GEO-LOCATION CLUSTERING USING THE k-MEANS ALGORITHM

Motivation:

The goal of this project is the same as the real life scenario of service providers. The movie recommendation project is the one I already did in one of my undergrad projects. Also, I never worked with location data so this motivated me to do this project.

Project Goal:

Implement k-means in SPARK and use it for geo-location clustering on various datasets of spatial locations.

Problem-2:

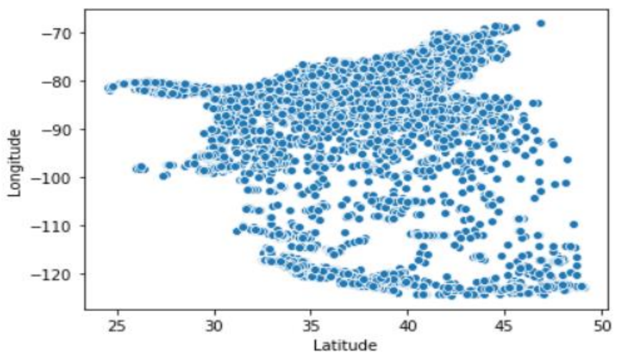
Log into your aws and type s3 in search bar. Then create a new bucket and create a bucket by typing the name and allow public access by unchecking the check box. Then upload the 3 .txt files into the bucket.

Step-1:

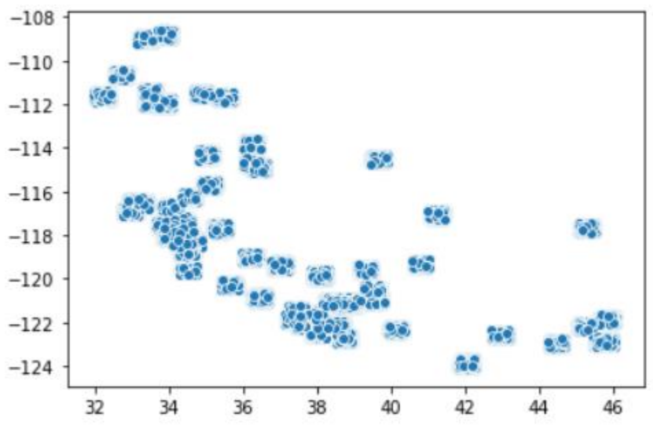
- We create the sparkcontext object and use it to load the data from the devicestatus.txt.
- Then we split the data by line and substitute the '[\|/]' with ',' using string functions from the module 're'.
- Then split the data with ','.
- Then we take latitude, longitude, date and time, manufacturer and model_name into a dataframe.
- Then we specify the latitude and longitude as float and data and time as date and time and the manufacturer and model_name as string.
- Then we save this cleaned dataframe into the s3 bucket.

Step-2:

- Now we do the same cleaning and loading of data from sample_geo.txt instead splitting on '\n' we split on '\t'.
- Then we convert them to appropriate datatypes.
- Now we use seaborn to plot the data by extracting it from the s3 bucket.

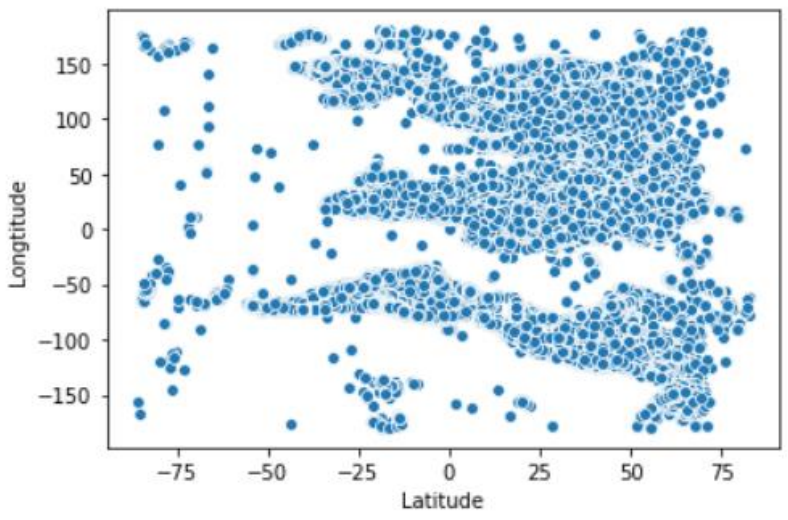


- Now if we use seaborn to visualize the devicestatus .txt then we get.



Step-3:

- Now if we use seaborn to visualize the lat_long.txt using the same methods as above we get the following:



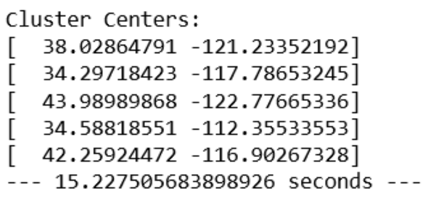
Problem-3:

Step-2:

The K-Means in the spark.ml.clustering has the convergence threshold(which is epsilon) is set by it's own calculation.

Step-3:

We first use the devicestatus.txt to get the clustering predictions. Then using this we obtain the cluster's longitude and latitude.



```
Cluster Centers:
[  38.02864791 -121.23352192]
[  34.29718423 -117.78653245]
[  43.98989868 -122.77665336]
[  34.58818551 -112.35533553]
[  42.25924472 -116.90267328]
--- 15.227505683898926 seconds ---
```

```
+-----------------+------------------+----------+---------------+----------------+
|original_latitude|original_longitude|prediction|center_latitude|center_longitude|
+-----------------+------------------+----------+---------------+----------------+
|        33.689476|       -117.543304|         1|      34.297184|      -117.78653|
|         37.43211|        -121.48503|         0|       38.02865|      -121.23352|
|         39.43789|        -120.93898|         0|       38.02865|      -121.23352|
|        39.363518|        -119.40034|         0|       38.02865|      -121.23352|
|        33.191357|        -116.44824|         1|      34.297184|      -117.78653|
+-----------------+------------------+----------+---------------+----------------+
only showing top 5 rows
```

We calculate the Great-Circle-Distance and the Euclidean distance using the geodesic and distance.euclidean methods respectively.

```
+-----------------+------------------+----------+---------------+----------------+------------------+------------------+
|original_latitude|original_longitude|prediction|center_latitude|center_longitude|           gc_dist|           eu_dist|
+-----------------+------------------+----------+---------------+----------------+------------------+------------------+
|        33.689476|       -117.543304|         1|      34.297184|      -117.78653| 71197.25683187979|0.4284674337977763|
|         37.43211|        -121.48503|         0|       38.02865|      -121.23352| 69922.61967336302|0.4191158261528471|
|         39.43789|        -120.93898|         0|       38.02865|      -121.23352| 158769.1391050153| 2.0727134647313505|
|        39.363518|        -119.40034|         0|       38.02865|      -121.23352|217572.52162612986|  5.142437243892346|
|        33.191357|        -116.44824|         1|      34.297184|      -117.78653| 174441.9225295172| 3.0138671277672984|
+-----------------+------------------+----------+---------------+----------------+------------------+------------------+
only showing top 5 rows
```

The calculation procedure if done manually is:

Great Circle Distance in Haversine Formula

dlat = math.radians(lat2 - lat1)

dlon = math.radians(lon2 - lon1)

a = (math.sin(dlat / 2) * math.sin(dlat / 2) +

math.cos(math.radians(lat1)) * math.cos(math.radians(lat2)) *

math.sin(dlon / 2) * math.sin(dlon / 2))

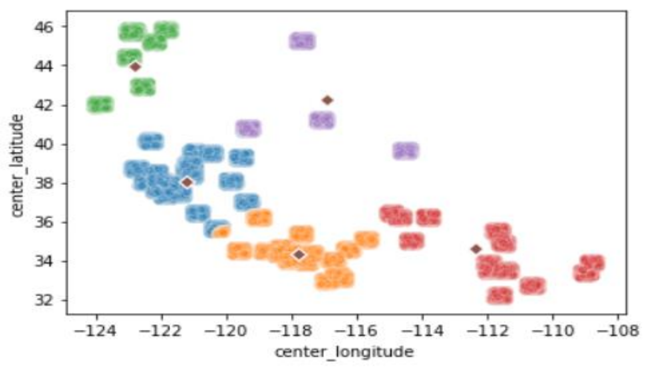
c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

d = radius * c

Euclidean Distance

distance = math.sqrt(sum([(a - b) ** 2 for a, b in zip(x, y)]))

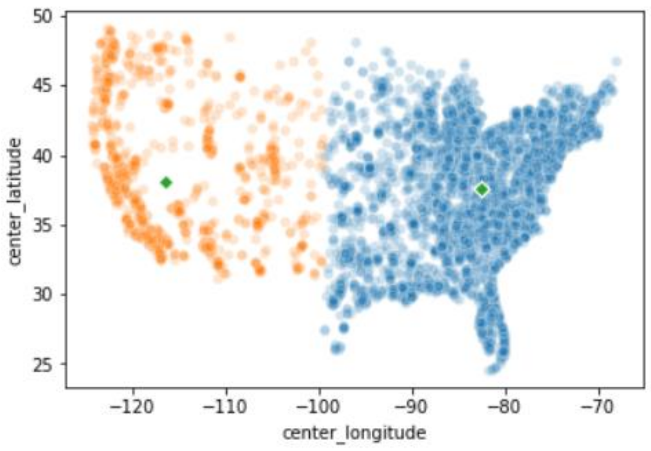
The cluster plot is shown below:



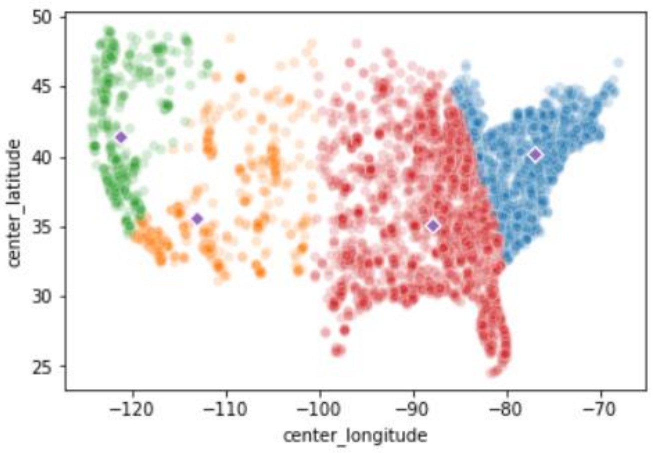
Note:

The cluster centers are shaped as parallelograms or like square rotated to 45 degrees angle.

Now by using sample_geo.txt and K=2 we get:

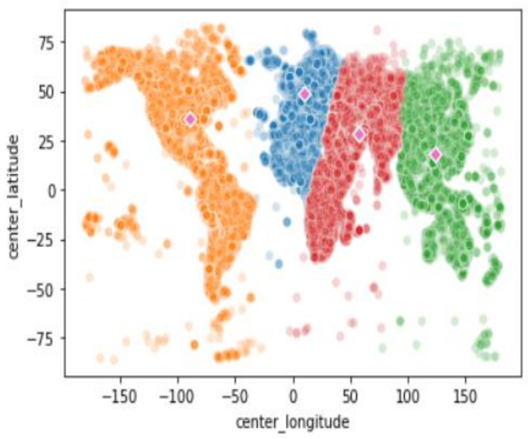


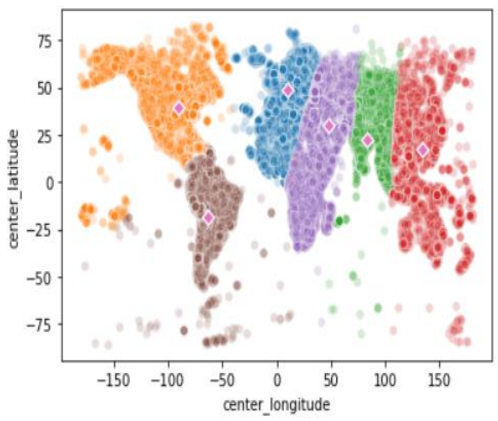
And by using sample_geo.txt and K=4 we get:



Now we do the same for lat_long.txt data:

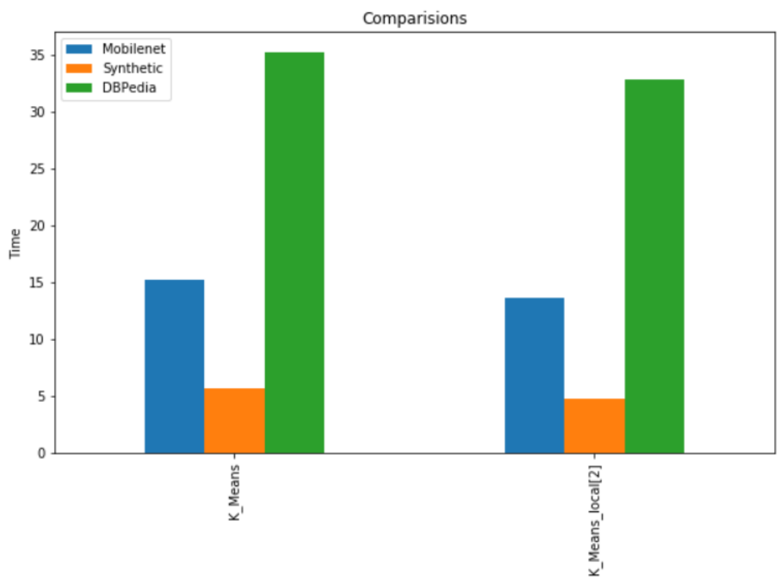
K=4: K=4:





Step-4:

Now we compare the times for all 3 datasets in emr cluster environment and in local environment.



The times in the EMR cluster environment and the local environment are same.

Conclusion:

Thus we created clusters to know which locations cover most of the services and which are of importance. Thus we can connect the people who have service down to those ones and maintain the main ones with more than required for so that to accommodate and satisfy all users. We can also see where the service is low or unstable so we can implement and improve the service there.

Also, since it is a fast growing service(Loudacre) they also need to know the how the people in different locations are using the service like for phone calls, for internet, for location etc.,