1. **Unsupervised Machine Learning**
   1. **Chapter Objectives**

This chapter begins by introducing the concept of unsupervised learning and contrasting it with supervised learning. We will use ‘Netflix Userbase Dataset’ (available at <https://www.kaggle.com/datasets/arnavsmayan/netflix-userbase-dataset> )

as a case study to enhance your understanding of how Unsupervised Machine Learning techniques can be applied to unstructured business data to address business challenges. We will focus on five commonly used methods: K Means Clustering, PAM (K Medoids) clustering, Hierarchical Clustering, Gaussian Mixture Model clustering (EM algorithm), and Density-Based Spatial Clustering of Applications with Noise (DBSCAN). We hope you will enjoy developing your skills from the perspective of a ‘Business Analyst’ or ‘Business Consultant’.

* 1. **Supervised vs unsupervised machine learning**

In Chapter 6, you learned that Supervised learning is “*the process of learning a function that maps feature variables to target variables. (Dive Into Data Science, Page 122).”*

Suppose your client wants to run a personalized marketing campaign and asks for your advice on segmenting their customers into different groups based on characteristics such as geographical locations, driving habits, income level, education level, and so on. In this scenario, you would need to determine how many groups have similar characteristics without annotation. An unsupervised learning approach is beneficial in this case as it can provide invaluable data-driven insights and ideas compared to intuition-based or experience-based approaches. Therefore, ‘one of the main goals of unsupervised learning is to understand how subsets of data relate to one another’ (Page 143).

A map of a person

Description automatically generated with medium confidence

The process of identifying similar instances and segmenting them based on their similarities is known as clustering. There are two types of clustering methods: ‘hard clustering’ and ‘soft clustering’. In hard clustering, each data point is assigned to exactly one cluster. This method is beneficial when the data is clearly separated and non-overlapping. For example, suppose you are asked to run a personalized marketing campaign for frequent buyers (first cluster) and occasional buyers (second cluster) at an online store. The store has purchasing history, income, age, location, gender, and web browsing history in their database. In this scenario, the hard clustering method is useful as each data point is assigned to one of two clusters: frequent buyers or occasional buyers. We will learn about K-Means Clustering, PAM (K-Medoids) clustering, and Hierarchical Clustering methods as hard clustering methods.

Soft clustering can be useful when your data distribution is complex, overlapping, and may contain hidden patterns that are not immediately apparent. This method, also known as density estimation, helps understand how the data is spread out in space. In this chapter, we will analyze the Gaussian Mixture Model clustering (EM algorithm), a type of soft clustering.

Choosing the right clustering method depends on the size, dimensionality, and nature of your data, as well as the specific requirements of the project, such as budget and timeline. It’s important to understand the assumptions and limitations of each method to make an informed decision. To better illustrate the benefits of each method in a business use case, we will do the analysis under a store scenario in this chapter.

* 1. **Data Preparation and Exploratory Data Analysis**

In this section, let’s take a moment to perform the crucial step of Exploratory Data Analysis through our hands-on analysis using Python code. This will help us understand the structure, quality, and relationships within our data, and prepare it for further analysis. Codes are written in **bold format** and explanations are added after # so that you can copy and paste the codes when necessary.

If you are already familiar with this process, please feel free to jump ahead to section 8.4 K-Means Clustering.

* + 1. **Import All Necessary Libraries**

#Pandas for powerful data manipulation

**import pandas as pd**

#Numerical Python library that supports large multi-dimensional arrays and matrices

**import numpy as np**

#Visualization library to draw statistical graphics

**import matplotlib.pylab as plt**

#Python's visualization library

**import seaborn as sns**

#Module to scale the data using z-score

**from sklearn.preprocessing import StandardScaler**

#importing clustering algorithms

**from sklearn.cluster import KMeans**

**from sklearn.mixture import GaussianMixture**

**from sklearn.cluster import AgglomerativeClustering**

**from sklearn.cluster import DBSCAN**

#Silhouette score to compare model performance.

**from sklearn import metrics**

**from sklearn.metrics import silhouette\_score**

* + 1. **Preparation**

The following process is carried out after downloading the ‘Netflix Userbase Dataset’ from this URL:

<https://www.kaggle.com/datasets/arnavsmayan/netflix-userbase-dataset>

# Import the file from the folder you saved and unzip the file.

**import zipfile**

**def read\_zip\_file(filepath):**

**with zipfile.ZipFile(filepath, 'r') as zip\_ref:**

**zip\_ref.extractall()**

# Please replace 'filepath' with the path to the zip file you want to read

**read\_zip\_file('archive.zip')**

# Reads the CSV file and stores the data in a pandas DataFrame, df.

**df = pd.read\_csv('Netflix Userbase.csv')**

# displays the first five rows of the DataFrame to give quick observation of the data.

**df.head()**

A screenshot of a computer

Description automatically generated

* + 1. **Exploratory Data Analysis (EDA)**

This section showcases some examples of EDA. To reiterate, our objective in conducting EDA is to acquire a comprehensive understanding of our data. This involves taking necessary measures such as normalization, standardization, and data cleansing to guarantee more precise results and insights.

# Let's get a summary of the dataframe to view its structure and contents at a high level.

**df.info()**

A screenshot of a computer screen

Description automatically generated

**Observations:**

* There are 2,500 observations and 9 columns in the dataset.
* All columns have 2,500 non-null values i.e. there are no missing values.
* User ID, Monthly Revenue, and Age are numeric data types and all the others are object data types.

# Checking unique values in each column to prepare for data analysis and plan for necessary preprocessing steps.

**df.nunique()**

A screenshot of a computer screen

Description automatically generated

**Observations:**

* All User IDs are unique.

# Checking if there are any duplicated values. When duplicates are found, it's recommended to understand the cause and decide whether or not to remove the duplicates.

**df[df.duplicated()]**

**Observations:**

* There are no duplicates.
* As all User IDs are unique, they would not add value to our analysis. Hense, we can drop the User ID column.
* We will also drop the Join Date and Last Payment Date columns as well due to the irrelevance for clustering.

# Dropping columns that are not relevant for clustering.

**df = df.drop(['User ID', 'Join Date', 'Last Payment Date', 'Plan Duration'], axis=1)**

Given that our data comprises two types of variables, numerical and categorical, we will segregate them and store them in distinct lists.

# Creating a list of numerical columns

**num\_cols=['Monthly Revenue','Age']**

# Creating a list of categorical variables

**cat\_cols= ['Subscription Type', 'Country', 'Gender', 'Device']**

# Checking summary statistics of the numerical columns to understand their distribution and potential outliers.

**df[num\_cols].describe().T**

A close-up of numbers

Description automatically generated

**Observations:**

* The average Monthly Revenue is 12, whereas the average Age is 39.
* The standard deviation is low for Monthly Revenue, meaning there is not much variance.
* The standard deviation of Age is 7. The youngest individual is 26 years old and the oldest is 51 years old.

# Printing the percentage of subcategories in each category.

**for i in cat\_cols:**

**print(df[i].value\_counts(normalize=True))**

**print('\*'\*40)**

A screenshot of a computer

Description automatically generated

# Checking how Subscription Type is related to other categorical variables.

**for i in cat\_cols:**

**if i!='Subscription Type':**

**(pd.crosstab(df[i],df['Subscription Type'],**

**normalize='index')\*100).plot(kind='bar',figsize=(8,4),**

**stacked=True)**

**A chart of a bar graph

Description automatically generated with medium confidence plt.ylabel('Percentage Subscription Type %')**

**A graph of different colored bars

Description automatically generated**

**A graph of different colored squares

Description automatically generated**

Observation: We can see that subscription types are equally distributed among device and gender categories. However, we observe some unique characteristics in the ‘Country’ category. For instance, both the United Kingdom and Mexico have a high number of Standard subscribers. The distribution of Basic subscribers is similar in Brazil, Germany, and Italy.

We have ovserved that our dataset contains two types of values: numerical and categorical. To compute similarities or to feed the data into a machine learning model, we often need to convert categorical values into a numerical format.

Two techniques commonly employed for this conversion are Label Encoding and One-Hot Encoding.

***Label Encoding***

It is a simple and intuitive method that assigns a unique numerical label to each category within a feature. This encoding method is preferred when the categorical variable has an inherent order or hierarchy, such as clothing sizes where 'S' (Small) could be represented as 0, 'M' (Medium) as 1, 'L' (Large) as 2, and 'XL' (Extra Large) as 3. However, label encoding could misinterpret the numerical labels as ordinal data, hence we should carefully analyze the characteristics of our data.

***One-Hot Encoding***

This method is preferred when the categorical data has no inherent order, like countries or devices. In this method, each category for a feature is converted into a new binary feature (0 or 1), effectively spreading the categories out into multiple dimensions in the feature space. This prevents any notion of order or priority among the categories, which is beneficial for many machine learning models.

However, one-hot encoding can significantly increase the dimensionality of the data, which might lead to the curse of dimensionality if not handled properly.

Another tip for converting categorical values of no inherent order is using the **get\_dummies** method, provided in the Pandas libraries in Python. This method is simpler and more handy for simple data analysis.

# Creating a list of columns to create dummy variables

**to\_get\_dummies\_for = ['Subscription Type', 'Country', 'Gender', 'Device']**

#creating dummy variables

**df\_converted= pd.get\_dummies(data = df, columns= to\_get\_dummies\_for, drop\_first= False)**

# Printing the first five rows in our dataframe to visually observe the converted values.

**df\_converted.head()**

**A screenshot of a computer

Description automatically generated**

After successfully running the **get\_dummies** method, the categorical columns are converted to numerical values.

However, continuous features such as 'Age' and 'Revenue' remain in their original scale, and we should standardize them.

Some machine learning algorithms such as K-Means are sensitive to the scale of the features, as it measures the distance of each data point to determine similarities. For this reason, we will apply the **'StandardScaler'** in the following steps.

# Initializing the StandardScaler

**scaler = StandardScaler()**

# Selecting the numerical data from the dataframe

**numeric\_data = df\_converted.select\_dtypes(include = ['float64', 'int64'])**

# Fitting the scaler to the numerical data and transforming it

**scaled\_data = scaler.fit\_transform(numeric\_data)**

# Replacing the original numerical data in the dataframe with the scaled data

**df\_converted[numeric\_data.columns] = scaled\_data**

# Printing the first five rows in our dataframe to visually observe the standardized values.

A screenshot of a computer

Description automatically generated**df\_converted.head()**

Having prepared our data, we are now set to embark on our clustering analysis, initiating with K-Means Clustering.

* 1. **K-Means Clustering**

K-means clustering is a popular method due to its simplicity, speed, and efficiency. Businesses often see these benefits, especially when dealing with large datasets. The following are the steps involved in K-means clustering:

1. Specify the number of clusters, *K*.
2. Initialize the centroids of each cluster.
3. Assign each observation to the nearest centroid.
4. Repeat the Classification and Adjustment steps until reaching a stopping condition.

When applying K-means clustering, a critical step is determining the appropriate number of clusters, K. In real-world scenarios, choosing the optimal K value can be challenging, even after visualizing the data through boxplots, scatterplots, or heatmaps like ours. Furthermore, you may need to justify your choice of K to stakeholders or decision-makers.

One effective method for determining K is the Elbow Plot. This technique involves plotting the explained variation as a function of the number of clusters, and picking the elbow of the curve as the number of clusters to use. The “elbow” is indicative of the point where adding another cluster provides diminishing returns in terms of explained variation.

Let’s apply the Elbow Plot to our Netflix Userbase Data so that we can have better understanding visually. The goal of K-means clustering is to minimize dissimilarities within clusters. This is achieved by minimizing the Sum of Squared Errors (SSE), which is the sum of the squared distances between each member of a cluster and its centroid.

# We start by initializing an empty dictionary to store the SSE for each number of clusters.

**sse = {}**

# We then iterate over different values of K (the number of clusters) and fit the scaled data to the algorithm.

# We use the inertia attribute from the clustering object and store the inertia value for that k.

**for k in range(1, 10):**

**kmeans = KMeans(n\_clusters=k, random\_state=1).fit(df\_converted)**

**sse[k] = kmeans.inertia\_**

# Finally, we plot the SSE values against the number of clusters to create an Elbow plot.

**plt.figure()**

**plt.plot(list(sse.keys()), list(sse.values()), 'bx-')**

**plt.xlabel("Number of cluster")**

**plt.ylabel("SSE")**

**plt.show()**

A graph with a blue line

Description automatically generated

Observation: The above elbow plot shows a gentle curve. However, after K=4, there doesn't seem to be much change. Therefore, we chose 4 as our initial number of clusters.

We will continue to write code for our K-means clustering.

**num\_clusters = 4**

# Initializing a K-Means clustering model with 4 clusters and setting the random\_state to 0 for reprducibility.

**kmeans = KMeans(n\_clusters=num\_clusters, random\_state=0)**

**df\_converted['Cluster'] = kmeans.fit\_predict(df\_converted)**

# Calculating the mean of all columns in the ‘df\_converted’ dataset.

**cluster\_stats = df\_converted.groupby('Cluster').mean()**

# Fitting the K-means model to our data stored in the 'df\_converted' dataframe.

**kmeans.fit(df\_converted)**

# Assigning each data point in the 'df\_converted' dataframe to one of the K clusters by using the 'predict' method.

**cluster = kmeans.predict(df\_converted)**

# Creating a copy to avoid modifying the original dataframe.

**df\_cluster = df\_converted.copy()**

# Adding the cluster assignments to our dataframe to store the results of our clustering analysis.

**df\_cluster['cluster']=cluster**

# Using a for loop to go through each of the four clusters to calculate the mean of each feature of the data points, and store the calculated means in a new column, cluster 0, 1, 2, or 3.

**for i in range(4):**

**df\_results[f'cluster {i}'] =**

**df\_cluster[df\_cluster['cluster'] == i].mean().tolist()**

# Printing first five rows to see our results.

**df\_cluster.head()**

A screenshot of a computer

Description automatically generated

# To make the data table more visually understandable, we're setting the index and dropping unnecessary rows ('Cluster', 'cluster') from our dataset.

**df\_results = df\_results.set\_index(df\_cluster.columns)**

**df\_results = df\_results.drop(['Cluster', 'cluster'], errors='ignore')**

**df\_results**

A table of numbers and symbols

Description automatically generated

As we have seen during our EDA, some features such as Subscription Type, Gender, and Device are fairly equally distributed. Yet, if we pay attention to the values that are the highest among each cluster, we can observe the following characteristics in the groups:

[Cluster 0]

* Monthly revenue: Highest
* Age: Youngest
* Dominant Subscription Type: Basic
* Countries: Italy, Spain
* Gender distribution: More male
* Most used device: Tablet

[Cluster 1]

* Monthly revenue: Second Highest
* Age: Second Oldest
* Dominant Subscription Type: Basic
* Countries: France, United States
* Gender distribution: More female
* Most used device: Laptop

[Cluster 2]

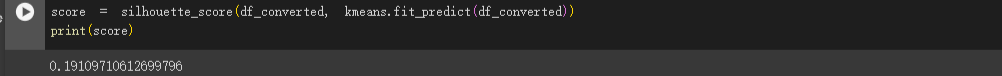
* Monthly revenue: Second Lowest
* Age: Second Youngest
* Dominant Subscription Type: Basic
* Countries: Australia, Germany, Mexico
* Gender distribution: More male
* Most used device: Smartphone

[Cluster 3]

* Monthly revenue: Lowest
* Age: Oldest
* Dominant Subscription Type: Basic
* Countries: United Kingdom
* Gender distribution: More female
* Most used device: Smartphone

Now you see how the K-means clustering method can help us observe the multi-dimensional data and find unseen characteristics from our dataset. Based on the results, we could formulate our marketing plan, choose an advertising method, or decide how to run a campaign. Also, we could study what channels people in each cluster watch, then be able to build a more welcoming recommendation system.

However, K-means clustering does have some disadvantages. It is structured to identify clusters of roughly equal sizes, lacking the adaptability to discover clusters of varying sizes and densities. Deciding how many clusters to use at the beginning could also be a disadvantage. This can be further illustrated by the calculation of silhouette score – as shown in the following:



As we can see, the calculated silhouette score is around 0.19, which is close to 0, indicating that the clusters are close to each other. In other words, our choice of 4 clusters does not perform well in this case.

* 1. **PAM (K Medoids) clustering**

Partitioning Around Medoids (PAM), also known as K-Medoids clustering, is another clustering algorithm used in data analysis. The key difference between K-means clustering, which we previously studied, and PAM clustering lies in their reference points. PAM uses the most centrally located object within a cluster, known as the ‘medoid’, as a reference point, while K-means uses the mean value of the objects in a cluster.

The primary advantage of PAM clustering lies in its ability to mitigate the effects of noise and outliers. Furthermore, since PAM employs actual data points as cluster centers, the clusters can be more readily interpreted. For instance, the medoid of a cluster could exemplify a ‘typical’ Netflix user within that cluster, thereby simplifying the understanding of the group’s characteristics.

PAM clustering has two phases: the Building Phase and the Swapping Phase. In the Building Phase, the algorithm initializes the most centrally located points, medoids, and assigns each data point to the closest medoid. In the Swapping Phase, the algorithm fine-tunes the medoids and ensures that it converges to a stable solution. By combining the two phases, the PAM clustering algorithm can cluster datasets efficiently. Let’s break down each phase and understand the necessary steps.

**Building Phase**

* + 1. Choose a value for K to determine the number of clusters that the algorithm generates.
    2. Select K points within the dataset as the initial representative points (initial medoids) to start the algorithm.
    3. Calculate the distance from each medoid to all other points. Euclidean distance or Manhattan distance are popular methods used for the calculation. This step helps partition the data into clusters around medoids.
    4. Assign each point to the cluster with the closest medoid in order to minimize the dissimilarities within each cluster.
    5. Calculate the total cost of the current cluster assignments.

After the completion of the building phase, we proceed to the Swapping Phase, also known as the “Repeating Phase”, to optimize the clustering.

**Swapping Phase**

* + 1. Randomly select a non-medoid object, *Orandom*, that is not currently a medoid.
    2. Calculate the distance from *Orandom* to all other points to partition the data into clusters around the newly selected medoids.
    3. Assign each point to the cluster with the closest medoid (that has fewer dissimilarities) to minimize the dissimilarities within each cluster.
    4. Calculate the total cost to assess if *Orandom i*s a better choice than the current one, and if it should replace the current medoid.
    5. If the swap decreases the cost function (less than zero), replace the current medoid with *Orandom.*
    6. Repeat Steps 6 to 10 until the cost no longer decreases. In other words, you need to continue the “Swapping Phase” until you find the most representative medoids and the most cohesive clusters.

Now, let’s proceed to execute the PAM clustering on the Netflix Userbase Dataset and

juxtapose the outcomes.

#Installing the scikit-learn-extra package in Python then import KMedoids class (method)

**pip install scikit-learn-extra**

**from sklearn\_extra.cluster import KMedoids**

#Creating an instance of the K-Medoids class, setting the number of cluster as 4. We also set the random\_state to ensure the results are reproducible.

**kmedo = KMedoids(n\_clusters = 4, random\_state=0)**

# Applying the K-Medoids clustering to the df\_converted dataset.

**kmedo.fit(df\_converted)**

# First predicting the closest datapoint in each df\_converted dataset by using the already fitted model. Then adding these predictions as a new column kmedoLabels to the df\_converted dataframe.

**df\_converted['kmedoLabels'] = kmedo.predict(df\_converted)**

# First making a copy of the original dataset before moving forward by creating a new variable: original\_features.

**original\_features = df\_converted.copy()**

# Calculating mean and median of all features of each cluster to understand the central tendency of each cluster.

**mean = df\_converted.groupby('kmedoLabels').mean()**

**median = df\_converted.groupby('kmedoLabels').median()**

#Stacking the median and mean dataframe and storing the result in df\_kmedoids

**df\_kmedoids = pd.concat([mean, median], axis=0)**

# Replacing the index to easily understand each cluster's mean and median.

**df\_kmedoids.index = ['cluster\_0 Mean', 'cluster\_1 Mean', 'cluster\_2 Mean', 'cluster\_3 Mean', 'cluster\_0 Median', 'cluster\_1 Median', 'cluster\_2 Median', 'cluster\_3 Median']**

# For our visualization purpose, transposing swaps the rows and columns of the DataFrame.

**df\_kmedoids.T**

A table of numbers with numbers

Description automatically generated

Since our Netflix data has more categorical values, we do not get the median like when we have more numerical values. Let’s give our observation based on the mean of each cluster:

[Cluster 0]

* Monthly revenue: Lowest
* Age: Second Youngest
* Dominant Subscription Type: Standard
* Countries: Germany, Mexico, United States
* Gender distribution: More Male
* Most used device: Smartphone

[Cluster 1]

* Monthly revenue: Highest
* Age: Youngest
* Dominant Subscription Type: Premium
* Countries: Australia, France, Spain
* Gender distribution: More male
* Most used device: Tablet

[Cluster 2]

* Monthly revenue: Second Highest
* Age: Oldest
* Dominant Subscription Type: Basic
* Countries: Brazil, Italy
* Gender distribution: More female (significantly higher values)
* Most used device: Laptop

[Cluster 3]

* Monthly revenue: Second lowest
* Age: Second highest
* Dominant Subscription Type: Basic
* Countries: Canada, United Kingdom
* Gender distribution: More Male (significantly higher values)
* Most used device: Smartphone

In comparison to the K-Means clustering, we can observe that some characteristics of each cluster have changed. We choose a clustering method depending on our data characteristics, goals, and objectives, or we evaluate our methods and choose the best model.

* 1. **Hierarchical Clustering**

While we have seen that K-means and PAM clustering use partitional methods, Hierarchical Clustering takes a different approach. It is an agglomerative method that progressively merges or splits existing groups. Hierarchical clustering provides a tree-like visual representation, known as a dendrogram, which allows us to understand relationships at multiple scales. One of the key benefits of Hierarchical clustering is that you do not have to determine the number of clusters ‘K’ in advance.

When using Hierarchical clustering, the choice of linkage and distance method is crucial as it determines how the distance between clusters is calculated, which in turn affects the results. The objective of Hierarchical clustering is to end up with a single cluster that contains all data points.

Now let’s dive into how Hierarchical clustering is built by understanding each step and method to use.

* + 1. Treat each data point as its own cluster. In other words, we consider all data points for this clustering method.
    2. Measure the distance between each pair of clusters using the distance method such as Euclidean distance and Manhattan Distance, then record them in a distance matrix to identify which ones are closest to each other.
    3. Merge clusters that are closest to each other, calculated by one of the following linkage methods:
* *Single Linkage*: Takes the shortest distance between any two points in the cluster.
* *Average Linkage:* Takes the average distance between each point in one cluster to every point in the other cluster.
* *Complete Linkage:* Takes the longest distance between any two points in the cluster.
* *Ward Linkage*: Forms clusters that are most similar to each other to minimize the total within-cluster variance and reduce the overall spread of the data.
  + 1. Adjust the distance matrix to reflect the distance between the newly formed cluster and the original clusters.
    2. Repeat steps 3 and 4 (the merging process) until all the data points are merged into a single cluster.

Based on the above understanding, we will now plot the dendrogram for visual intuition on how well each linkage method is able to separate our Netflix Userbase Dataset.

# Importe the dendrogram and linkage functions from the scipy.cluster.hierarchy

**from scipy.cluster.hierarchy import dendrogram, linkage**

# List of all linkage methods to check

**methods = ['single',**

**'average',**

**'complete',**

**'ward']**

# Create a subplot image

**fig, axs = plt.subplots(len(methods), 1, figsize=(20, 15))**

# Enumerate through the list of all methods above, get linkage and plot dendrogram

**for i, method in enumerate(methods):**

**Z = linkage(df\_converted, metric='euclidean', method=method)**

**dendrogram(Z, ax=axs[i]);**

**axs[i].set\_title(f'Dendrogram ({method.capitalize()} Linkage)')**

**axs[i].set\_ylabel('Distance')**

**A close-up of a graph

Description automatically generatedA screenshot of a graph

Description automatically generated**

In the realm of hierarchical clustering, the dendrogram’s height is a visual representation of the dissimilarity or distance between clusters. The higher the linkage merges on the diagram, the greater the dissimilarity between those clusters. From the above results, we observe that the Ward linkage with the number of clusters set to 4,

A screenshot of a computer

Description automatically generated appears to provide a well-spread and clear picture, indicating a more balanced and well-defined result.

Throughout the exercise, you might have noticed that hierarchical clustering took longer to yield results. Indeed, one of the disadvantages of hierarchical clustering is its time-consuming nature and the requirement for substantial computational power. This can become particularly costly, especially with large datasets, as the algorithm needs to calculate and compare distances among multiple data points or clusters. Therefore, it is recommended that you consider the size and nature of your dataset prior to your analysis.

* 1. **Gaussian Mixture Model clustering (EM algorithm)**

So far, we have explored clustering methods where each data point belongs only to the closest cluster, or nearest mean, as part of the ‘hard clustering’ methods. In the context of our Netflix scenario, we would prefer to conduct our marketing activities by analyzing our customers from multiple dimensions. This is because a user often exhibits characteristics that could place them into multiple groups. The Gaussian Mixture Model (GMM) could be beneficial for this multi-dimensional approach, as the model assumes that all data points are generated from a finite number of Gaussian distributions, also known as normal distributions, which we often see in a bell curve.

The Gaussian Mixture Model is a density estimation algorithm used to estimate the probability density function of the underlying distributions. Therefore, we only need two numbers: a mean that determines the center of the bell curve, and a variance that shows how widely the bell curve is spread out. Since each of these distributions is characterized by parameters (mean and covariance), the Expectation-Maximization (EM) algorithm, which estimates these parameters by maximizing the likelihood of the observed data, is crucial to the GMM model.

Let’s now observe how we can implement the EM algorithm:

1. *Random Selection*: Randomly determine the mean and the variance of every cluster.
2. ***Expectation (E-step)***: Calculate the probability that each data point belongs to each Gaussian distribution given the current parameters.
3. ***Maximization (M-step)***: Adjust the position and cluster boundaries for each cluster by using the expected values of the hidden variables that we obtained during the E-step, to maximize the parameter.
4. *Repetition*: Repeat the above E-step and M-step until they stop changing significantly (reach a local optimum) or reach a threshold we have set in advance.

Now that we have a basic understanding of the GMM and the EM algorithm, let’s apply them to our Netflix data and analyze the results.

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

Let’s enhance the clarity of our DataFrame by discarding columns that are not needed. Also, in order to provide a more visual representation of our analysis, we will generate a heatmap, which can offer a more instinctive comprehension of our data.

# Create a copy to avoid modifying the results.

**gmm\_table = df\_gmm\_results.copy()**

# Drop the unnecessary columns from the copy

**gmm\_table = gmm\_table.drop(['Cluster', 'kmedoLabels', 'GMM\_Cluster'], axis=0)**

# Pring the modified results.

**gmm\_table**

A table of numbers and symbols

Description automatically generated

# Set the figure size of our Heatmap.

**plt.figure(figsize=(10, 8))**

# Create a heatmap

**sns.heatmap(gmm\_table, cmap='coolwarm', annot=True)**

# Show the plot

**plt.show()**

A screenshot of a data table

Description automatically generated

Let’s give our observation based on the results.

[Cluster 0]

* Monthly revenue: Highest
* Age: Second youngest
* Dominant Subscription Type: Basic
* Countries: Brazil, Canada, France, Spain
* Gender distribution: More Male
* Most used device: Tablet

[Cluster 1]

* Monthly revenue: Second Highest
* Age: Oldest
* Dominant Subscription Type: Basic
* Countries: United Kingdom, United States
* Gender distribution: More female
* Most used device: Laptop and Smart Phone

[Cluster 2]

* Monthly revenue: Lowest
* Age: Youngest
* Dominant Subscription Type: Basic
* Countries: Canada, Italy
* Gender distribution: Female only
* Most used device: Smartphone and Tablet

[Cluster 3]

* Monthly revenue: Second Lowest
* Age: Second Highest
* Dominant Subscription Type: Basic
* Countries: Australia, Canada, Germany, Mexico, United States
* Gender distribution: Male only
* Most used device: Smart TV and Smartphone

Based on the above observations, let’s explore how we can tailor our marketing campaigns.

* One common feature among all clusters is that the Basic subscription type is the most selected. We could then run discounts or limited-time promotions to encourage them to upgrade their subscription type to Premium.
* In Cluster 0, we can observe that Spain has a significantly higher ratio compared to other countries, with the exception of the United States. Therefore, it would be a good idea to run country-specific promotions in Spain, such as using popular celebrities, songs, and characters, in the Spanish language.
* Clusters 2 and 3 share some common or similar observations, yet the gender distribution is completely opposite in each. It might be beneficial to conduct an A/B testing type of marketing campaign to analyze how results differ across genders. Subsequently, we could see if we can apply the insights gained to the other cluster group.
* The most frequently used devices—Tablets, Laptops, and Smartphones—suggest that subscribers often watch Netflix outside their homes. Our marketing campaign is not only tailored to be device-specific, but we also recommend collecting additional data such as timestamps and geographical information. This would help us understand subscribers’ behavioral patterns and refine our marketing campaign.

As we conclude this session, we have a few recommendations. When using the GMM with the EM algorithm, it’s crucial to establish a limit or maximum iteration count to avoid endless execution of the algorithm. Please note that convergence might take time as the E step and M step are executed repeatedly until the highest optimum is reached. Occasionally, to enhance performance, K-means clustering is used as a preliminary step before applying the GMM, given that the initial parameter values can have a substantial impact on the outcomes.

* 1. **Density-Based Spatial Clustering of Applications with Noise (DBSCAN)**

Lastly, we will examine the Density-Based Spatial Clustering of Applications with Noise (DBSCAN). DBSCAN is a density-based clustering algorithm that groups together points in high-density regions and separates them from low-density regions. One of DBSCAN’s most significant advantages is its ability to detect outliers and categorize them as noise, rather than clustering them with their ‘nearest neighbors’. Other benefits include not requiring us to predefine the number of clusters, being well-suited to analyzing complex datasets due to its density-based nature, and flexibility in handling clusters of various shapes. It can also identify the center of dense areas by ignoring noise or less dense areas.

You might wonder how we can apply this DBSCAN method to a business context like our Netflix data. Let’s brainstorm some examples of how we can benefit from our analysis. Suppose we have run various marketing campaigns such as limited-time offers, country-specific promotions, A/B testing, or used popular celebrities to target specific groups. We might then observe, or hope to see, unusual behavior or patterns from the targeted groups due to the campaign. Then we want our model to identify these unusual customer activities as outliers or noise so that we can gain more valuable insights, leveraging the strengths of the DBSCAN algorithm.

In order to understand the concept of DBSCAN, we will first understand two key parameters: ***ε (eps)*** and ***Minimum Points (min\_samples)***. Then, we will describe the types of points that DBSCAN determines: ***Core Points, Border Points***, and ***Outlier Points (Noise Points)***.

**Selecting Two Key Parameters:**

The initial step in running the model involves selecting certain parameters. This is a crucial step, as choosing appropriate values can significantly impact our results. If our ε (eps) value is too small, it may result in many data points being classified as noise, whereas a large ε (eps) may cause multiple clusters to merge. Each parameter is defined as follows:

* **ε (eps):** This parameter represents the maximum distance between two points for them to be considered as neighbors. If the distance is equal to or less than the ε value, those points are considered as neighbors.
* **Minimum Points (min\_samples):** This parameter represents the minimum number of points required to form a dense region, which is then considered a cluster.

Once the parameters are set, DBSCAN begins its operation by visiting each data point in the dataset. For each point, it calculates whether it’s a Core Point, Border Point, or Noise Point based on the predefined parameters and the point’s position relative to others.

**Types of Points:**

DBSCAN uses these types of points to identify dense regions in the data space, group them together, and form clusters. The algorithm begins by selecting an arbitrary point and continues until all points have been either assigned to a cluster or identified as outliers.

* **Core Points**: A point is classified as a Core Point if there are at least a minimum number of points *min\_samples*, including the point itself, within the *ε* (eps) radius.
* **Border Points**: A point is considered a Border Point if it is within the *ε* (eps) radius of at least one Core Point but does not have *min\_samples* within its own ε radius.
* **Outlier Points:** A point is considered an Outlier Point or Noise Point if it does not meet the criteria to be either a Core Point or a Border Point.

With the basic understanding of how DBSCAN forms clusters, let’s apply it to our Netflix data and analyze the results.

# Import the DBSCAN class from the sklearn.cluster module

**from sklearn.cluster import DBSCAN**

# Apply eps=0.5 and min\_samples=5 to the model

**dbscan = DBSCAN(eps=0.5, min\_samples=5)**

# Fit the DBSCAN model and predict the cluster each data point belongs to.

**df\_converted['DBSCAN\_Cluster'] = dbscan.fit\_predict(df\_converted)**

# Group the data and calculate the mean of each group.

**dbscan\_cluster= df\_converted.groupby('DBSCAN\_Cluster').mean()**

# Create a copy of the DataFrame to preserve the original DataFrame

**dbscan\_copy = dbscan\_cluster.copy()**

# Drop the unnecessary columns from the copy

**dbscan\_copy = dbscan\_copy.drop(['Cluster', 'kmedoLabels', 'GMM\_Cluster'], axis=1)**

# Print the result

**dbscan\_copy**

A screenshot of a computer

Description automatically generated

Since the above table is too wide to fit in this chapter, let’s plot a heatmap for a better view.

# Set a figure size of the heatmap.

**plt.figure(figsize=(20, 3))**

# Create a heatmap of the dbscan\_copy DataFrame.

**sns.heatmap(dbscan\_copy, annot=True, cmap='coolwarm')**

# Set the title of the heatmap as 'DBSCAN Clustering Results'

**plt.title('DBSCAN Clustering Results')**

# Display the heatmap

A diagram of a number

Description automatically generated with medium confidence**plt.show()**

From the above results, we find -1 as a part of the DBSCAN cluster. The negative one, -1, represents the Outlier Points (Noise Points). Then we also find many points are assigned to these Outlier Points. Let’s see how many data points are assigned to each cluster by using the following codes:

# Calculate the size of each DBSCAN cluster

**cluster\_sizes = df\_converted['DBSCAN\_Cluster'].value\_counts()**

# Print the size of each cluster

**print(cluster\_sizes)**

A white background with black text

Description automatically generated

When we ran the Exploratory Data Analysis (EDA), we found that there are 2500 observations. Out of these 2500 observations, 2458 are assigned to the outlier points. This result implies that we did not assign the optimal value to ε (eps). Since DBSCAN is sensitive to the hyperparameters, it is critical to have optimal values to improve our results. One commonly used data-driven method to help us find the optimal ε (eps) value is using the ‘K-nearest neighbors (k-NN). The basic idea of using k-NN in the DBSCAN model is that the algorithm calculates the average distance between each point and its k-nearest neighbors, where k is defined as *min\_samples*. These averages are then plotted, and the point of maximum curvature is usually selected as the optimal ε (eps) value. We will observe this process through our hands-on exercise.

# Import the NearestNeighbors class from the sklearn.neighbors module

**from sklearn.neighbors import NearestNeighbors**

# Define the minimum number of samples required to form a dense region

**min\_samples=5**

# Apply the NearestNeighbors class with n\_neighbors set to min\_samples

**nearest\_neighbors = NearestNeighbors(n\_neighbors=min\_samples)**

# Fit the model to the data

**nearest\_neighbors.fit(df\_converted)**

# Compute the k-nearest neighbors for each point.

**distances, indices = nearest\_neighbors.kneighbors(df\_converted)**

# Sort the distances in ascending order for each point

**distances = np.sort(distances, axis=0)**

# Select the distance to the second nearest neighbor (index 1) for each point

**distances = distances[:,1]**

# Set a figure size

**plt.figure(figsize=(10, 5))**

# Plot the distances

**plt.plot(distances)**

# Set the title of the plot to 'k-NN Graph'

**plt.title('k-NN Graph)**

# Set the label of the x-axis to 'Data Points sorted by distance'

**plt.xlabel('Data Points sorted by distance')**

# Set the label of the y-axis to 'Epsilon'

**plt.ylabel('Epsilon')**

# Display the plot

**plt.show()**

A graph with a line drawn on it

Description automatically generated with medium confidence

From the above results, we could observe the Epsilon value of 0.8 is the point where the distance between data points begins to increase sharply. Hence, let’s replace the ε(eps) value from 0.5 to 0.8, and observe its result.

# Change the eps value

**dbscan = DBSCAN(eps=0.8, min\_samples=5)**

# Fit and predict clusters

**df\_converted['DBSCAN\_Cluster'] = dbscan.fit\_predict(df\_converted)**

# Analyze the updated clusters

**dbscan\_afterkNN= df\_converted.groupby('DBSCAN\_Cluster').mean()**

# Create a copy of the updated DataFrame, drop the unnecessary columns from the copy, and print the results

**dbscan\_afterkNN = dbscan\_afterkNN.copy()**

**dbscan\_afterkNN = dbscan\_afterkNN.drop(['Cluster', 'kmedoLabels', 'GMM\_Cluster'], axis=1)**

A screenshot of a computer

Description automatically generated**dbscan\_afterkNN**

Below section is just to give you some ideas on how we observe results from different point of view to compare results of modifying the parameters, ε=0.5 and ε=0.8. Please do not hesitate to write your own code to plot results visually by using other methods.

# Set figure size to the point that can be printable in one page and print the heatmap

**plt.figure(figsize=(25, 15))**

**sns.heatmap(dbscan\_afterkNN, annot=True, cmap='coolwarm')**

**plt.title('DBSCAN Clustering Results (e=0.8)')**

**plt.show()**

A screenshot of a graph

Description automatically generated

# Calculate the size of each DBSCAN cluster and print its result.

**cluster\_sizes = df\_converted['DBSCAN\_Cluster'].value\_counts()**

**print(cluster\_sizes)**

A computer screen shot of a computer

Description automatically generated

Even though the model seems to have slightly improved, we still observe that 1836 out of 2500 data points are assigned to Outlier points. This indicates that our DBSCAN model may not be clustering our data effectively. Instead of giving our insights into the results, let’s discuss how we measure the model’s accuracy.

One common metric used to evaluate the quality of a clustering model is the ‘Silhouette Score’. This score measures the similarity within clusters and the dissimilarity between clusters, providing a measure of the ‘tightness’ of the clusters. The score ranges from -1 to 1. Here are some interpretations of this range:

* A score close to 1: indicates that the data points are well clustered and far away from neighboring clusters.
* A score close to 0: suggests that the data points are on or very close to the decision boundary between two neighboring clusters.
* A score close to -1: implies that the data points are poorly clustered and could potentially belong to a different cluster.

In our case, we ran the model with two different parameters: ε=0.5 as the initial value and ε=0.8 as an improvement. Let’s calculate the Silhouette Score for both and discuss our observations.

# Create dictionary and define DBSCAN models with different parameters

**dbscan\_models = {**

**'DBSCAN\_eps\_0.5': DBSCAN(eps=0.5, min\_samples=5),**

**'DBSCAN\_eps\_0.8': DBSCAN(eps=0.8, min\_samples=5)}**

# Set an empty dictionary to store the results

**dbscan\_results = {}**

# Fit models and calculate metrics by looping over each model in the dictionary.

**for name, model in dbscan\_models.items():**

**model.fit(df\_converted)**

**labels = model.labels\_**

**dbscan\_results[name] = metrics.silhouette\_score(df\_converted,**

**labels)**

# Print results

**for name, score in dbscan\_results.items():**

**print(f"\n{name} Silhouette Score: {score}")**

A close-up of words

Description automatically generated

When we first ran DBSCAN, the Silhouette Score returned a negative value that was very close to -1. After adjusting the hyperparameters, the score became positive, but it was still around 0, indicating that the clusters are not well separated and are close to the decision boundaries. Given these results, we suggest trying different clustering algorithms to potentially improve the quality of the clustering.

Indeed, let’s run the Silhouette Score for all of our clustering methods we have observed to compare their qualities. This not only gives us a way to visualize the models’ qualities but also could be helpful for you to explain your model selection in a data-driven approach.

# Create a dictionaly to define the five models

**models = {**

**'KMeans': KMeans(n\_clusters=4, random\_state=0),**

**'KMedoids': KMedoids(n\_clusters=4, metric='euclidean',**

**method='pam', random\_state=0),**

**'Hierarchical': AgglomerativeClusteringn\_clusters=4,**

**linkage='ward'),**

**'GaussianMixture': GaussianMixture(n\_components=4,**

**random\_state=0),**

**'DBSCAN\_eps\_0.8': DBSCAN(eps=0.8, min\_samples=5) }**

# Create an empty dictionary to store the results

**results = {}**

# Fit models and calculate metrics

**for name, model in models.items():**

**model.fit(df\_converted)**

**labels = model.predict(df\_converted) if name ==**

**'GaussianMixture' else model.labels\_**

**results[name] = metrics.silhouette\_score(df\_converted, labels)**

# Print results

**for name, score in results.items():**

**print(f"\n{name} Silhouette Score: {score}")**

A number and numbers on a white background

Description automatically generated

If we rank our score results from high to low (from the score that is closer to positive one to negative one), we can observe that KMedoids gives us the best result followed closely by KMeans and Hierarchical clustering. GaussianMixture and DBSCAN did not perform as well according to the silhouette scores.

1. KMedoids: 0.7780589864718633 (BEST)
2. KMeans: 0.7698724498044935
3. Hierarchical : 0.7677584235322338
4. DBSCAN : 0.1020286324166116
5. GaussianMixture: -0.2248229255697769

The Silhouette score is just one example of evaluation methods. Yet, it enables us to take a more data-driven approach to justify our model selection.

* 1. **Chapter Summary**

In this chapter, we have learned about Unsupervised Machine Learning techniques and practiced interpreting results to make suggestions or recommendations to business stakeholders using the ‘Netflix Userbase Dataset’. We explored various unsupervised learning algorithms, including K-Means Clustering, PAM (K-Medoids) Clustering, Hierarchical Clustering, Gaussian Mixture Model Clustering (EM algorithm), and Density-Based Spatial Clustering of Applications with Noise (DBSCAN).

We calculated the Silhouette Scores for each model to evaluate the quality of the clustering performance. Please note that scoring models and improving their evaluation scores is just one aspect of the process. In the real business world, a company has unique characteristics that give it a competitive advantage. Because of this uniqueness, business decision-makers or owners might have concerns or goals that they want to improve but are not yet clear, such as ‘I know we should improve xyz, but I do not know what and how’.

Let’s conclude this chapter with some advice when you consult your clients: listen to your clients carefully, understand their business uniqueness, needs, and goals. Then start building your analysis or models to work closely together to become their ‘Business Partner’. Enjoy continuing your learning journey through this book!