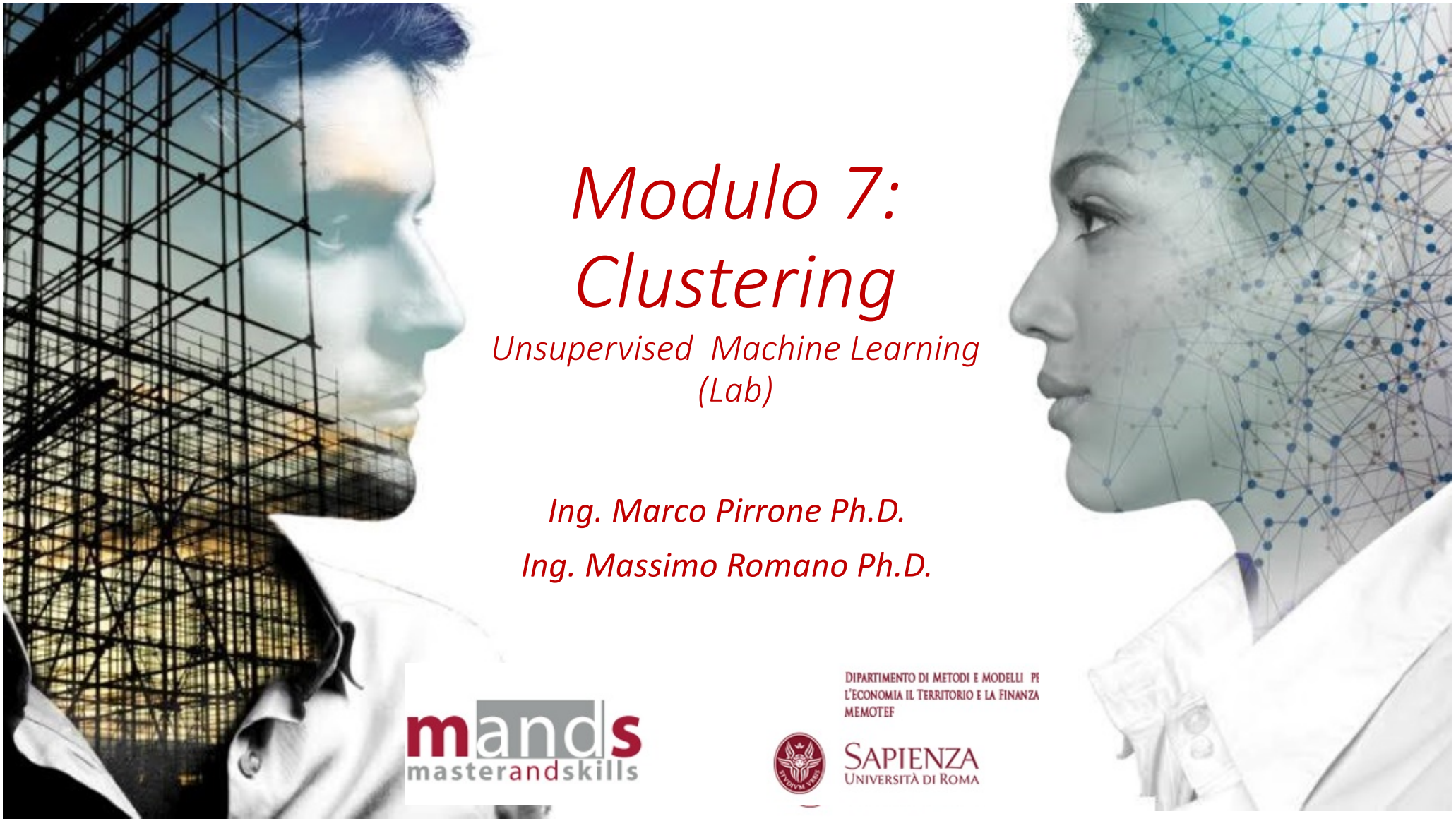


Modulo 7: Clustering

Unsupervised Machine Learning (Lab)

Ing. Marco Pirrone Ph.D.

Ing. Massimo Romano Ph.D.

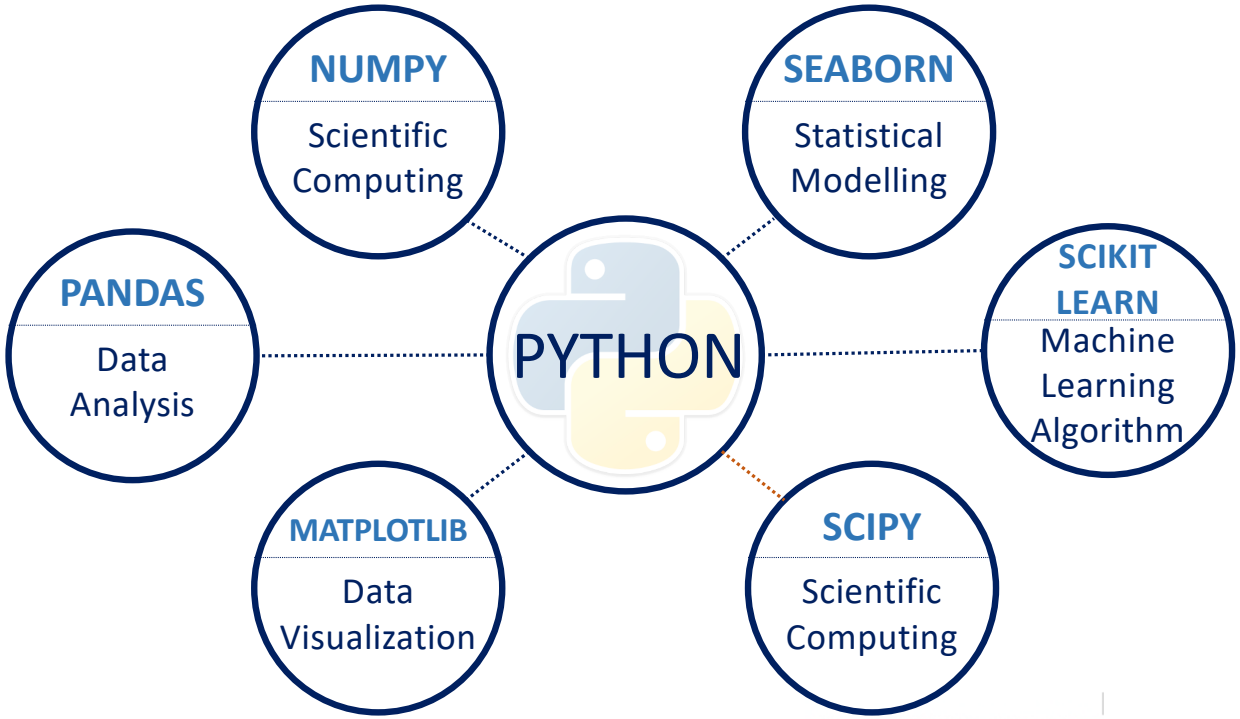
mands masterandskills

DIPARTIMENTO DI METODI E MODELLI PE L'ECONOMIA IL TERRITORIO E LA FINANZA MEMOTEF

SAPIENZA UNIVERSITÀ DI ROMA

Pillars

Library



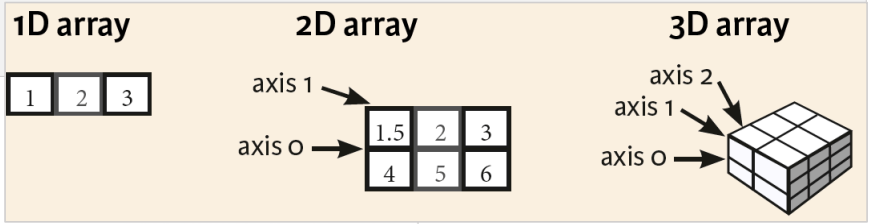
NumPy (1/2)

- **NumPy** (Numerical Python or Numeric Python), is an open source Python module which provides fast mathematical computation on arrays and matrices.
- Arrays and matrices are an essential part of the Machine Learning ecosystem

```
np.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)
  object:array_like - An array, any object exposing the array interface, an
  object whose array method returns an array, or any (nested) sequence.

  out:ndarray - An array object satisfying the specified requirements.

  ndarray -> N-dimensional
```



NumPy (2/2)

Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

`>>> np.zeros((3,4))`	Create an array of zeros
`>>> np.ones((2,3,4),dtype=np.int16)`	Create an array of ones
`>>> d = np.arange(10,25,5)`	Create an array of evenly spaced values (step value)
`>>> np.linspace(0,2,9)`	Create an array of evenly spaced values (number of samples)
`>>> e = np.full((2,2),7)`	Create a constant array
`>>> f = np.eye(2)`	Create a 2X2 identity matrix
`>>> np.random.random((2,2))`	Create an array with random values
`>>> np.empty((3,2))`	Create an empty array

Pandas (1/4)

- **Pandas** (PANel + DAta) is a library based on NumPy that allows easy and fast data analysis and manipulation by providing numerical tables and time series data structures called **DataFrame** and **Series**, respectively.
- **Series** -> One-dimensional **ndarray** with axis labels (including time series).

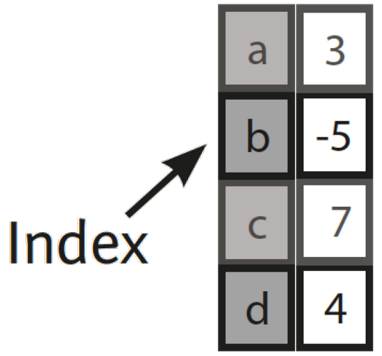
```
pd.Series(data=None, index=None, dtype=None, name=None, copy=False,
fastpath=False)
  data: array-like, Iterable, dict, or scalar value
  index: array-like or Index (1d) - Values must be hashable and have the same
  length as data.
  name: The name to give to the Series.

  One-dimensional ndarray with axis labels (including time series).
```

Pandas (2/4)

Series

A **one-dimensional** labeled array capable of holding any data type




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

Pandas (3/4)

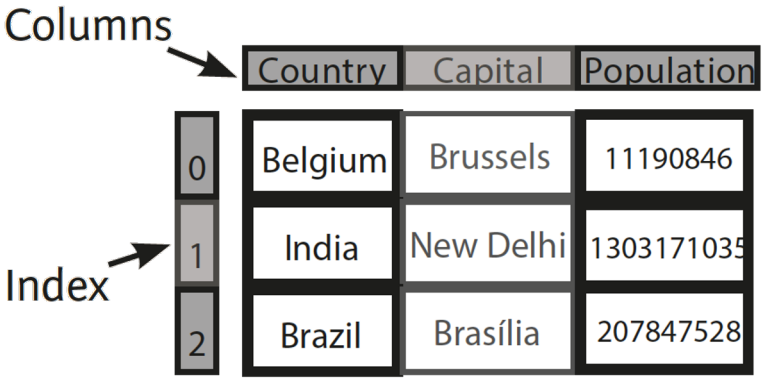
- **DataFrame** -> Two-dimensional, size-mutable, potentially heterogeneous tabular data.

```
pd.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)

data: ndarray (structured or homogeneous), Iterable, dict, or DataFrame;
index: Index or array-like - Index to use for resulting frame;
columns: Index or array-like - Column labels to use for resulting frame;
```

Pandas (4/4)

DataFrame



A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Matplotlib (1/3)

- **Matplotlib** is a python library used to create 2D graphs and plots by using python scripts.
- It has a module named **pyplot** which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc.
- Two type of interfaces for plotting
  - For simple plotting the ***pyplot*** module provides a MATLAB-like interface.
  - For the power user, you have full control of line styles, font properties, axes properties, etc, via an ***Object Oriented Interface***.

Matplotlib (2/3)

- MATLAB-like interface.

Import Pyplot

```
>>> import matplotlib.pyplot as plt
```

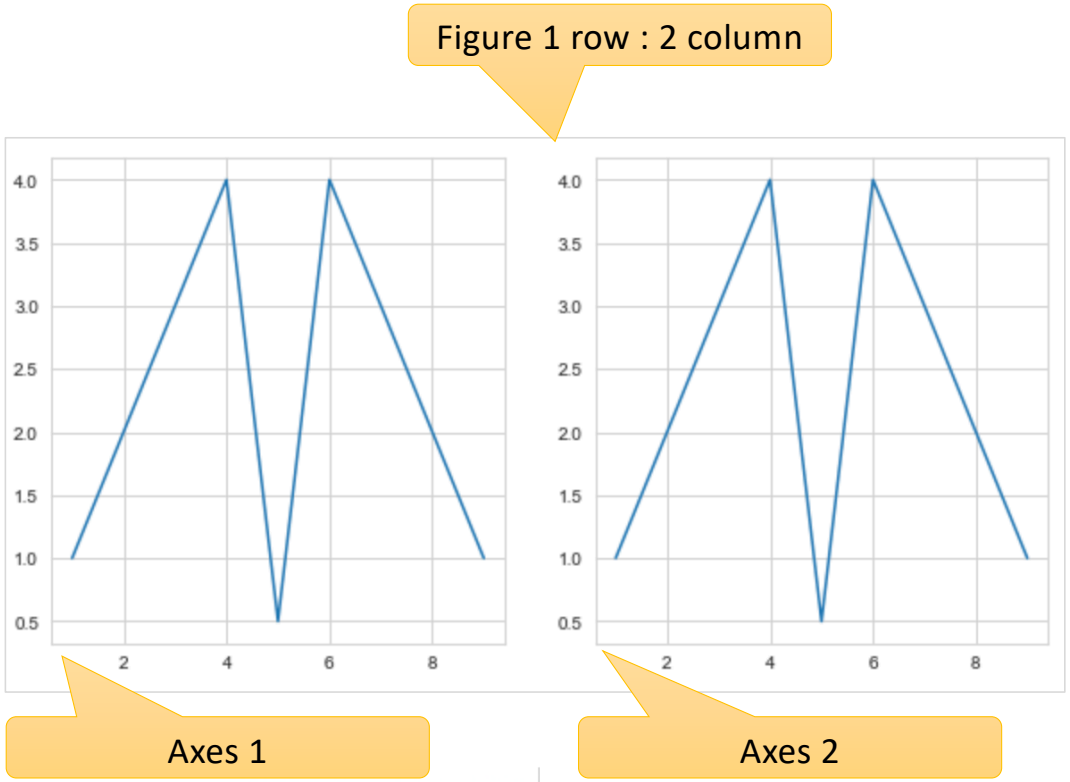
Plotting

```
>>> plt.plot(…)
>>> plt.scatter(…)
>>> plt.bar(…)
```

- Pandas has a built in **.plot()** function as part of the **DataFrame** class using matplotlib.

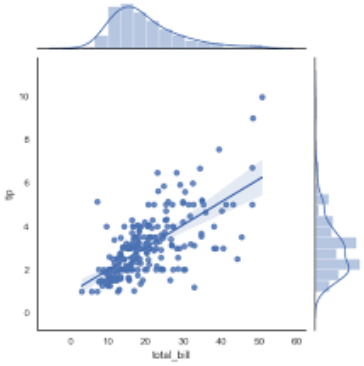
Matplotlib (3/3)

- Object Oriented Interface.
  - A Figure in matplotlib is divided into two different objects.
    - Figure object
    - Axes object
  - A Figure object can contain one or more axes objects. One axes represents one plot inside figure.

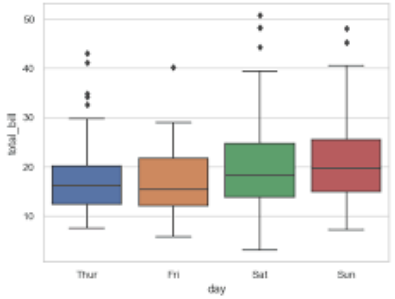


Seaborn

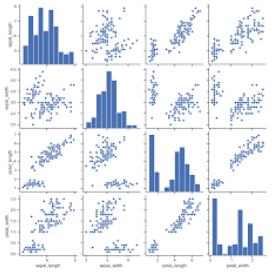
- **Seaborn** is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- **Seaborn** is designed to work very well with Pandas DataFrame objects.



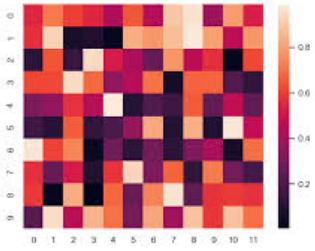
Jointplot



Boxplot



Pairplot

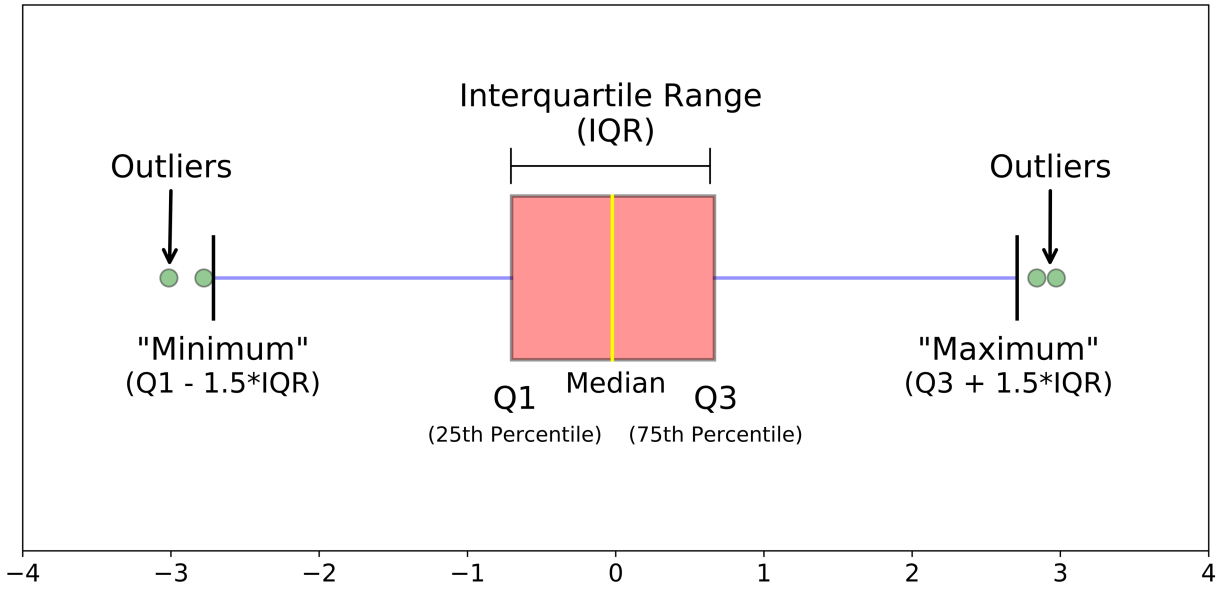


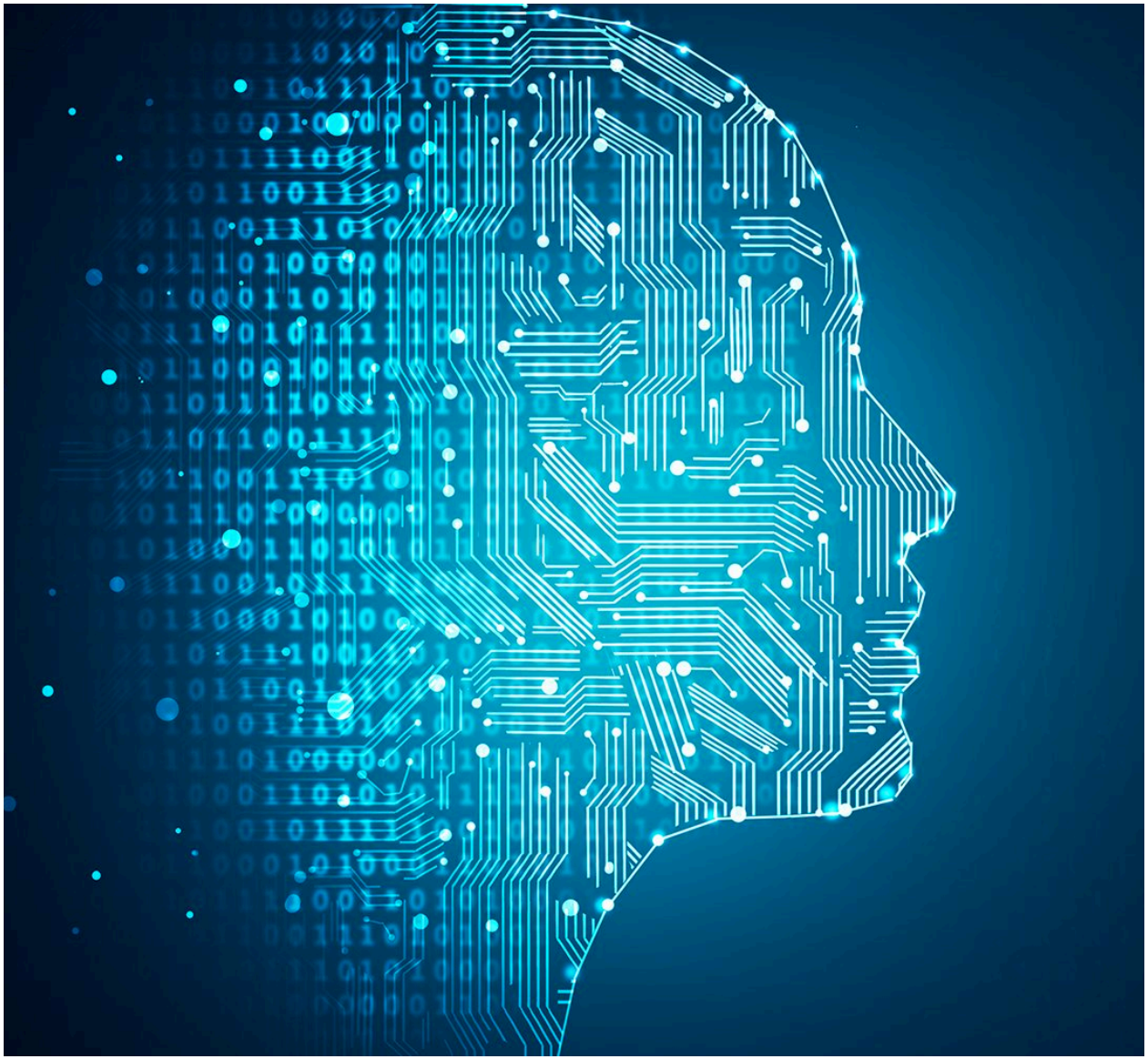
Heatmap

Boxplot

- A boxplot is a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”). It can tell you about your outliers and what their values are.

- **median (Q2/50th Percentile)**: the middle value of the dataset.
- **first quartile (Q1/25th Percentile)**: the middle number between the smallest number (not the “minimum”) and the median of the dataset.
- **third quartile (Q3/75th Percentile)**: the middle value between the median and the highest value (not the “maximum”) of the dataset.
- **interquartile range (IQR)**: 25th to the 75th percentile.
- **whiskers (shown in blue)**
- **outliers (shown as green circles)**
- **“maximum”**: Q3 + 1.5*IQR
- **“minimum”**: Q1 -1.5*IQR





Scikit Learn

Scikit-Learn

- **Scikit-Learn** is python's core machine learning package that has most of the necessary modules to support a basic machine learning project.
- It features various classification, regression and clustering algorithms including: support vector machines, random forests, gradient boosting, k-means and DBSCAN, etc.
- The package is written heavily in python, and it incorporates C++ libraries like LibSVM and LibLinear for support vector machines and generalized linear model implementation.
- The package depends on **Pandas** (mainly for the dataframe processes), **NumPy** (for the ndarray construct) and **SciPy** (for sparse matrices).

https://scikit-learn.org/

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, …

— Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, …

— Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, …

— Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization.

— Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics.

— Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction.

— Examples

Preprocessing

Data Preparation

- Data **pre-processing** refers to the transformations applied to our data before feeding it to the algorithm.
  - Missing Value
  - Categorical Data
  - Feature Scaling

Missing Values (1/2)

- Very common to have data where values are missing for some of the attributes.
- Different ways of handling missing data have difference effects on the performance of our models.
- The approach to deal with missing values is heavily dependent on the nature of the dataset:
  - Ignore the data row.
  - Back-fill or forward-fill to propagate next or previous values respectively.
  - Replace with some constant value outside fixed value.
  - Replace with mean, median value.

Missing Values (2/2)

- Ignore the data row.
  - Preferred in cases where the percentage of missing values is relatively low (<5%).
  - You drop one whole observation just because one of the features had a missing value.
- Back-fill or forward-fill to propagate next or previous values respectively.
  - Note that the NaN value will remain even after forward filling or back filling if a next or previous value isn't available or it is also a NaN value.
- Replace with some constant value outside fixed value range.
  - This method is useful as it gives the possibility to group missing values as a separate category represented by a constant value.
  - The downside is that performance of linear models can suffer.
- Replace with mean, median value.
  - Treat every variable individually, ignoring any interrelationships with other variables.

Categorical Data (1/2)

- Categorical data are variables that contain label values rather than numeric values.
- Some algorithms can work with categorical data directly: **Decision Tree**.
- Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric.
- This involves two steps:
  - Integer Encoding
  - One-Hot Encoding

Categorical Data (2/2)

INTEGER ENCODING

POSITION	ENCODE
First	1
Second	2
Third	3
Fourth	4
…	…

ONE-HOT ENCODING

COLOUR	IS_RED	IS_YELLOW	IS_GREEN
Red	1	0	0
Yellow	0	1	0
Green	0	0	1
…	…	…	…
…	…	…	…

Scaling (1/2)

- Data may contain attributes with a **mixtures of scales** for various quantities such as dollars, kilograms and sales volume.
- Many machine learning methods expect or are **more effective** if the data attributes have the same scale.
- Feature Scaling is a technique to **standardize** the independent features present in the data in a fixed range.
- If feature scaling is not done, then a machine learning algorithm **tends to weigh greater values**, higher and **consider smaller values as the lower values**, regardless of the unit of the values.

Scaling (2/2)

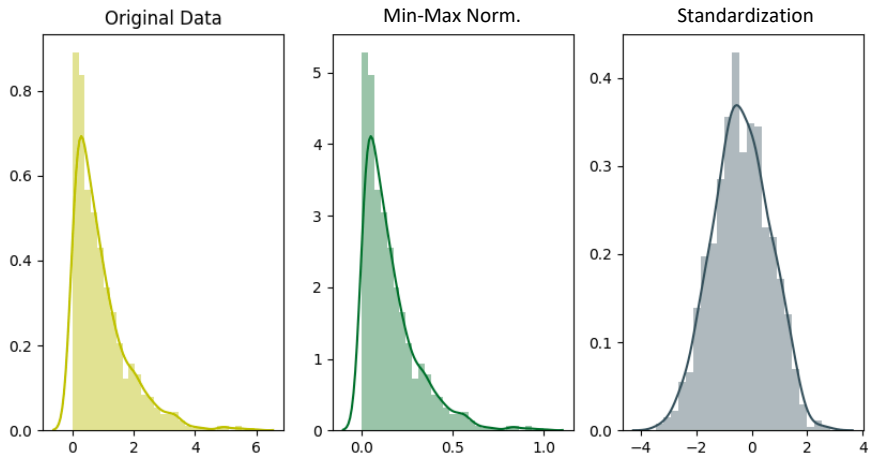
- Two popular data scaling methods are **normalization** and **standardization.**

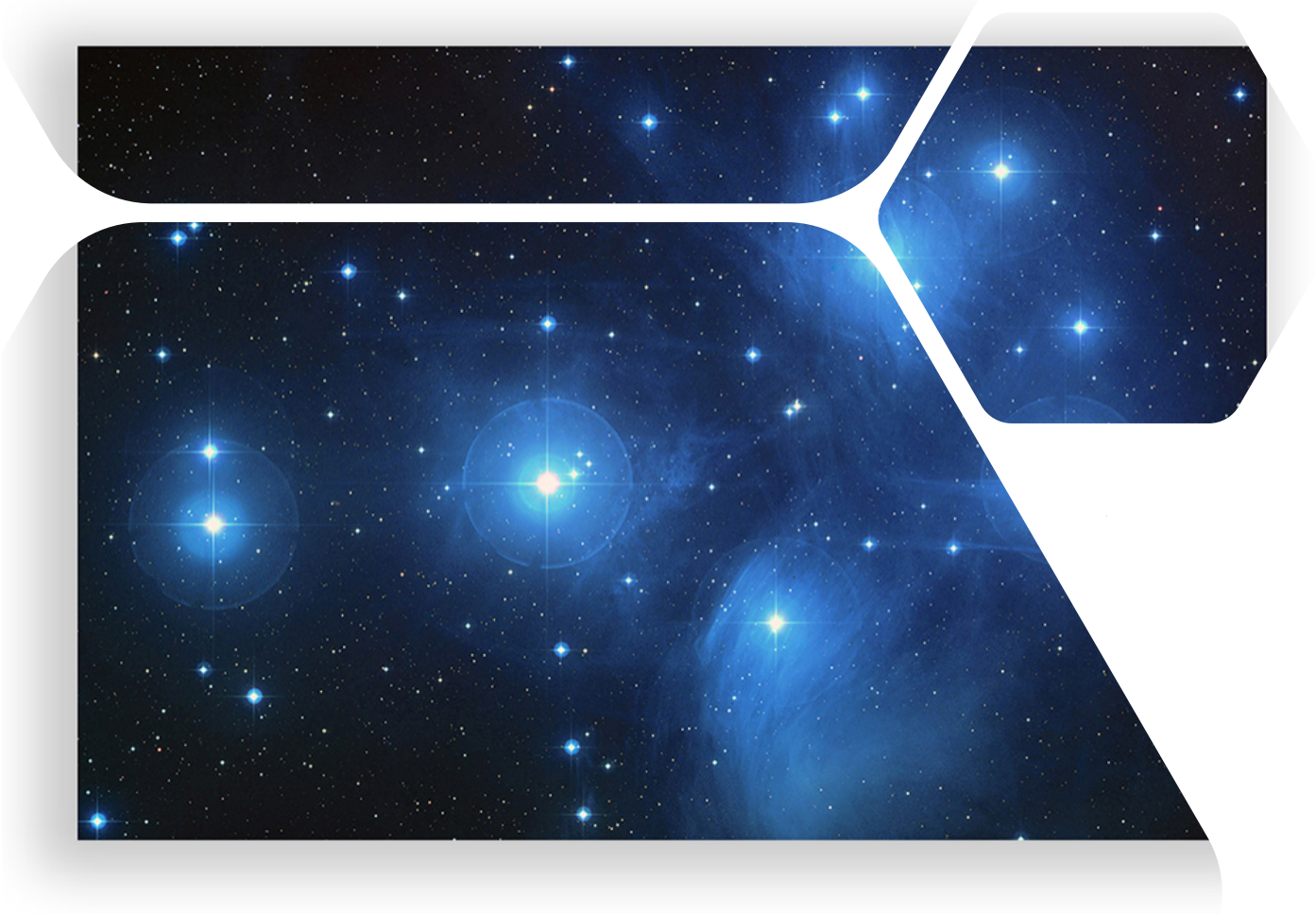
Min-Max Normalization: rescales a feature or observation value with distribution value between 0 and 1.

$$X_{new} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$

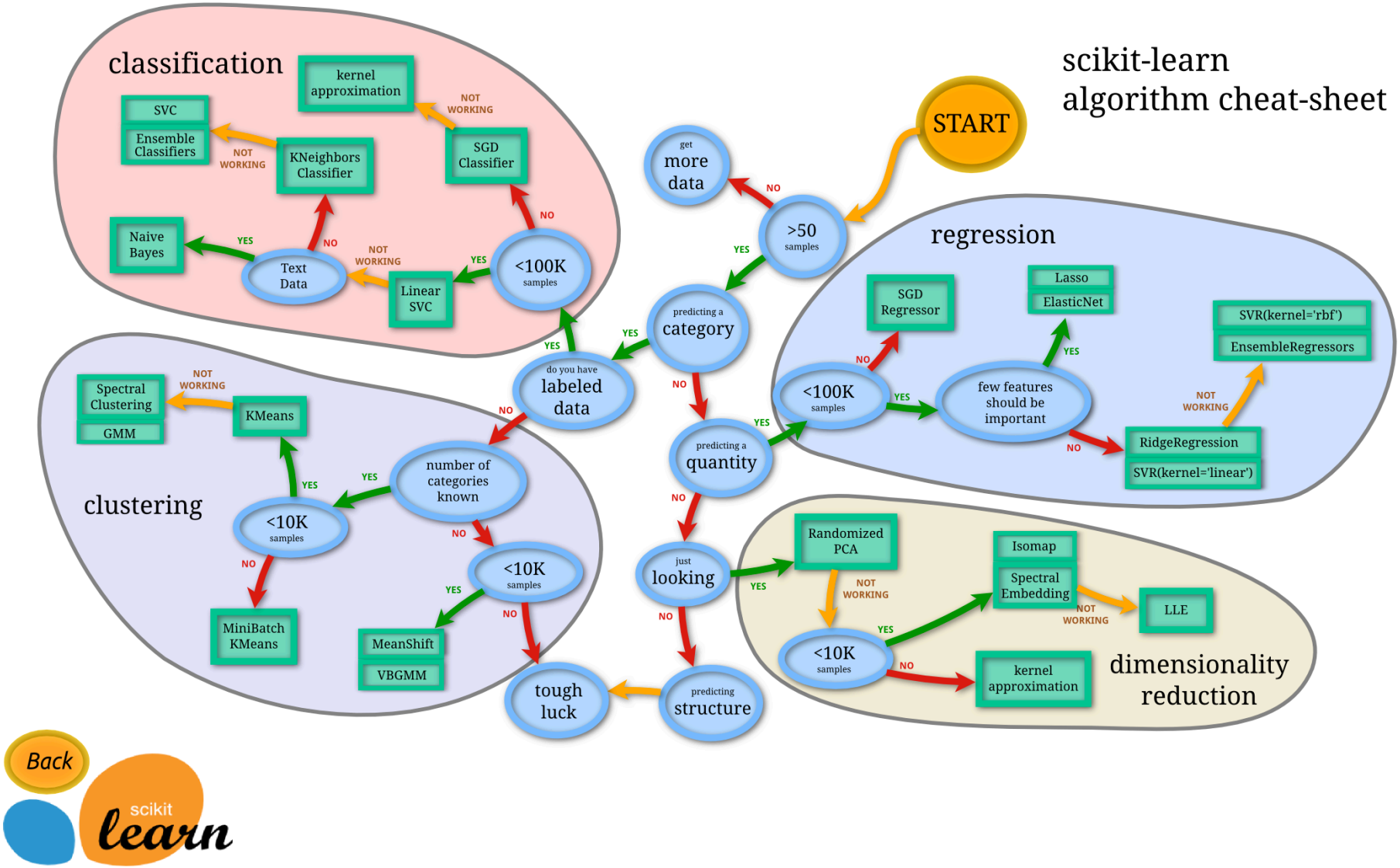
Standardization: rescales a feature value so that it has distribution with 0 mean value and variance equals to 1.

$$X_{new} = \frac{X_i - X_{mean}}{\text{Standard Deviation}}$$

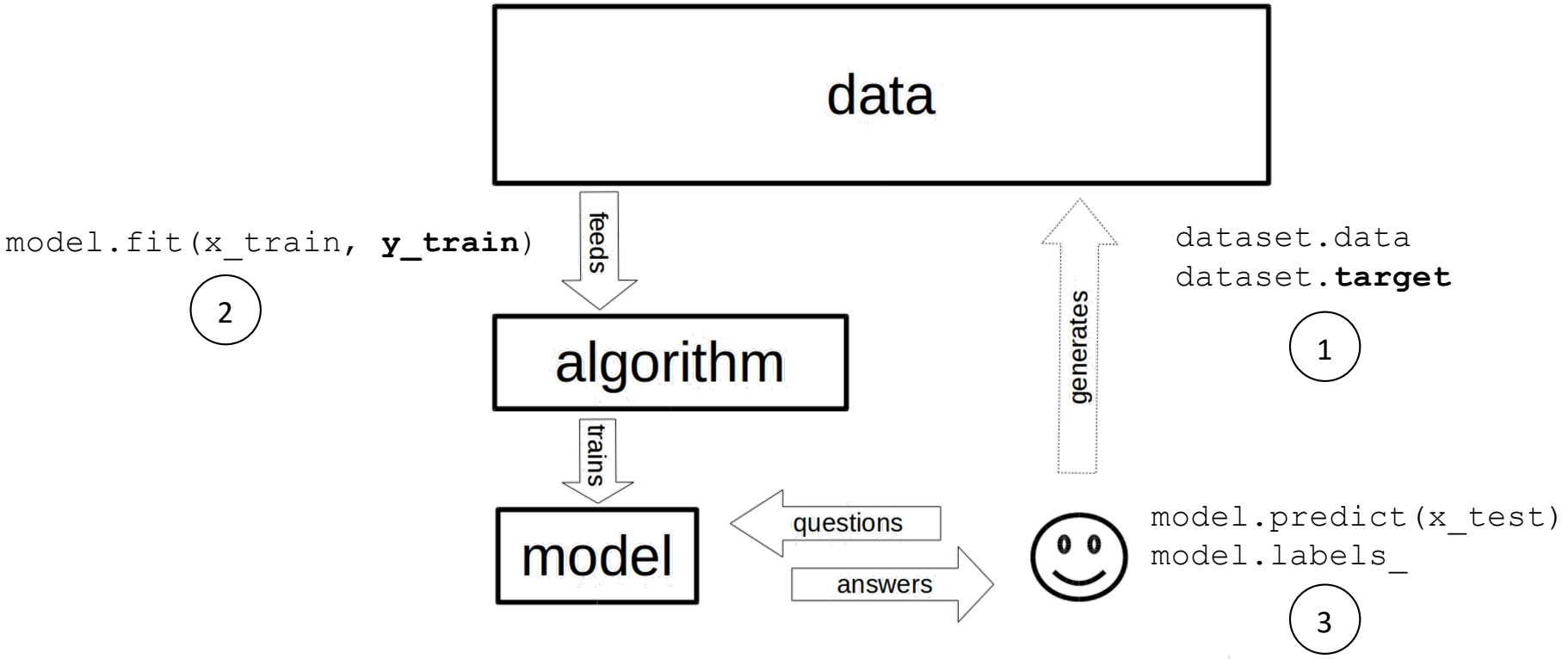




Clustering

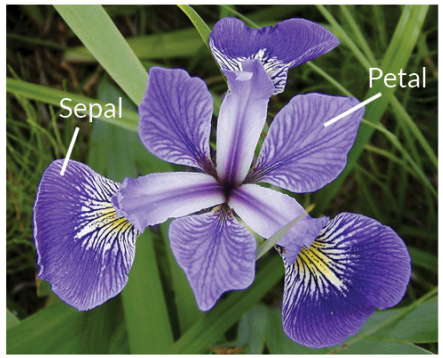

scikit-learn
algorithm cheat-sheet
START
>50 samples
get more data
predicting a category
do you have labeled data
predicting a quantity
just looking
predicting structure
tough luck
<100K samples
SVC
Ensemble Classifiers
kernel approximation
KNeighbors Classifier
SGD Classifier
Naive Bayes
Text Data
Linear SVC
NOT WORKING
YES
NO
classification
Spectral Clustering
GMM
KMeans
number of categories known
<10K samples
MiniBatch KMeans
MeanShift
VBGMM
clustering
SGD Regressor
Lasso
ElasticNet
few features should be important
SVR(kernel='rbf')
EnsembleRegressors
RidgeRegression
SVR(kernel='linear')
regression
Randomized PCA
Isomap
Spectral Embedding
LLE
kernel approximation
dimensionality reduction
Back
scikit learn

Learning Process



Iris Dataset

- Number of Instances: 150 (50 in each of three classes)
- Number of Attributes: 4 numeric
  - sepal length in cm
  - sepal width in cm
  - petal length in cm
  - petal width in cm
- Class:
  - Iris Setosa
  - Iris Versicolour
  - Iris Virginica



Iris Versicolor


Iris Setosa


Iris Virginica

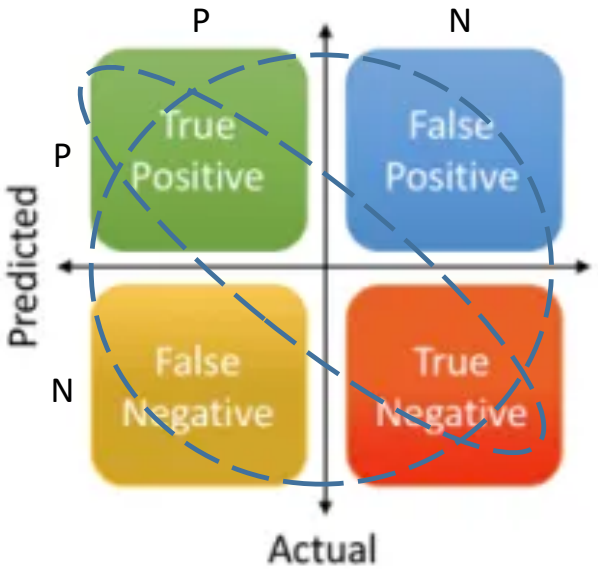
Evaluating Classification Algorithm (1/4)

Accuracy: is the most intuitive performance metric, it is simply the ratio of correctly predicted observations to total observations

$$Accuracy = \frac{\boldsymbol{correcly\ predicted}\ observation}{total\ observations}$$

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

$$Accuracy = \frac{True\ Positive + True\ Negative}{Total}$$



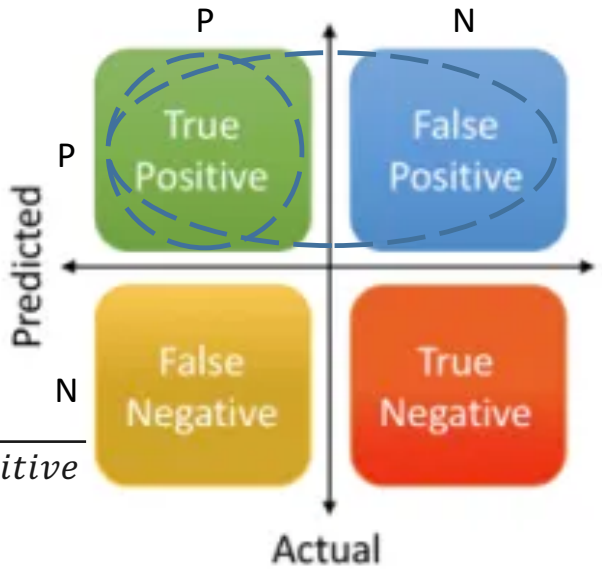
Evaluating Classification Algorithm (2/4)

Precision: is the ratio of correctly predicted ***positive*** observations of the total predicted ***positive*** observations.

$$Precision = \frac{\textbf{correctly}\ predicted\ \textbf{positive}\ observation}{total\ \textbf{predicted positive}\ observations}$$

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

$$Precision = \frac{True\ Positive}{Predicted\ Results}\ or\ \frac{True\ Positive}{True\ Positive + False\ Positive}$$



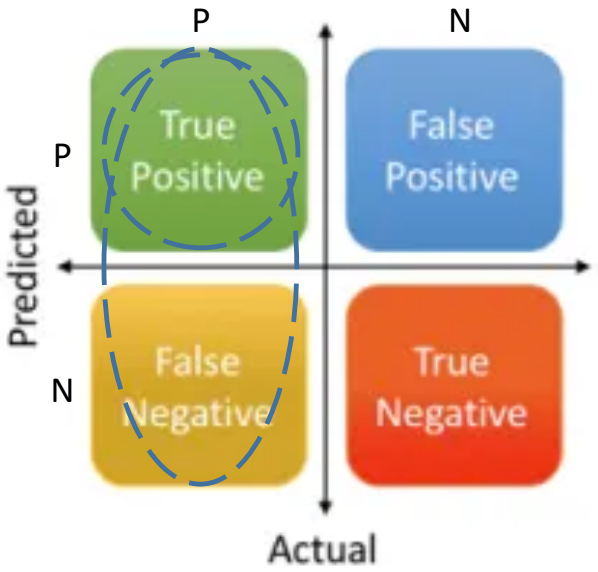
Evaluating Classification Algorithm (3/4)

Recall (Sensitivity): is the ratio of the **positive** observations correctly predicted to all observations in the actual class – yes.

$$Recal = \frac{\textbf{correctly}\ predicted\ \textbf{positive}\ observation}{total\ \textbf{actual positive}\ observations}$$

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

$$Recal = \frac{True\ Positive}{Actual\ Results}\ or\ \frac{True\ Positive}{True\ Positive + False\ Negavite}$$

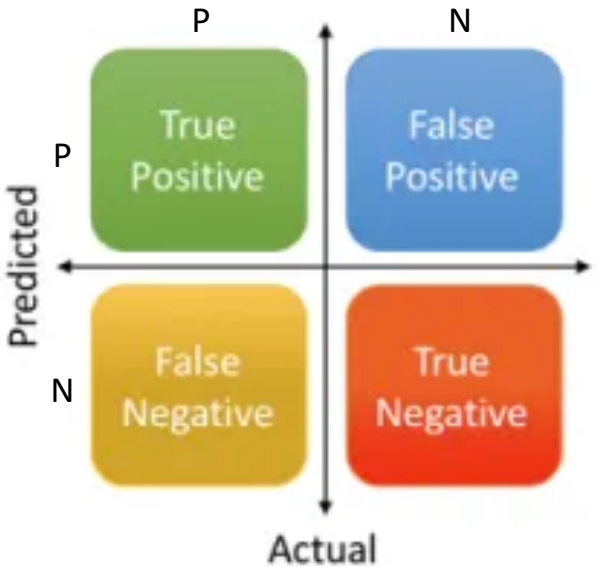


Evaluating Classification Algorithm (4/4)

F1 Score: is the harmonic mean of Precision and Recall.

$$F1\ score = 2\ \frac{recall \times accuracy}{recall + accuracy}$$

- Accuracy is best if false positives and false negatives have similar costs.
- If the cost of false positives and false negatives is very different, then it's best to look at Precision and Recall at the same time.

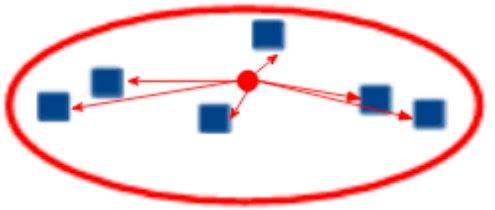


Evaluating Clustering Algorithm (1/5)

Inertia

- Within-cluster sum of squares of distances to the cluster center.
- It's a measure of how internally coherent clusters are.
- It suffers from various drawbacks:
  - Inertia makes the assumption that clusters are convex and isotropic.
  - Inertia is not a normalized metric, we just know that lower values are better and zero is optimal.

$$\sum_{i=0}^{n} \min_{\mu_j \in C}(||x_i - \mu_j||^2)$$

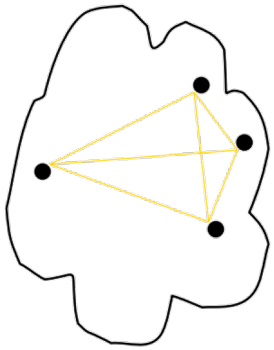


Evaluating Clustering Algorithm (2/5)

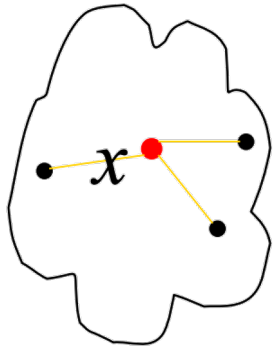
Cohesion

- Measures how closely related are points in a cluster.

$$Cohesion(C) = \sum_{x,y \in C} proximity(x, y)$$



$$Cohesion(C) = \sum_{x \in C} proximity(x, centroid)$$

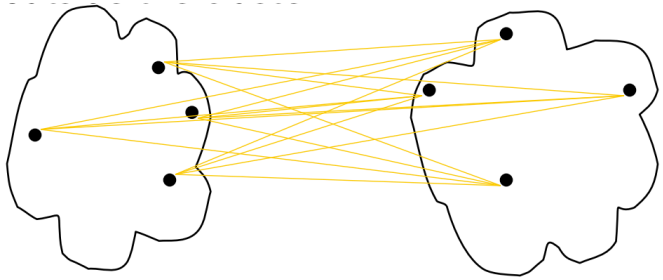


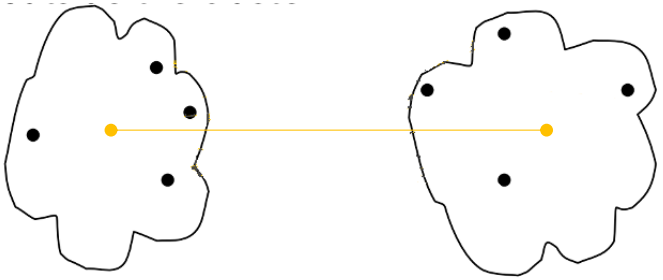
Evaluating Clustering Algorithm (3/5)

Separation

- Measure how distinct or well separated a cluster is from other clusters.

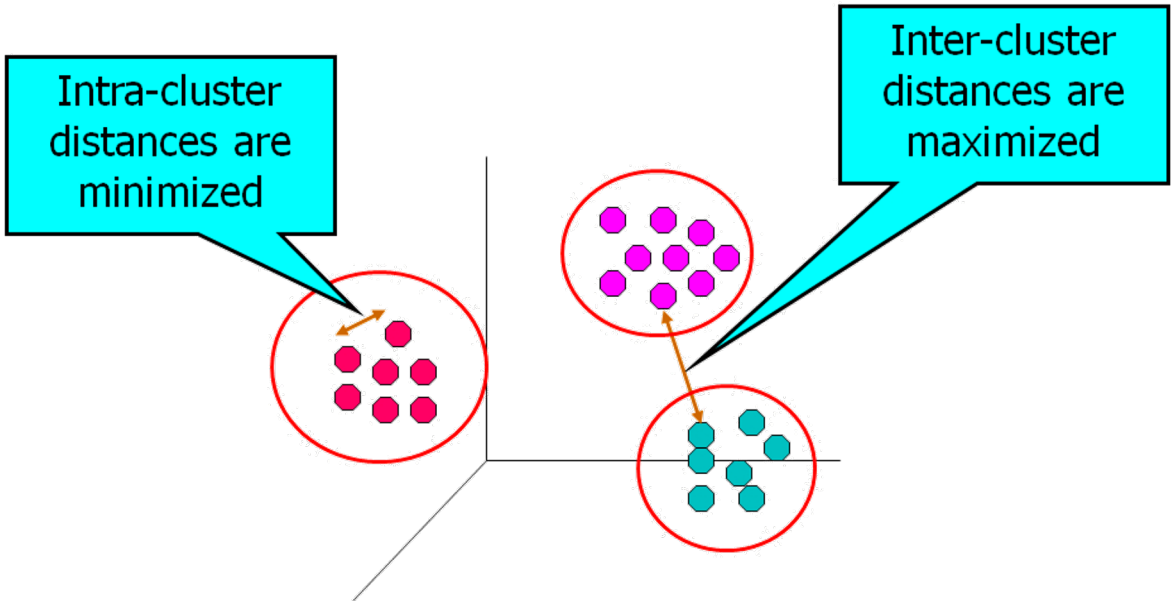
$$Separation(C_i, Cj) = \sum_{\substack{x \in Ci \\ y \in Cj}} proximity(x, y) \qquad Separation(C_i, Cj) = proximity(centroid_i, centroid_j)$$





Evaluating Clustering Algorithm (4/5)

- **Cohesion vs Separation**



Evaluating Clustering Algorithm (3/5)

Silhouette Coefficient

Combination of the mean cohesion and separation.

$$Silhouette = \frac{b - a}{\max(a, b)}$$

$a = cohesion$
$b = separation$

Normalization factor

Range from [-1, 1]

- -1: poor clustering
- 0: indifferent
- +1: excellent clustering

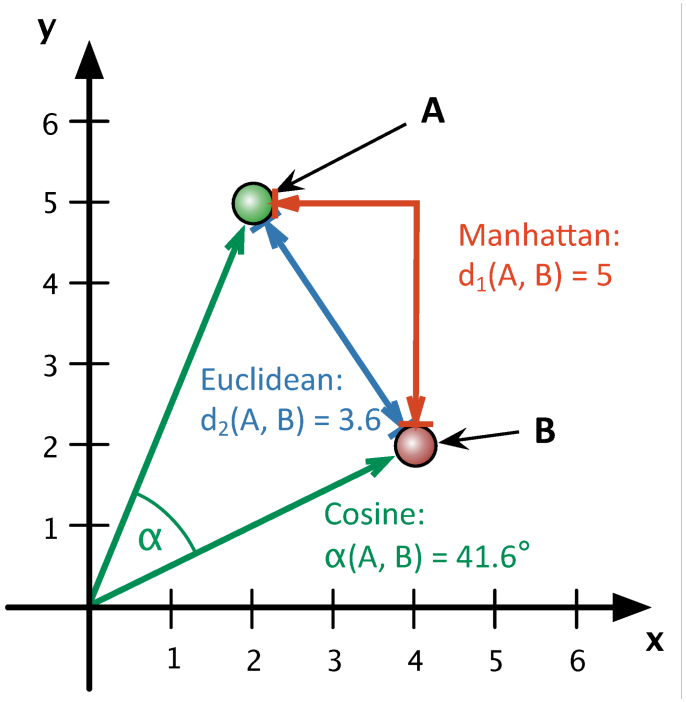
Agglomerative Clustering (1/2)

Affinity

$$Manhattan \Rightarrow d_1(A,B) = \sum_{i=1}^{n} |a_i - bi|$$

$$Euclidean \Rightarrow d_2(A,B) = \sqrt{\sum_{i=1}^{n} (ai - bi)}$$

$$Cosine \Rightarrow \cos(\alpha) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sum_{i=1}^{n} a^2{}_i \sum_{i=1}^{n} b^2{}_i}$$

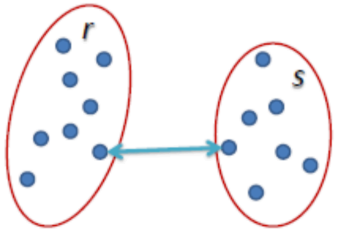


Agglomerative Clustering (2/2)

Linkage

Single

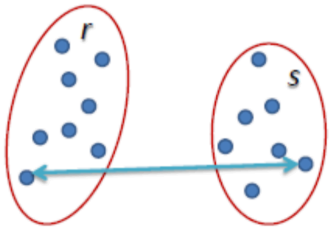
The distance between two clusters is the shortest distance between two points in each cluster.



$$L(r,s) = \min(D(x_{ri}, x_{sj}))$$

Complete

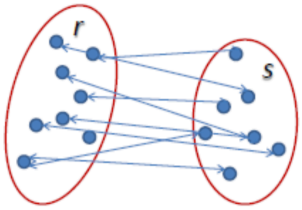
The distance between two clusters is the longest distance between two points in each cluster.



$$L(r,s) = \max(D(x_{ri}, x_{sj}))$$

Average

The distance between clusters is the average distance between each point in one cluster to every point in other cluster.



$$L(r,s) = \frac{1}{n_r n_s} \sum_{i=1}^{n_r} \sum_{j=1}^{n_s} D(x_{ri}, x_{sj})$$

Reference

- Pandas (http://pandas.pydata.org)
- Scikit-Learn (https://scikit-learn.org)

- NumPy (http://www.numpy.org/)
- Matplotlib (https://matplotlib.org/)
- Seaborn (https://seaborn.pydata.org/)

- Jupyter (https://jupyter.org/)