Implementation of K-Means algorithm in determining pickup hotspots for Cabs

Naru Venkata Pavan Saish
(20BDS0167)

AIM

Determining the pick-up hotspots for a Cab Company using the K-Means Algorithm

About the Project

- ★ Collect the data and clean it
- ★ Analyze the pick-up points
- ★ Implement K-means algorithm
- ★ Finding the hot-spots using the graphs

K-Means Algorithm

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third step, if any reassignment occurs, then go to step-4 else stop the algorithm.

Advantages

- Relatively simple to implement
- Less in complexity
- Scales to large data sets
- K means may be computationally faster than hierarchical clustering for larger no of variables (if K is small)

Disadvantages

- Difficult to predict the number of clusters (K-Value)
- Different initial partitions can result in different final clusters
- Clustering data of varying sizes and density.
- Clustering outliers

Elbow Method

In the Elbow method, we are actually varying the number of clusters from 3 - 11. For each value of K, we calculate WCSS (Within-Cluster Sum of Square).

WCSS is the sum of squared distance between each point and the centroid in a cluster. When we plot the WCSS with the K value, the plot looks like an **Elbow**. As the number of clusters increases, the WCSS value will start to decrease. WCSS value is large when K = 1. When we analyze the graph we can see that the graph will rapidly change at a point and thus creating an elbow shape. From this point, the graph starts to move almost parallel to the X-axis. The K value corresponding to that point will be the optimal K value.

Code

```
from scipy.spatial import ConvexHull
Import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
import numpy as np
df = pd.read_csv('Rides.csv')
### BUILD A TWO DIMENSIONS CLUSTER AGAIN ###
# k means
```

Code

```
kmeans = KMeans(n_clusters=3, random_state=0)
df['cluster'] = kmeans.fit_predict(df[['Longitude', 'Latitude']])
# get centroids
centroids = kmeans.cluster_centers_
cen_x = [i[0] for i in centroids]
cen_y = [i[1] for i in centroids]
## add to df
df['cen_x'] = df.cluster.map({0:cen_x[0], 1:cen_x[1], 2:cen_x[2]})
df['cen_y'] = df.cluster.map({0:cen_y[0], 1:cen_y[1], 2:cen_y[2]})
# define and map colors
colors = ['#DF2020', '#81DF20', '#2095DF']
df['c'] = df.cluster.map({0:colors[0], 1:colors[1], 2:colors[2]})

x = df.iloc[:, 4:6]  # 1t for rows and second for columns
kmeans = KMeans(3)
```

Code

```python
kmeans.fit(x)
wcss = []
for i in range(1, 7):
  kmeans = KMeans(i)
  kmeans.fit(x)
  wcss_iter = kmeans.inertia_
  wcss.append(wcss_iter)
number_clusters = range(1, 7)
plt.plot(number_clusters, wcss)
plt.title('The Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
```

Code

```
kmeans = KMeans(n_clusters=3, random_state=0)
df['cluster'] = kmeans.fit_predict(df[['Longitude', 'Latitude']])
# get centroids
centroids = kmeans.cluster_centers_
cen_x = [i[0] for i in centroids]
cen_y = [i[1] for i in centroids]
## add to df
df['cen_x'] = df.cluster.map({0:cen_x[0], 1:cen_x[1], 2:cen_x[2]})
df['cen_y'] = df.cluster.map({0:cen_y[0], 1:cen_y[1], 2:cen_y[2]})
# define and map colors
colors = ['#DF2020', '#81DF20', '#2095DF']
df['c'] = df.cluster.map({0:colors[0], 1:colors[1], 2:colors[2]})
```

Code

```python
from scipy import interpolate
fig, ax = plt.subplots(1, figsize=(8,8))
plt.scatter(df.Longitude, df.Latitude, c=df.c, alpha = 0.6, s=10)
plt.scatter(cen_x, cen_y, marker='^', c=colors, s=70)

for i in df.cluster.unique():
    # get the convex hull
    points = df[df.cluster == i][['Longitude', 'Latitude']].values
    hull = ConvexHull(points)
    x_hull = np.append(points[hull.vertices,0],
                       points[hull.vertices,0][0])
    y_hull = np.append(points[hull.vertices,1],
                       points[hull.vertices,1][0])
    # interpolate
```

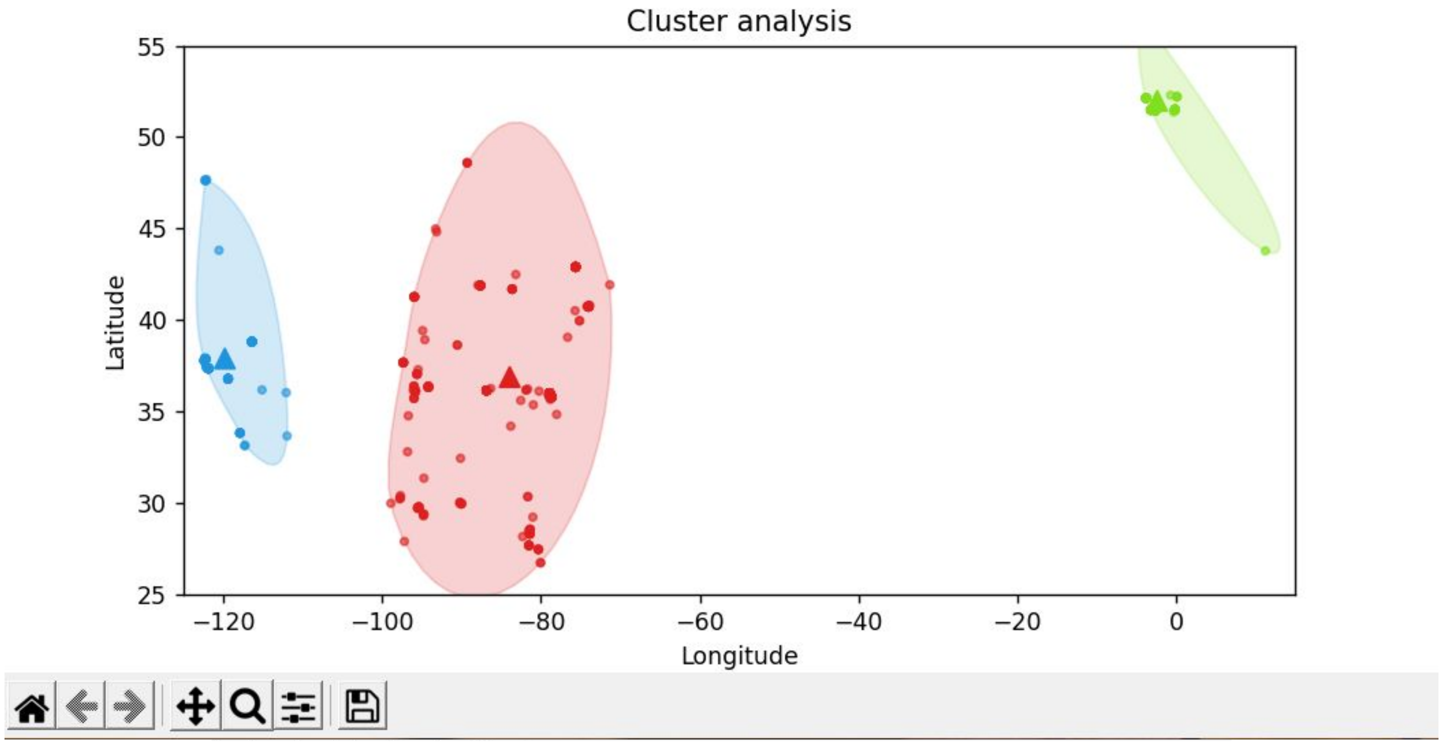
Code

```
    dist = np.sqrt((x_hull[:-1]-x_hull[1:])**2+(y_hull[:-1]-y_hull[1:])**2)
    dist_along = np.concatenate(([0], dist.cumsum()))
    spline, u = interpolate.splprep([x_hull, y_hull],
                                    u=dist_along, s=0)
    interp_d = np.linspace(dist_along[0], dist_along[-1], 50)
    interp_x, interp_y = interpolate.splev(interp_d, spline)
    # plot shape
    plt.fill(interp_x, interp_y, '--', c=colors[i], alpha=0.2)
# plt.xlim(0,200)
# plt.ylim(0,200)
z=1
p=1
```

Code

```
plt.ylim(25*z, 55*z)
plt.xlim(-125*p, 15*p)
plt.title('Cluster analysis')
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
```

Output

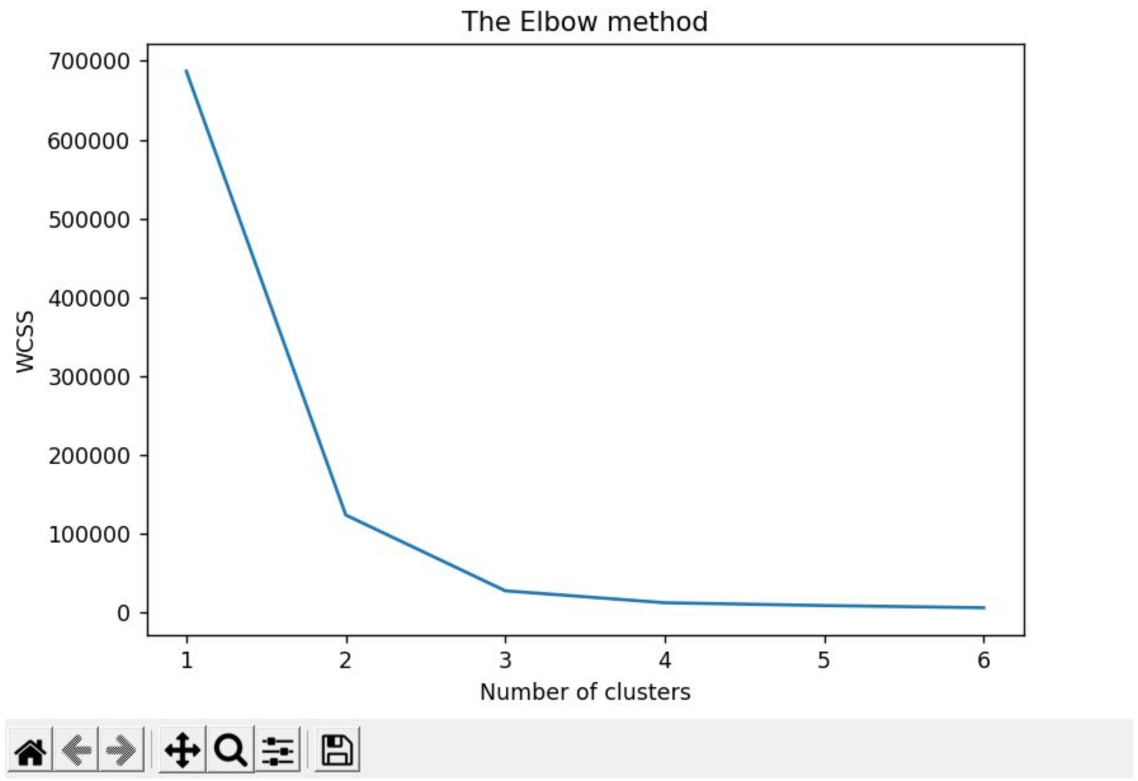

Cluster analysis
Latitude
Longitude
55
50
45
40
35
30
25
−120
−100
−80
−60
−40
−20
0

Choosing K using Elbow method

In the Elbow method, we are actually varying the number of clusters (K) from 1 – 7. For each value of K, we are calculating WCSS (Within-Cluster Sum of Square). WCSS is the sum of squared distance between each point and the centroid in a cluster. When we plot the WCSS with the K value, the plot looks like an Elbow. The K value corresponding to the point where **the bend** occurs, is the optimal K value.

In the graph we obtained, the bend occurs as K = 3. Hence, that is the optimum number of clusters.

Output


The Elbow method
700000
600000
500000
400000
300000
200000
100000
0
WCSS
1
2
3
4
5
6
Number of clusters

Conclusion

The above code helps us to find the hotspots by implementing k means algorithm and elbow method.

From the geographic data points (latitudes and longitudes), we clustered them with the help of elbow method and then implemented the k means algorithm to find the hotspots.