Data Analysis Lab

Clustering

Hierarchical Clustering

Fátima Leal

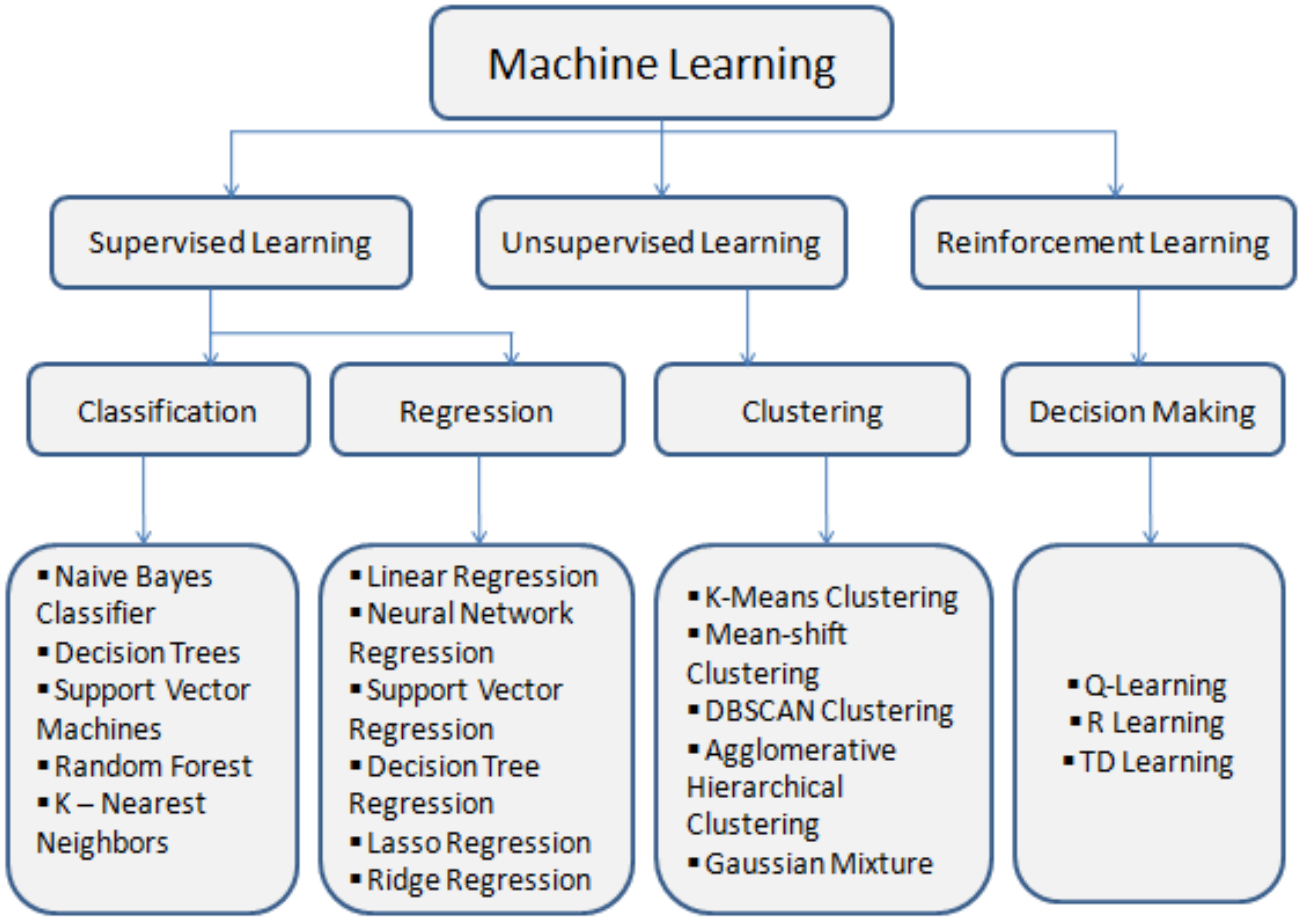
DCT DEPARTAMENTO CIÊNCIA E TECNOLOGIA

UPT UNIVERSIDADE PORTUCALENSE

Outline

- Clustering
- K-Means Clustering
- Hierarchical Clustering
- Examples

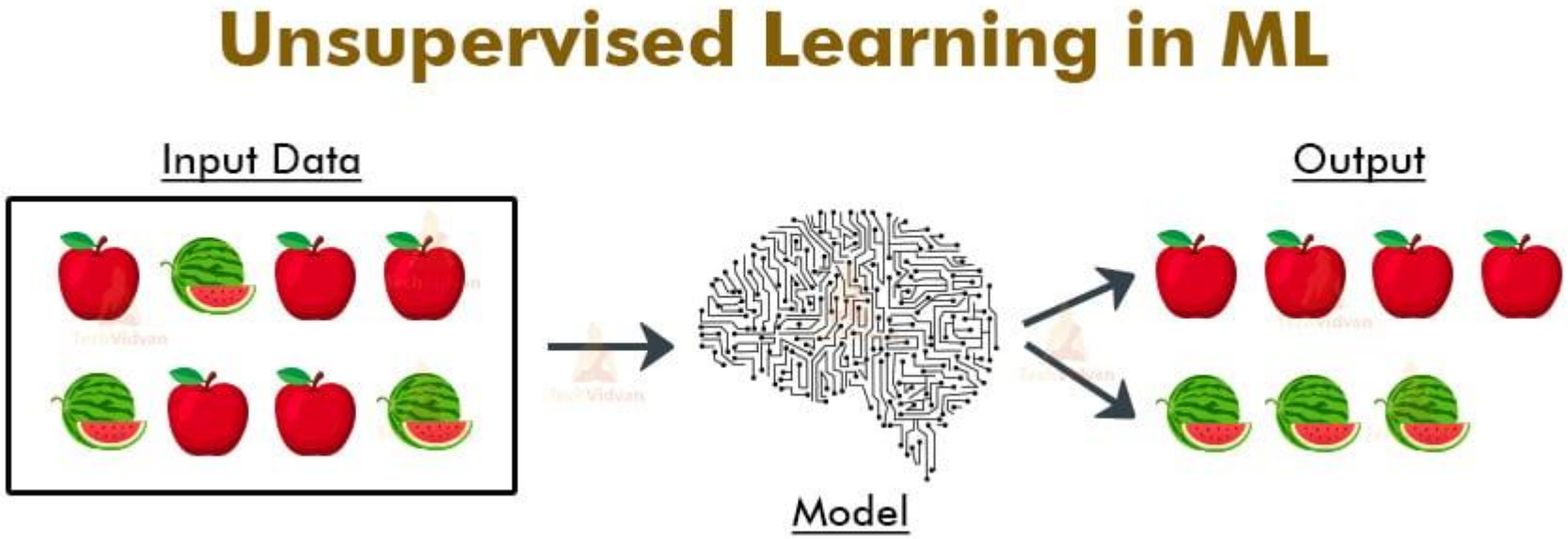
Supervised vs Unsupervised Learning



Unsupervised Learning

- **Unsupervised learning** cannot be directly applied to a regression or classification problem because we have the input data but no corresponding output data.
- The goal of unsupervised learning is to find the underlying structure of dataset, group that data according to similarities
- Example:

Unsupervised Learning



Clustering

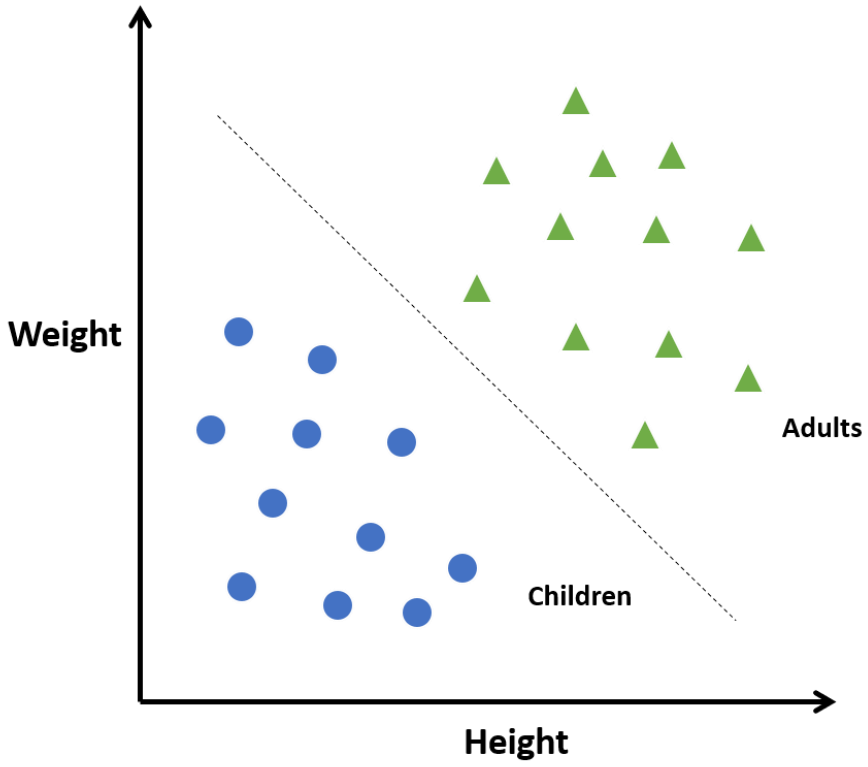
- Let's consider a dataset of points: $X = \{\bar{x}_1, \bar{x}_2, ..., \bar{x}_n\}$ where $\bar{x}_i \in \mathrm{R}^m$
- We assume that it's possible to find a criterion (not unique) so that each sample can be associated with a specific group: $g_k = G(\bar{x}_i)$ where $k = \{0, 1, 2, ..., t\}$
- Conventionally, each group is called a cluster and the process of finding the function G is called clustering
- Different clustering algorithms are based on alternative strategies to solve this problem, and can yield very different results.
- There's an example of clustering based on four sets of bidimensional samples; the decision to assign a point to a cluster depends only on its features and sometimes on the position of a set of other points (neighborhood)

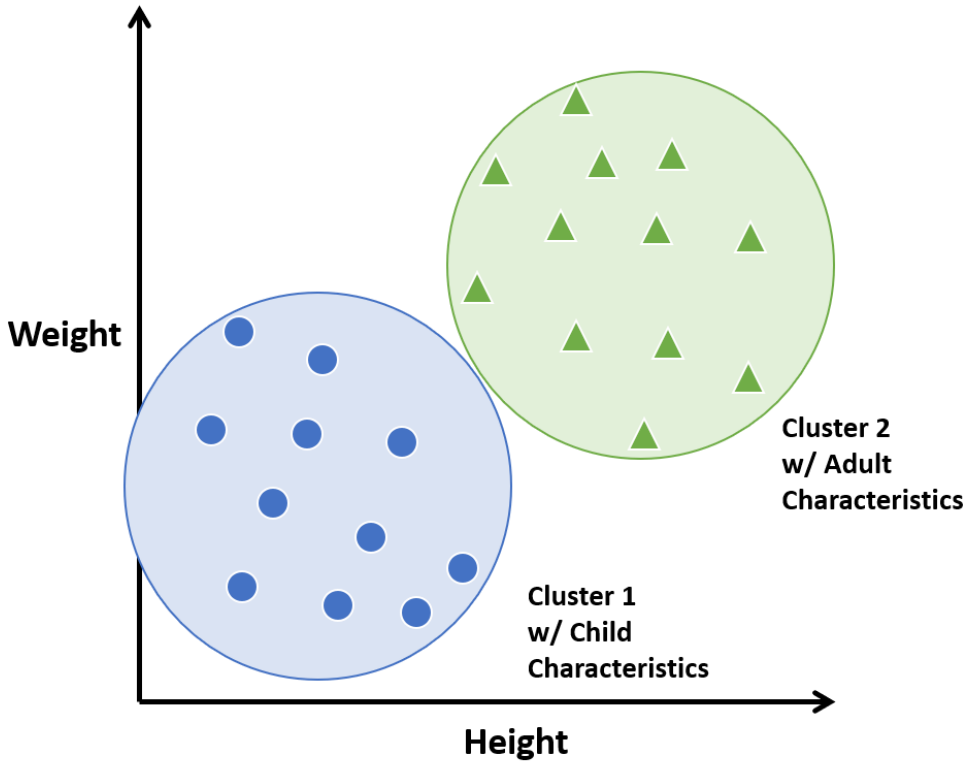
Clustering

- We can have hard clustering techniques, where each element must belong to a single cluster. The alternative approach, called soft clustering, is based on a membership score that defines how much the elements are "compatible" with each cluster.
- The generic clustering function becomes: $\bar{m}_i = F(\bar{x}_i)$ where $\bar{m}_i = (m_i^0, m_i^2, ..., m_i^t)$ and $m_i^k \in [0, 1]$
- A vector m_i represents the relative membership of x_i, and it's often normalized as a probability distribution.

Clustering

Classification vs Clustering

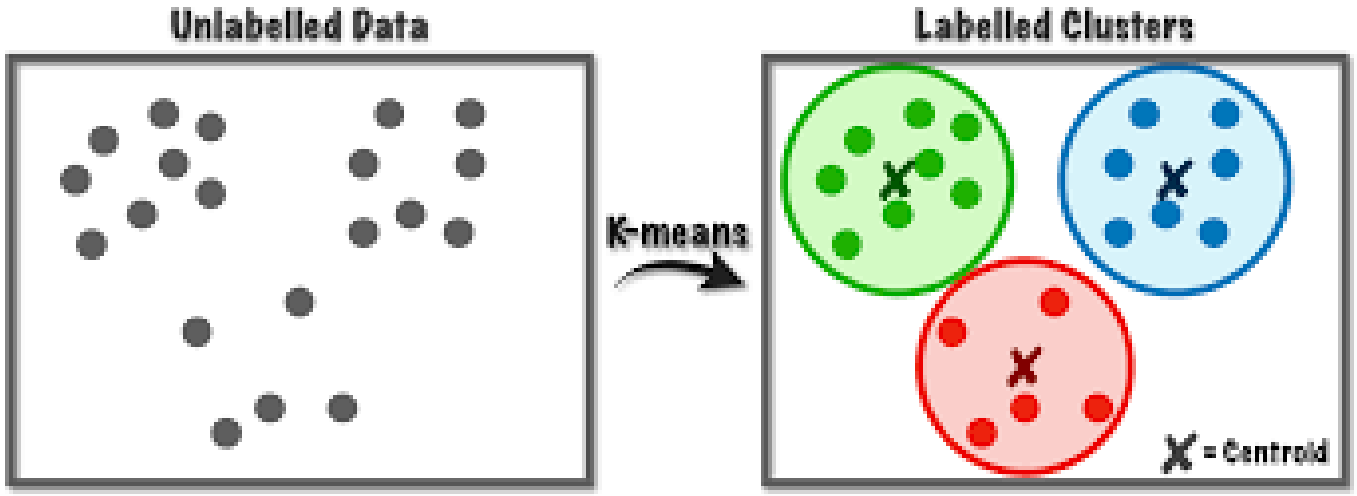




K-means

- The k-means algorithm is based on the (strong) initial condition to decide the number of clusters through the assignment of k initial centroids or means: $K_{(0)} = \{\mu_1^{(0)}, \mu_2^{(0)}, ..., \mu_k^{(0)}\}$
- Then the distance between each sample and each centroid is computed and the sample is assigned to the cluster where the distance is minimum. This approach is often called minimizing the inertia of the clusters, which is defined as follows:
$SS_{w_i} = \sum ||x_t - \mu_i||^2 \forall i \in (1, k)$

K-means

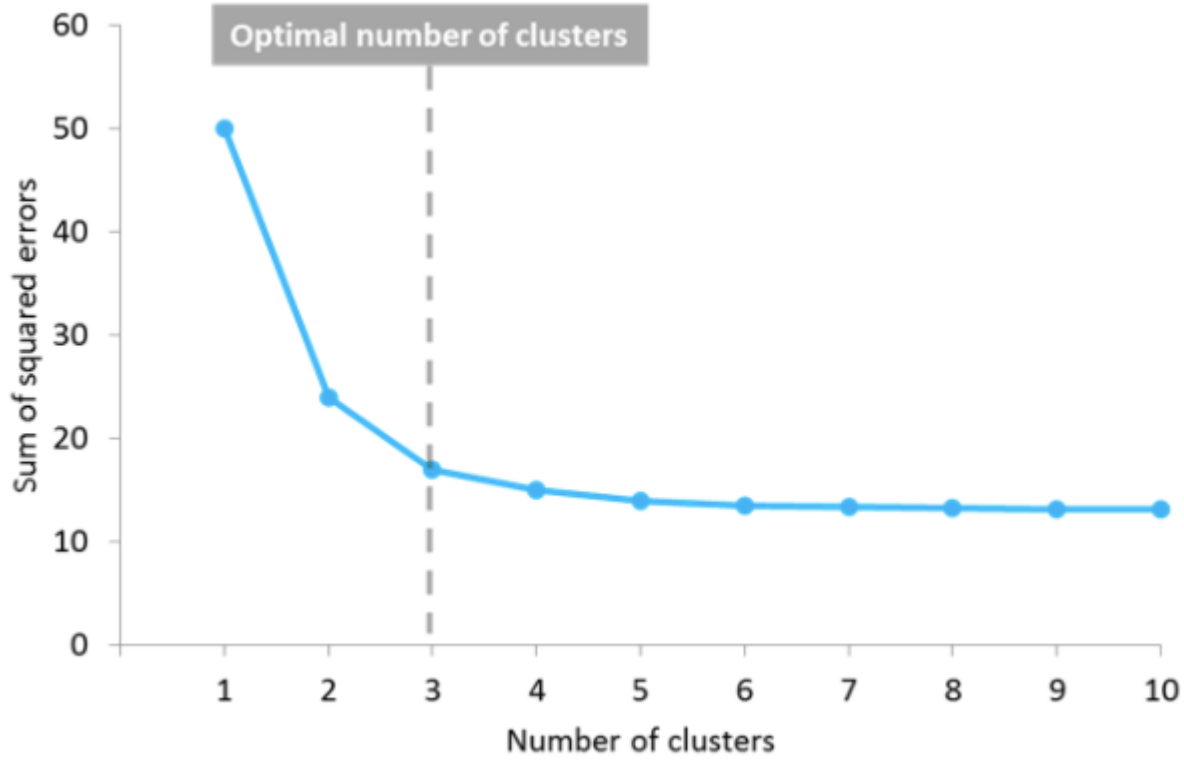


K-means

- The process is iterative-once all the samples have been processed, a new set of centroids $K_{(1)}$ is computed (now considering the actual elements belonging to the cluster), and all the distances are recomputed.
- The algorithm stops when the desired tolerance is reached, or in other words, when the centroids become stable and, therefore, the inertia is minimized.
- Of course, this approach is quite sensitive to the initial conditions, and some methods have been studied to improve the convergence speed

Optimal value of k – Elbow point

- **Elbow point** - variation vs number of clusters (k) plot



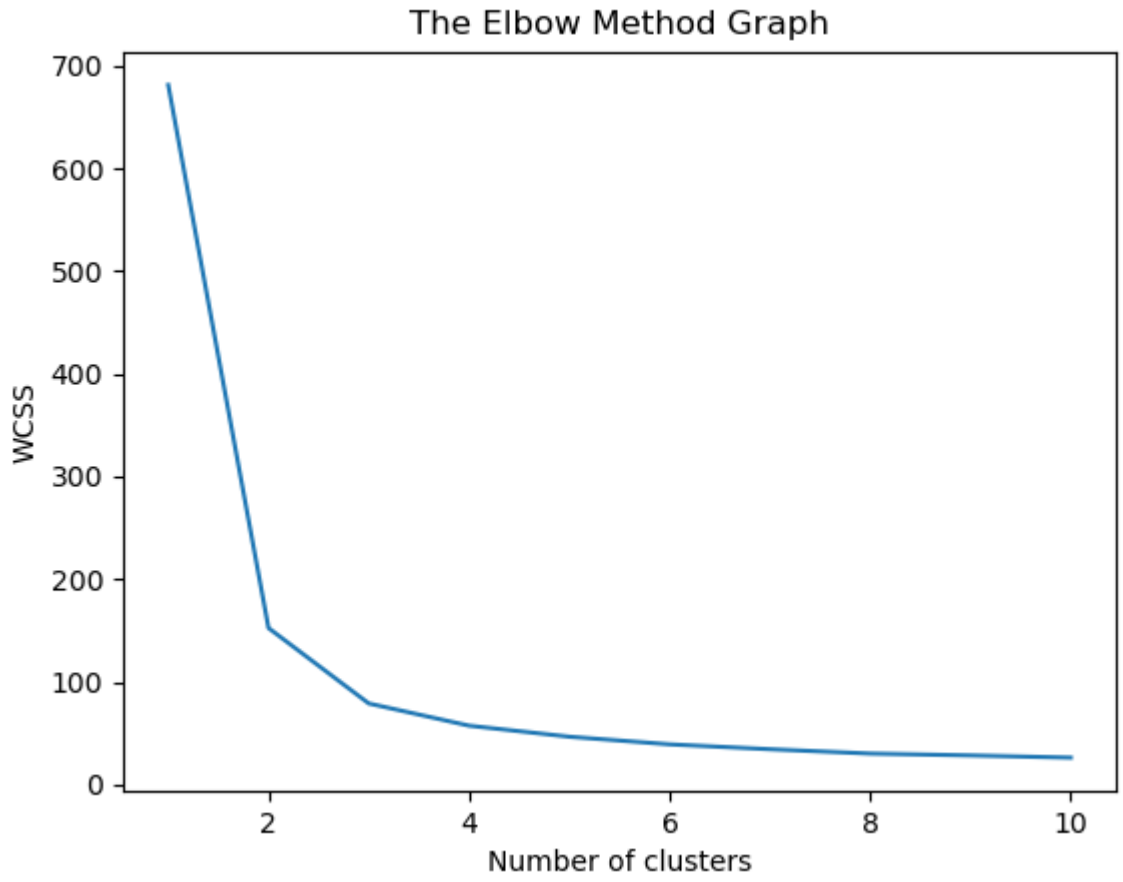
Optimal value of k – silhouette

- The silhouette score is based on the principle of "maximum internal cohesion and maximum cluster separation".
- We would like to find the number of clusters that produce a subdivision of the dataset into dense blocks that are well separated from each other.
- Every cluster will contain very similar elements and, selecting two elements belonging to different clusters, their distance should be greater than the maximum intracluster
- After defining a distance metric (Euclidean is normally a good choice), we can compute the average intracluster distance for each element: $a(\bar{x}_i) = E_{\bar{x}_{j \in C}}[d(\bar{x}_i, \bar{x}_j)] \forall \bar{x}_i \in C$

Optimal value of k – silhouette

- We can also define the average nearest-cluster distance (which corresponds to the lowest intercluster distance):
  $b(\bar{x}_i) = E_{\bar{x}_{j \in C}}[d(\bar{x}_i, \bar{x}_j)] \forall \bar{x}_i \in C$ where $D = argmin\{d(C, D)\}$
- The silhouette score for an element xi is defined as:
  $S(\bar{x}_i = \frac{b(\bar{x}_i) - a(\bar{x}_i)}{max a(\bar{x}_i), b(\bar{x}_i)}$
- This value is bounded between -1 and 1, with the following interpretation:
  - A value close to 1 is good (1 is the best condition) because it means that $a(x_i) << b(x_i)$
  - A value close to 0 means that the difference between intra and inter cluster measures is almost null and therefore there's a cluster overlap
  - A value close to -1 means that the sample has been assigned to a wrong cluster because $a(x_i) >> b(x_i)$

```
from sklearn import datasets
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

iris=datasets.load_iris()

wcss = [] # sum of the squared distance
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i,
                    init='k-means++',
                    max_iter=300,
                    n_init=10,
                    random_state=0)
    kmeans.fit(iris.data)
    wcss.append(kmeans.inertia_)

plt.plot(range(1,11),wcss)
plt.title('The Elbow Method Graph')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```


The Elbow Method Graph
WCSS
Number of clusters
0
100
200
300
400
500
600
700
2
4
6
8
10

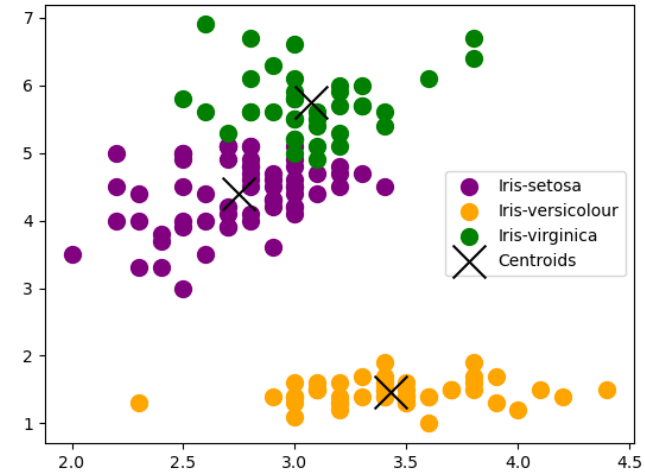
Silhouette and centroids

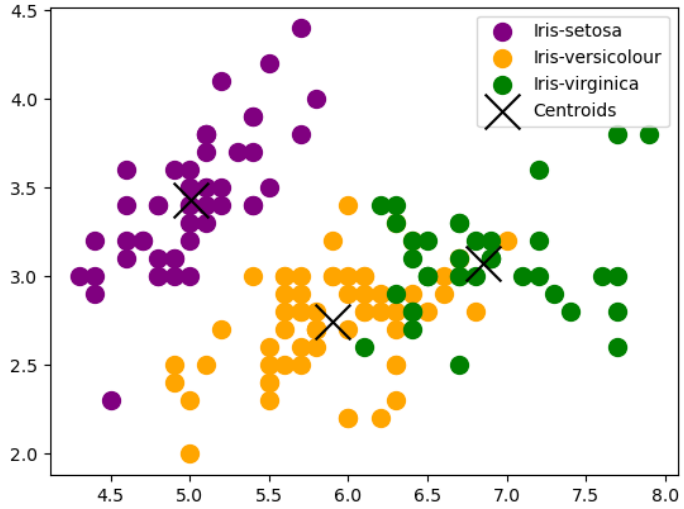
```
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

iris=datasets.load_iris()

rf=KMeans(n_clusters=3)
clf=rf.fit(iris.data)
centroids=clf.cluster_centers_
score=silhouette_score(iris.data,clf.labels_)
print(centroids)
print(score)
```

KMeans

```
X=iris.data
y_kmeans=clf.labels_

y_kmeans = clf.fit_predict(X)

# Visualising the clusters
cols = iris.feature_names

plt.scatter(X[y_kmeans == 0, 0],
            X[y_kmeans == 0, 1],
            s=100, c='purple',
            label='Iris-setosa')
plt.scatter(X[y_kmeans == 1, 0],
            X[y_kmeans == 1, 1],
            s=100, c='orange',
            label='Iris-versicolour')
plt.scatter(X[y_kmeans == 2, 0],
            X[y_kmeans == 2, 1],
            s=100, c='green',
            label='Iris-virginica')

# Plotting the centroids of the
clusters
plt.scatter(centroids[:, 0],
            centroids[:, 1],
            s=400, c='black',
            marker="x",
            label='Centroids')

plt.legend()
plt.show()
```

The cluster for each line of data

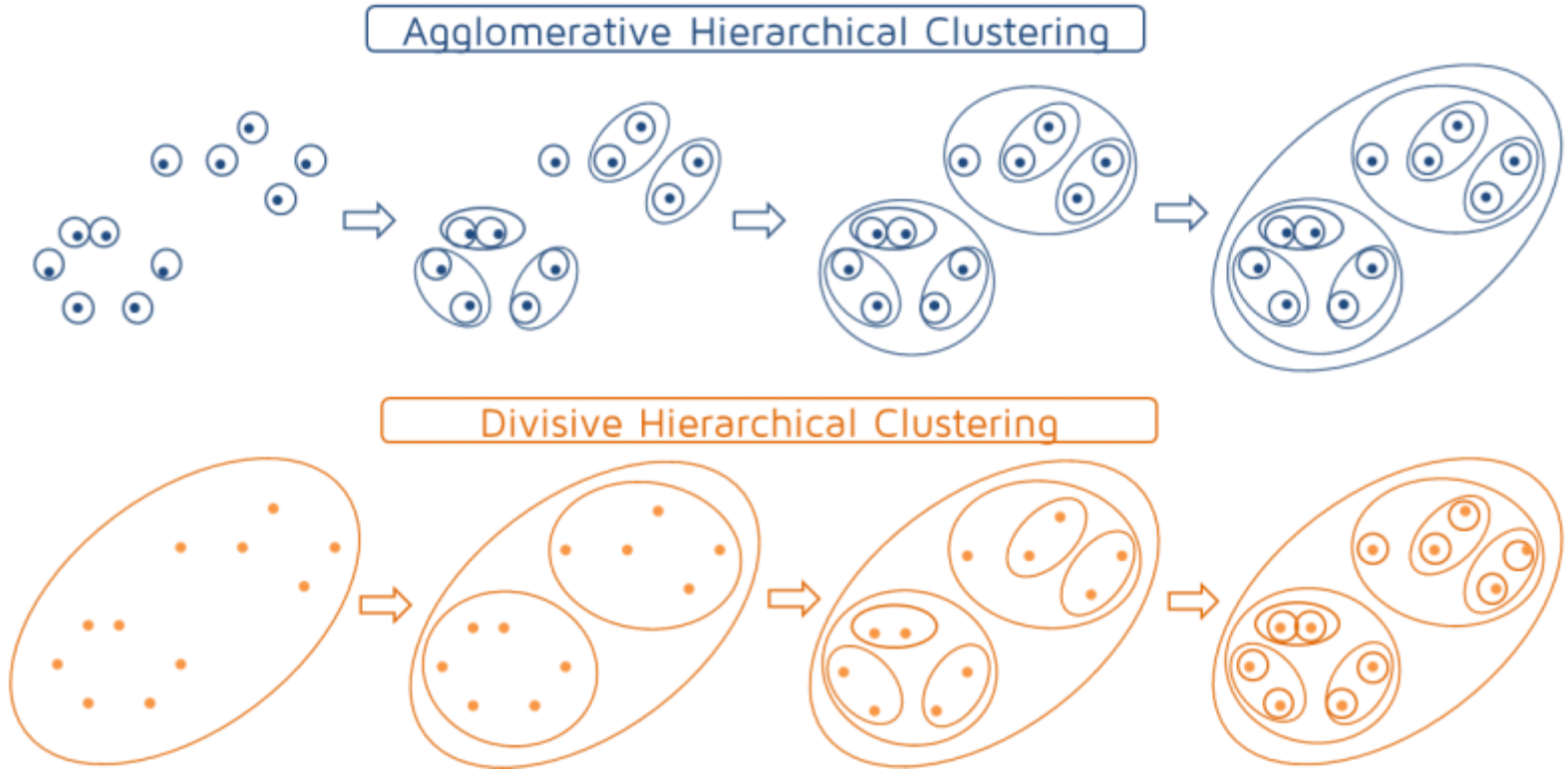




Hierarchical Clustering

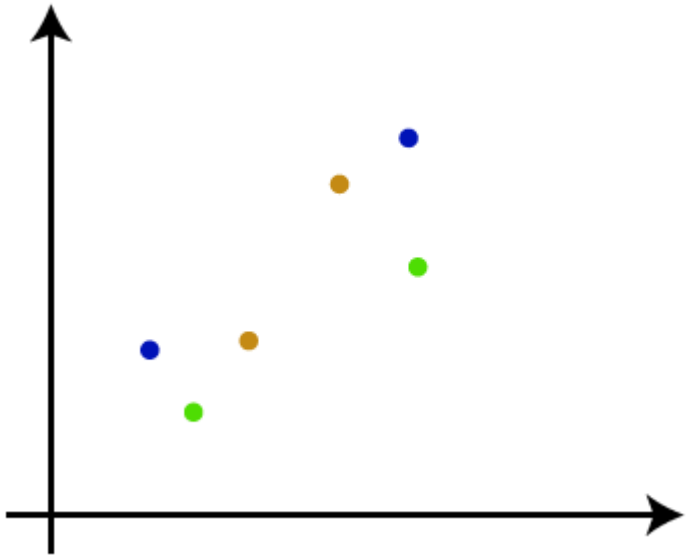
- Hierarchical clustering is based on the general concept of finding a hierarchy of partial clusters, built using either a bottom-up or a top-down approach.
- Agglomerative clustering: The process starts from the bottom (each initial cluster is made up of a single element) and proceeds by merging the clusters until a stop criterion is reached.
- Divisive clustering: In this case, the initial state is a single cluster with all samples and the process proceeds by splitting the intermediate cluster until all elements are separated. At this point, the process continues with an aggregation criterion based on the dissimilarity between elements

Hierarchical Clustering



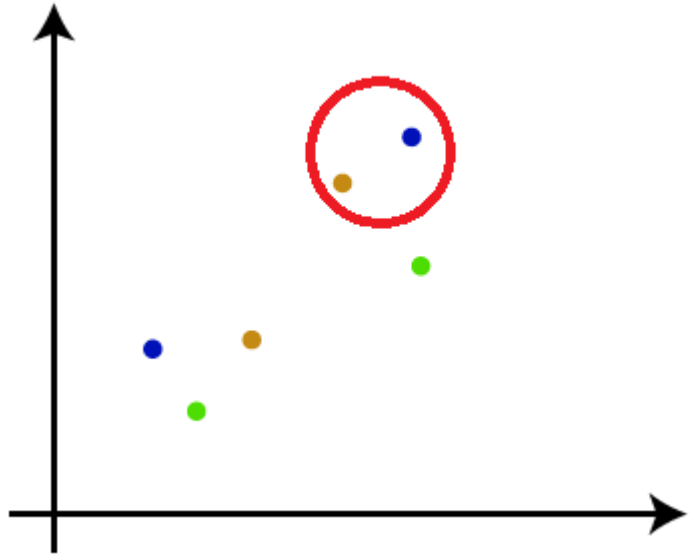
Agglomerative Clustering

- Step-1: Create each data point as a single cluster. Let's say there are N data points, so the number of clusters will also be N.

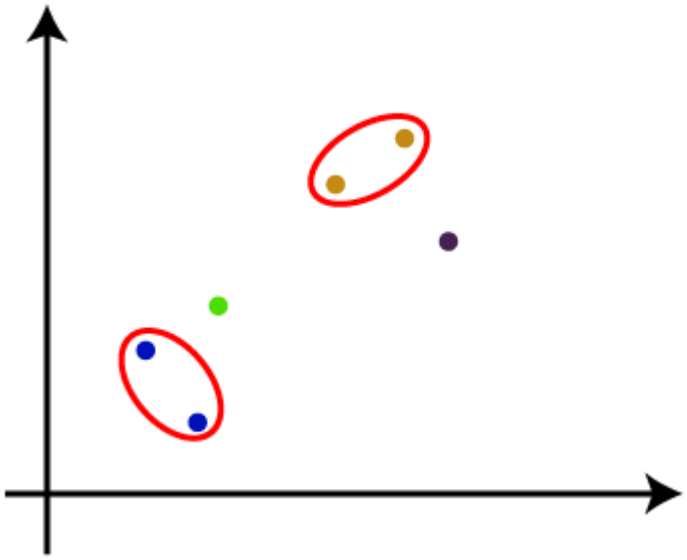


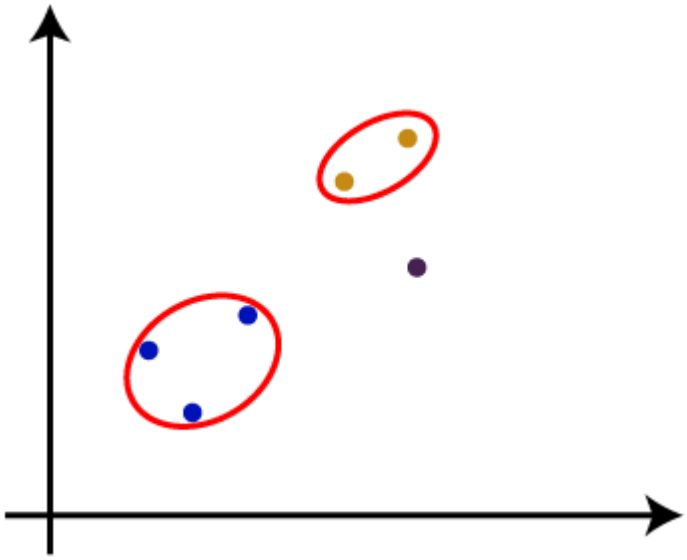
Agglomerative Clustering

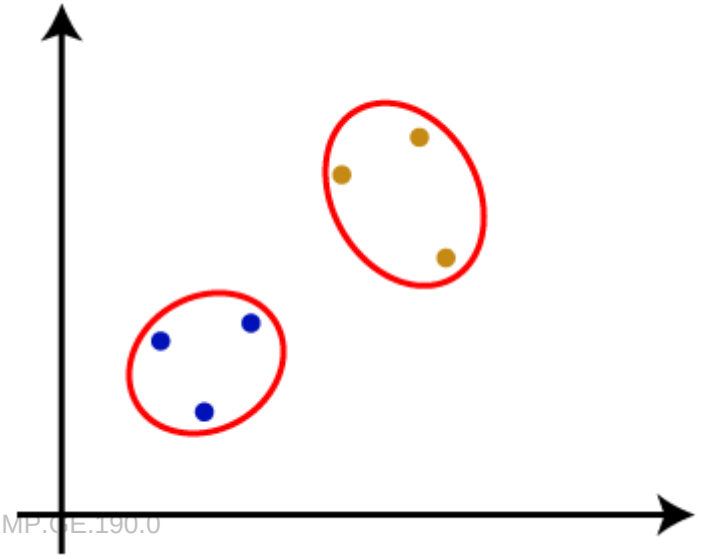
- Step-2: Take two closest data points or clusters and merge them to form one cluster. So, there will now be N-1 clusters.

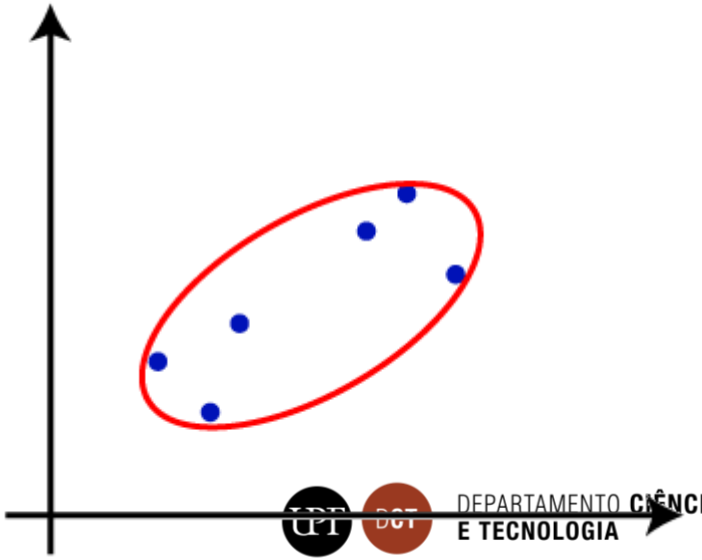


Agglomerative Clustering









Agglomerative Clustering

- Let's consider the following dataset: $X = \{\bar{x}_1, \bar{x}_2, ..., \bar{x}_n\}$ where $\bar{x}_i \in \mathrm{R}^m$
- We define affinity, a metric function of two arguments with the same dimensionality m.
- The most common metrics are: Euclidean or L2, Manhattan or L1 and Cosine distance
- The Euclidean distance is normally a good choice, but sometimes it's useful to a have a metric whose difference with the Euclidean one gets larger and larger. The Manhattan metric has this property; to show it, in the following figure there's a plot representing the distances from the origin of points belonging to the line $y = x$:

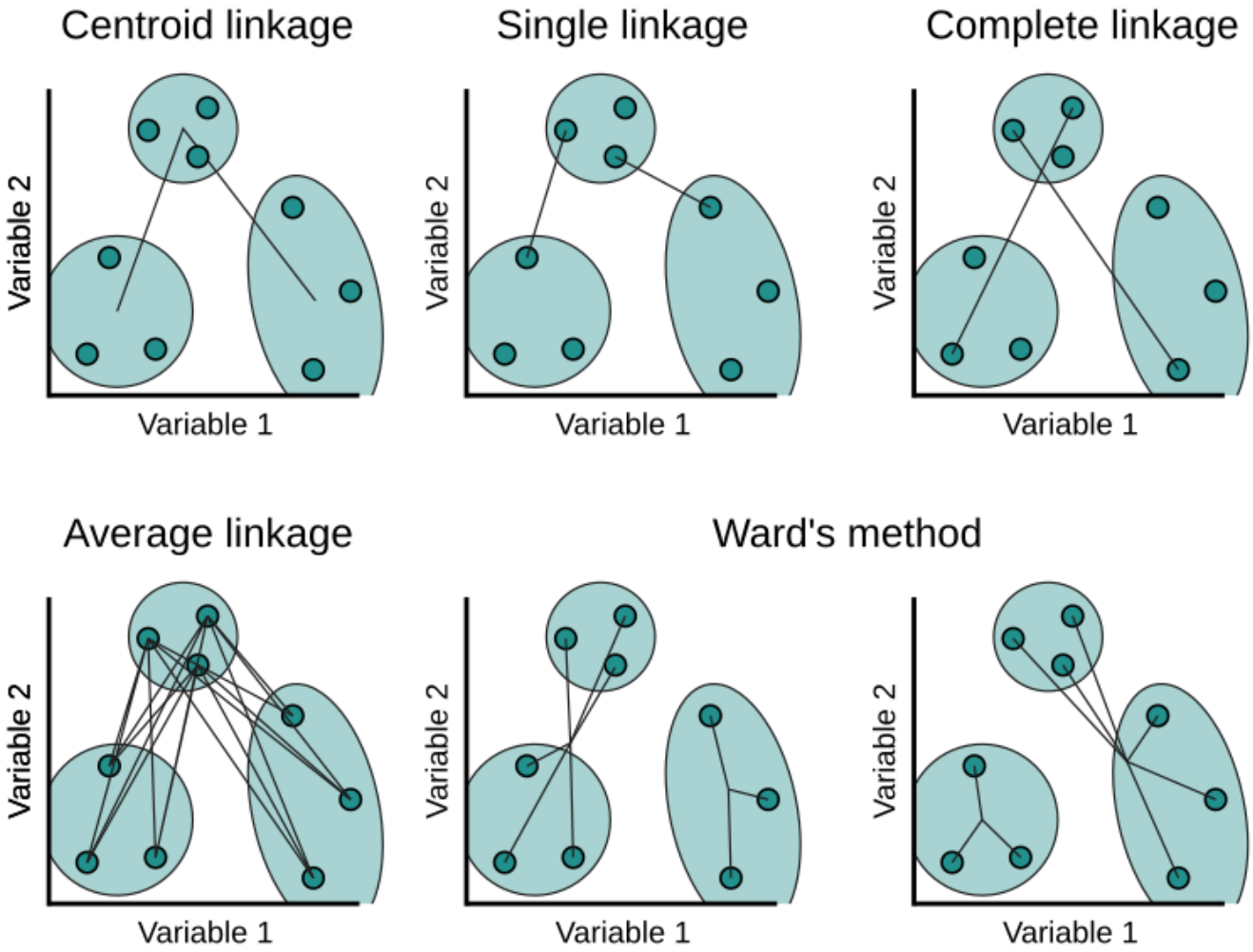
Agglomerative Clustering

- The cosine distance, instead, is useful when we need a distance proportional to the angle between two vectors
- This distance can be employed when the clustering must not consider the L2 norm of each point
- Once a metric has been chosen ($d(x, y)$), the next step is defining a strategy (called linkage) to aggregate different clusters.
- Complete linkage: For each pair of clusters, the algorithm computes and merges them to minimize the maximum distance between the clusters (in other words, the distance of the farthest elements): $\forall C_i, C_j L_i, j = max\{d(x_a, x_b) \forall x_a \in C_i and x_b \in C_j\}$

Agglomerative Clustering

- Average linkage: It's similar to complete linkage, but in this case the algorithm uses the average distance between the pairs of clusters: $\forall C_i, C_j L_{i,j} = \frac{1}{|C_i||C_j|} \sum_{x_a \in C_i} \sum_{x_b \in C_j} d(x_a, x_b)$
- Ward's linkage: In this method, all clusters are considered and the algorithm computes the sum of squared distances within the clusters and merges them to minimize it. From a statistical viewpoint, the process of agglomeration leads to a reduction in the variance of each resulting cluster. The measure is: $\forall C_i, C_j L_{i,j} = \sum_{x_a \in C_i} \sum_{x_b \in C_j} ||x_a - x_b||^2$
- The Ward's linkage supports only the Euclidean distance.

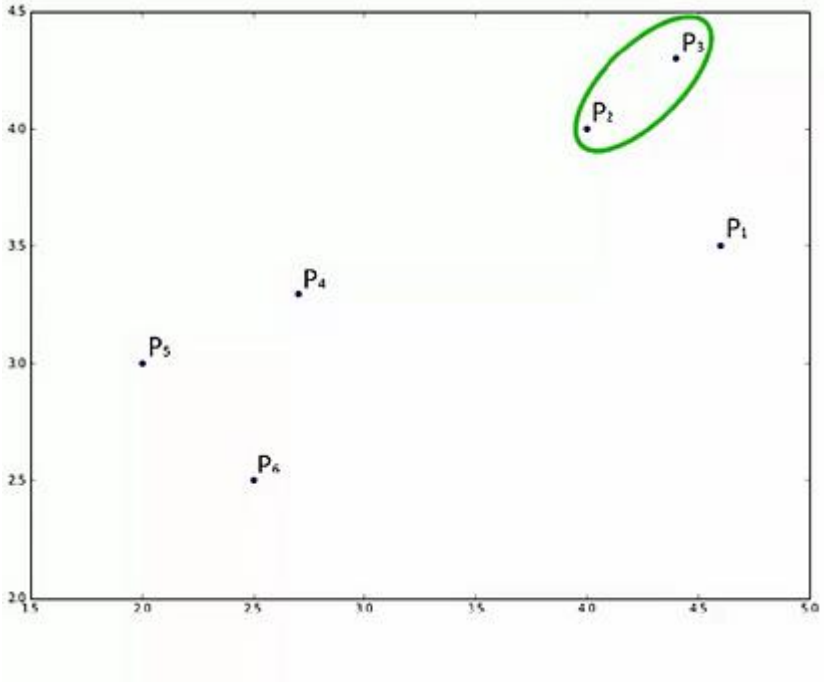
Agglomerative Clustering

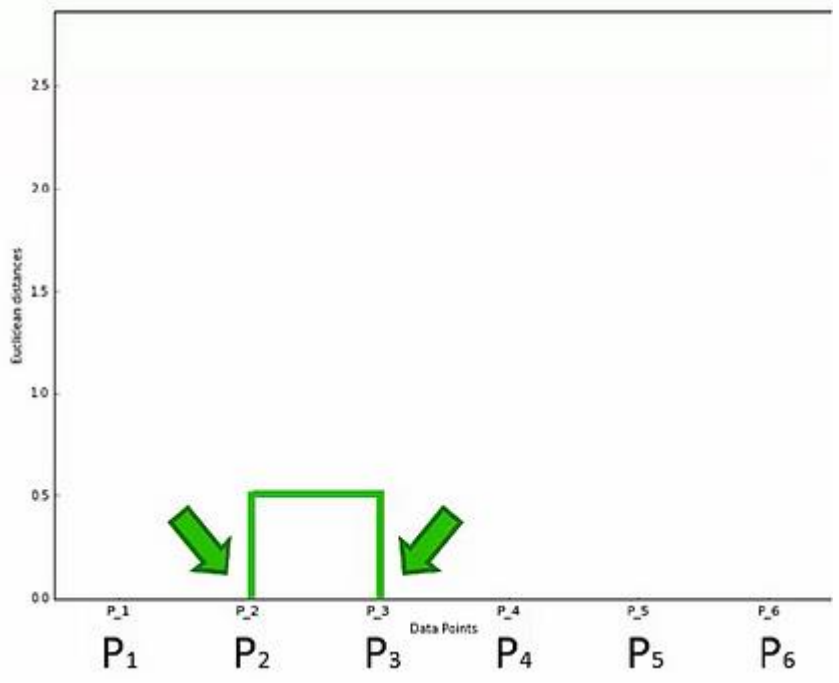


Dendrograms

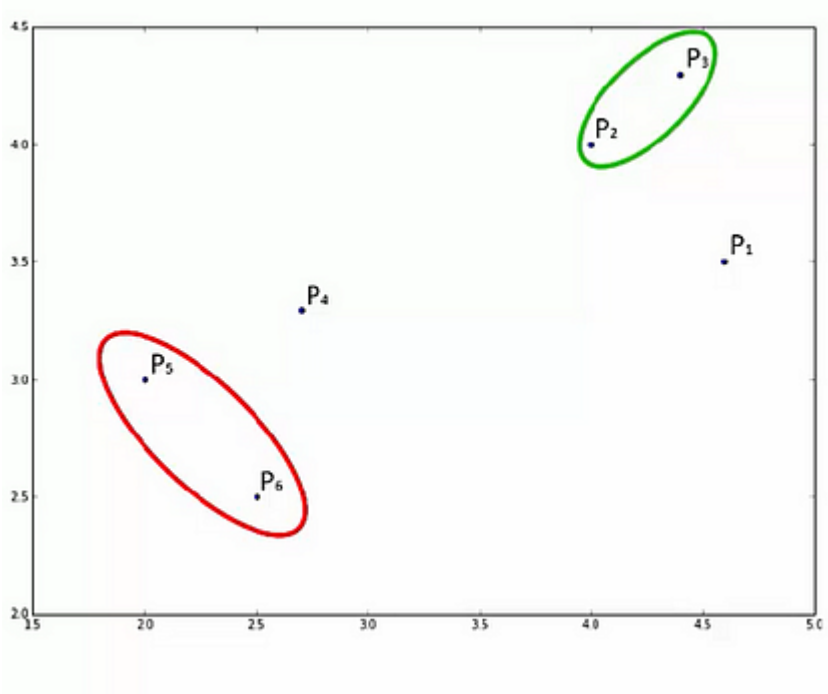
- A graphical method called a dendrogram, shows in a static way how the aggregations are performed, starting from the bottom (where all samples are separated) till the top (where the linkage is complete).
- In the x axis, there are the samples (numbered progressively), while the y axis represents the distance. Every arch connects two clusters that are merged together by the algorithm

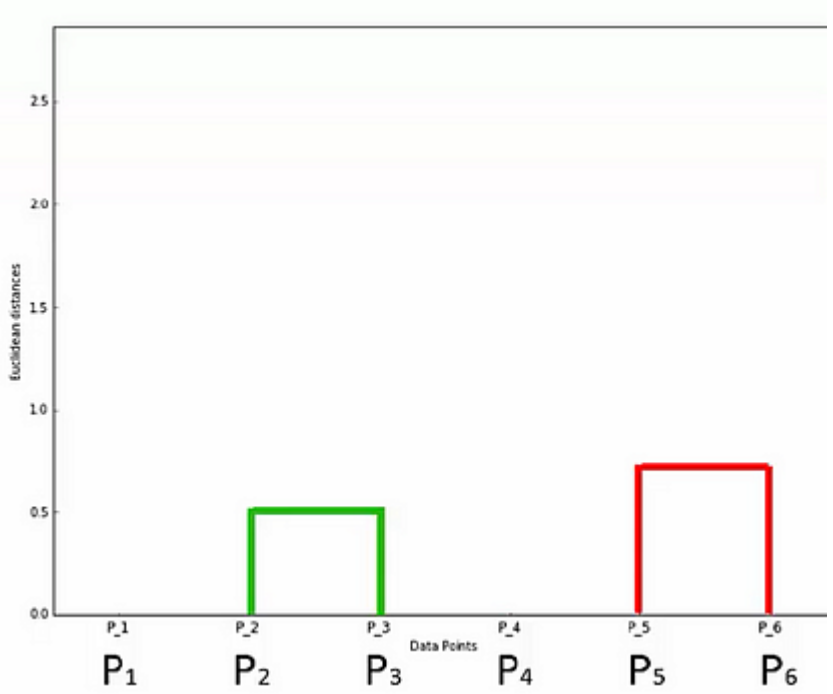
Dendrograms



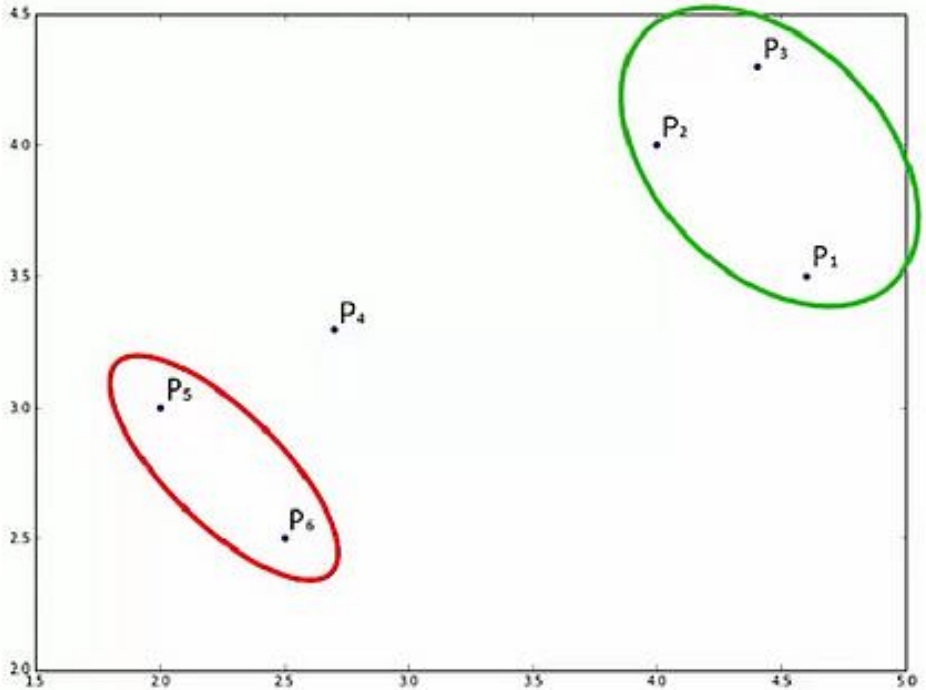


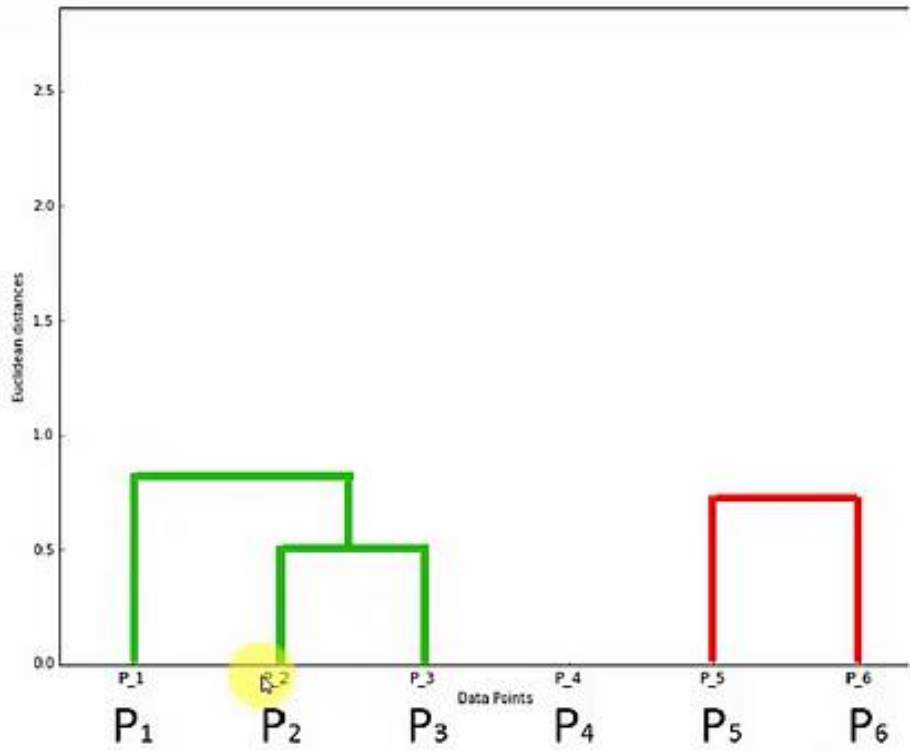
Dendrograms



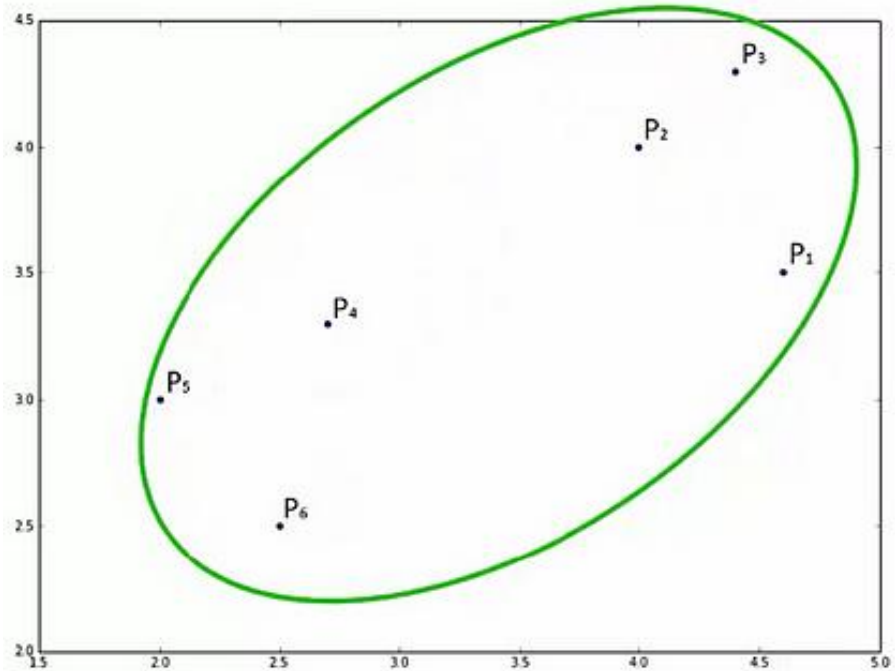


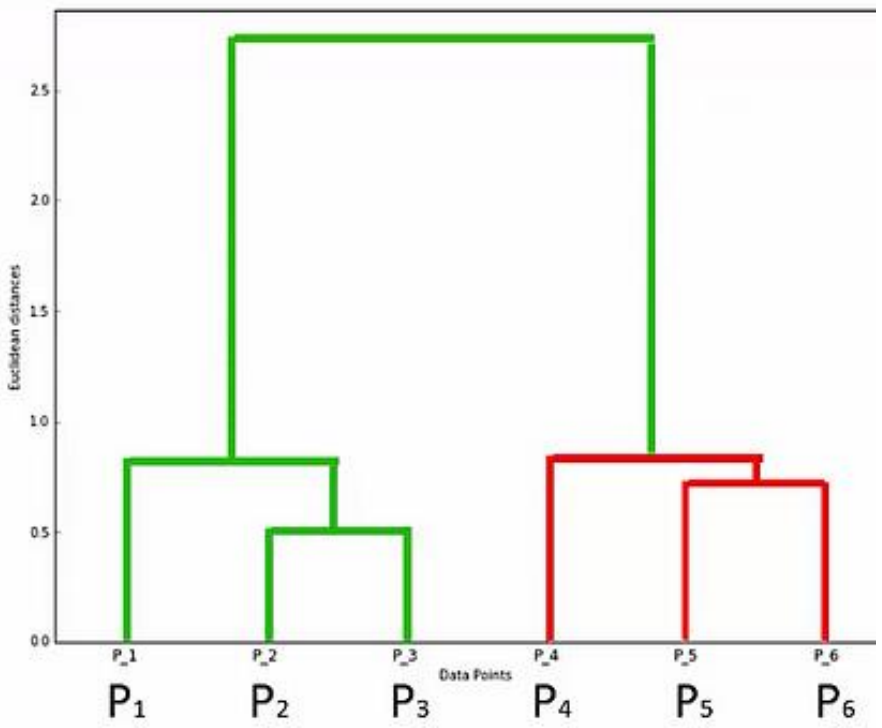
Dendrograms

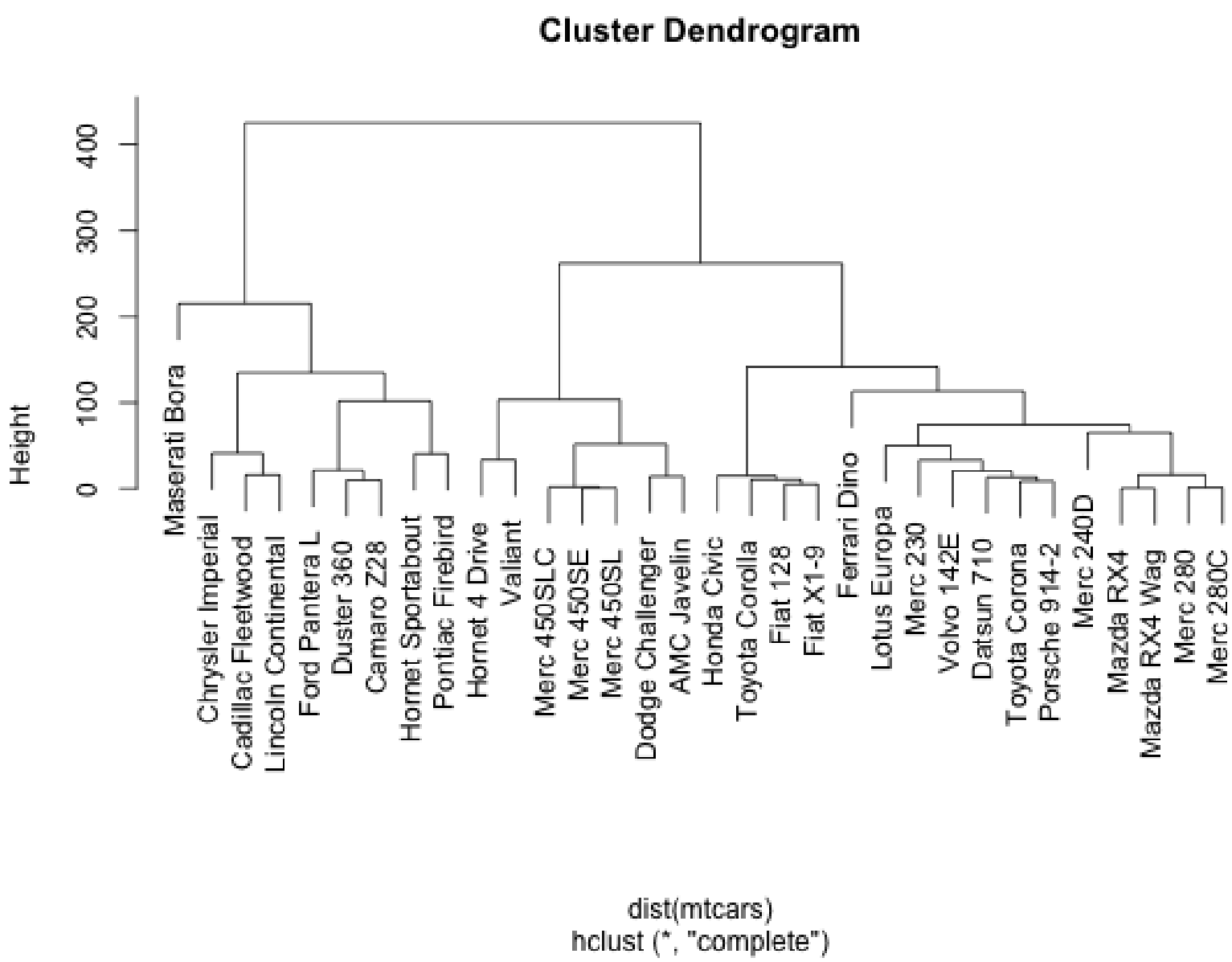




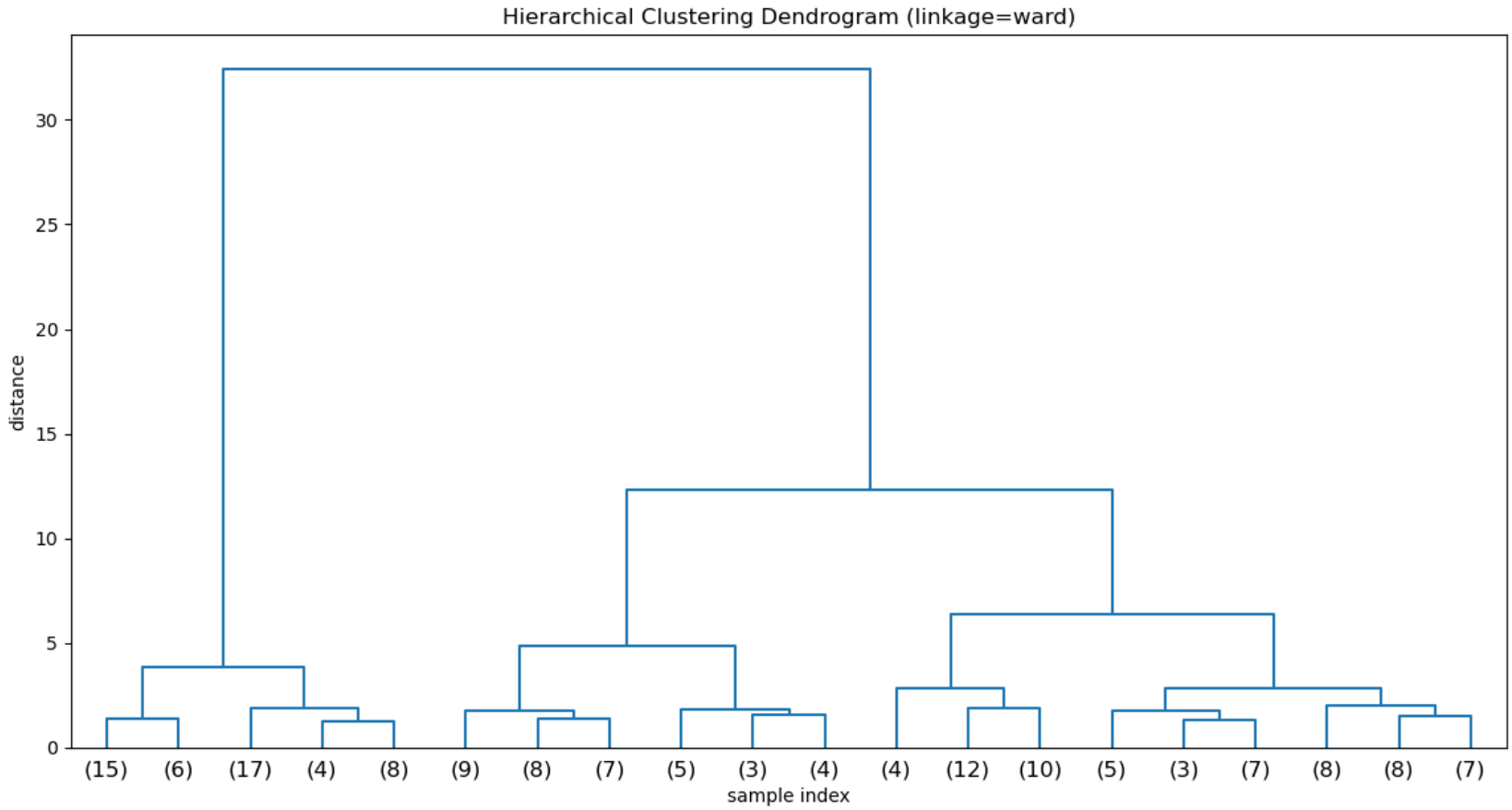
Dendrograms

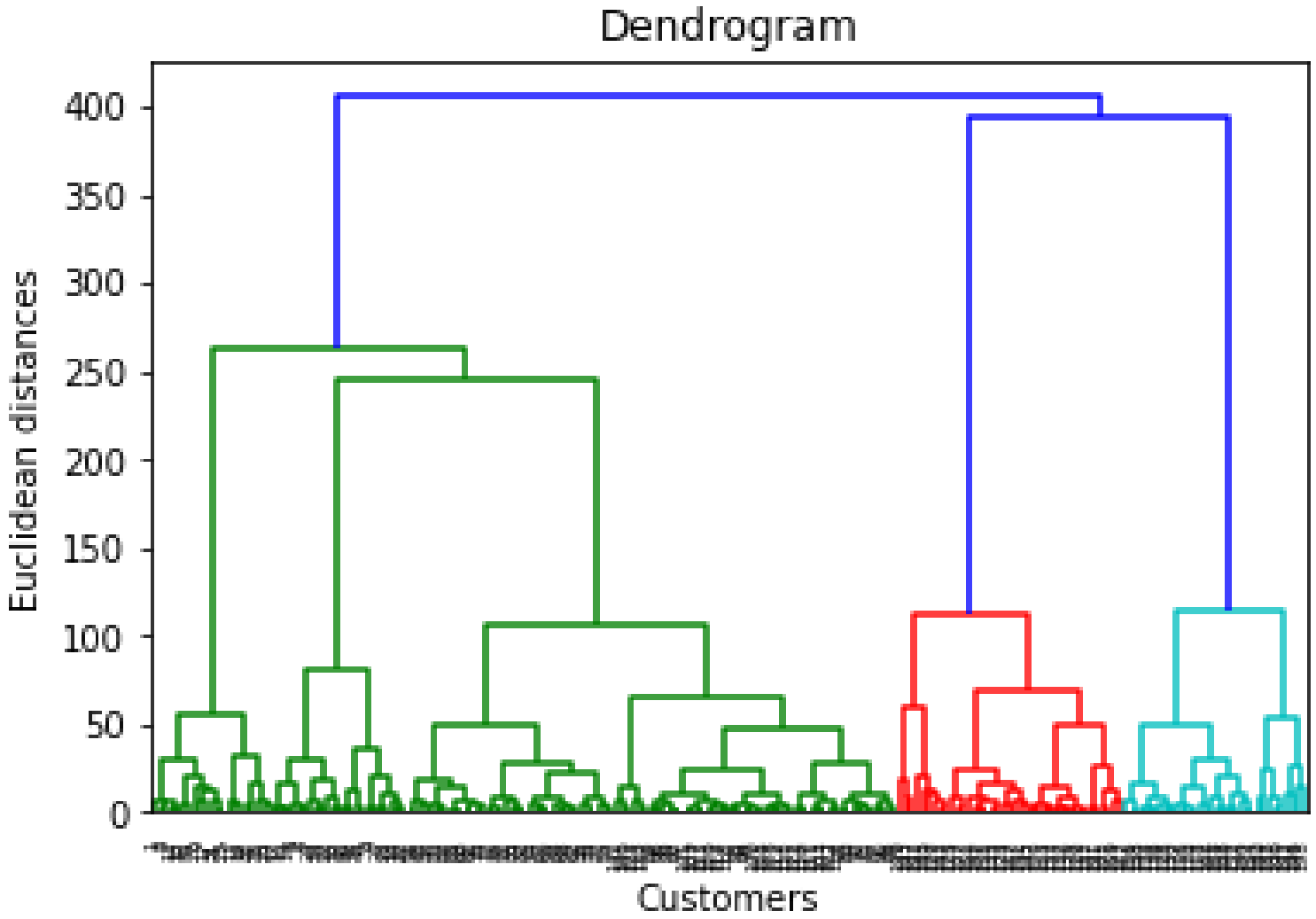





Cluster Dendrogram
Height
0
100
200
300
400
Maserati Bora
Chrysler Imperial
Cadillac Fleetwood
Lincoln Continental
Ford Pantera L
Duster 360
Camaro Z28
Hornet Sportabout
Pontiac Firebird
Hornet 4 Drive
Valiant
Merc 450SLC
Merc 450SE
Merc 450SL
Dodge Challenger
AMC Javelin
Honda Civic
Toyota Corolla
Fiat 128
Fiat X1-9
Ferrari Dino
Lotus Europa
Merc 230
Volvo 142E
Datsun 710
Toyota Corona
Porsche 914-2
Merc 240D
Mazda RX4
Mazda RX4 Wag
Merc 280
Merc 280C
dist(mtcars)
hclust (*, "complete")

```
from sklearn import datasets
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

iris = datasets.load_iris()

# separate features and class labels
X_features = iris.data
y_labels = iris.target

model = AgglomerativeClustering(linkage="ward", n_clusters=3)
model.fit(X_features)
predicted_labels = model.labels_

linkage_matrix = linkage(X_features, 'ward')
plot = plt.figure(figsize=(14, 7))
dendrogram(
    linkage_matrix,
    truncate_mode='lastp',
    p=20,
    color_threshold=0,
)
plt.title('Hierarchical Clustering Dendrogram (linkage=ward)')
plt.xlabel('sample index')
plt.ylabel('distance')
plt.show()
```


Hierarchical Clustering Dendrogram (linkage=ward)
distance
0
5
10
15
20
25
30
(15)
(6)
(17)
(4)
(8)
(9)
(8)
(7)
(5)
(3)
(4)
(4)
(12)
(10)
(5)
(3)
(7)
(8)
(8)
(7)
sample index


Dendrogram
Euclidean distances
400
350
300
250
200
150
100
50
0
Customers

Exercises

- Download the dataset Mall_Customers.csv from moodle
- Create a data frame with the features Annual Income (k$) and Spending Score()1-100
- Determine the optimal number of clusters
- Implement k-means clustering and Agglomerative Clustering
- Plot the clusters as well as the dendogram

References

- Bonaccorso, G. (2020). Mastering Machine Learning Algorithms: Expert techniques for implementing popular machine learning algorithms, fine-tuning your models, and understanding how they work. Packt Publishing Ltd.
- Bonaccorso, G. (2018). Machine Learning Algorithms: Popular algorithms for data science and machine learning. Packt Publishing Ltd.
- Lee, W. M. (2019). Python machine learning. John Wiley & Sons.
- Hauck, T. (2014). scikit-learn Cookbook. Packt Publishing Ltd.
- Montgomery, D. C., Peck, E. A., & Vining, G. G. (2021). Introduction to linear regression analysis. John Wiley & Sons.
- Hosmer Jr, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). Applied logistic regression (Vol. 398). John Wiley & Sons.

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.