04 – Unsupervised Learning

Data Science and Management

Corso di Laurea Magistrale in Ingegneria Gestionale

Marco Mamei, Natalia Hadjidimitriou, Fabio D'Andreagiovanni, Matteo Martinelli

{marco.mamei, selini, fabio.dandreagiovanni, matteo.martinelli}@unimore.it

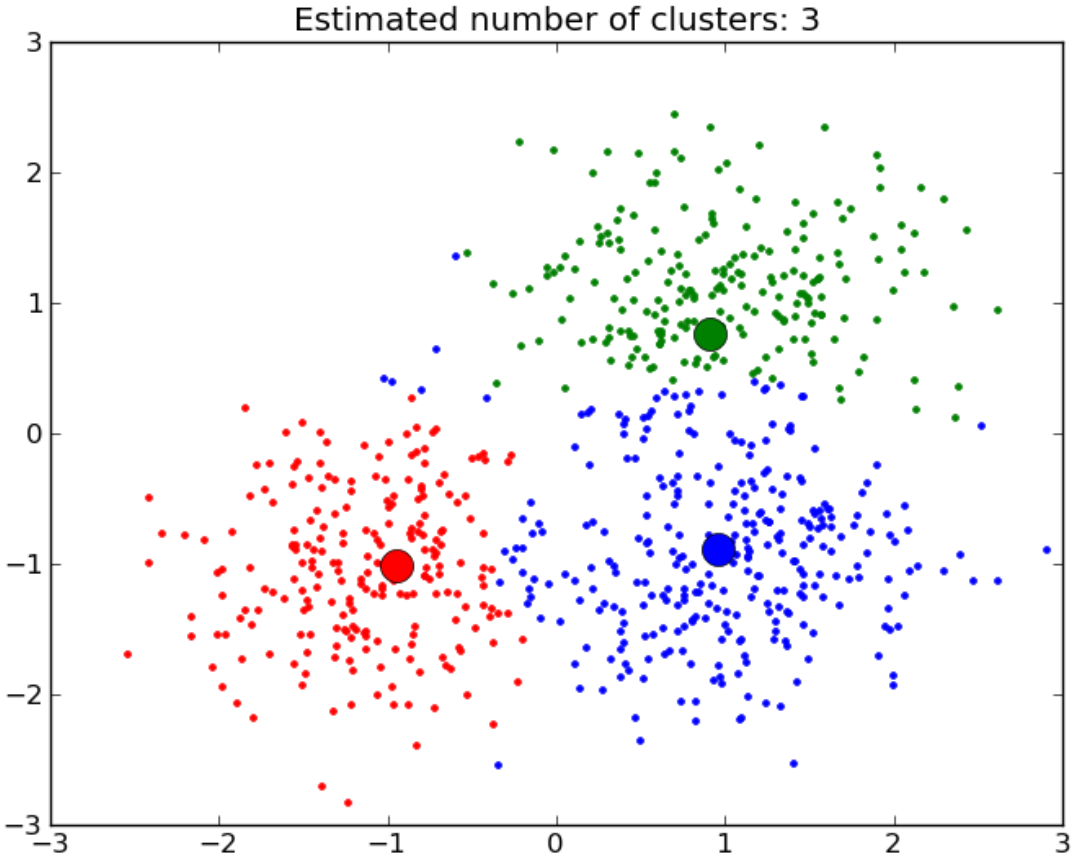
- Clustering
- Dimensionality Reduction

Clustering

Clustering is a form of **unsupervised** learning

Supervision is costly and often hard to get

Given a set of **examples** (points in some high-dimensional space) find **groups** of similar elements



Clustering

Examples

- Find Amazon **users** sharing **similar** characteristics: for the **kind of objects** they buy, for the **budget** they exploit, for the **frequency** of orders they make
- Group Facebook **users** into communities because of their **friendships** and the **contents** they like
- Given a collection of **text documents**, find those regarding the **same topic**

Clustering

No supervisions implies…

- No **target** to be computed
- No straightforward **performance measure** to **evaluate** the success of our algorithm
- **Heuristic** approaches to assess validity of results
- Partial lack of **theoretical** grounds

Clustering

Main idea of clustering algorithms

- Partition data samples into K sets (named **clusters**)
  - All examples belong to one and only one cluster
- High **intra**-cluster **similarity**
- High **inter**-cluster **dissimilarity**
- Based on a **distance/metric** between examples

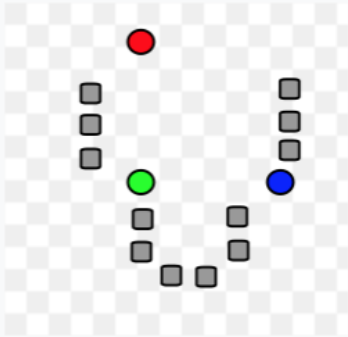
Clustering

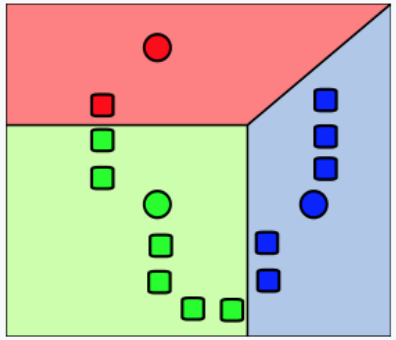
The most famous clustering algorithm is **K-Means**:

- Choose a number of clusters K (**in advance**!!!)
- Start with K **random** cluster centroids
- **Assign** each point to the **closest** centroid
- Compute **new** cluster centroids
- Iterate until **convergence** (?)

Clustering

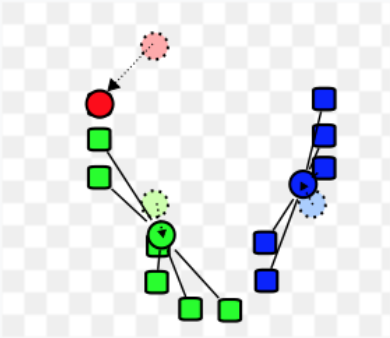
1. INIT

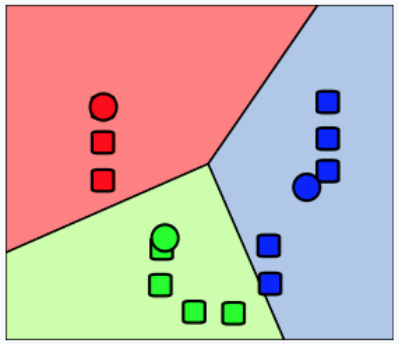




2. ASSIGN

3. COMPUTE





4. CONVERGE

Image from Wikipedia

Clustering

K-Means algorithm

```
1. init centroids (K random points)

2. repeat

2a.    assignment

2b.    compute

3. until stopping criterion
```

$$s(x) = \arg\min_{c_i \in C} d(c_i, x)$$

$$c_i = \frac{1}{|x, s(x) = c_i|} \sum_{x, s(x)=c_i} x$$

Clustering

Pros

- Straightforward and **intuitive** algorithm

Cons

- Depends on **random initialization**
- Choose K **in advance**
- Clustering only by **linear separations**

Clustering

Clustering indices:

INTERNAL: LABELS NOT NEED TO BE KNOWN

- Dunn Index, Silhouette, …
- Homogeneity, Completeness, V-Measure, …

EXTERNAL: LABELS NEED TO BE KNOWN

Clustering

Dunn Index

Minimum intra-cluster similarity

Maximum inter-cluster similarity

$$DI = \frac{\min_{i,j|C(i)=C(j)} s(x_i, x_j)}{\max_{i,j|C(i)\neq C(j)} s(x_i, x_j)}$$

Silhouette

Average dissimilarity of i with points in the same cluster

$$s = \frac{1}{n}\sum_{I=1}^{n} \frac{b(i) - \boxed{a(i)}}{max\{a(i), \boxed{b(i)}\}}$$

-1 <= s <= 1

Minimum average dissimilarity of i with points of other clusters

Clustering

Homogeneity (H)
- Each cluster contains only points of a single label

Completeness (C)
- All points of a label are assigned to the same cluster

V-Measure
- Harmonic mean between H and C

Clustering

Hierarchical clustering

- Start with **pairwise distances** between data points
- Iteratively **agglomerate** clusters
- Need to define a **distance between groups**
- At the end we obtain a **tree**, named **dendrogram**

Clustering

Dendrogram

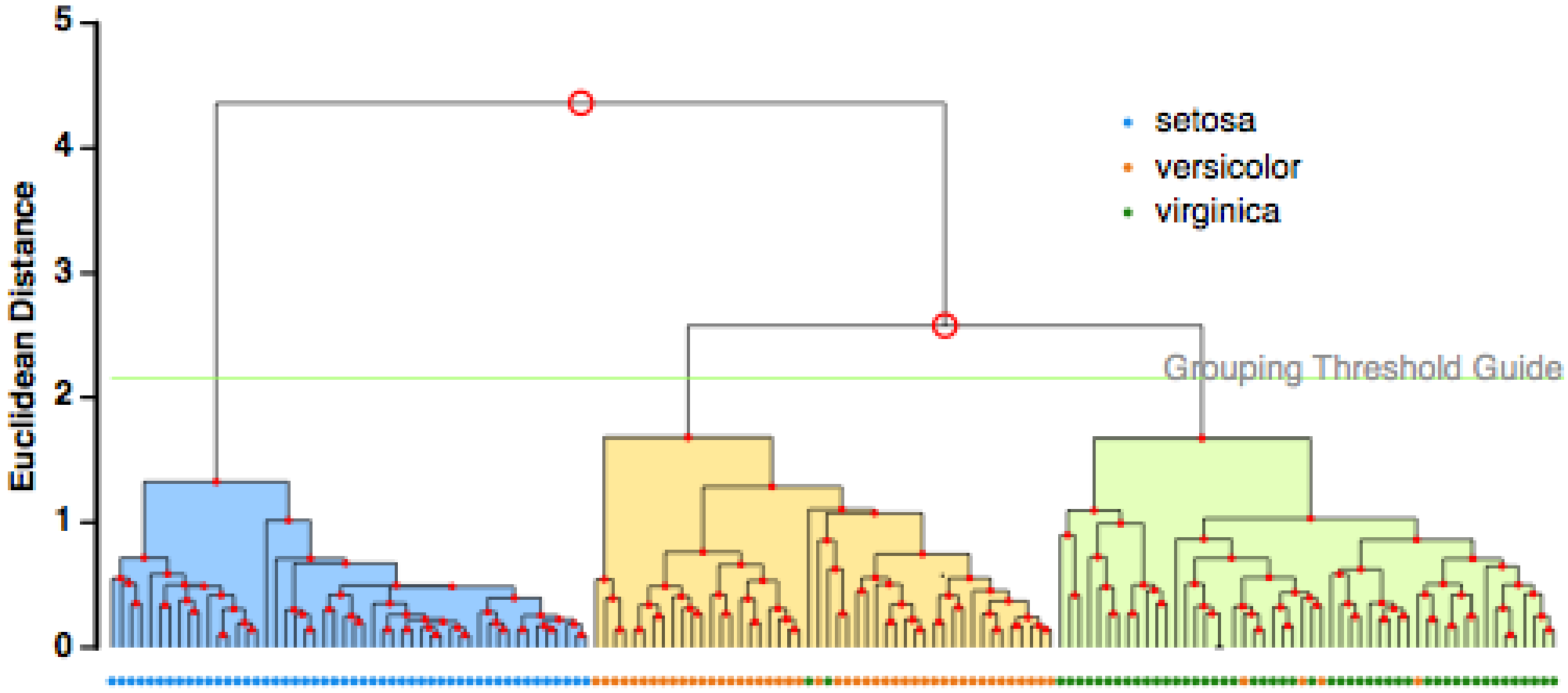
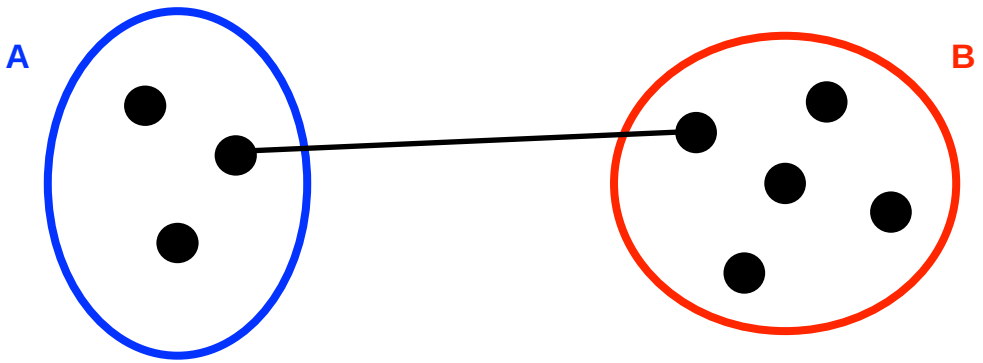


Figure from https://www.gigawiz.com/hcluster2.html

Clustering

Distances between groups of points:

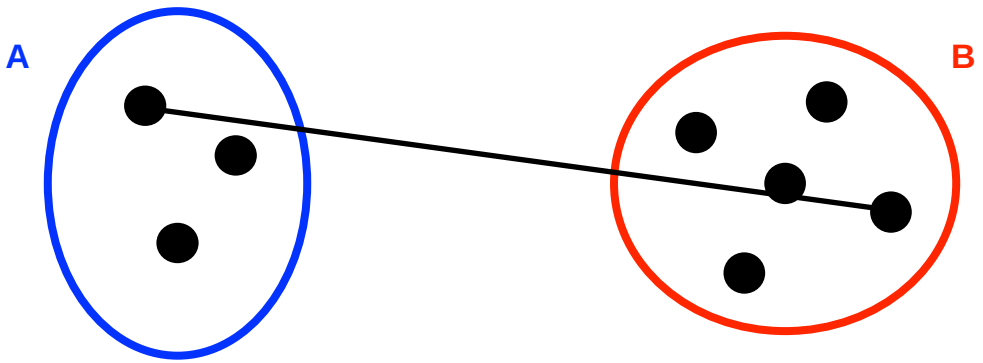


SINGLE LINKAGE

$$d(A,B) = \min_{x \in A, y \in B} d(x,y)$$

Clustering

Distances between groups of points

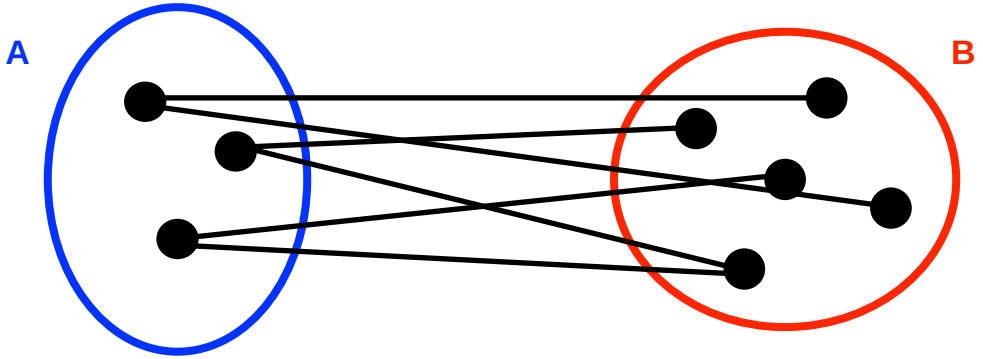


COMPLETE LINKAGE

$$d(A, B) = \max_{x \in A, y \in B} d(x, y)$$

Clustering

Distances between groups of points

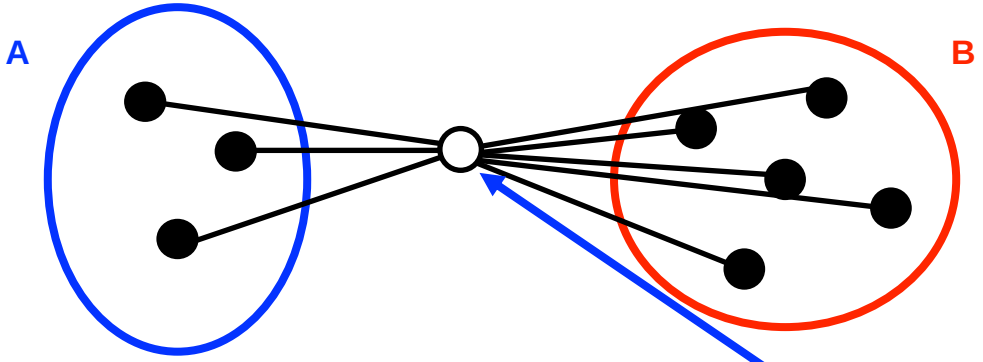


AVERAGE LINKAGE

$$d(A,B) = \frac{1}{|A||B|} \sum_{x \in A, y \in B} d(x,y)$$

Clustering

Improvement in total distance in case of merging



WARD'S METHOD

$$d(A,B) = \sum_{x \in A \cup B} d(x,\mu)$$

NEW CENTROID IN CASE OF MERGING

Clustering

Many other algorithms…

- Spectral clustering
- Affinity propagation
- DBScan (it also estimates K)
- …

Example with Scikit-Learn

Iris dataset

- A very famous dataset in machine learning
- Measurements in cm of sepal/petal width/length for 50 **flowers** from each of 3 species of **iris**
- 150 examples, 4 features, 3 classes

Example with Scikit-Learn

```
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.metrics import homogeneity_completeness_v_measure

X, y = datasets.load_iris(return_X_y=True)

kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
print(kmeans.inertia_)
print(kmeans.cluster_centers_)
print(kmeans.labels_)
print(homogeneity_completeness_v_measure(y, kmeans.labels_))
```

Example with Scikit-Learn

```
predictions = kmeans.labels_
colors = np.array(['red', 'blue', 'green'])
plt.scatter(X[:, 2], X[:, 3], color=colors[predictions])
plt.show()
```

PLOT EACH POINT AT A TIME (NOT EFFICIENT)

```
predictions = kmeans.labels_
colors = np.array(['red', 'blue', 'green'])
markers = ['o', 'x', 's']
for i in range(len(X)):
    plt.scatter(X[i,2], X[i,3], marker=markers[y[i]], color=colors[predictions[i]])
plt.show()
```

Example with Scikit-Learn

```
from sklearn import datasets
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
import numpy as np

X, y = datasets.make_moons(n_samples=1000, noise=0.05)

clusters = AgglomerativeClustering(n_clusters=2, linkage='single').fit(X)
predictions = clusters.labels_
colors = np.array(['red', 'blue'])
plt.scatter(X[:, 0], X[:, 1], color=colors[predictions])
plt.show()
```

Dimensionality Reduction

Unsupervised

- Principal Component Analysis (PCA)
- …

Supervised

- Linear Discriminant Analysis (LDA)
- …

Principal Component Analysis

Principal Component Analysis (PCA) is a technique for **mapping** a set of examples, represented by **many** features, to a **lower-dimensional** space, where the main data characteristics are **preserved**

It is particularly useful for **exploratory data analysis**, for example to **visualize** data with many features

Principal Component Analysis

Suppose to have N examples with P features

How to **project** the P-dimensional space into a **two-dimensional** space, preserving distances between data points, so that we can **plot** the examples?

We look for **principal components** of data

Principal Component Analysis

PCA is an orthogonal **linear** transformation that maps the original matrix X of data from the original feature space into a lower-dimensional space

$$T = XW$$

$$X \in \mathbb{R}^{N \times P}, \quad W \in \mathbb{R}^{P \times Q}, \quad T \in \mathbb{R}^{N \times Q}$$

In particular, we search for the projection W that maximizes the **variance** within data (in order to better **discriminate** among data)

Principal Component Analysis

There are several ways to derive PCA…
One of the most general algorithms is based on the computation of the **covariance matrix**

$$C_{ij} = \mathrm{cov}(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)]$$

Then, the **eigenvectors** of matrix C are computed and **ordered** by **descending eigenvalues**

The **largest eigenvectors** correspond to the principal components of the original data

Example with Scikit-Learn

```
from sklearn import datasets
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np

X, y = datasets.load_iris(return_X_y=True)

pca = PCA(n_components=2)
T = pca.fit_transform(X)
colors = np.array(['red', 'blue', 'green'])
plt.scatter(T[:,0], T[:,1], color=colors[y])
plt.show()
```