## **Unsupervised Learning with Clustering (partitioning vs. hierarchical)**

# Part 1: Clustering via partitioning with K-Means

We will be working on a wholesale customer segmentation problem. You can download the dataset using [this link](https://archive.ics.uci.edu/ml/machine-learning-databases/00292/Wholesale%20customers%20data.csv). The data is hosted on the UCI Machine Learning repository.

**The aim of this problem is to segment the clients of a wholesale distributor based on their annual spending on diverse product categories, like milk, grocery, region, etc.** So, let’s start coding!

**1) Wholesale dataset EDA:** Data profiling is the process of analysing and summarising data. Exploratory Data Analysis (EDA) is used to gain insights into its structure, content, and relationships, ultimately helping to identify data quality issues and trends. We will first install and import the **ydata-profiling** library.

!pip install ydata-profiling

**ydata-profiling** is a leading tool in the data understanding step of the data science workflow as a pioneering Python package. ydata-profiling is a leading package for data profiling that automates and standardises the generation of detailed reports, complete with statistics and visualisations. The significance of the package lies in how it streamlines the process of understanding and preparing data for analysis in a single line of code!

import pandas as pd

from ydata\_profiling import ProfileReport

#Load your dataset

data = pd.read\_csv('/content/Wholesale customers data.csv')

#Create a data visualisation profile report for your data

profile = ProfileReport(data, title="Profiling Report")

#Save your profile report as HTML web page

profile.to\_file("Wholesle\_Profile.html")

A screenshot of a computer

Description automatically generated

We have the spending details of customers on different products like Milk, Grocery, Frozen, Detergents, etc. Now, we have to segment the customers based on the provided details. Before doing that, let’s pull out some statistics related to the data. Launch the HTML file **Wholesale\_Profile.html**

Here, we see that there is a lot of variation in the magnitude of the data. Variables like Channel and Region have low magnitude, whereas variables like Fresh, Milk, Grocery, etc. have a higher magnitude.

A screenshot of a computer

Description automatically generated

**2) Prepare your dataset for clustering:** First, start by removing any descoped variables from the analysis; in this case, we remove both **Region and Channel** variables.

#list the variables names

list(data.columns)

A screen shot of a computer

Description automatically generated

# Exclude Region and Channel

df = data.drop(['Region', 'Channel'], axis=1)

df.head()

A screenshot of a computer

Description automatically generated

When examining the **ydata profile** report, you will notice a variation in magnitude values by looking at the mean of sales per product. Since K-Means is a distance-based algorithm, scaling your input features before clustering is advised since this magnitude difference can create a problem. Let’s scale the wholesale data with Standard Scaler to bring all the variables to the same magnitude.

# Import the standard scaler

from sklearn.preprocessing import StandardScaler

# Initiate the standard scaler

scaler = StandardScaler()

# Scale your dataset

df\_scaled = scaler.fit\_transform(df)

# Convert df scaled values to a new data frame df\_scaled

df\_scaled = pd.DataFrame(df\_scaled, columns=df.columns)

df\_scaled.head()

A screenshot of a computer

Description automatically generated

**3) Modelling Wholesale dataset with partitioning clustering:** First, the magnitude looks similar now. Next, let’s create a k-means function and fit it into the data:

# Import K-Means algorith

from sklearn.cluster import KMeans

# defining the kmeans function with initialization as random

kmeans = KMeans(n\_clusters=2, init='random')

# fitting the k means algorithm on scaled data

kmeans.fit(df\_scaled)

A blue and white rectangular sign with black text

Description automatically generated

We have initialised two clusters (k=2). Pay attention – the initialisation is random here. However, other initialisation methods, like the k-means++, are known to produce better results.

**4) Evaluating your model:** Let’s evaluate how well the formed clusters are. To do that, we will calculate the inertia of the clusters:

# inertia (Total variations) on the fitted data

kmeans.inertia\_

A number on a grey background

Description automatically generated

The inertia measures how well a dataset was clustered by K-Means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster. A good model is one with low inertia AND a low number of clusters ( K ).

**5) Tuning your model:** We got an inertia value of almost 2000. Now, let’s see how we can use the elbow curve to determine the optimum number of clusters in Python. We will first fit multiple k-means models, and in each successive model, we will increase the number of clusters. We will store the inertia value of each model and then plot it to visualise the result:

# Import matplot to plot the elbow chart

import matplotlib.pyplot as plt

# Import numpy to generate a number sequance

import numpy as np

# Before running the algorithm, there are no variations

# Start with an empty list of total variations values

Total\_variations =[]

# Select a range of k values (clusters) between 1 and 11 clusters

# Build a k-means model for each k value, and append the total variations value

for k in range(1, 11):

kmeans = KMeans(n\_clusters=k, init='random').fit(df\_scaled)

Total\_variations.append(kmeans.inertia\_)

# Plot the K range of values against the Total variations

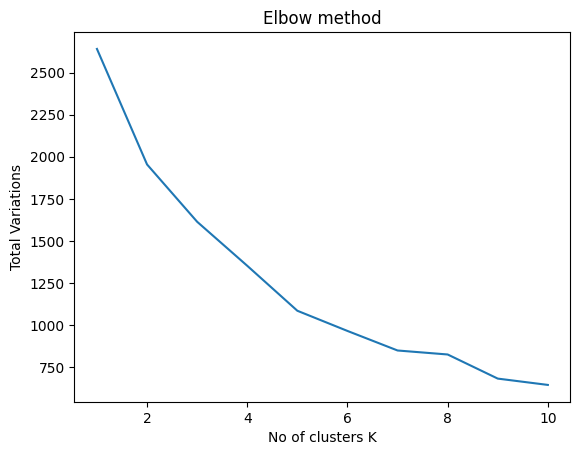
plt.plot(range(1, 11), Total\_variations)

plt.title('Elbow method')

plt.xlabel('No of clusters K')

plt.ylabel('Total Variations')

plt.show()



Alternatively, we can use **Yellowbrick**, a Python library that provides visualisation tools to help with model selection and hyperparameter tuning, particularly for evaluating clustering algorithms and finding the optimal number of clusters (k) using elbow and silhouette analysis methods.

# Install yellowbrick package for elbow method

!pip install yellowbrick

# Import KElbowVisualizer from yellowbrick

from yellowbrick.cluster import KElbowVisualizer

# Initiate the elbow method

elbow\_visualizer = KElbowVisualizer(kmeans, k=(1,11))

# Fit the KElbowVisualizer to the scaled data

elbow\_visualizer.fit(df\_scaled)

elbow\_visualizer.show()

A graph with a line and a line

Description automatically generated with medium confidence

**6) Build your Kmeans model with best k:** Can you tell the optimum cluster value from the above plot? Looking at the above elbow curve, **we can choose any number of clusters between 5**. Let’s set the number of clusters as 5 and fit the model:

# defining the kmeans function with initialization as random

kmeans5 = KMeans(n\_clusters=5, init='random')

# fitting the k means algorithm on scaled data

kmeans5.fit(df\_scaled)

A blue and black rectangular sign

Description automatically generated

**7) Use the clustering model to label your data:**  once the data is partitioned into 5 clusters, we can use the model to label each instance from the training data with the cluster it belongs to.

# Use kmeans (k=5) to label the data points into their 5 clusters

y\_kmeans5 = kmeans5.fit\_predict(df\_scaled)

y\_kmeans5

A screenshot of a computer

Description automatically generated

Add these labels as a new column to the original data, so it becomes known for each instances to which cluster it belongs to.

# Create a new column with the clusters labels

data['Cluster'] = y\_kmeans5

data.head()

A screenshot of a grocery list

Description automatically generated

shuffled\_data = data.sample(frac=1)

shuffled\_data.head(10)

A graph with numbers and lines

Description automatically generated with medium confidence

Finally, let’s look at the value count of points in each of the above-formed clusters and this is how we can implement K-Means Clustering in Python.

data['Cluster'].value\_counts()

A screenshot of a computer

Description automatically generated

# Part 2: Agglomerative Hierarchical Clustering via

In this example, we will perform hierarchical clustering on real-world data and see how it can be used to solve an actual problem.

The problem that we are going to solve in this section is to segment customers into different groups based on their shopping trends. The dataset for this problem can be downloaded from Kaggle. To cluster this data into groups we will follow the same steps that we performed in the previous section.

**1) Import the required libraries for analysis:** Execute the following script to import the desired libraries:

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

**2) Understand your dataset:** Also, understand the dimensions, the number of records and attributes, and the various features in your dataset.

customer\_data = pd.read\_csv('/content/Customer shopping data.csv')

customer\_data.shape



The script above will return (200, 5) which means that the dataset contains 200 records and 5 attributes. To eyeball the dataset, execute the head() function of the data frame.

customer\_data.head()

A screenshot of a graph

Description automatically generated

Our dataset has five columns: CustomerID, Genre, Age, Annual Income, and Spending Score. To view the results in two-dimensional feature space, we will retain only two of these five columns. We can remove CustomerID column, Genre, and Age column. We will retain the Annual Income (in thousands of dollars) and Spending Score (1-100) columns. The Spending Score column signifies how often a person spends money in a mall on a scale of 1 to 100 with 100 being the highest spender. Execute the following script to filter the first three columns from our dataset:

# Exclude Region and Channel

data = customer\_data.drop(['CustomerID', 'Genre', 'Age'], axis=1)

data.head()

A screenshot of a graph

Description automatically generated

**3) Create a dendrogram:** Next, we need to know the clusters that we want our data to be split to. We will again use the **SciPy**library to create the dendrograms for our dataset. We will use **“Ward” linkage** to merge the various groups. **Linkage Methods** are methods used to determine the distance or similarity between clusters when deciding which ones to merge.

In hierarchical clustering, Ward's linkage minimizes the increase in within-cluster variance when merging clusters, aiming for compact and well-separated clusters by minimizing the spread of data points within them.

# import the SciPy library and the hierarchal clustering package

import scipy.cluster.hierarchy as shc

# specify the size of your figure and add a title

plt.figure(figsize=(10, 10))

plt.title("Customer Dendograms")

# plot the dendrogram

dendrogram = shc.dendrogram(shc.linkage(data, method='ward'))

In the script above we import the hierarchy class of the **scipy.cluster** library as shc. The hierarchy class has a dendrogram method which takes the value returned by the linkage method of the same class. The linkage method takes the dataset and the method to minimize distances as parameters. We use 'ward' as the method since it minimizes the variants of distances between the clusters (see lecture notes).

A diagram of a diagram

Description automatically generated

**4) Determine the desired number of clusters:** If we draw a horizontal line that passes through longest distance without crossing a horizontal line, we get 5 clusters as shown in the following figure:

A screenshot of a diagram

Description automatically generated

**5 ) Build your Clustering model with 5 clusters:** Now we know the number of clusters for our dataset, the next step is to group the data points into these five clusters.

To do so we will again use the**AgglomerativeClustering** class of the **sklearn.cluster**library. Take a look at the following script:

# import and instantiate the bottom-up hierarchal clustering algorithm

from sklearn.cluster import AgglomerativeClustering

#specify the linkage and number of clusters hyperparameters

cluster = AgglomerativeClustering(n\_clusters=5, linkage='ward')

# build your clustering model

agglomerative\_model = cluster.fit(data)

agglomerative\_model

A blue and white rectangle with black text

Description automatically generated

**6) Use the clustering model to label the data:** Here we use the clustering model to label each instance with the cluster it belongs to using the **fit\_predict** method.

customers\_lables = agglomerative\_model.fit\_predict(data)

customers\_lables

The output of the script above looks like this:

A black background with white numbers

Description automatically generated

You can see the cluster labels from all of your data points. Since we had five clusters, we have five labels in the output i.e. 0 to 4. We can tidy this up by creating a data frame with each instance and its label.

# Create a new column with the clusters labels

data['Cluster'] = customers\_lables

shuffled\_data = data.sample(frac=1)

shuffled\_data.head(10)

A screenshot of a computer program

Description automatically generated

# To find the count of customers within each cluster

data['Cluster'].value\_counts()

A screenshot of a computer

Description automatically generated

**7) Visualise the customers segments:** As a final step, let's plot the clusters to see how actually our data has been clustered:

plt.figure(figsize=(10, 10))

plt.scatter(data['Annual Income (k$)'], data['Spending Score (1-100)'], c = cluster.labels\_, cmap='coolwarm')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.title('Customer segments')

plt.show()

The output of the code above looks like this:

A graph with red blue and white dots

Description automatically generated

You can see the data points in the form of five clusters. The data points in the bottom right belong to the customers with high salaries but low spending. These are the customers that spend their money carefully. Similarly, the customers at top right (green data points), these are the customers with high salaries and high spending. These are the type of customers that companies target. The customers in the middle (blue data points) are the ones with average income and average salaries. The highest numbers of customers belong to this category. Companies can also target these customers given the fact that they are in huge numbers, etc.