Data Analytics MSc Dissertation MTHM038, 2021/22

Comparison of Clustering Methods and Algorithms

Kanchan Kumari

Student ID 210937251

Supervisor: Dr. Matthew Lewis

A thesis presented for the degree of
Master of Science in *Data Analytics*

School of Mathematical Sciences
Queen Mary University of London

Declaration of original work

This declaration is made on September 5, 2022.

Student's Declaration: I, Kanchan Kumari, hereby declare that the work in this thesis is my original work. I have not copied from any other students' work, work of mine submitted elsewhere, or from any other sources except where due reference or acknowledgement is made explicitly in the text, nor has any part been written for me by another person.

Referenced text has been flagged by:

1. Using italic fonts, **and**
2. using quotation marks "...", **and**
3. explicitly mentioning the source in the text.

Acknowledgements

This work is dedicated to my beautiful family and especially my dear husband, Vishal. Without their constant support, I would not have been here.
I am also extremely grateful to my supervisor Dr Matthew Lewis for guiding me on each step of this project. I hope this has turned out to be more than what we expected.

Abstract

Clustering is one of the main techniques in Machine Learning and Data Analysis and is used to get insight into the underlying nature and structure of data. The purpose of clustering is to partition a set of data into clusters such that data points belonging to the same cluster, have similar characteristics, whereas data points from different clusters are dissimilar to each other.

For this project our goal is provide detailed explanation and comparison of clustering algorithms with the help of a user review data set from a popular online search engine. We will investigate how different clustering techniques perform on this particular data. We consider some of the best known clustering techniques such as K-means, Gaussian mixture models, Spectral clustering, Affinity propagation, BIRCH, and Hierarchical clustering. We evaluate the results of these algorithms using a measure called 'Silhouette score'.

This analysis has many practical uses in the travel industry, specifically for tailoring recommendations for each of its user based on their previous reviews.

Keywords: Clustering Algorithms, Cluster Evaluation, Recommender systems, Unsupervised Learning

Contents

Chapter 1

Introduction

This dissertation is aimed at comparing different clustering methods and algorithms. Clustering is one of the most widely used unsupervised machine learning technique which consists of representing a data set in form of vectors and partitioning the resulting set of these vectors into 'clusters'. Different clustering methods use different approaches to group the data from the data sets into clusters. The clustering algorithms extract patterns and inferences from the data points and then create different clusters from them based on features/characteristics of data.

There is a wide range of clustering algorithms, including distances-based, density-based, hierarchical-based, and distribution based. These clustering algorithms are used in various applications such as - statistical data analysis, social network analysis, market segmentation, image Segmentation, pattern recognization, anomaly detection, and recommendation systems.

The cluster formation depends upon factors like **shortest distance between data points, connectivity of graphs created using data points, and density of the data points.** Grouping into clusters is conducted by finding a **similarity measure** between the data points such as Euclidean distance, Manhattan distance etc. It is easier to find similarity measures for a data set with lesser number of features. Finding similarity measures becomes a complex process as the number of features increases.

1.1 Problem Statement

For this project, we have used ‘Google Review Data Set’ to compare the performance of different clustering algorithms. The data set is picked keeping in mind that it has enough number of samples and attributes to perform clustering. It is a medium-sized data set with very practical use cases. The data set is uniformly distributed, see figure 2.4. It does not contain any special shapes such as - blobs, circles, or half moons.

It is well established fact that, not all algorithms scale efficiently for a particular data set. In this project, we will discuss six clustering algorithms **(K-Means, Gaussian Mixture, Spectral Clustering, Affinity Propagation, BIRCH, Hierarchical Clustering)** in detail, which scale well for this particular data set, and in the end, we will talk about why other algorithms- such as DBSCAN or OPTICS do not perform as well for this data set.

We will discuss in detail how each of these algorithms work, and we will then use ‘Silhouette Score’ to assess the performance of these algorithms as well as visualize the results using diagrams. We will review what the benefits and shortcomings of each algorithm are.

Chapter 2

Understanding the Data

2.1 Data Set Introduction

The data set used for this project has been sourced from the Machine Learning Repository of University of California, Irvine (UC Irvine) - Travel Review Ratings Data Set: https://archive.ics.uci.edu/ml/datasets/Tarvel+Review+Ratings

The data set has user ratings extracted from Google reviews, on attractions from 24 categories across Europe. A Google review is a voluntary, unpaid, online review that customers write about places they've visited. Google reviews are tied directly to Google Search and Google Maps. Google reviews make it easy for people to leave ratings, add pictures, and tell others about their experiences. These ratings range from 1 to 5. In our data set, average user rating per category is calculated.

The data set has 5456 different data points and 25 different attributes.
Attribute 1 : Unique user id
Attribute 2 : Average ratings on churches
Attribute 3 : Average ratings on resorts
Attribute 4 : Average ratings on beaches
Attribute 5 : Average ratings on parks

Attribute 6 : Average ratings on theatres
Attribute 7 : Average ratings on museums
Attribute 8 : Average ratings on malls
Attribute 9 : Average ratings on zoo
Attribute 10 : Average ratings on restaurants
Attribute 11 : Average ratings on pubs/bars
Attribute 12 : Average ratings on local services
Attribute 13 : Average ratings on burger/pizza shops
Attribute 14 : Average ratings on hotels/other lodgings
Attribute 15 : Average ratings on juice bars
Attribute 16 : Average ratings on art galleries
Attribute 17 : Average ratings on dance clubs
Attribute 18 : Average ratings on swimming pools
Attribute 19 : Average ratings on gyms
Attribute 20 : Average ratings on bakeries
Attribute 21 : Average ratings on beauty & spas
Attribute 22 : Average ratings on cafes
Attribute 23 : Average ratings on view points
Attribute 24 : Average ratings on monuments
Attribute 25 : Average ratings on gardens

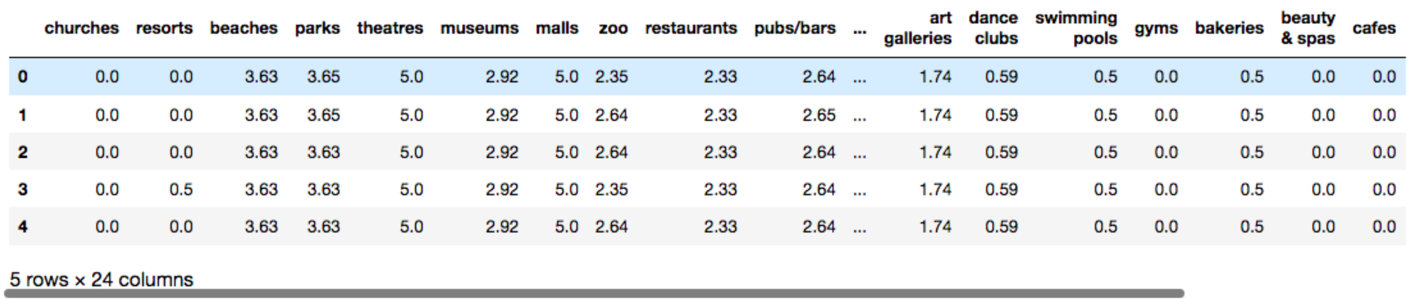
2.2 Cleaning the Data Set and Exploratory Data Analysis

The following steps were taken to clean the data set:

- Dropped the first column which contained values such as- user 1, user 2 etc. We do not need it, as we have Data Frame index to identify different users.
- All columns should be of float data type as they contain a value between 0 to 5. Changed the data type of column 'Local service' from object to float.

- Dropped 2 rows with null values using Python function *dropna()*.

After perfoming the steps mentioned above, the data set looks as following:



	churches	resorts	beaches	parks	theatres	museums	malls	zoo	restaurants	pubs/bars	...	art galleries	dance clubs	swimming pools	gyms	bakeries	beauty & spas	cafes
0	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.35	2.33	2.64	...	1.74	0.59	0.5	0.0	0.5	0.0	0.0
1	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.64	2.33	2.65	...	1.74	0.59	0.5	0.0	0.5	0.0	0.0
2	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	2.33	2.64	...	1.74	0.59	0.5	0.0	0.5	0.0	0.0
3	0.0	0.5	3.63	3.63	5.0	2.92	5.0	2.35	2.33	2.64	...	1.74	0.59	0.5	0.0	0.5	0.0	0.0
4	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	2.33	2.64	...	1.74	0.59	0.5	0.0	0.5	0.0	0.0

5 rows × 24 columns

Figure 2.1: A snippet of Data Set

The rating distribution for all 24 categories can be seen in figure 2.2.

- Overall ratings lie between 0 and 5.
- Categories 'Malls' and 'Restaurants' have wide spread rating distribution and high center of distribution.
- Categories 'Parks' and 'Theaters' have wide spread rating distribution but average center of distribution.
- Category 'Art Galleries' has wide spread rating distribution and quite low center of distribution, no other category shows such distribution.
- Categories 'Swimming pools', 'Gyms', 'Bakeries', 'Beauty & Spa' have narrow rating distribution and very low center of distribution.
- We do not have any target variable in the data set. All the clustering algorithms used will be 'Unsupervised Learning Algorithms'.

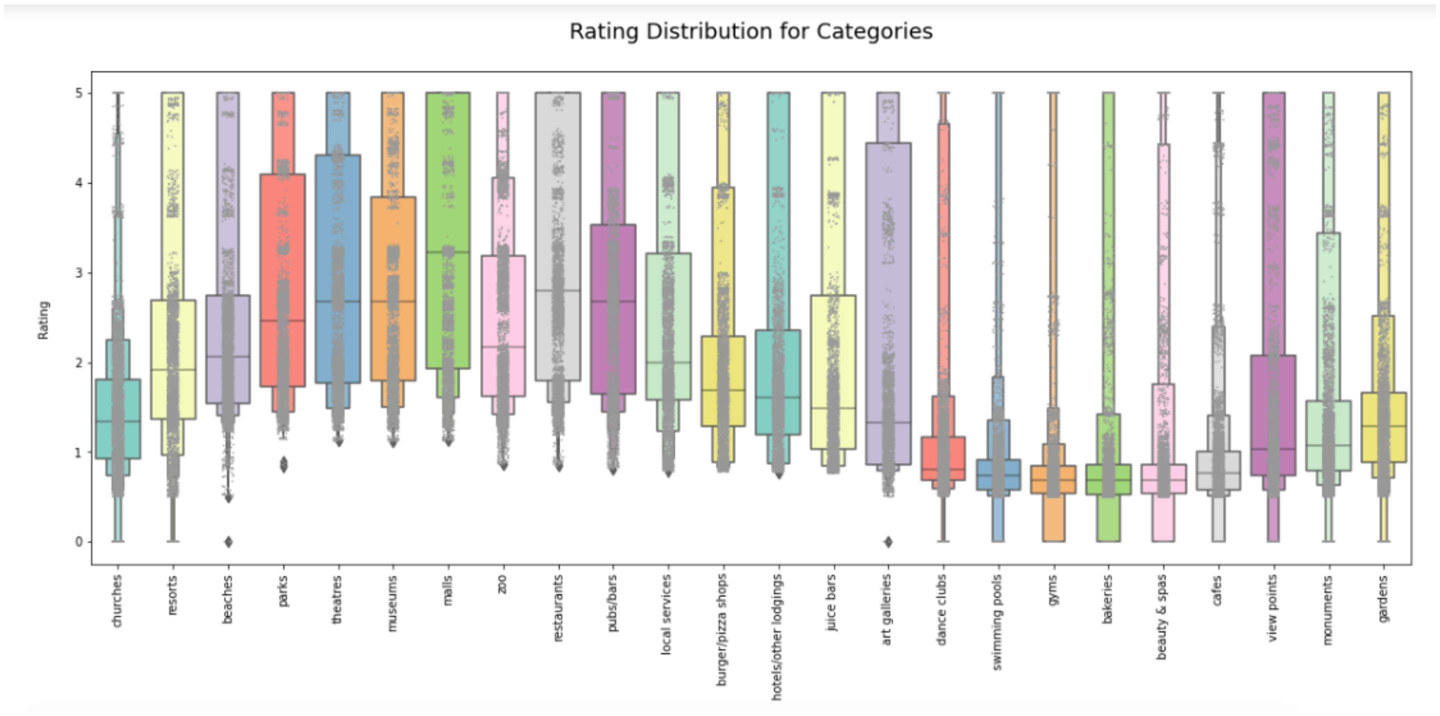


Figure 2.2: Data Set Rating Distribution for 24 Categories

The Heatmap which shows correlation among different categories of data set can be seen in Figure 2.3

2.3 Preparing Data for Clustering

We have a clean data set after performing the above steps, but the data set has 24 attributes which makes it prone to the "Curse of Dimensionality". Running a dimensionality reduction algorithm such as **Principal component analysis (PCA)** prior to clustering can alleviate this problem and speed up the computations.

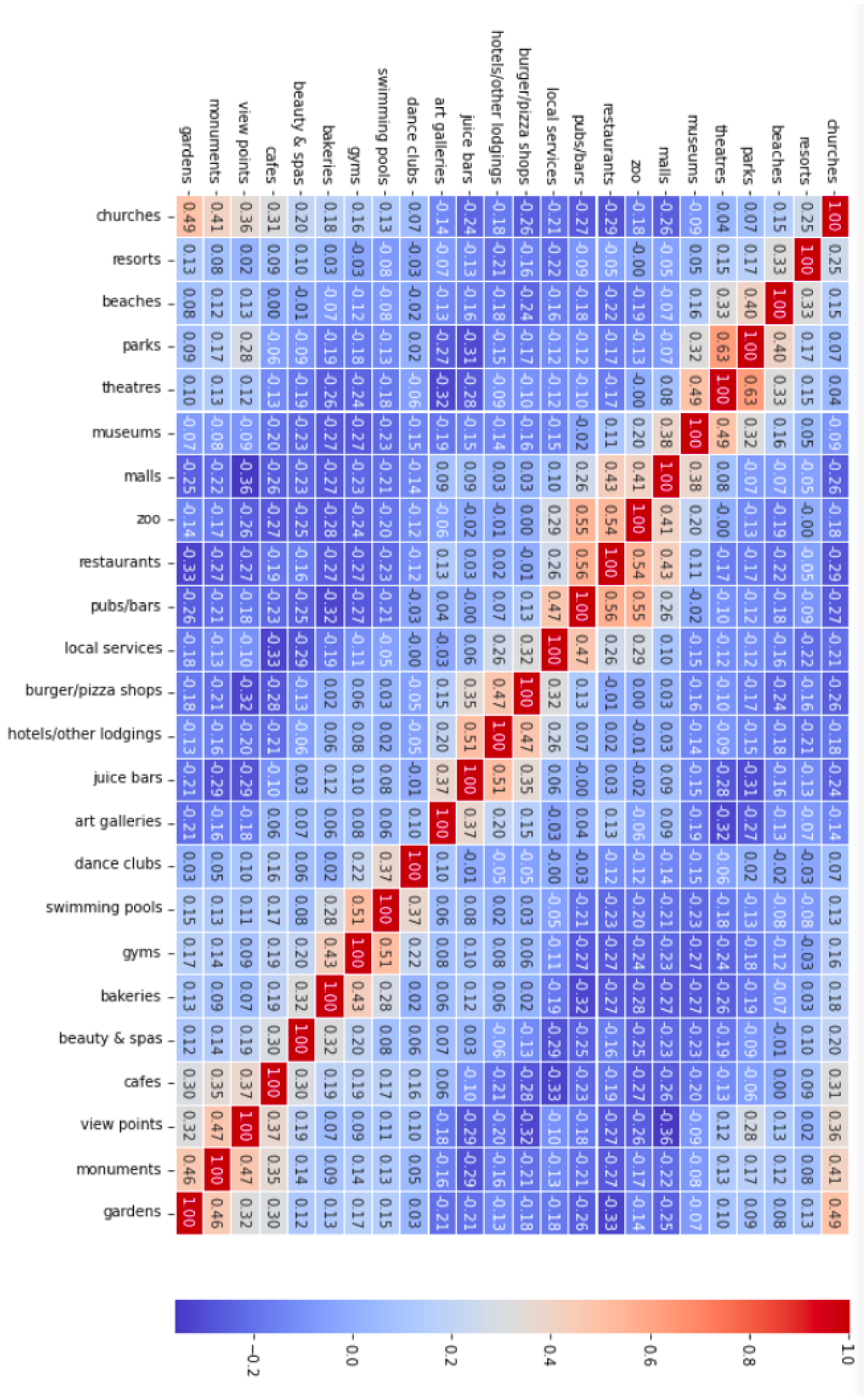


Figure 2.3: HeatMap Showing Correlation Among Different Categories

2.3.1 Principal Component Analysis

PCA is a popular machine learning technique for reducing the dimensionality of data while retaining patterns and trends and minimizing information loss. It allows us to summarize the information contained in a large data set by finding the most significant feature in the data set while maximizing variance. It uses eigenvalues and eigenvectors to do so because eigenvalues represent the total amount of variance that can be explained by a principal component(eigenvector). To transform a data set into n components, PCA uses largest n eigenvalues and corresponding eigenvectors. These n eigenvectors can be considered as new uncorrelated n components that maximize variance.

- We use the Python function *sklearn.decomposition.PCA()*, which has parameter n_components (number of components) to transform our data set with 24 variables. n_components controls how many new variables will be created.

- Reducing the complexity of data sets using PCA can also help us visualize the data better. We use PCA, and reduce the dimensions of data set to two, to visualize our Google review data set. See figure 2.4.

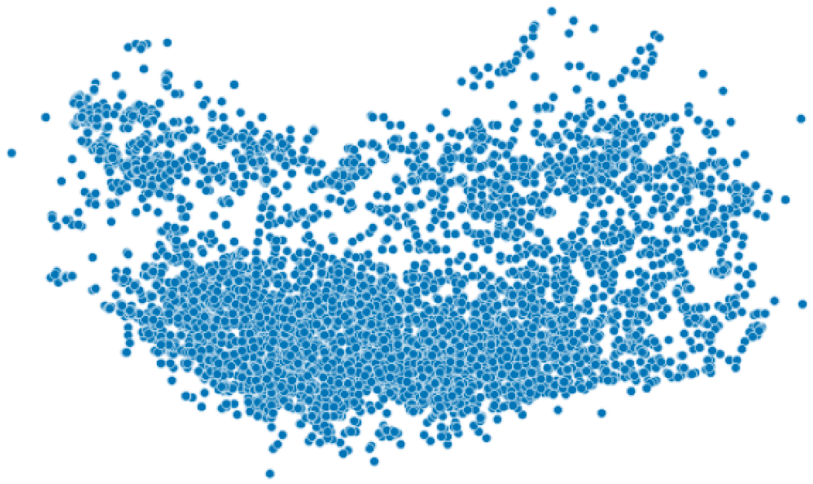


Figure 2.4: Distribution of Data Points in 2D Space

- We also need to make sure that our data is scaled, and for that we use Python function, *StandardScaler().fit_transform()*
- We create 4 data sets using the following Python code:

```python
#Original data
df_original = df

#Scaled Data
df_scaled = StandardScaler().fit_transform(df)

#PCA without scaling
pca = PCA(n_components = 5,random_state=42)
df_PCA = pca.fit_transform(df_original)

#PCA with scaling
df_PCA_scaled = pca.fit_transform(df_scaled)
```

Chapter 3

K-Means Algorithm

The K-means Algorithm, also known as Lloyd's Algorithm, is one of the most widely used clustering algorithms due to its ease of implementation. The K-means algorithm is an unsupervised learning algorithm that divides data set into K clusters through an iterative process while minimizing a criterion known as 'inertia' or 'within-cluster-sum-of-squares'. It is a distance-based algorithm and scales well for a large number of samples.

3.1 Algorithm Outline

1. First, we need to decide on how many clusters we want to create. To do this, we use the elbow method. In the elbow method, we try different values of K (1,2,....n) and plot it against inertia which is the sum of squared distances of samples to their closest cluster center. We chose K, where we see the curve flattening after a sudden drop, see figure 3.1.

   We can see in figure 3.1 that the curve flattens around 5 or 6 clusters. We will pick K as 5 according to the elbow method graph. We can also see that inertia is minimum for *df_PCA_scaled* data set, and out of the original data sets, it is minimum for *df_scaled* data set. We will focus our analysis on these two data sets going forward.

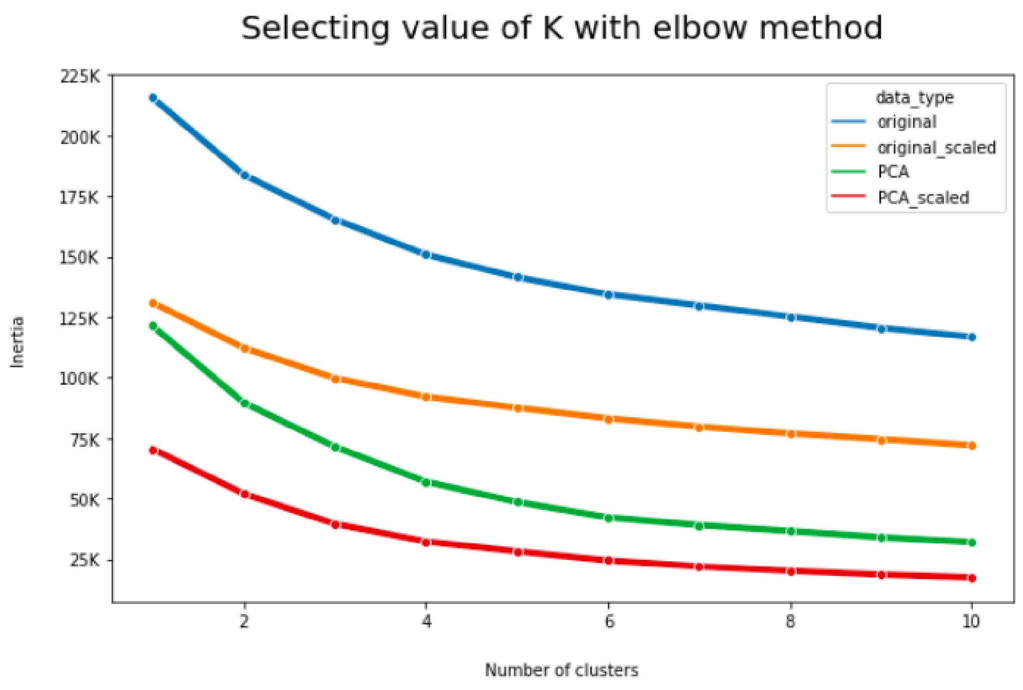


Figure 3.1: Elbow Method

2. The next step is to choose initial centroids for clusters. K-means chooses K random samples from the data set and initializes the process. The final results of K-means highly depend on the initialization of the centroids.

3. After initializing the centroids, K-means consists of looping between two steps.

   - Step 1, assigns each data point to its nearest centroid using some measurement such as 'euclidean distance' or 'cosine distance'.
   - Step 2, creates new centroids by taking the mean value of all the data points assigned to each previous centroid.

   These two steps are repeated until the centroids do not move significantly and the algorithm converges.

4. To evaluate the performance of K-means we use 'silhouette score'.

3.1.1 Silhouette Score

The Silhouette score is used to measure the goodness of fit of a clustering algorithm. It is a measure of how similar a data point is to its own cluster (cohesion) compared to other clusters (separation). It is especially useful in the case of unsupervised learning, where we do not have a target variable to compare our results with.
To get the Silhouette score for a data set, we calculate the Silhouette score of each data sample and then calculate the mean Silhouette score of all the samples.

The value of Silhouette score varies from -1 to 1. If the score is 1, the clusters are dense and well-separated from each other. A value near 0 represents overlapping clusters with data samples very close to the decision boundary of the neighboring clusters. A negative score, between -1 and 0, indicates that the samples might have got assigned to the wrong clusters.
To calculate the Silhouette score for a sample, we first calculate two things:

1. Mean distance between the sample and all other data points in the same cluster. This distance is called a 'mean intra-cluster distance' and is denoted by a.

2. Mean distance between the sample and all other data points of the next nearest cluster. This distance is called a 'mean nearest-cluster distance' and is denoted by b.

Once we have a and b, we calculate the Silhouette score(S) using the formula -

$$S = \frac{(b - a)}{max(a, b)}$$

3.2 K-means Application on Data Set

We implement the K-means algorithm using the Python library Scikit-learn.

```python
#df_scaled data set
from sklearn.cluster import KMeans
model = KMeans(n_clusters=5,random_state=30)
model.fit(df_scaled)
cluster = model.labels_
label = model.fit_predict(df_scaled)
score = silhouette_score(df_scaled, label, metric='euclidean'
    )
print('K-means - Silhouette Score:  %.3f' % score)
#Silhouette Score is 0.151 for df_scaled data.

#df_PCA_scaled data set
model = KMeans(n_clusters=5,random_state=30)
model.fit(df_PCA_scaled)
cluster = model.labels_
label = model.fit_predict(df_PCA_scaled)
score = silhouette_score(df_PCA_scaled, label, metric='
    euclidean')
print('Silhouette Score: %.3f' % score)
#Silhouette Score is 0.309 for df_PCA_scaled data set
```

Clusters created from data Set df_scaled using K-means can be seen in figure 3.2.

Cluster 0 : likes to visit art galleries, dance clubs, swimming pools, and gyms.
Cluster 1 : likes to visit malls, zoo, restaurants, pubs/bars, and local services.
Cluster 2 : likes to visit parks, theatres, and museums.
Cluster 3 : Does not like anything too much, but prefers churches, resorts, beaches, viewpoint, monuments, and gardens over other.
Cluster 4 : likes to visit burger/pizza shop, hotels, juice bars, and art galleries.

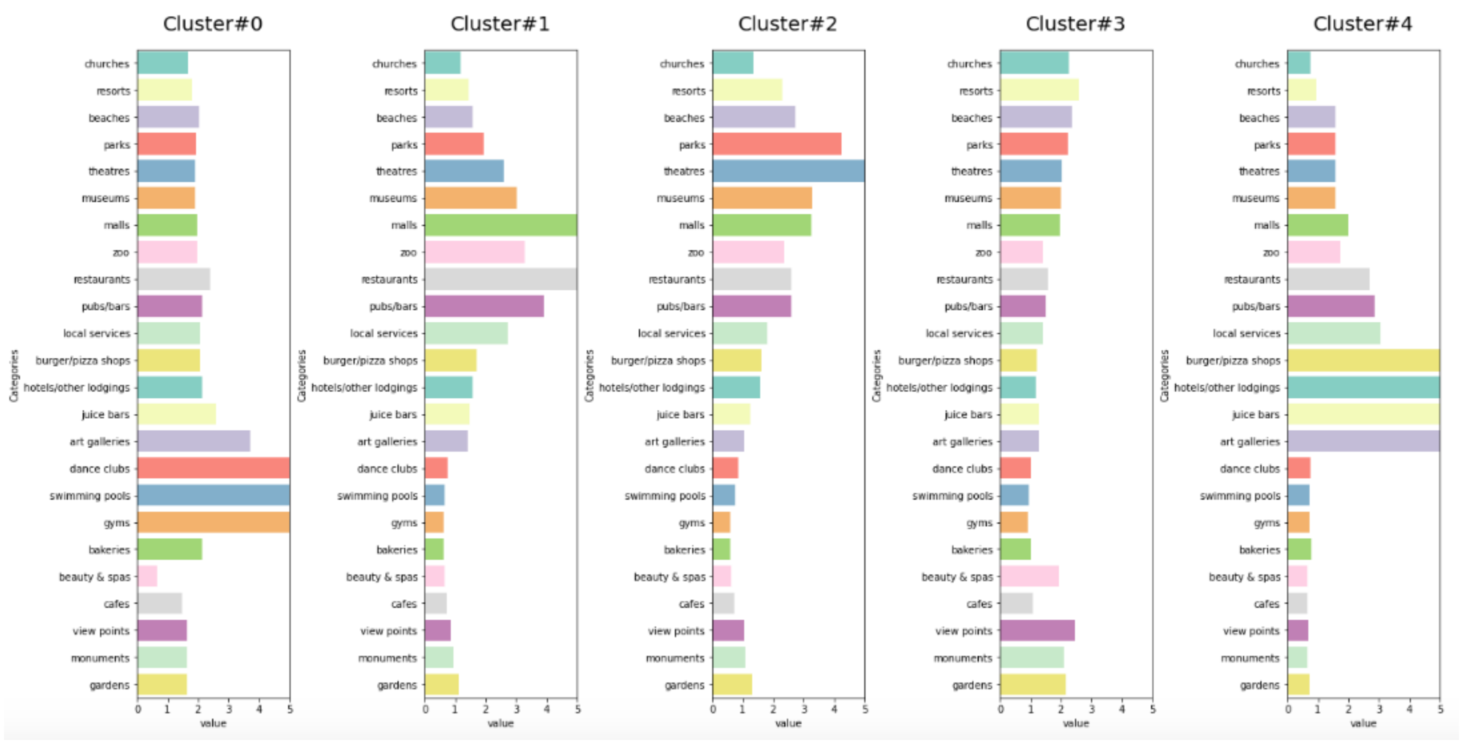


Figure 3.2: Clusters Created from df_scaled Data Set

Clusters created from data Set df_PCA_scaled using K-means can be seen in figure 3.3.

Cluster 0 : likes to visit art galleries, dance clubs, swimming pools, and gyms.
Cluster 1 :likes to visit parks, theatres, and museums.
Cluster 2 : likes to visit burger/pizza shop, hotels, juice bars, and art galleries.
Cluster 3 : Does not like anything too much- but prefers churches, resorts, beaches, viewpoint, monuments, and gardens over others.
Cluster 4 : likes to visit malls, zoo, restaurants, pubs/bars, and local services.

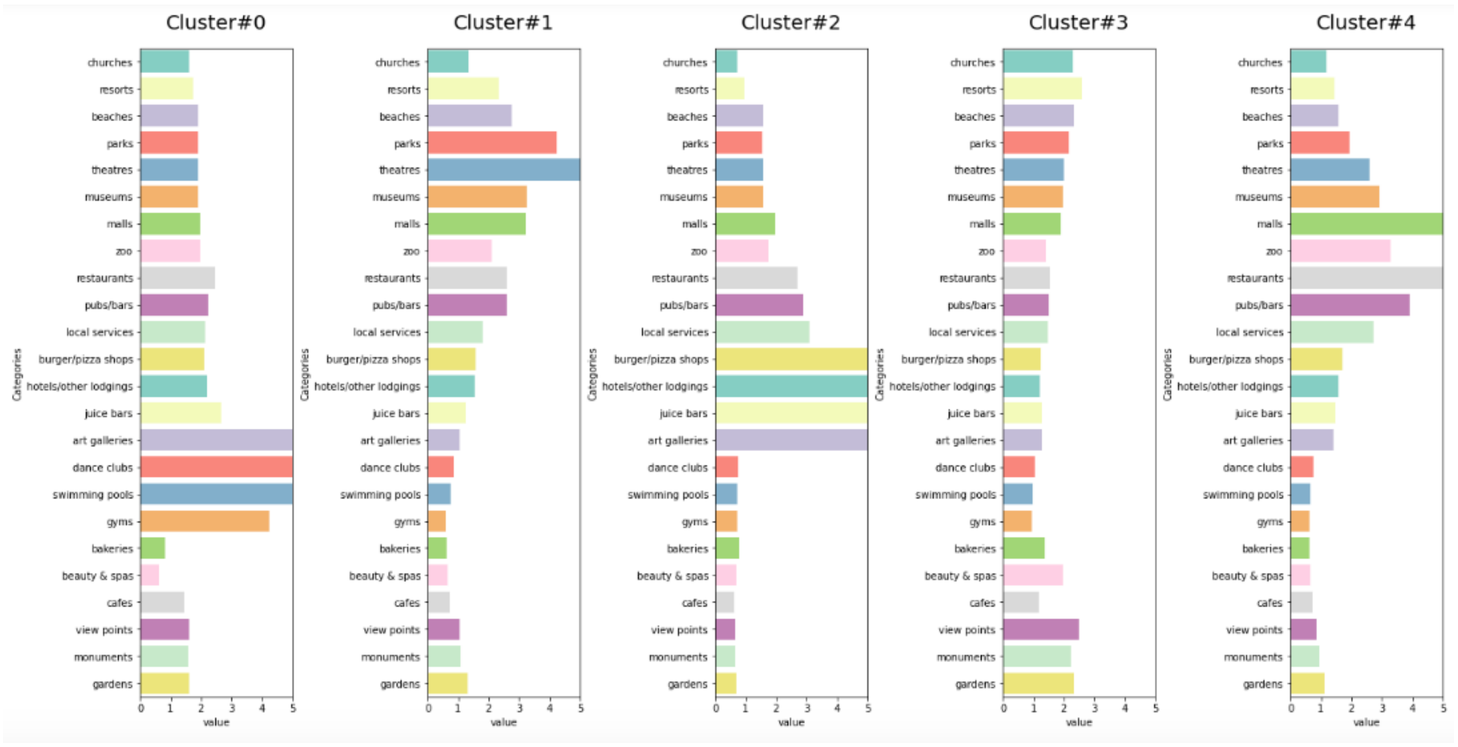


Figure 3.3: Clusters Created from df_PCA_scaled Data Set

As we can conclude from the above two figures, the clusters created from the original scaled data and the PCA scaled (with five components) data are consistent. Only the order of clusters has changed. We will name these clusters, and going forward, we will compare them with the results we get from other clustering algorithms. Both data sets are resulting in the same clusters, but PCA data is computationally efficient and has a better Silhouette score of 0.309, compared to silhouette score of original scaled data, 0.151. Hence, from now on wards, we will only use the PCA scaled data set to create clusters and evaluate results.

3.3 Naming the Clusters

Cluster A : likes to visit art galleries, dance clubs, swimming pools, and gyms. Similar to ‘Cluster 0’ from df_scaled data set(3.2) and ‘Cluster 0’ from df_PCA_scaled data set(3.3)

Cluster B : likes to visit parks, theatres, and museums.
Similar to 'Cluster 2' from df_scaled data set(3.2) and 'Cluster 1' from df_PCA_scaled data set(3.3)

Cluster C : likes to visit burger/pizza shop, hotels, juice bars, and art galleries.
Similar to 'Cluster 4' from df_scaled data set(3.2) and 'Cluster 2' from df_PCA_scaled data set(3.3)

Cluster D : likes to visit malls, zoo, restaurants, pubs/bars, and local services.
Similar to 'Cluster 1' from df_scaled data set(3.2) and 'Cluster 4' from df_PCA_scaled data set(3.3)

Cluster E : Does not like anything too much but prefers churches, resorts, beaches, viewpoints, monuments, gardens over others
Similar to 'Cluster 3' from df_scaled data set(3.2) and 'Cluster 3' from df_PCA_scaled data set(3.3)

We can see in figure 3.4 how data set looks like after being divided in 5 clusters.

3.4 K-means ++ Algorithm

As mentioned above, the K-means is highly dependent on the initialization of the centroids, and while K-means always converges, it might converge to local minima. As a result, the computation is often done several times, with different initializations of the centroids.

To address this issue, we use K-means++ initialization, which initializes the cen-

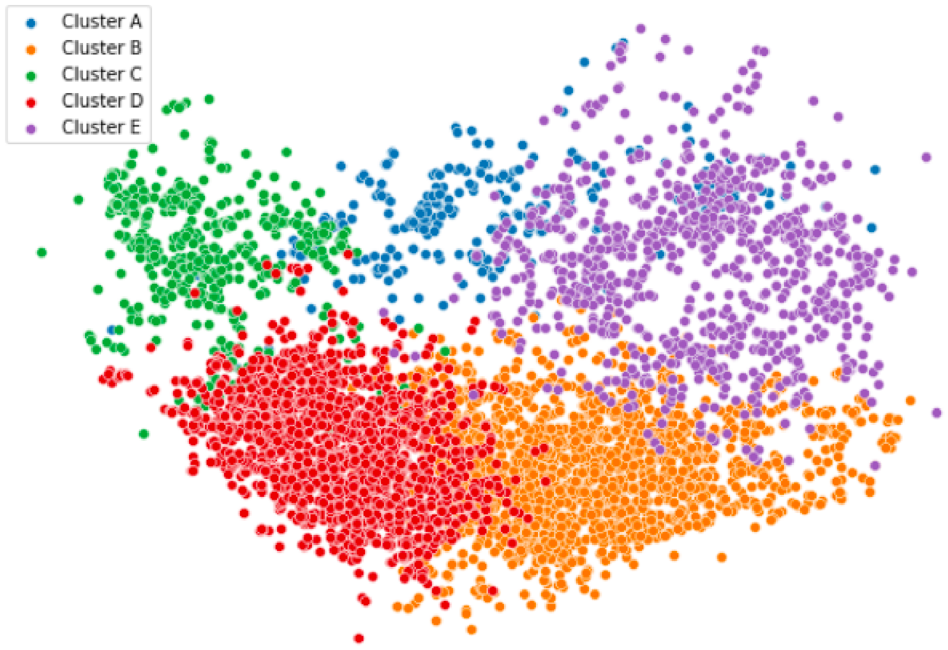


Figure 3.4: Data Set Divided in Five Clusters

troids distant from each other. This improves both the speed and the accuracy of K-means. The rest of the steps remain the same.

Using the Python package to implement K-means++ will lead to the same results as K-means because, in the Python package, computation is done several times to make sure K-means converges to global maxima. Hence, instead of using existing Python packages, we will implement K-means++ from scratch here.

```python
#Euclidean distance between centroid and all points in data set
def euclidean_distance(centroid, df):
    return np.sqrt(np.sum((centroid - df)**2, axis=1))
```

```python
class KMPlus:
    def __init__(self, k , max_iter):
```

```python
    #k represents no. of clusters
        self.k = k
        self.max_iter = max_iter


    def initialize(self, df):
        #choosing first centroid at random.

        self.centroids = [random.choice(df)]
        for i in range(self.k-1):
            # Calculate distances from points to the centroids
            distances = np.sum([euclidean_distance(centroid, df) for centroid in self.centroids], axis=0)
            distances /= np.sum(distances)
            # Picking the remaining centroids farthest from the first centroid
            new_centroid, = np.random.choice(range(len(df)), size = 1, p=distances)
            self.centroids += [df[new_centroid]]

        itr = 0
        prev_centroids = None
        while np.not_equal(self.centroids, prev_centroids).any() and itr < self.max_iter:
            # assigning each data point to nearest centroid
            #Creating a list of lists
            clustered_points = [[] for i in range(self.k)]

            for sample in df:
                distances = euclidean_distance(sample, self.centroids)
                centroid_index = np.argmin(distances)
                clustered_points[centroid_index].append(sample)

            prev_centroids = self.centroids
            self.centroids = [np.mean(_ , axis=0) for _ in clustered_points]
            for i, centroid in enumerate(self.centroids):
            # Catch any np.nans, resulting from a centroid having no points
                if np.isnan(centroid).any():
                    self.centroids[i] = prev_centroids[i]
            itr += 1

    def labels(self, df):
        centroids = []
        labels = []
        for sample in df:
            distances = euclidean_distance(sample, self.centroids)
            centroid_index = np.argmin(distances)
            centroids.append(self.centroids[centroid_index])
            labels.append(centroid_index)
        return centroids, labels

```

3.5 K-means++ Application on Data Set

Implementing the above K-means++ algorithm on PCA Scaled Data:

```
kmeans = KMPlus(5, 500)
kmeans.initialize(df_PCA_scaled)
class_centers, classification = kmeans.labels(df_PCA_scaled)
score = silhouette_score(df_PCA_scaled, classification, metric='euclidean')
print('Silhouette Score: %.3f' % score)
#Silhouette Score for K-means++ is 0.308 for df_PCA_scaled data set.
```

We get silhouette score of 0.308 using K-means++.

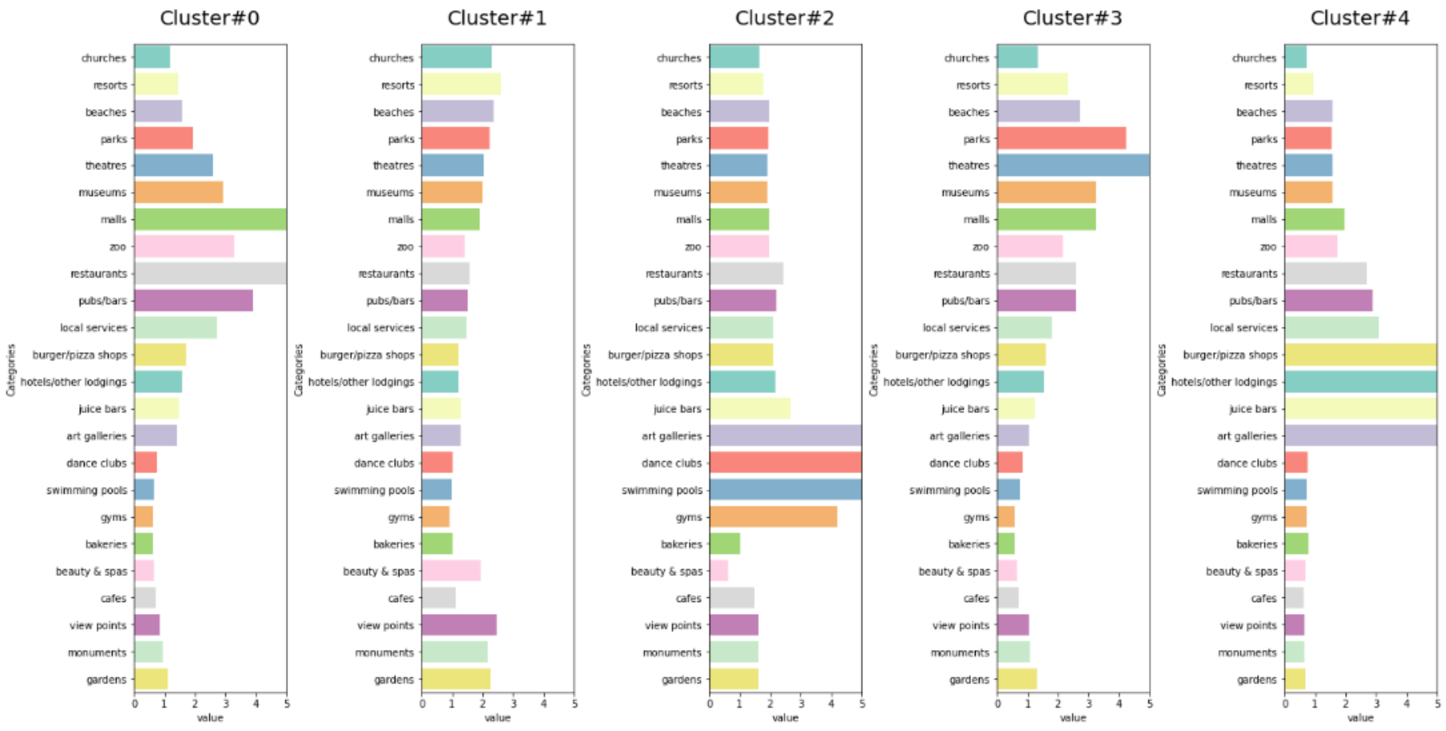


Figure 3.5: Clusters Created from K-means++ Algorithm

Cluster 0: Similar distribution as Cluster D

Cluster 1: Similar distribution as Cluster E

Cluster 2: Similar Distribution as Cluster A

Cluster 3: Similar Distribution as Cluster B

Cluster 4: Similar Distribution as Cluster C

Chapter 4

Gaussian Mixture Models Algorithm

Gaussian Mixture Models (GMMs) assume that there are a certain number of Gaussian distributions (with specific mean and variance), and each of these distributions represents a cluster. It uses 'Expectation-Minimization' Algorithm for fitting GMMs , which we will discuss in detail in later sections.
Unlike K-means, which gives us a hard classification of data points, GMM performs a soft classification i.e. for a given data set, the GMM algorithm will identify the probability of each data point belonging to each distribution.
K-means always provides us with circular clusters, with a centroid as the mean of all points and radius defined by the most distant point of the cluster from the centroid. It works fine in the case of clusters with circular boundaries, but when data has an oblong or ellipsoid shaped clusters, then K-means will perform poorly. In contrast, GMM is flexible in this case and can take any shape.

4.1 Algorithm Outline

1. Similar to K-means, we need to decide how many clusters we want to create for GMMs. We have number of clusters, K, is 5 that we decided using Elbow Method, see section 3.1.

Five Gaussian Clusters means that we have five different Gaussian Distributions with mean $(\mu_1, \mu_2, \mu_3, \mu_4, \mu_5)$ and covariance $(\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5)$. There is another parameter, density, which defines number of points in each distribution , represented by $(\pi_1, \pi_2, \pi_3, \pi_4, \pi_5)$. The sum of all the densities will always be one. We need to find values of these parameters to determine Gaussian Distributions.

2. To initialise, we randomly assign values to mean, covariance, and density parameters and then perform E-step and M-step of Expectation- Minimization Algorithm iteratively.

3. **E-Step :** For each point x_i, we calculate the probability that it belongs to cluster/distribution $c_1, c_2, \ldots c_5$. We calculate what is the likelihood that the i-th sample came from distribution k represented by $P(x_i = k|\mu_k, \sigma_k, \pi_k)$.

$$P(x_i = k|\mu_k, \sigma_k, \pi_k) = \frac{Probability\ x_i\ belongs\ to\ c_k}{Sum\ of\ Probability\ x_i\ belong\ to\ c_1, c_2, \ldots c_5}$$

   This value will be high when the point is assigned to the right cluster and low otherwise.

4. **M-Step :** In this step, we update the Gaussian parameters (μ, σ, π) to fit points assigned to them, using the likelihood value obtained in E-step. The new density is defined by the ratio of the number of points in the cluster and the total number of points.
   The mean and the covariance parameters are updated in proportion with the likelihood/probability values for the data point. A data point with a higher probability of being a part of that distribution will contribute a more significant portion.

   Based on the updated values generated from M-step, we calculate the new probabilities for each data point and update the values iteratively until the algorithm converges.

4.2 Application on Data Set

We Implement the GMMs algorithm using the Python library Scikit-learn.

```python
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components = 5,covariance_type = '
    spherical', max_iter = 300,random_state=30)
gm_labels = gmm.fit_predict(df_PCA_scaled)
gm_centers = gmm.means_
print ("gmm: silhouttte: ", silhouette_score(df_PCA_scaled,
    gm_labels))
# Silhouette Score for GMMS is 0.307 for df_PCA_scaled Data
    set
```

We get Silhouette Score of 0.307 using GMMs algorithm.

Visualizing Clusters for GMMs:

Cluster 0: Similar Distribution as Cluster E
Cluster 1: Similar Distribution as Cluster D
Cluster 2: Similar Distribution as Cluster A
Cluster 3: Similar Distribution as Cluster C
Cluster 4: Similar Distribution as Cluster B

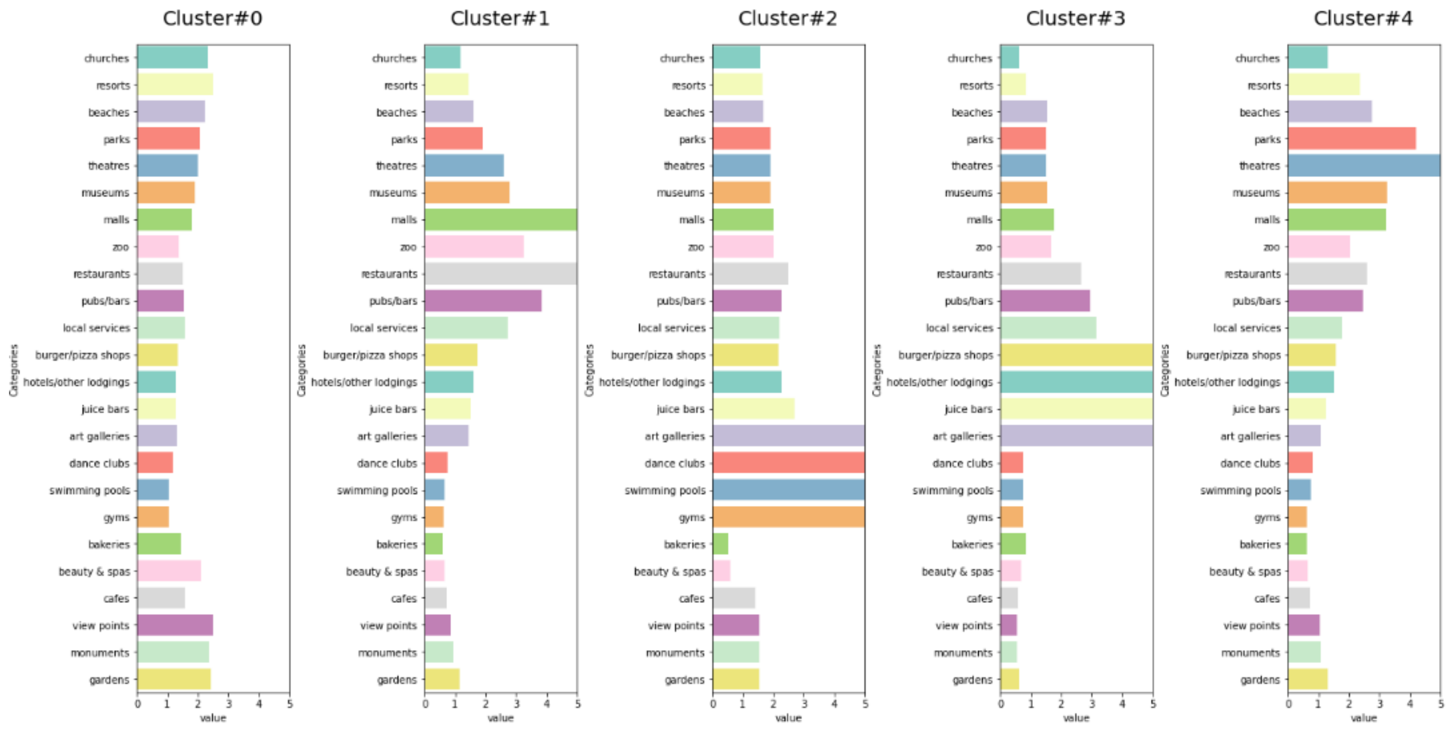


Figure 4.1: Clusters Created from GMMs Algorithm

Chapter 5

Spectral Clustering Algorithm

Spectral Clustering has its roots in graph theory. It divides data into clusters using eigenvectors of a matrix. This matrix is known as *Affinity Matrix* and is derived from pairwise similarities between data points. If pairs of points are very dissimilar, then the affinity should be 0. If the points are identical, then the affinity might be 1. The algorithm is flexible and allows us to cluster non-graph data as well.

One remarkable advantage of spectral clustering is its ability to cluster 'points' that are not necessarily vectors; for this, it uses a parameter 'similarity', which is less restrictive than a distance. The second advantage of spectral clustering is its flexibility; it can find clusters of arbitrary shapes under realistic separations.

5.1 Algorithm Outline

1. Similar to K-means and GMMs, for spectral clustering, we need to decide the number of clusters, K, which is five for our Google review data set.

2. The next step is to transform the data set to *Affinity Matrix* which is like an adjacency matrix except the value for each pair of points expresses how similar those points are. There are entire fields dedicated to how to create Affinity Matrix. In our case, the Python function *sklearn.cluster.SpectralClustering()*

has affinity parameter which can take four values.

Four ways of constructing the affinity matrix (according to Sci-Kit Learn official documentation) are following:

- 'nearest_neighbors': construct the affinity matrix by computing a graph of nearest neighbors.
- 'rbf': construct the affinity matrix using a radial basis function (RBF) kernel. (default value)
- 'precomputed': interpret X(input matrix) as a precomputed affinity matrix, where larger values indicate greater similarity between instances.
- 'precomputed_nearest_neighbors': interpret X(input matrix) as a sparse graph of precomputed distances, and construct a binary affinity matrix from the n_neighbors nearest neighbors of each instance.

3. Once we have the Affinity Matrix, Spectral Clustering uses its eigenvectors to decide clusters. We compute Largest K(=5) eigenvalues $(\lambda_1, \lambda_2, ..., \lambda_k)$ of affinity matrix and corresponding eigenvectors $(V_1, V_2, ..V_k)$.

4. Next, we embed the data in the K-th Principal subspace, creating a new data set of dimension NXK using the K eigenvectors $(V_1, V_2, ..V_k)$, where N is the number of data points in the original data set. $X = (V_1, V_2, ..V_k)$

5. Apply K-means or any other clustering algorithm on new data set X to create clusters.

5.2 Application on Data Set

We Implement the spectral clustering algorithm using the Python library Scikit-learn.

```python
from sklearn.cluster import SpectralClustering
model = SpectralClustering(n_clusters=5, eigen_solver=None,
    n_components=5,
    random_state=30,
    #n_init=10,
    #n_neighbors=10,
    eigen_tol=0.0,
    assign_labels='discretize')
spec_labels = model.fit_predict(df_PCA_scaled)

print ("SC: silhouttte: ", silhouette_score(df_PCA_scaled,
   spec_labels))
#Silhouette Score for Spectral Clustering is 0.299 for
   df_PCA_scaled Data set
```

We get Silhoutte Score of 0.299 using spectral clustering algorithm.

Visualizing Clusters for Spectral Clustering:

Cluster 0: Similar Distribution as Cluster D
Cluster 1: Similar Distribution as Cluster C
Cluster 2: Similar Distribution as Cluster E
Cluster 3: Similar Distribution as Cluster A
Cluster 4: Similar Distribution as Cluster B

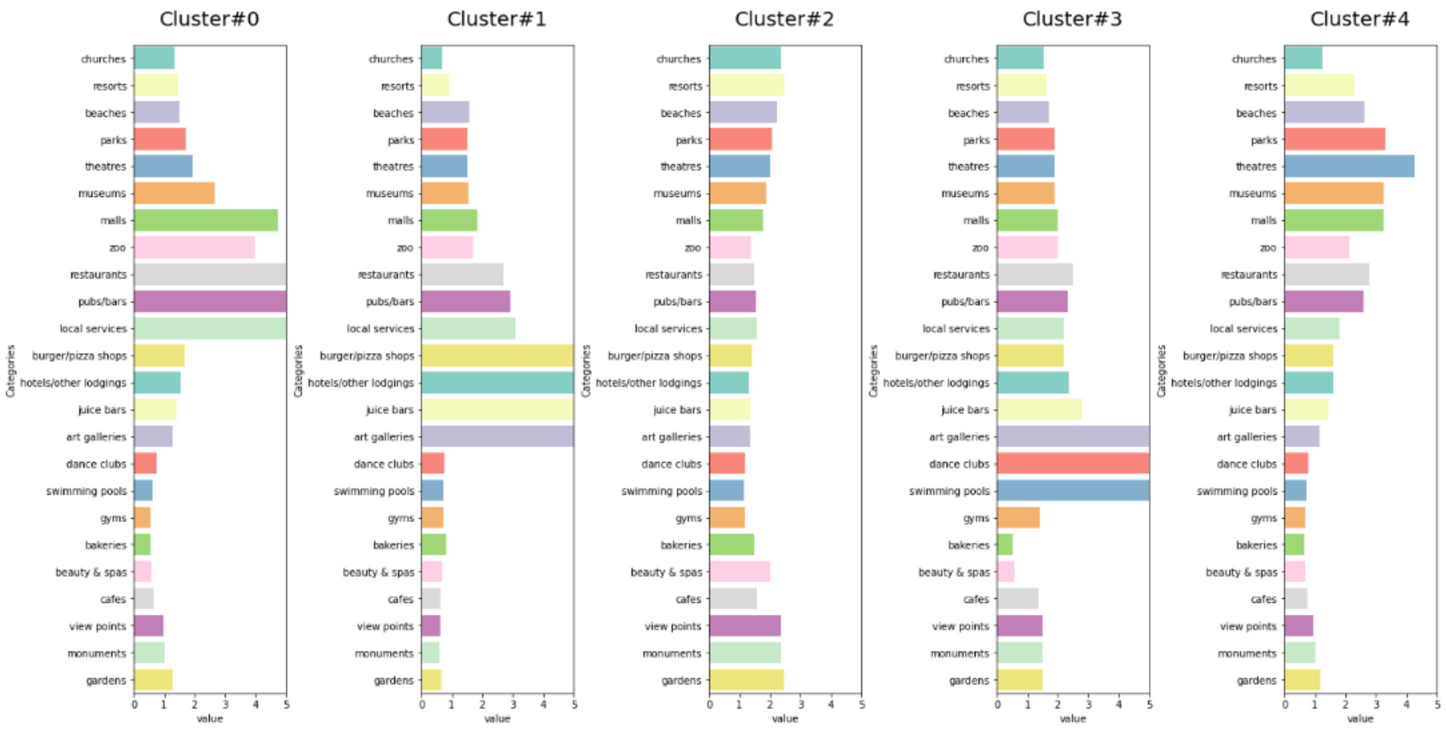


Figure 5.1: Clusters Created from Spectral Clustering Algorithm

Chapter 6

Affinity Propagation Algorithm

Affinity propagation (AP) is a clustering algorithm based on the concept of 'message passing' between data points. The subject of these messages is the willingness of the data points to be exemplars. *Exemplars* are points that explain the other data points 'best' and are the most significant of their cluster. The algorithm identifies exemplars among data points and forms clusters of data points around these exemplars.

In contrast to other clustering methods that we have seen so far, Affinity Propagation does not require us to specify the number of clusters(K). The main drawback of Affinity Propagation is its Time complexity of $O(N^2T)$ where N is number of samples in the data set and T is number of iterations until convergence. Hence it is most suitable for small or medium-sized data sets.

6.1 Algorithm Outline

In Layman's term, Affinity Propagation has the following steps-

1. Initialize: Each data point sends messages to all other points informing its targets (exemplars) of each target's relative attractiveness to the sender.

2. Availability Message: Each target (exemplar) then responds to all senders with a reply informing each sender of its availability to associate with the sender, given the attractiveness of the messages that it has received from all other senders.

3. Responsibility Message: Senders reply to the targets with messages informing each target of their revised relative attractiveness to the sender, given the availability of messages it has received from all targets.

The message-passing procedure (steps 2 and 3) proceeds until a consensus is reached. Once the sender is associated with one of its targets, that target becomes the point's exemplar. All points with the same exemplar are placed in the same cluster.

Affinity Propagation has two important parameters - **1. preference**, which controls how many exemplars i.e. clusters are created, and **2. damping factor**, which damps the responsibility and availability messages to avoid numerical oscillations when updating these messages.

6.2 Application on Data Set

We Implement the affinity propagation algorithm using the Python library Scikit-learn.

```python
from sklearn.cluster import AffinityPropagation
clustering = AffinityPropagation(damping = 0.9, preference=
    -3000, affinity = 'euclidean',random_state=30 ).fit(
    df_PCA_scaled)

APlabels = clustering.labels_
```

```
print ("Affinity Propagation: silhouette: ", silhouette_score
   (df_PCA_scaled, APlabels))
#Silhouette Score is 0.306 for affinity prop. algorithm.
```

We get Silhouette Score of 0.306 using affinity propagation algorithm.

Visualizing Clusters for AP:

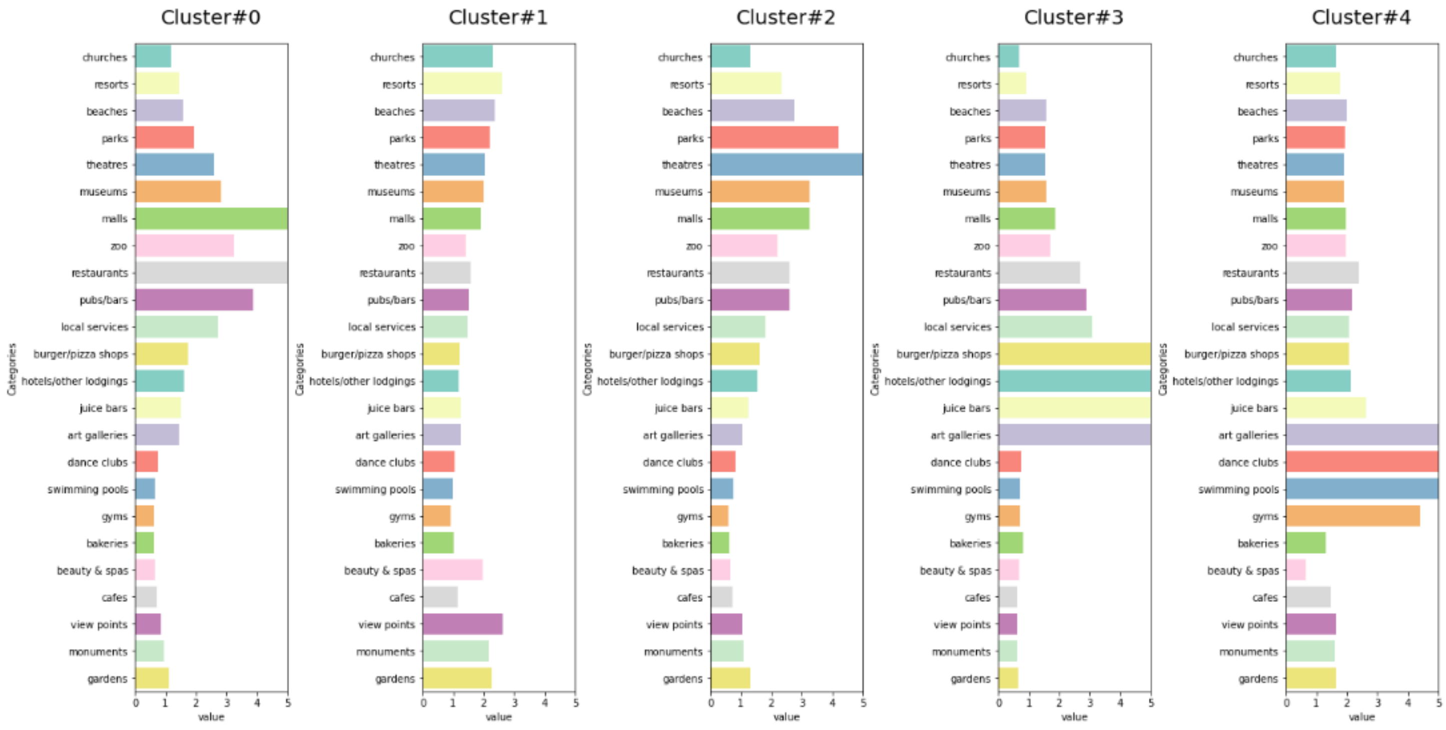


Figure 6.1: Clusters Created from Affinity Propagation Algorithm

Cluster 0: Similar Distribution as Cluster D

Cluster 1: Similar Distribution as Cluster E

Cluster 2: Similar Distribution as Cluster B

Cluster 3: Similar Distribution as Cluster C

Cluster 4: Similar Distribution as Cluster A

Chapter 7

BIRCH Algorithm

Balanced Iterative Reducing and Clustering using Hierarchies, or BIRCH for short, is an unsupervised data mining algorithm used to perform cluster analysis over particularly large data-sets. It generates a compact summary of data that retains as much information as possible and then creates clusters from the data summary instead of the original data set.

BIRCH is often used to complement other clustering algorithms by creating a summary of the data set that the other clustering algorithm can use. It is especially useful for huge data sets or streaming data because of its ability to find a good clustering solution with only a single scan of data. Optionally, it can further scan the data to improve clustering.

7.1 Algorithm Outline

1. The BIRCH builds a Clustering Feature Tree(CFT) for the given data. In CFT, each leaf node contains a sub-cluster. Every CF tree entry (CF entry) contains a pointer to a child node, and a CF entry comprises the sum of CF entries in the child nodes. There is a maximum number of entries in each

leaf node. This maximum number is called the threshold, which is also a parameter of BIRCH.

2. Entries in CFT are known as Cluster Feature Entries/Nodes. Cluster Feature entry is defined as an ordered triple, (N, LS, SS) where 'N' is the number of data points in the cluster, 'LS' is the linear sum of the data points, and 'SS' is the squared sum of the data points in the cluster. A CF entry can be composed of other CF entries.

3. BIRCH algorithm has two parameters, the threshold and the branching factor. The branching factor specifies the maximum number of CF sub-clusters in each node and the threshold is the maximum number of data points a sub-cluster in the leaf node of the CF tree can hold.

7.2 Application on Data Set

We Implement the BIRCH algorithm using the Python library Scikit-learn.

```python
from sklearn.cluster import Birch
brc = Birch(n_clusters = 5,threshold = 0.1, branching_factor
    = 50)
brc.fit(df_PCA_scaled)
blabels = brc.predict(df_PCA_scaled)
print ("birch: silhouette: ", silhouette_score(df_PCA_scaled,
     blabels))
#Silhouette Score is 0.292 for BIRCH.
```

We get Silhouette Score of 0.292 using BIRCH algorithm.

Visualizing Clusters for BIRCH:

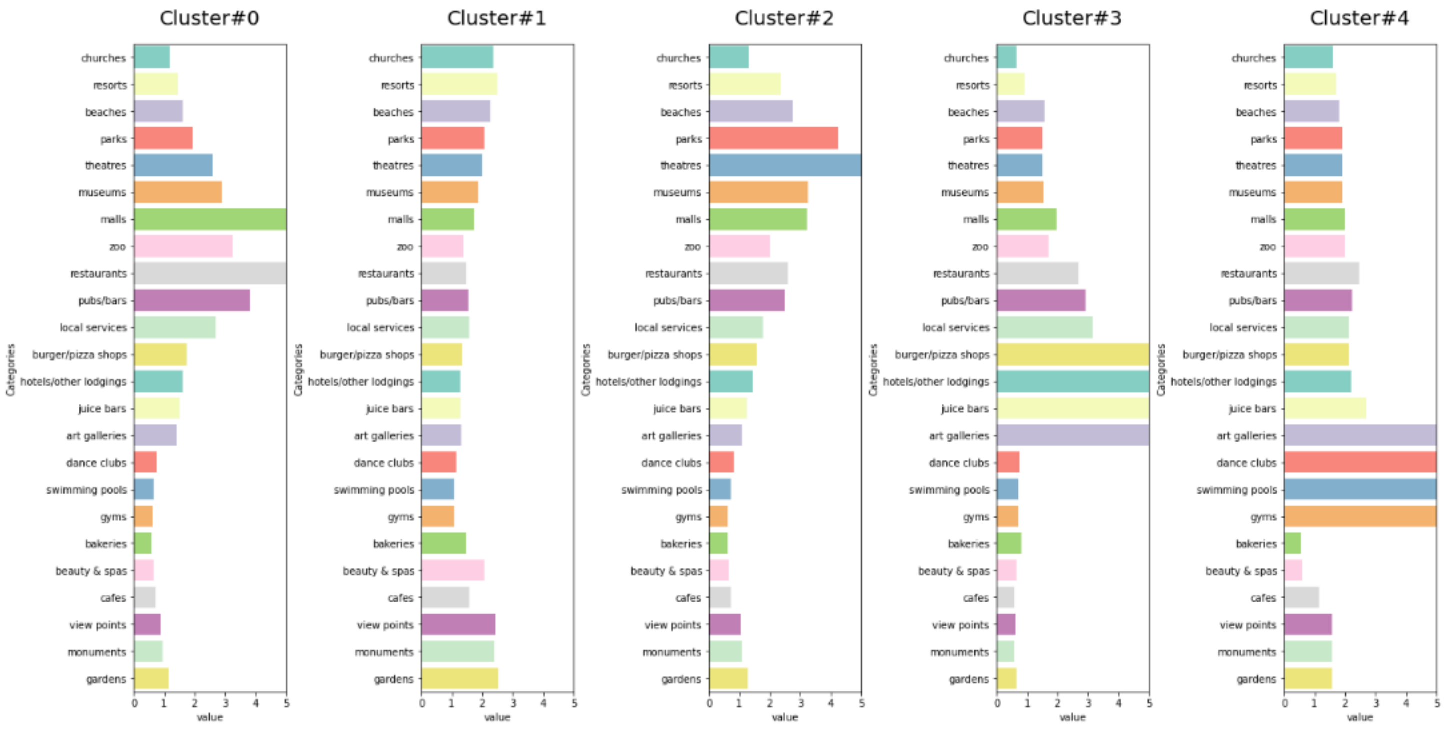


Figure 7.1: Clusters Created from BIRCH Algorithm

Cluster 0: Similar Distribution as Cluster D

Cluster 1: Similar Distribution as Cluster E

Cluster 2: Similar Distribution as Cluster B

Cluster 3: Similar Distribution as Cluster C

Cluster 4: Similar Distribution as Cluster A

Chapter 8

Agglomerative Hierarchical Clustering Algorithm

Hierarchical clustering is a clustering technique that groups similar data points into clusters. There are two types of Hierarchical clustering -
1. Divisive (top-down): This a top-down approach; all observations start in one cluster, and at each step most heterogeneous cluster is divided into two, as one moves down the hierarchy.
2. Agglomerative (bottom - up): This is a "bottom-up" approach; each observation starts in its own cluster, and similar pairs of clusters are merged as one moves up the hierarchy.
We are going to focus on Agglomerative Clustering here.

8.1 Algorithm Outline

1. Initially, every data point is considered a unique cluster. Then Agglomerative Clustering performs two steps Iteratively until the desired output is reached -
   1. Identify which 2 clusters are most similar
   2. Merge the 2 most similar clusters.

   These two steps are repeated iteratively to get the desired number of clusters.

2. The criteria for merging the clusters is known as ‘Linkage Criteria’. There are four different *Linkage Criteria* in Agglomerative Clustering: Ward, Average Linkage, Minimum Linkage, and Single Linkage.

   - Ward minimizes the variance(sum of squared differences) of clusters being merged. It is a variance-minimizing approach and, in this sense, is similar to the K-means.
   - Average Linkage uses the average of the distances of each observation of the two clusters.
   -Maximum linkage uses the maximum distances between all observations of the two clusters.
   - Single Linkage uses the minimum of the distances between all observations of the two clusters.

   We can select which linkage criteria to use while implementing agglomerative clustering.

8.2 Application on Data Set

We Implement the agglomerative clustering algorithm with linkage criteria ‘ward’, using the Python library Scikit-learn.

```
from sklearn.cluster import AgglomerativeClustering
clustering = AgglomerativeClustering(n_clusters = 5, affinity
    ='euclidean',linkage='ward').fit(df_PCA_scaled)
wlabels = clustering.labels_
print ("ward: silhouette: ", silhouette_score(df_PCA_scaled,
    wlabels))
#Silhouette score is 0.291 for agglomerative clustering.
```

We get Silhouette Score of 0.291 using agglomerative clustering (ward) algorithm.

Visualizing Clusters for Agglomerative Clustering:

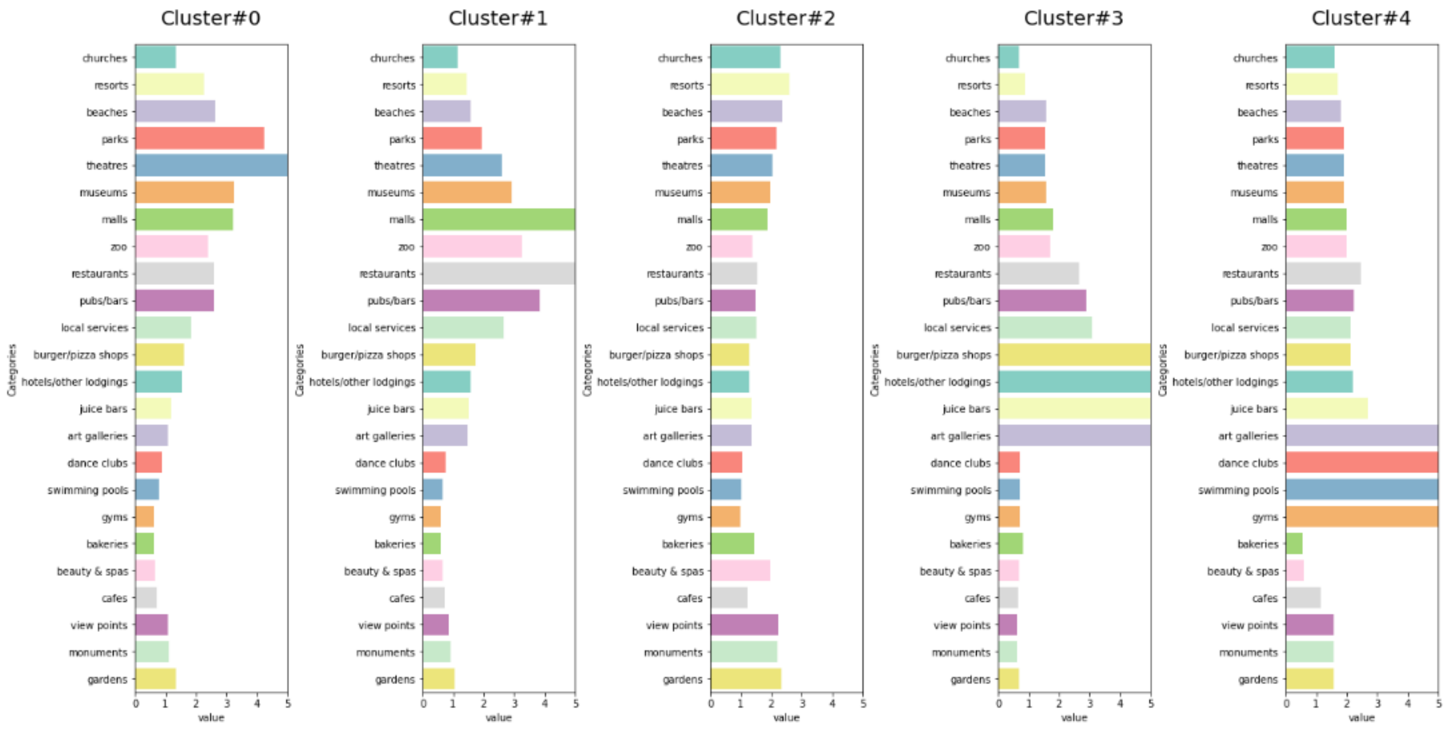


Figure 8.1: Clusters Created from Agglomerative Clustering Algorithm

Cluster 0: Similar Distribution as Cluster B

Cluster 1: Similar Distribution as Cluster D

Cluster 2: Similar Distribution as Cluster E

Cluster 3: Similar Distribution as Cluster C

Cluster 4: Similar Distribution as Cluster A

Chapter 9

Conclusions

Algorithm	Silhouette Score
K-means	0.309
K-means++	0.308
GMM	0.307
Spectral Clustering	0.299
Affinity Propagation	0.306
BIRCH	0.292
Agglomerative Hierarchical	0.291

The results of the six clustering algorithms discussed above are consistent. All clusters created are extremely similar through all methods which validates the fact that our data can be partitioned in these five clusters. The Silhouette scores are almost equal for all the algorithms but we can see that K-means, K-means++, and GMM algorithm performed best for this data set. The data set is uniformly distributed which is why these six algorithms have performed well for this data set.

We have also tried DBSCAN (Density-Based Spatial Clustering of Applications with Noise) and OPTICS (Ordering Points To Identify Cluster Structure) algorithms for this data set . Both algorithms are density-based algorithms and fail in case of varying density clusters and high dimensional data. Since our data set is uniformly distributed, DBSCAN puts all the points in single cluster and creates only one cluster. OPTICS is also an extension of DBSCAN algorithm and which is why it fails as well.

Now we will discuss an interpretation of resulting clusters. The five clusters created by the algorithms were the following:

Cluster A : likes to visit art galleries, dance clubs, swimming pools, and gyms.

Cluster B : likes to visit parks, theatres, and museums.

Cluster C : likes to visit burger/pizza shop, hotels, juice bars, and art galleries.

Cluster D : likes to visit malls, zoo, restaurants, pubs/bars, and local services.

Cluster E : Does not like anything too much but prefers churches, resorts, beaches, viewpoints, monuments, and gardens over others.

Cluster A shows that demographic that likes to visit dance clubs, swimming pools and gyms is more likely to visit art galleries. So it makes sense for art galleries to invest money in advertising in dance clubs, swimming pools, gyms and fitness centers. Fitness centers can also advertise in art galleries to attract more customers. It's also worth noting that gyms and swimming pools etc. are more likely to be found in the same area and already have the same clientele, so there's likely less of a need for them to advertise in each other's spaces.

Cluster B represents a demographic which likes to visit theaters, parks and museums. We can share this information with travel companies so they can offer customised tours to cater for the type of person represented in cluster B.
Travel companies can offer joint tickets to access several famous museums and theatres. We know from our data analysis that the same demographic is likely to be interested in both and they're often found in similar city areas (generally on the outskirts, where land is affordable enough to accommodate large buildings).

Armed with this information, special deals could be made between travel companies and theatres/museums, to allow a share of the profits to be split amongst all participating businesses.

Cluster C shows that the demographic that likes to go to burger/pizza shop, hotels and juice bars, also likes art galleries. So it makes sense for art galleries to invest money in having on-site catering, given that their demographic has shown an interest in burger/pizza shop and juice bars. Burger/pizza shop, hotels, and juice bars could advertise in art galleries to attract more customers and vice-versa.

Cluster D shows that the demographic that likes to visit restaurants, pubs and malls also likes zoos. It therefore makes sense to recommend for zoos to focus on advertising in places that serve food/drink.

Cluster E represents a demography which do not like any thing in extreme but are still fond of visiting churches, resorts, beaches, viewpoints, monuments, and gardens. We can recommend resorts to provide day tours of such places for their guests.

We can share this information with tourism companies and holiday providers combined with advertising offers to help them cater to needs of their customers. This project shows the value in taking a data analytical approach to the tourism industry. It suggests that travel agencies and tour companies could invest money into conducting surveys of their own, in order to maximise profits by optimising marketing strategy for their business, as we have done here.

Bibliography

[1] *Machine Learning Repository of University of California*,

[2] *Sci-Kit Learn Official Documentation*,

[3] Shini Renjith, A. Sreekumar, M. Jathavedan Evaluation of Partitioning Clustering Algorithms for Processing Social Media Data in Tourism Domain, 2018 IEEE Recent Advances in Intelligent Computational Systems (RAICS) — December 06 - 08, 2018 — Trivandrum

[4] *K Means Clustering Git hub pages*

[5] *KMeans Silhouette Score Explained With Python Example*

[6] *Affinity Propagation Algorithm Explained*

[7] *ML — BIRCH Clustering*, GeeksforGeeks