$$\min_w \sum_{i=1}^n V(x_i \cdot w_i, y_i) + \lambda \sum_{i=1}^n |w_i|$$

In scikit-learn, the C parameter ( $1/\lambda$ ) determines the strength of the regularization for classification algorithms (for regression, parameter alpha =  $\lambda$ ). A large value of C corresponds to less regularization, i.e. the model will better fit the training set, whereas a small value will tend to bring the coefficients closer to 0, making the model more general.



# Multiclass or multinomial classification – One-vs.-Rest

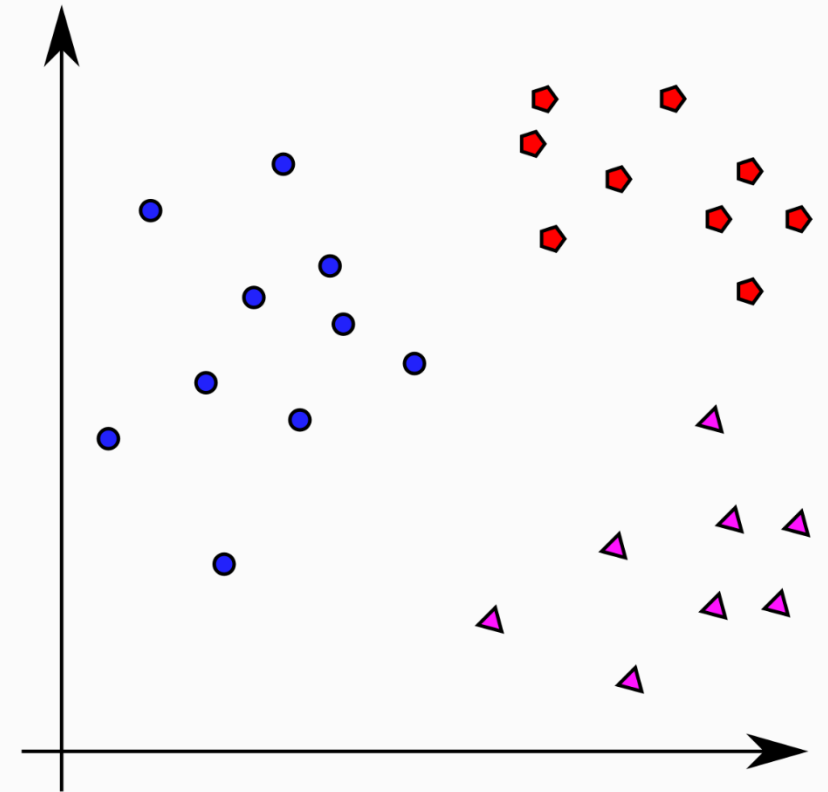
Many classification algorithms only work for 2 classes.

One common technique to extend binary classification to multinomial classification is the one-vs.-rest approach.

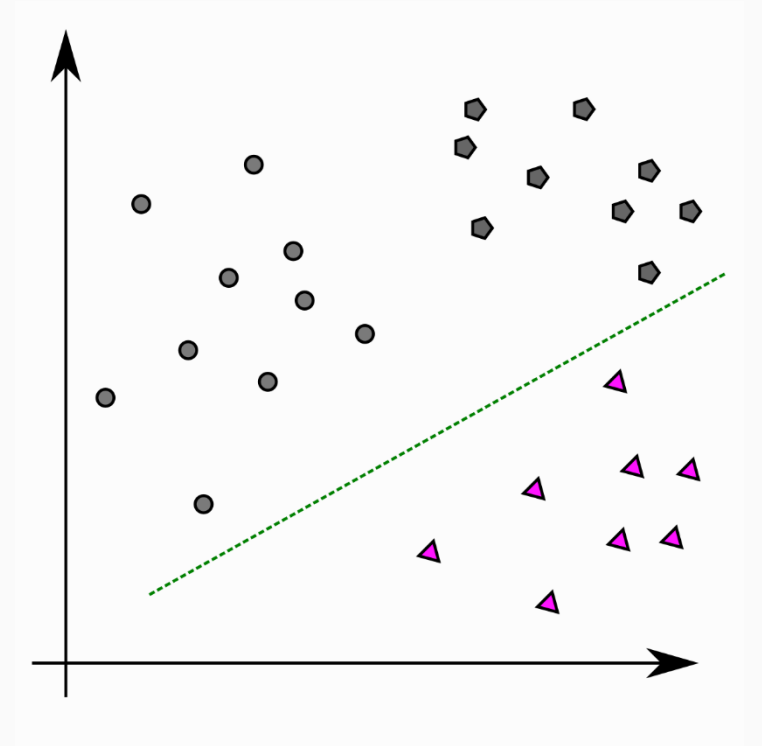
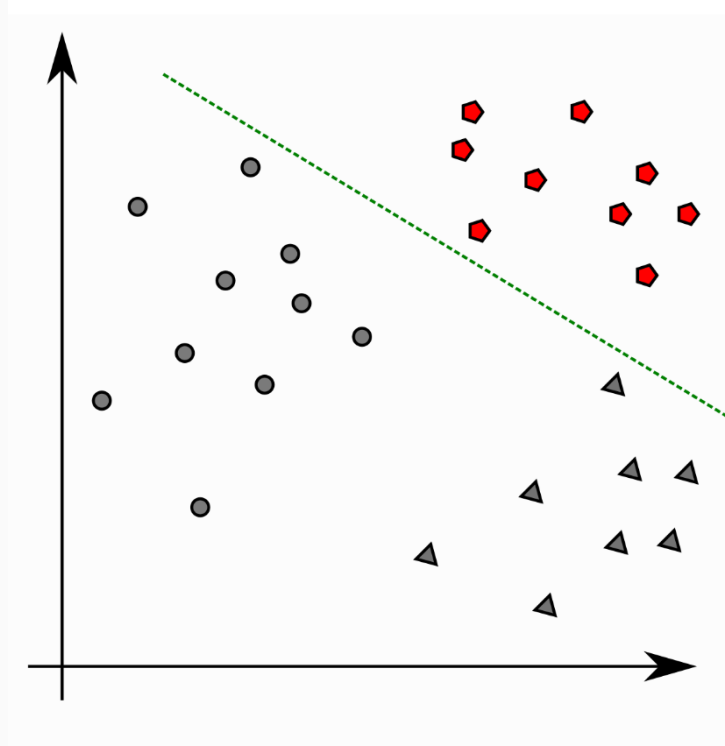
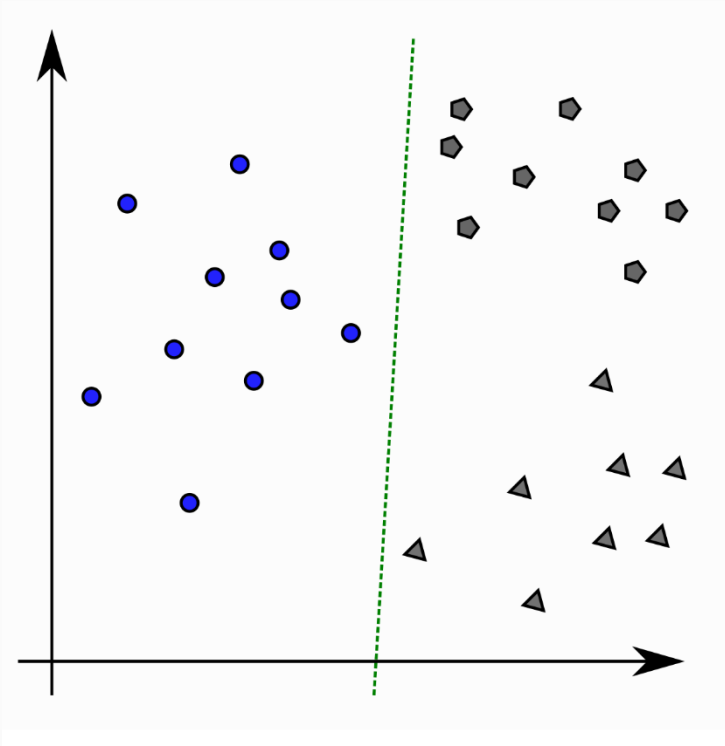
## **One-vs.-Rest**

As many binary models as there are classes are built. Each model deals with one specific class versus the rest of the classes.

To make a prediction for a given point, all the binary classifiers are tested on that point and the classifier with the highest score "wins" and the point is assigned the specific class predicted by this classifier.



# One-vs.-Rest



# Considerations for Linear Classifiers

The value of parameter  $C$  is important but there are few other hyperparameters to tune.

If we suppose that few features/columns in the data are really important, then usually  $L_1$  regularization is chosen. Otherwise,  $L_2$  regularization is chosen.

$L_1$  regularization is useful when the interpretability of the model is important: few features are used, therefore it is easier to understand which of the features are more important for the model, and what their effects are.

Linear models are generally more easily interpretable than other models. However, the relationship between the value of coefficients and their importance is not necessarily clear, in particular when features are correlated.

Linear models usually still exhibit good behavior when the number of features is large with respect to the number of observations.