University of New Haven
Tagliatela college of Engineering
Data Science, Computer Engineering and Computer Science

GEO-LOCATION CLUSTERING USING K-MEANS ALGORITHM

DISTRIBUTED AND SCALABLE DATA ENGINEERING

DSCI – 6007

Date: 12/8/2020

By,

Namitha Nagaraju

Supervisor:
Dr.Vahid Behzadan

Table of contents

1. Introduction

Clustering is the process of grouping a set of data points into k clusters. Data points that are similar will be grouped to in one cluster and the dissimilar ones will be grouped in another cluster.

Clustering is used in applications such as Pattern recognition, Image analysis, Information Retrieval, Bioinformatics, Document classification, Logistics, Marketing, Computer graphics, Machine learning etc.,

There are many algorithms that can be used to achieve this goal. Some of them are:

- Connectivity-based clustering
- Distribution-based clustering
- Density-based clustering
- Grid-based clustering

In this project a *Centroid-based* clustering technique- *k-means algorithm* is implemented. K-means calculates distances that iteratively updates the location of k cluster centroids until convergence. The goal is to minimize the distances between the each data point and their centroid.

The most important parameters of the k-means algorithm are:

- **Distance function**

To measure the distance between the geo-points, one of the two below distance measures are used. Here d is the distance, R is the radius of earth, θ is latitude and ϕ is longitude.

  - Euclidian distance
    - $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$
    - $x = R\ \cos\theta \cos\phi$
    - $y = R\ \cos\theta \sin\phi$
    - $z = R\ \sin\theta$
  - Haversine Distance:
    - $d = R.c$
    - $c = 2.\text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$
    - $a = sin^2\left(\frac{\Delta\theta}{2}\right) + \cos\theta_1 . \cos\theta_2 . sin^2(\frac{\Delta\phi}{2})$
- **Cluster size (k)**

  This parameter needs to be set according to the application or problem domain.

Time taken to cluster the data points with respect to different dataset sizes, cluster sizes and distance measures is also analysed in this project. Finally, a larger geo-location dataset is clustered on Amazon EMR. The datasets used for the analysis are devicestatus.txt, sample-geo.txt, lat-longs.txt. These datasets are stored on Amazon S3.

2. Data pre-processing and visualization

Data pre-processing is transforming raw data into an understandable format. The three datasets of sizes 10000, 9970, and 450150 respectively are cleaned to obtain desired features. This cleaned data is saved on Amazon S3.

2.a. Device status data:

The input data contains information collected from mobile devices on Loudacre's network, including device ID, current status, location and so on. Loudacre Mobile is a (fictional) fast- growing wireless carrier that provides mobile service to customers throughout western USA. Because Loudacre previously acquired other mobile provider's networks, the data from different subnetworks has a different format. The following steps are conducted at this stage –

- Loading the dataset
- Determine which delimiters to use
- Filter out any records which have erroneous data
- Extract date, model, Device ID, latitude and longitude.
- Filter out locations that have a latitude and longitude of magnitude 0.
- Separate the manufacturer from the model field.
- Save data on S3.

The below Figure 1 is the visualization of the cleaned data. Python's *Geopandas* library is used to visualize the longitude and latitude pairs.

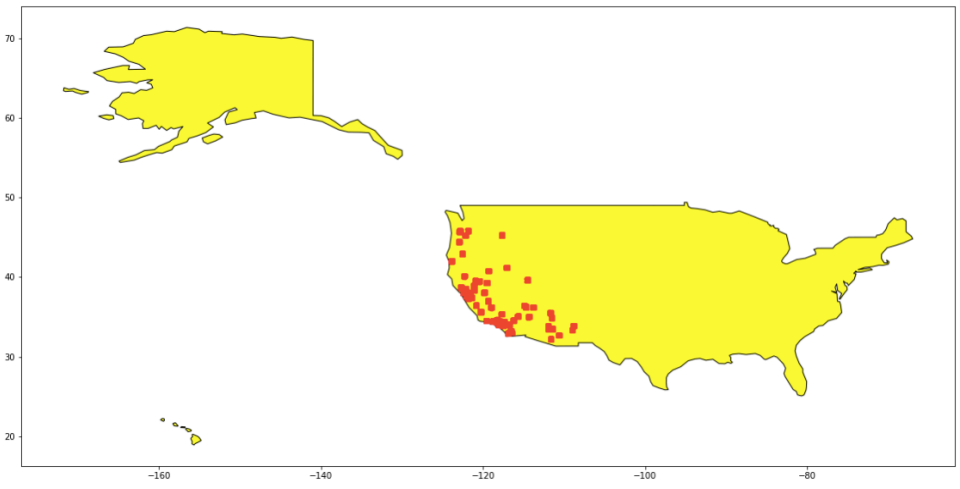

Figure 1: Visualization of cleaned Device status data

2.b Synthetic location data:

The above steps are carried out on a dataset with 9970 points. Similar pre-processing is done on the dataset and the resulting dataset is uploaded to Amazon S3. The below figure is the visualization of longitude and latitude pairs obtained from sample-geo.txt dataset.

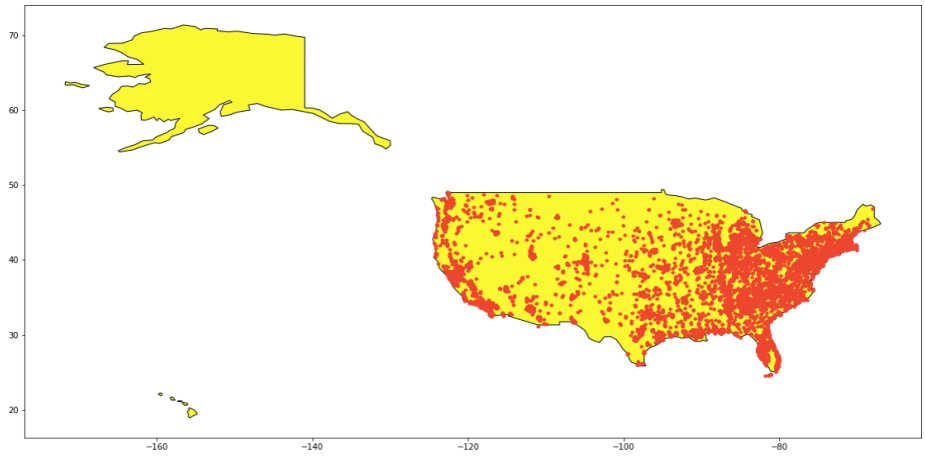

Figure 2: Visualization of cleaned Synthetic location data

2.b DBpedia location data:

This is a large-scale data where (latitude, longitude) pairs are extracted from DB-pedia (lat_longs.zip). Each record represents a location that has a Wikipedia article and latitude-longitude information. The format is: latitude, longitude and the name_of_page. In total, there are 450,151 data points. Figure 3 is obtained by plotting the geo-points on a Geopandas's world dataset file.

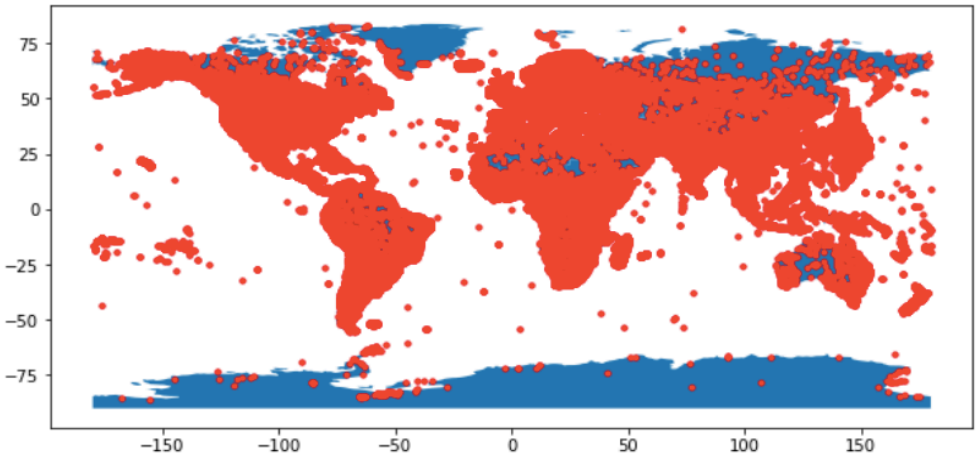

Figure 3: Visualization of cleaned DBPedia location data

3. Implementation of the K-means algorithm

Given below is the pseudocode of geo-clustering using k-means algorithm. To initialize the iterative algorithm, a random sample of k centroids are taken from the data points. In every iteration, the data points are assigned to their nearest centroid. This is measured using either the Euclidian or Haversine distance. Next, mean of each of the k clusters is calculated. This becomes the new centroid. Iteration distance is calculated by adding all the distances from every data point to the centroid. The goal is to minimize this distance. The middle point is calculated again using the new centroid. This process is repeated until convergence.

In this project, Apache Spark framework is used. Spark is especially useful for parallel processing of distributed data with iterative algorithms on big data workloads**.** The implementation of the k-means algorithm is done using PySpark, which is the Python API to support Apache Spark**.**

- *initialCentroids = random k data points*
- *convergenceDistance = 1*
- *WHILE IterationDistance > convergenceDistance*
    - *Find and assign the nearest centroid to each point*
    - *Find the mean of each k clusters*
    - *Assign the mean as the new centroid*
    - *Compute the iterationDistance which is the sum of all the data points in a single cluster.*

The following figures show the visualization of a few iterations of the k-means clustering algorithm applied on the Synthetic dataset with k=4. Euclidian distance is used to measure the closest centroid and the convergence distance is taken to be 1 kilo meter.

Figure 4: Initial random centroids

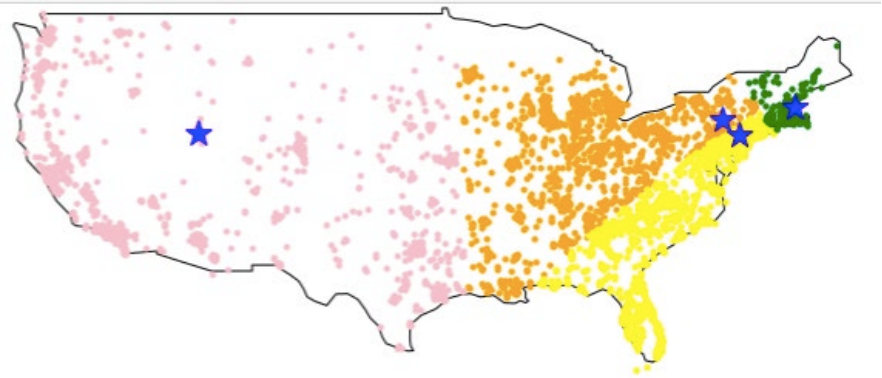

Figure 5: Finding closest centroid (clusters)

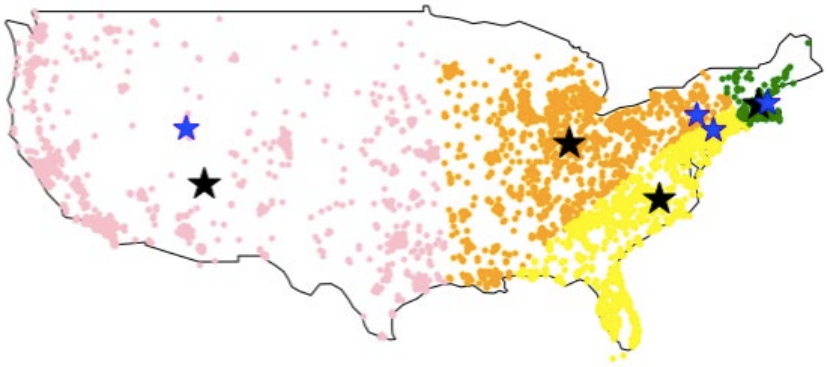

Figure 6: Finding mean of the clusters (Black)

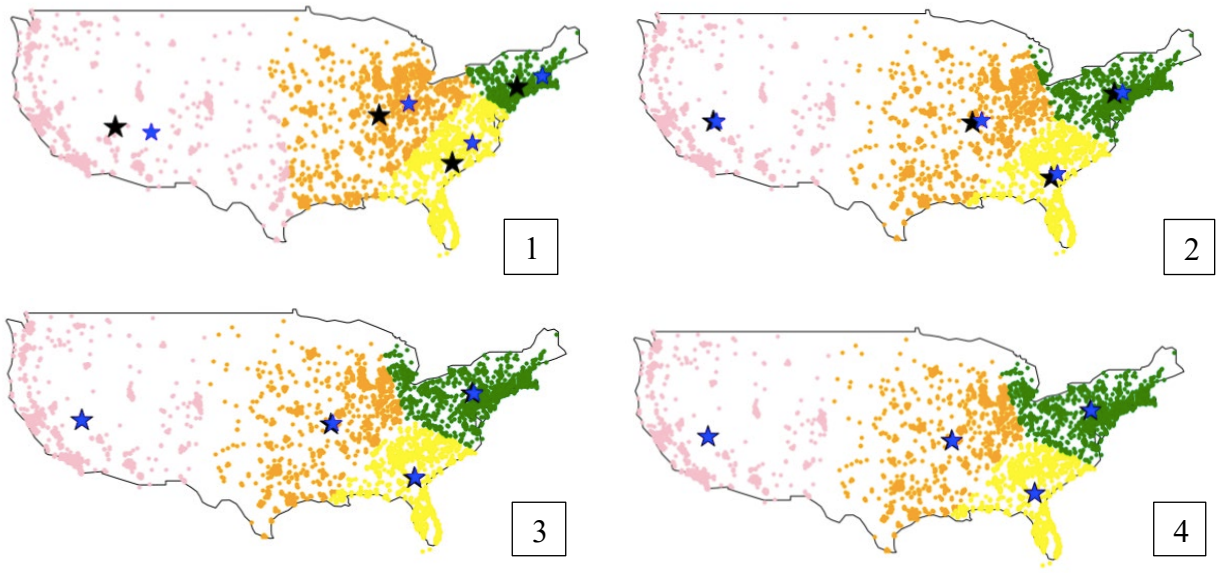

Figure 7: Subsequent iterations till convergence

3.a Device location data with k=5

The below graph is obtained clustering the pre-processed Device status data into 5 zones depicted in red, pink, yellow, orange and green colours. We can observe that the algorithm has horizontally partitioned California into 4 zones. This application of finding out the centre location could be used to enhance the networking experience of Loudacre Mobile users.

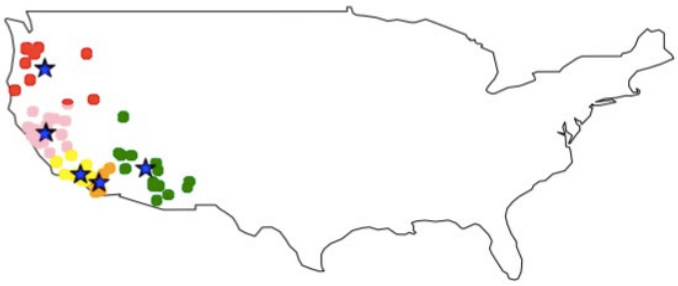

Figure 8: Clustering Device location data with k=5

3.b Synthetic location data with k=2 and k=4

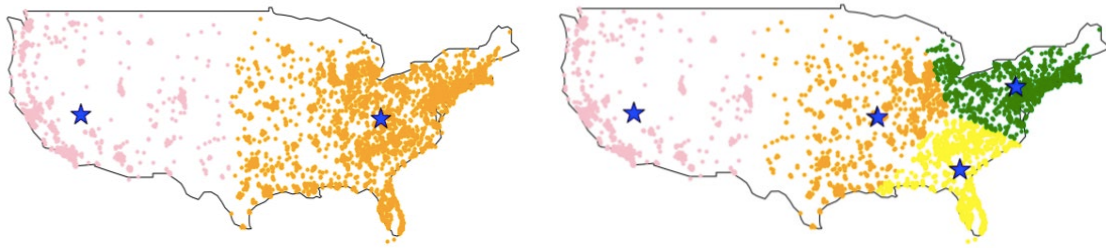

Figure 9: Clustering Synthetic location data with k=2 and k=4

It can be seen that when number of clusters were set to 2, the data points are divided into West and East of America. When k=4, we obtain 4 regions- West, Central, North-Eastern and South-Eastern regions.

3.c Dbpedia location data with k=6

Dbpedia is a larger dataset consisting of 4 lakhs longitude and latitude pairs. The below map was obtained by taking a subset of 1 lakh data points. The clusters were obtained by applying the algorithm with the stop condition of 100 kilo metres. The algorithm has grouped a few continents together when the number of clusters were set to 6. However, the parameter k changes based on the application and

inferences can be made accordingly from the resulting clusters.

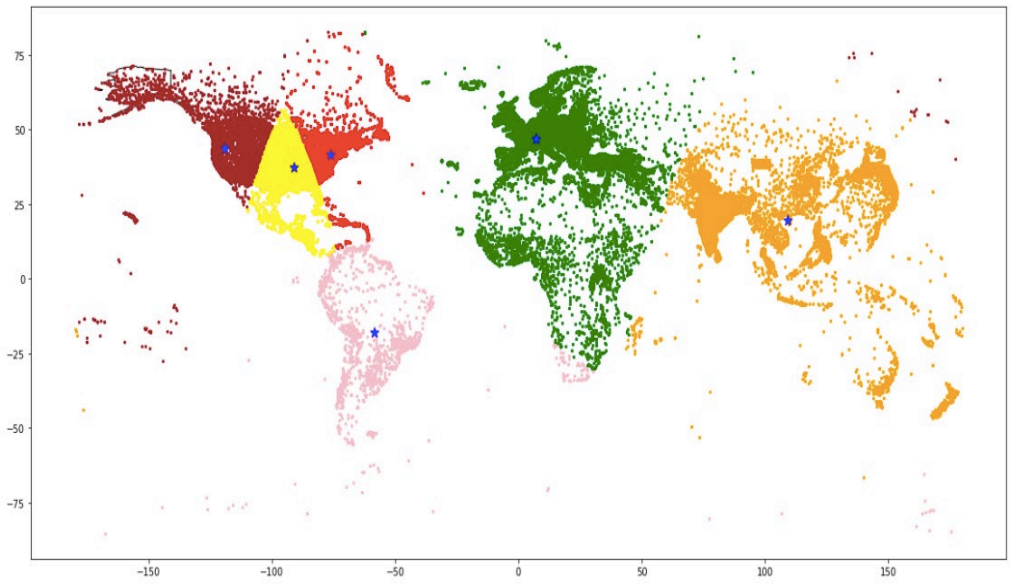

Figure 10: Clustering DBpedia data with k=2 and k=4

4. Runtime analysis

The algorithm is analysed by setting different k values and distance measure type. The below observations were made by keeping the random initial centroids constant. Spark job can be submitted as follows.

Spark-submit generic.py <input-file-name> <output-file-name> <distance-type> <k>

<input-file-name> is the file name of the processed device status, synthetic and DBPedia data uploaded to S3. It can also be any new data with first field being the latitude and second being the longitude. <output-file-name> can be any name that we want our clusters to be outputted in a text file on Amazon S3. Distance type can either be “euclidian” or “haversine” (case-sensitive) . <k> is the number of clusters needed .It must be a whole number greater than 0. The following table shows the time taken for the spark job to complete for different cluster sizes and distance measures on different datasets without and with caching (Haversine distance).

Device status data	**K=2**	**K=4**	**K=6**
Euclidian distance	143.47 s	352.55 s	309.50 s
Haversine distance	135.06 s	344.90 s	328.02 s
With caching	133.35 s	223.43 s	326.5 s

Table 1: Time taken to compute clusters on Device status data using different settings

Synthetic data	**K=2**	**K=4**	**K=6**
Euclidian distance	97.65 s	150.60 s	194.04 s
Haversine distance	88.80 s	77.91 s	128.11 s
With caching	88.95 s	74.91 s	121.65 s

Table 2: Time taken to compute clusters on Synthetic data using different settings

DBPedia data	**K=2**	**K=6**
Euclidian distance	625.72 s	1145.63 s
Haversine distance	695.50 s	977.56 s
With caching	603.57	911.99 s

Table 3: Time taken to compute clusters on DBPedia data using different settings

From the above observed run times, we can infer the following-

- As the number of clusters to be computed increases, the computational time increases in most of the cases.
- The choice of the distance measure seems to be not affecting the time taken to compute clusters.
- Caching has decreased the time taken in all the cases except for Synthetic data with K=2. The decrease in the computational time after caching is evident in larger datasets.

5. Geo-clustering for used cars data

A larger dataset of size 343.6 MB with 324405 geo-points (as of 3rd December 2020) was downloaded from https://www.kaggle.com/austinreese/craigslist-carstrucks-data. To be able to compute clusters for such a large dataset, a Spark program was executed on an Amazon EMR master node. The EMR runtime for Spark can be over 3x faster than the standard Spark. This improved performance helps workloads run faster and saves compute time and costs. The EMR clusters (both master and slave nodes) were of the instance type m5.xlarge with 4vCore, 16 GiB memory, EBS only storage of 64 GiB.

The spark job was submitted with the command on the master node

```
spark-submit s3://geo-clustering/big_data.py s3://geo-clustering/big data/
s3://geo-clustering/output
```

A PySpark program was written and uploaded on to the master node. The third argument is the path where the raw used car data was uploaded onto Amazon S3.

The data is pre-processed in the same program before computing the clusters to obtain the latitude and longitude pairs. The number of clusters to be found was initialized to 5 and was computed by calculating the Euclidian distance. The run time was significantly lower and the resulting clusters were outputted to the path specified as the last argument to the command.

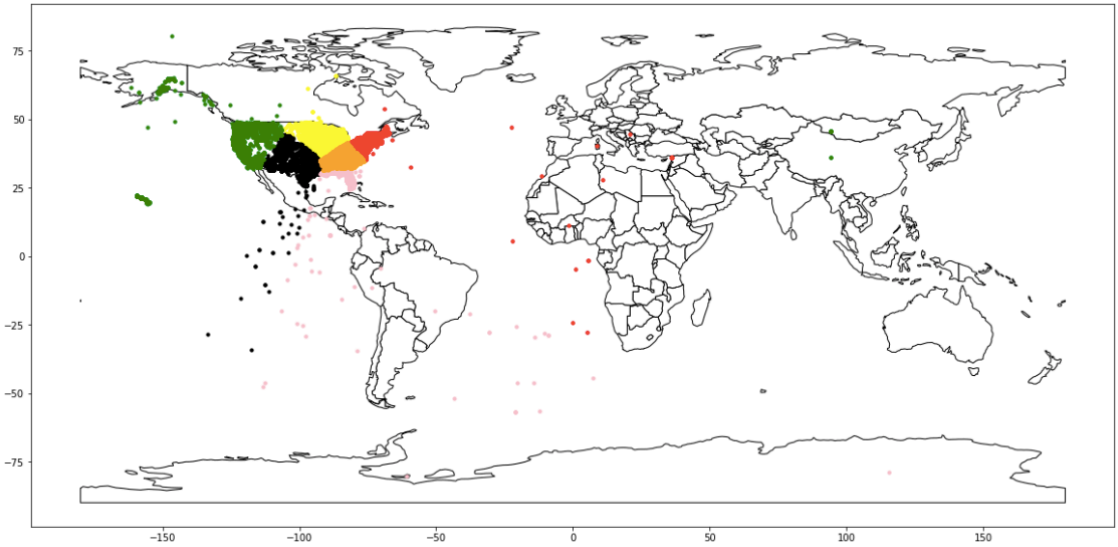

Figure 11: Clustering used car data with K=5 and Distance measure=Euclidian

The files were downloaded and converted into Pandas data frame to plot them using Geopandas library of Python. The above figure depicts the 5 different clusters that were obtained after successful completion of the Spark job.

This result can be used for useful purposes to manage the used cars from the different locations in the USA. More data pre-processing can be done to remove the outliers that can be observed in figure 11 to increase the accuracy of the middle location of each of the clusters which can be used to, for example, station a sale or a main waste management centre for a particular period.

6. Conclusion

Geo-location clustering finds similar regions in space which can be used in a whole range of applications. In this project, different sizes of datasets were used to form special clusters of sizes k, using the Apache's Spark framework. Spark is designed for in-memory processing. Spark achieves performance gains by caching the results of operations that repeat over and over again. The Spark Data frames which were used many times in the iteration of finding centroids were cached. A significant increase in the run time was observed as the size of the dataset increased. For an even larger dataset (USA used car data), the Spark program was executed on

Amazon EMR cluster for obtaining faster results. Visualizations were made to quickly understand the final result of the k-means clustering algorithm on the different special datasets.