Project Unsupervised ML – Clustering

Index

• Main objective of the analysis

In this project I'm going to use different models of **Clustering** to show the process to be followed to group a data set in different clusters.

a) Initially I'll compare the use of **K-means** and **Hierarchical Clustering** applied to a sample of points randomly generated.
b) And, afterward I will use a **k-means** model for the purpose of **Customer Segmentation**, when the label is unknown for the customer. So, I'll be using in this case Clustering instead of Classification.

• Brief description of the data set.

For the **case a)** I will be making random clusters of points by using the **make_blobs** class. The make_blobs class can take in many inputs, but I will be using these specific ones.

Input

n_samples: The total number of points equally divided among clusters. Value will be: **5000**

centers: The number of centers to generate, or the fixed center locations.

Value will be: **[[4, 4], [-2, -1], [2, -3], [1,1]]**

cluster_std: The standard deviation of the clusters. Value will be: **0.9**

Output

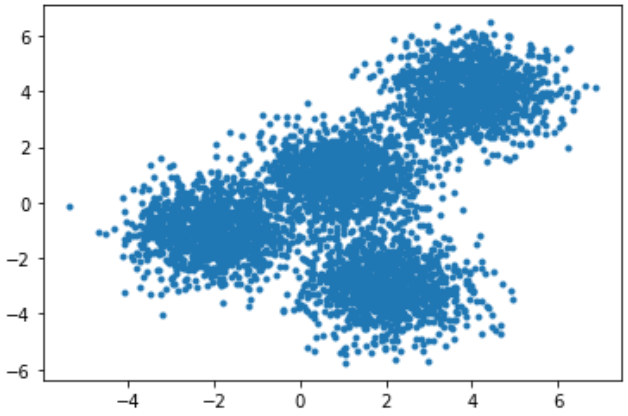
- **X**: Array of shape [n_samples, n_features]. (Feature Matrix)
  - The generated samples.

- **y**: Array of shape [n_samples]. (Response Vector)
  - The integer labels for cluster membership of each sample.

And, we generate the points

```
X, y = make_blobs(n_samples=5000, centers=[[4,4], [-2, -1], [2, -3], [1, 1]], cluster_std=0.9)
```

Displaying the scatter plot of the randomly generated data, we obtain the following graph



- ## Summary of training

In the case a.1) I will use **K-means.**

Now that I have our random data, let's set up our K-Means Clustering.

The KMeans class has many parameters that can be used, but I will be using these three:

- **init**: Initialization method of the centroids.
  - Value will be: "**k-means++**"
  - k-means++: *Selects initial cluster centers for k-mean clustering in a smart way to speed up convergence.*
- **n_clusters**: The number of clusters. Value will be: **4** (since we have 4 centers)
- **n_init**: Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.
  - Value will be: **12**

Initialize KMeans with these parameters, where the output parameter is called **k_means**.

```
k_means = KMeans(init = "k-means++", n_clusters = 4, n_init = 12)
```

Now let's **fit the KMeans model** with the feature matrix we created above, **X**

```
k_means.fit(X)
```

And using KMeans' **.labels_** attribute we can obtain labels for each point **with values between 0 and 3**, as we have 4 clusters.

And, I can also get the coordinates of the clusters centers with **.cluster_centers_**

Now let's grab the labels for each point in the model using KMeans' .labels_ attribute and save it as k_means_labels

```
k_means_labels = k_means.labels_
k_means_labels
```

```
array([0, 3, 3, ..., 1, 0, 0], dtype=int32)
```

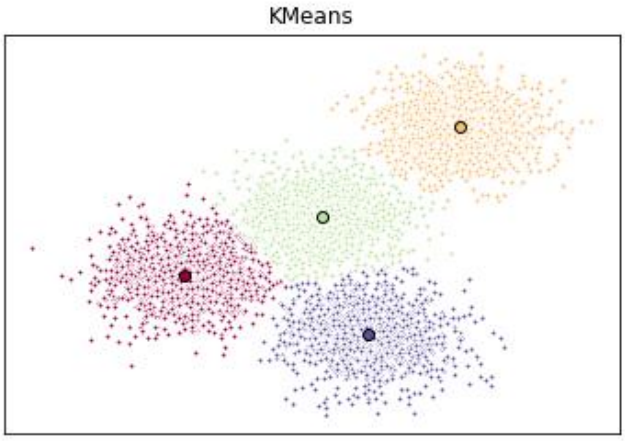
We will also get the coordinates of the cluster centers using KMeans' .cluster_centers_ and save it as k_means_cluster_centers

```
k_means_cluster_centers = k_means.cluster_centers_
k_means_cluster_centers
```

```
array([[-2.03743147, -0.99782524],
       [ 3.97334234,  3.98758687],
       [ 0.96900523,  0.98370298],
       [ 1.99741008, -3.01666822]])
```

And we can observe that these coordinates are the same than the ones we used to generate the points, so the result of the model is very good.

Finally, we can generate a **plot** obtaining the following **visualization.**



In the case a.2) I Will use **Agglomerative Hierarchical Clustering**

As commented befores, for the **cases a.1 and a.2)** I will be making random clusters of points by using the **make_blobs** class. The make_blobs class can take in many inputs, but in this second case I will be using the following specific ones.

Input these parameters into **make_blobs**:

n_samples: The total number of points equally divided among clusters. Choose initially **50**

centers: The number of centers to generate, or the fixed center locations. We'll have **4 centers**.

cluster_std: The standard deviation of the clusters. The larger the number, the further apart the clusters. Initially we'll use a value of **0.9**

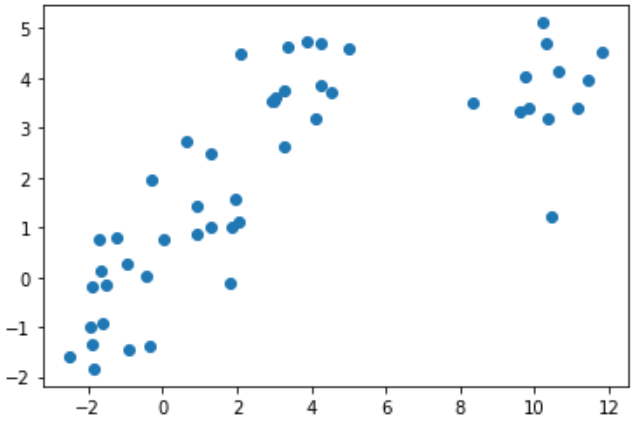
We generate the points with values of X, Y

X1, y1 = make_blobs(n_samples=**50**, centers=**[[4,4], [-2, -1], [1, 1], [10,4]],** cluster_std=**0.9**)

And the result is

```
plt.scatter(X1[:, 0], X1[:, 1], marker='o')
```

```
<matplotlib.collections.PathCollection at 0x7fe80697b550>
```

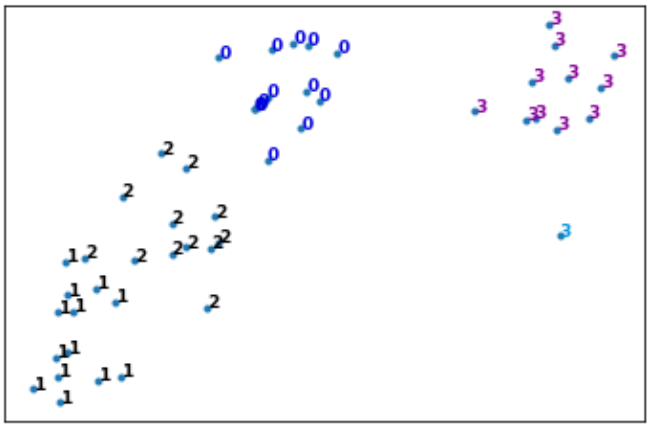


And, now we can start by clustering the random data points we just created. We'll use 4 clusters and Linkage "**Average**", and then fit the model

agglom = AgglomerativeClustering(n_clusters = **4**, linkage = **'average'**)

agglom.**fit**(X1,y1)

We can generate a plot to see the resulting **clustering**



As the distance matrix contains the distance from each point to every other point of a dataset, I'll use the function **distance_matrix**, which requires two inputs.

dist_matrix = distance_matrix(X1,X1)

print(dist_matrix)

```
[[0.         0.41689921 0.71333054 ... 0.35940605 0.62336998 0.67402686]
 [0.41689921 0.         0.32421296 ... 0.11465766 0.28523407 0.27601093]
 [0.71333054 0.32421296 0.         ... 0.35401271 0.13303007 0.05324675]
 ...
 [0.35940605 0.11465766 0.35401271 ... 0.         0.27119167 0.31636524]
 [0.62336998 0.28523407 0.13303007 ... 0.27119167 0.         0.13832993]
 [0.67402686 0.27601093 0.05324675 ... 0.31636524 0.13832993 0.        ]]
```

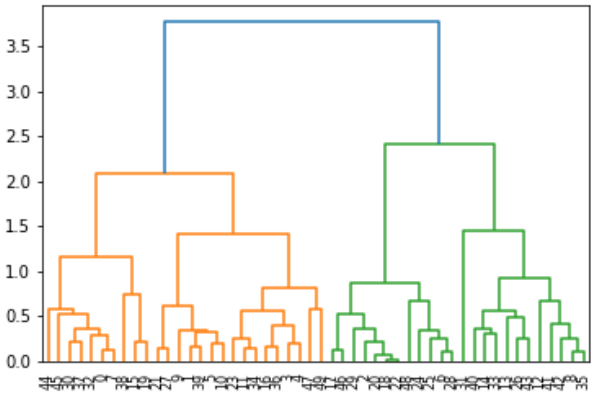
And, save the result to a variable called Z

Z = hierarchy.linkage(dist_matrix, 'average')

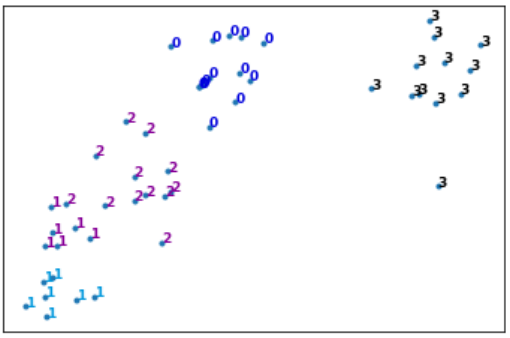
A Hierarchical clustering is typically visualized as a **dendrogram** as shown in the following cell.

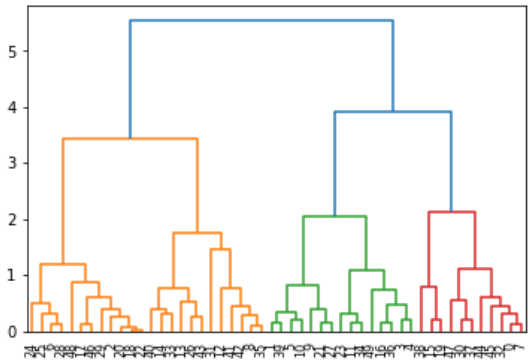
Each merge is represented by a horizontal line. The y-coordinate of the horizontal line is the similarity of the two clusters that were merged.

dendro = hierarchy.dendrogram(Z)



If we use **Linkage ="Complete"** instead of **Average**, the distances will be different and the clustering and dendogram will change, resulting as follows:





As in this **case a.2)** we have generated only **50 points** (to have a clearer configuration / visualization in the dendogram), the result of clustering is more "relative", and it depends on the type of **Linkage** ("distance to be considered for merges"),

In **case a.1)** the result using **K-means** is very clear, with 4 cluster and the centers coinciding with the coordinates used for generation. For this case we used **5000 points** and the distribution in the space is very clear.

Case b). I will use in this case a **k-means** model for the purpose of **Customer Segmentation**, when the label is unknown for the customer.

For this case we'll use the following **dataset**

https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/**Cust_Segmentation.csv**

and we can see the **columns** and the first **rows**

```
import pandas as pd
cust_df = pd.read_csv("Cust_Segmentation.csv")
cust_df.head()
```

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	Address	DebtIncomeRatio
0	1	41	2	6	19	0.124	1.073	0.0	NBA001	6.3
1	2	47	1	26	100	4.582	8.218	0.0	NBA021	12.8
2	3	33	2	10	57	6.111	5.802	1.0	NBA013	20.9
3	4	29	2	4	19	0.681	0.516	0.0	NBA009	6.3
4	5	47	1	31	253	9.308	8.908	0.0	NBA008	7.2

As pre-processing we'll keep all the features except Address, that is in this case a categorical value, and then is not appropriate for applying Euclidean distance.

We'll have now **9 variables**, one of them is **CustomerID**, and the following ones are

Age, Edu, Years Employed, Income, Card Debt, Other Debt, Defaulted, DebtIncomeRatio

We normalize data using **StandardScaler()** and will apply **modeling** with **k-Means**

```
clusterNum = 3
k_means = KMeans(init = "k-means++", n_clusters = clusterNum, n_init = 12)
k_means.fit(X)
labels = k_means.labels_
print(labels)
```

```
[1 2 1 1 0 2 1 2 1 2 2 1 1 1 1 1 1 1 2 1 1 1 1 2 2 2 1 1 2 1 2 1 1 1 1 1 1
 1 1 2 1 2 1 0 1 2 1 1 1 2 2 1 1 2 2 1 1 1 2 1 2 1 2 2 1 1 2 1 1 1 2 2 2 1
 1 1 1 1 2 1 2 2 0 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 2 1
 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 2 1
 1 1 1 1 1 1 2 1 2 2 1 2 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 2 1
 1 1 1 1 2 1 1 2 1 2 1 1 2 0 1 2 1 1 1 1 1 1 1 0 2 1 1 1 1 2 1 1 2 2 1 2 1 2
 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 0 2 1 1 1 1 1 1 1 2 1 1 1 1
 1 1 2 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1 2 1 2 2 1 1 1 1 1 1
 1 1 1 2 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 2 1 2 2 1
 1 1 1 1 2 1 1 1 1 1 1 2 1 1 2 1 1 2 1 1 1 1 1 2 1 1 1 0 1 1 1 2 1 2 2 2 1
 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1
 1 2 1 1 2 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 0
 1 1 1 1 1 1 2 1 1 1 0 1 1 1 1 2 1 0 1 1 1 1 2 1 2 2 2 1 1 2 2 1 1 1 1 1 1
 1 2 1 1 1 1 2 1 1 1 2 1 2 1 1 1 2 1 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1 2 1 1 1 1
 1 2 2 1 1 1 1 1 1 1 1 1 1 1 0 2 1 1 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 0 1 0 1
 1 0 1 1 1 1 1 1 1 1 1 2 1 2 1 1 0 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 2
 1 1 1 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2
 2 1 1 2 1 2 1 1 2 1 2 1 1 0 1 2 1 2 1 1 1 1 1 2 2 1 1 1 1 2 1 1 1 2 2 1 1
 2 1 1 1 2 1 0 1 1 2 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1
 1 1 2 1 1 2 1 2 1 2 2 1 1 1 2 1 2 1 1 1 1 1 2 1 1 1 1 2 2 1 1 2 2 1 1 1 1
 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 2 1 2 2 1 1 2 1 1 1 1 1 2 2
 1 1 1 1 1 1 1 2 1 1 1 1 1 1 0 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2]
```

Then **we assign the labels to each row** (customer) and we can also **obtain the centroids for the 3 clusters,** which are the **Insights clarifying the features for each cluster**.

We assign the labels to each row in dataframe.

```
df["Clus_km"] = labels
df.head(5)
```

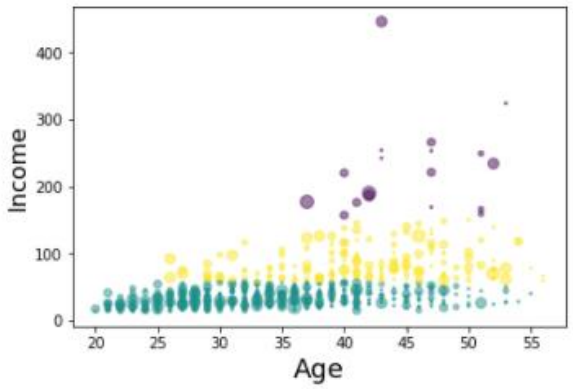
	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRatio	Clus_km
0	1	41	2	6	19	0.124	1.073	0.0	6.3	1
1	2	47	1	26	100	4.582	8.218	0.0	12.8	2
2	3	33	2	10	57	6.111	5.802	1.0	20.9	1
3	4	29	2	4	19	0.681	0.516	0.0	6.3	1
4	5	47	1	31	253	9.308	8.908	0.0	7.2	0

We can easily check the centroid values by averaging the features in each cluster.

```
df.groupby('Clus_km').mean()
```

Clus_km	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRatio
0	410.166667	45.388889	2.666667	19.555556	227.166667	5.678444	10.907167	0.285714	7.322222
1	432.006154	32.967692	1.613846	6.389231	31.204615	1.032711	2.108345	0.284658	10.095385
2	403.780220	41.368132	1.961538	15.252747	84.076923	3.114412	5.770352	0.172414	10.725824

And we can visualize it in the following scatterplot between **Age & Income**



Taking into account also the variable Education, we can consider the 3 Clusters following these types:

- AFFLUENT, EDUCATED AND OLD AGED (purple)
- MIDDLE AGED AND MIDDLE INCOME (Yellow)
- YOUNG AND LOW INCOME (Green)

- **Summary Key Findings and Insights, Suggestions for next steps**

As we have seen in this last **Case b),** the model **k-Means** has been very effective to group in different Clusters a dataset of customers (**segmentation**).

- The Insights can be clearly seen by using **visualization**, considering the variables showing a clear difference in values for the centers provided by the model.

- In this case **a value of k=3 seems to be very suitable for the dataset**, but a different number could be selected to compare

In other part, in **Cases a.1 & a.2)** using a set of points generated randomly we got the commented results:

- As in this **case a.2)** we have generated only **50 points** (to have a clearer configuration / visualization in the dendogram), the result of clustering is more "relative", and it depends on the type of **Linkage** ("distance to be considered for merges"),

- In **case a.1)** the result using **K-means** is very clear, with 4 cluster and the centers coinciding with the coordinates used for generation. For this case we used **5000 points** and the distribution in the space is very clear.

Some **Suggestions for next steps**, could be:

- For Segmentation purposes (case b)) using k-Means **check with different values of K**, and select the best value using the **Elbow Method.**

- DBSCAN (Density-based clustering) could be also used in case b) to see if could be possible to detect a "*cluster inside some of the clusters*".

  Particularly, as we have seen the type of shapes of the cluster resulting in this case.

- In the case a.2. using **Agglomerative Hierarchical Clustering**, could be also interesting to generate a **bigger number of points** (*for example, 100 or 500*) and see how is impacting depending of the type of Linkage ("Complete" or "Average"), generating for each case the resulting dendogram.