# GEO-LOCATION CLUSTERING USING THE K-MEANS ALGORITHM



| | | |
|---|---|---|
| Submitted By | : | MUKESH MUVVA |
| Guidance | : | 1) Prof. Vahid Behzadan |
| | | 2) Piazza, Nancirose G |
| Course | : | Distributed and Scalable Data Engineering |
| Submission Date | : | 08 December 2020 |

# TABLE OF CONTENTS:

# Introduction:

In this project we are using SPARK to implement an iterative algorithm that solves the clustering problem in an efficient distributed fashion. Clustering is the process of grouping a set of objects (or data points) into a set of k clusters of similar objects. Thus, objects that are similar should be in the same cluster and objects that are dissimilar should be in different clusters. Clustering has many useful applications such as finding a group of consumers with common preferences, grouping documents based on the similarity of their contents, or finding spatial clusters of customers to improve logistics.

More specific use cases are
• Marketing: given a large set of customer transactions, find customers with similar purchasing behaviors.
• Document classification: cluster web log data to discover groups of similar access patterns.
• Logistics: find the best locations for warehouses or shipping centers to minimize shipping times.

We are trying implement clustering algorithm to Geo-Locations. This kind of computations generally lot of resources so we are using AWS EMR to compute clusters around geolocation data.

# The K-Means Algorithm

kmeans is a distance-based method that iteratively updates the location of k cluster centroids until convergence. The main user-defined ingredients of the k-means algorithm are the distance function (often Euclidean distance) and the number of clusters k. This parameter needs to be set according to the application or problem domain. (There is no magic formula to set k.) In a nutshell, k-means groups the data by minimizing the sum of squared distances between the data points and their respective closest-centroid. We can also find the best K-value from elbow graph which is a plot against  K values and mean square error, This plot displays the best k value project against mean square errors.

Algorithm:

1. Initialize **cluster centroids** $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$ randomly.
2. Repeat until convergence: {

   For every $i$, set

   $$c^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||^2.$$

   For each $j$, set

   $$\mu_j := \frac{\sum_{i=1}^{m} 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)} = j\}}.$$

   }

# DATA

In order to implement geolocation based clustering, I need to have a dataset related to geolocations. For this purpose I have taken three datasets to compute their clusters. The first dataset is a device status data set from Loudacre Mobile network. The second one is from Statistical research website where it consists of pairs of Latitudes and Longitudes. The third data set is from DBPedia where it consists pairs of latitude , longitude and name of the webpage.

Out of all these three datasets Loudacre is of having 13.3 MB in size, The second one is of 0.5 MB in size and the third one is of 36 MB in size. The first dataset consists of 14 features.

Data Set 1:
https://github.com/mukeshmuvva/data/blob/main/devicestatus.txt

Data Set 2:
https://statistical-research.com/wpcontent/uploads/2013/11/sample_geo.txt

Data Set-3:
https://github.com/mukeshmuvva/data/blob/main/devicestatus.txt

## Setting Up The Environment

This project uses aws  extensively, Because of the fact that computing clusters needs lot of resources to calcute them. It takes much time in normal machines to compute them. So I have chosen aws to compute them faster and it's the best to get analysis for our calculations.

- Create a s3 bucket and upload all the data here
- Create an ec2 private key-pair key, this allows as password to get into your aws instances
- Create an latest emr for spark.
- SSH into aws emr instance.
- Update the emr instance.
- Install jupyter notebook and iparallel from ssh instance
- Change vim configuration and adjust it to open pyspark at 8888 port through jupyter notebook
- Configure ec2-security inbound rules to allow applications to access on  8888,22,18080 ports
- Open an another SSH instance.
- Install python3-developer tools.
- Install Geopandas (Without developer tools installed u can't access geopandas)
- Update pip version
- Install s3fs library(s3fs library is mainly used to access s3 from pandas)

# Data Preparation

**Prepare device status data:**

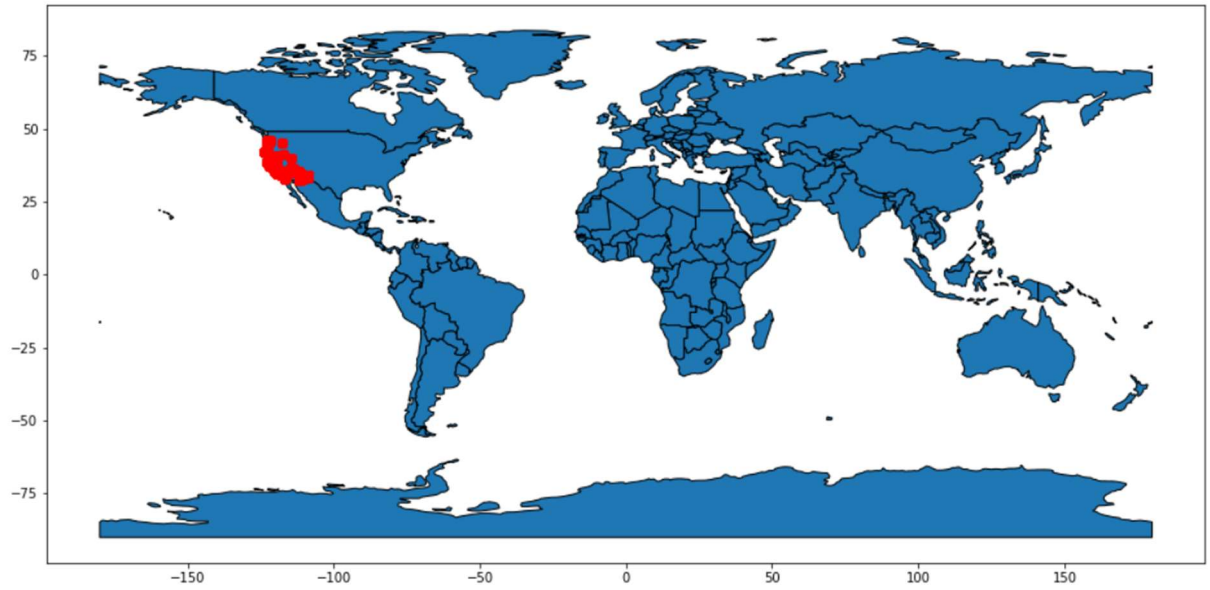
Upload this file to a S3 bucket. pre-process the data in order to get it into a standardized format for later processing. As part of the data cleaning I have done following taks.

- Reading the data set carefully
- Figure out the delimiters
- Use a Regex Expression to split the data
- After splitting the data remove all the unwanted data, In this case having more than 14 rows should be eliminated
- Create a pyspark dataframe from rdd
- Drop all the null values as they were having very less frequency
- Replace all unintelligible column names to intelligible column names
- Split the model column into combination of both model and manufacturer.
- Filter out all latitudes and longitudes which were having zero value
- Convert the pyspark dataframe to pandas dataframe and save it as csv file in s3.(Although u could convert pyspark dataframe to csv file, which is cumbersome.)
- After having all the preprocessing done, I have used the power of geopandas to plot all the latitude and longitude values.

## Visualizations for this data preprocessing:

| latitude | longitude | date | model | deviceID | manufacturer |
|---|---|---|---|---|---|
| 33.6894754264 | -117.543308253 | 2014-03-15:10:10:20 | Sorrento F41L | 8cc3b47e-bd01-448... | F41L |
| 37.4321088904 | -121.485029632 | 2014-03-15:10:10:20 | MeeToo 1.0 | ef8c7564-0a1a-465... | 1.0 |
| 39.4378908349 | -120.938978486 | 2014-03-15:10:10:20 | MeeToo 1.0 | 23eba027-b95a-472... | 1.0 |
| 39.3635186767 | -119.400334708 | 2014-03-15:10:10:20 | Sorrento F41L | 707daba1-5640-4d6... | F41L |
| 33.1913581092 | -116.448242643 | 2014-03-15:10:10:20 | Ronin Novelty Note 1 | db66fe81-aa55-43b... | 1 |
| 33.8343543748 | -117.330000857 | 2014-03-15:10:10:20 | Sorrento F41L | ffa18088-69a0-433... | F41L |
| 37.3803954321 | -121.840756755 | 2014-03-15:10:10:20 | Sorrento F33L | 66d678e6-9c87-48d... | F33L |
| 34.1841062345 | -117.9435329 | 2014-03-15:10:10:20 | MeeToo 4.1 | 673f7e4b-d52b-44f... | 4.1 |
| 32.2850556785 | -111.819583734 | 2014-03-15:10:10:20 | Ronin Novelty Note 2 | a678ccc3-b0d2-452... | 2 |
| 45.2400522984 | -122.377467861 | 2014-03-15:10:10:20 | Sorrento F41L | 86bef6ae-2f1c-42e... | F41L |

only showing top 10 rows



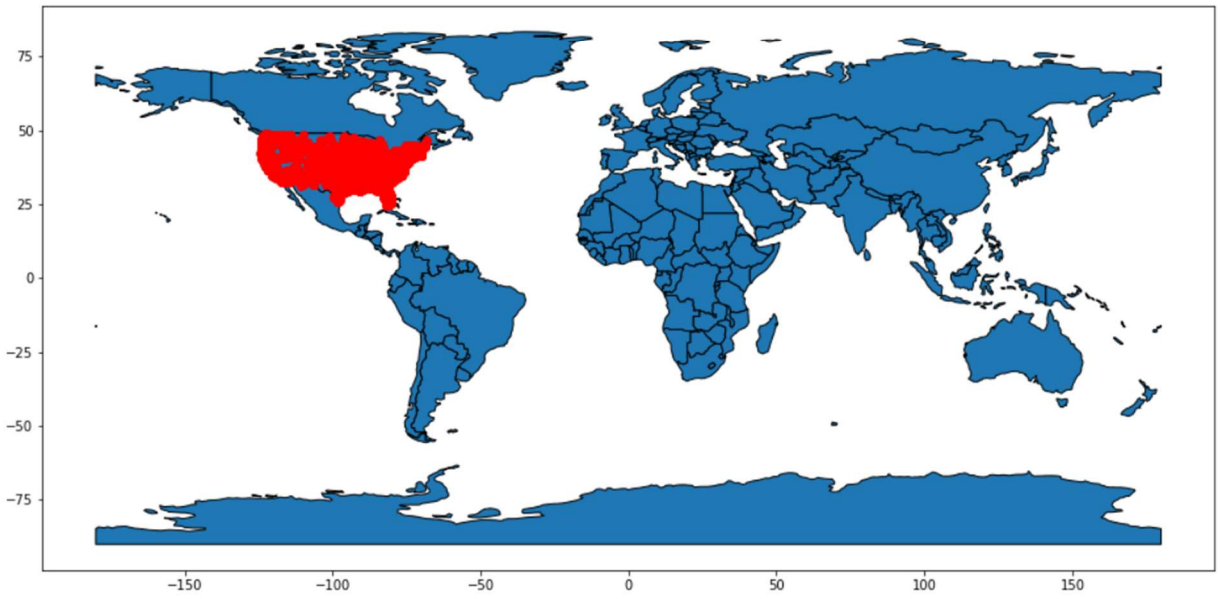

## Prepare Synthetic Data

- Read the dataset carefully
- Identify the delimiters
- Import the data from s3 into rdd
- Apply delimiter
- Change column names
- Convert it into csv file and store in s3 bucket

## Visualizations:

| latitude | longitude | locationID |
|---|---|---|
| 37.77253945 | -77.49954987 | 1.0 |
| 42.09013298 | -87.68915558 | 2.0 |
| 39.56341754 | -75.58753204 | 3.0 |
| 39.45302347 | -87.69374084 | 4.0 |
| 38.9537989 | -77.01656342 | 5.0 |
| 39.90031211 | -75.74486542 | 6.0 |
| 36.24009843 | -115.1586914 | 7.0 |
| 26.11330818 | -80.09202576 | 8.0 |
| 34.27036086 | -118.3162918 | 9.0 |
| 38.81664153 | -97.62573242 | 10.0 |

only showing top 10 rows




## Prepare DBPedia Data:

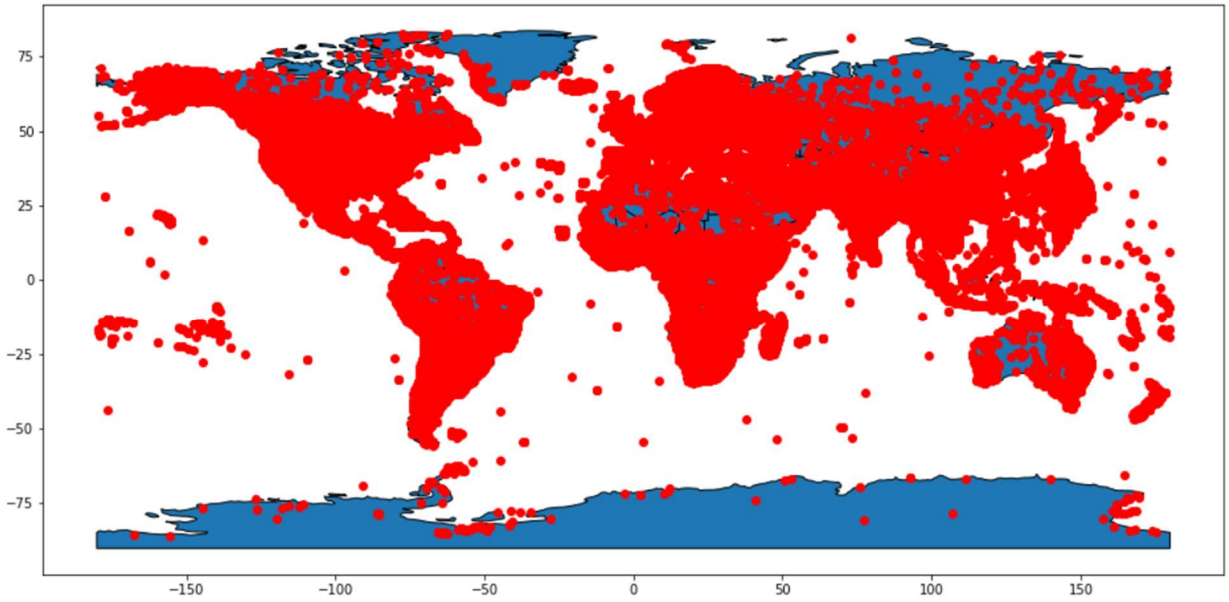
- Read the dataset carefully
- Identify the delimiters
- Import the data from s3 into rdd
- Apply delimiter
- Change column names

- Convert it into csv file and store in s3 bucket

## Visulizations:

| latitude | longitude | page |
|---|---|---|
| 36.7 | 3.216666666666667 | <http://dbpedia.o...> |
| 42.5 | 1.5166666666666666 | <http://dbpedia.o...> |
| 12.516666666666667 | -70.03333333333333 | <http://dbpedia.o...> |
| -8.833333333333334 | 13.333333333333334 | <http://dbpedia.o...> |
| 41.333333333333336 | 19.8 | <http://dbpedia.o...> |
| 34.53333333333333 | 69.13333333333334 | <http://dbpedia.o...> |
| 40.416666666666664 | 49.833333333333336 | <http://dbpedia.o...> |
| 39.93333333333333 | 32.86666666666667 | <http://dbpedia.o...> |
| 52.36666666666667 | 4.9 | <http://dbpedia.o...> |
| 50.46 | 2.13 | <http://dbpedia.o...> |
| 17.116666666666667 | -61.85 | <http://dbpedia.o...> |
| 57.04638888888889 | 9.919166666666667 | <http://dbpedia.o...> |
| 56.15 | 10.216666666666667 | <http://dbpedia.o...> |
| 34.929 | 138.601 | <http://dbpedia.o...> |
| 42.03472222222222 | -93.62 | <http://dbpedia.o...> |
| 33.41972222222222 | 43.3125 | <http://dbpedia.o...> |
| 50.78333333333333 | 6.083333333333333 | <http://dbpedia.o...> |
| 57.1526 | -2.11 | <http://dbpedia.o...> |
| 36.766666666666666 | 3.216666666666667 | <http://dbpedia.o...> |
| -15.518055555555556 | -71.76527777777778 | <http://dbpedia.o...> |

only showing top 20 rows



# IMPLEMENTATION OF KMEANS IN PYSPARK

Spark MLLIB can be used to implement clustering for our data. But the challenge here is our data. Our data is in the form of latitudes and longitudes, there are different kinds of distance metrics are there to measure the distance between them. Two familiar distance metrics are great circle distance and Euclidean distance. The inbuilt spark mllib uses Euclidean distance, but we have to use great circle distance too, for better results and better understanding of data. Spark mllib's Kmeans algorithm doesn't compute distances with great circle. Our requirement is such that we need a implementation which could use great circle distance too. In order to achieve this functionality there are two ways. The first one is transform the spark mllib output in terms of great circle distance. The second build a own custom spark kmeans algorithm from scratch.

It's best create a custom kmeans algorithm using spark RDD's from scratch. Creating your own algorithm makes the algorithm specific to the problem statement and yields better results.

**Custom Kmeans Algorithm:**

**1)Initialization**

- Choose No of Clusters
- Choose Distance Metric
- Choose No of Iterations
- Choose Convergence Distance
- Select Initial centroids

**2)Execution**

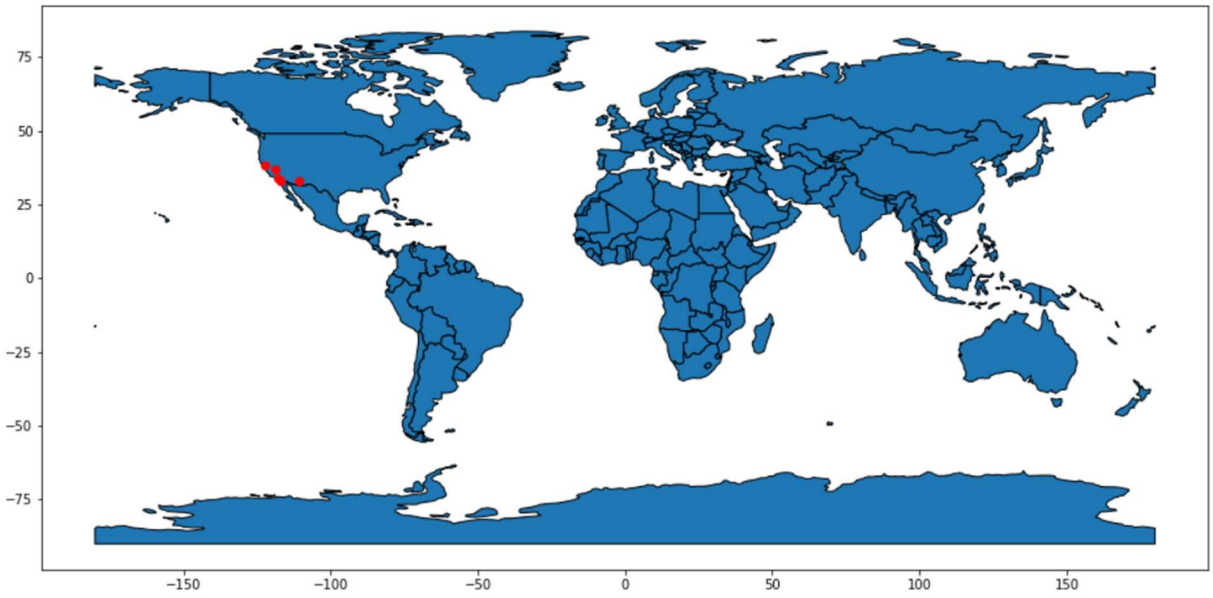
- Compute the difference between each latitude longitude pair to the centroids
- After computing the distance from each pair to the

number of clusters, assign labels to the pairs of longitude and latitude to which centroid they belong by seeing the short distance from the point to the centroid.

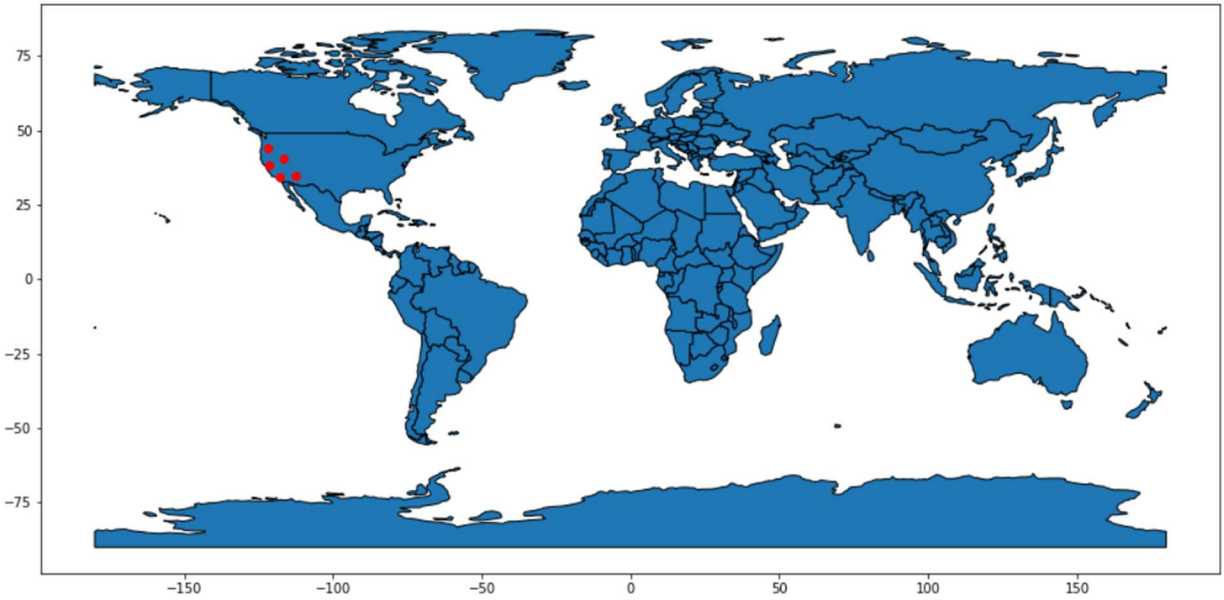
- After assigning Pairs to each of the point in our data, compute new centroids based on the labels we assigned using short distance.
- Compute the distance between new centroids to old centroids if the distance is less that converge rate or if old centroids and new centroids weren't equal or if we haven't done with iterations continue this process until it converges.
- After we have done with this process we will get new centroids for our data. These are the clusters for our data.

**Visualizations:**

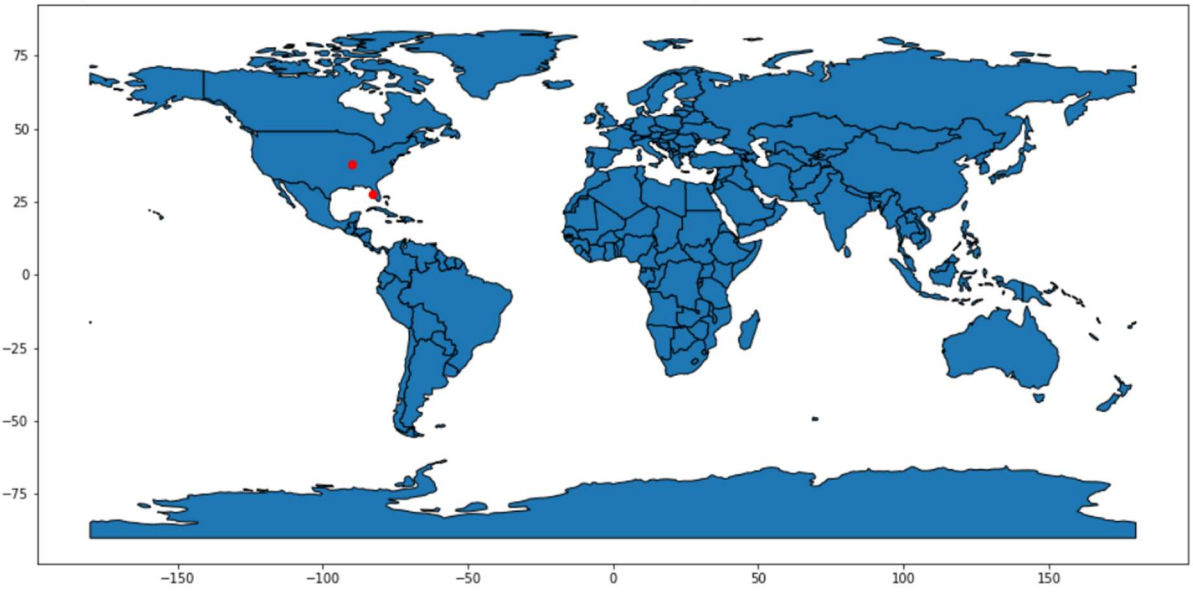
**1) For k=5 , Euclidean distance, DeviceLocation Data**



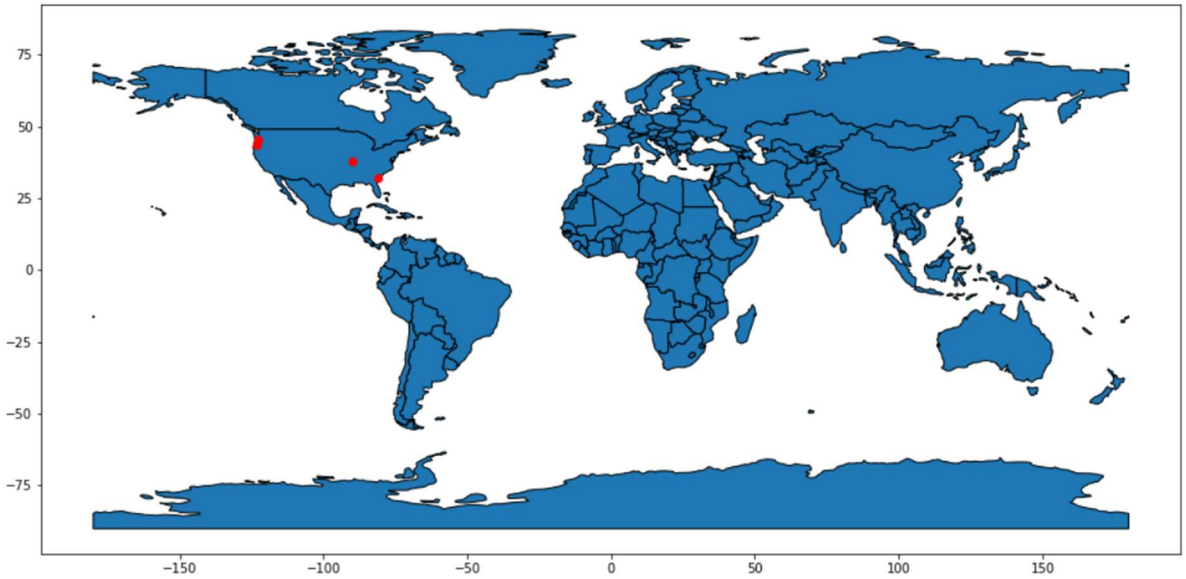
## 2) For k=5, Great Circle distance, Device Location Data



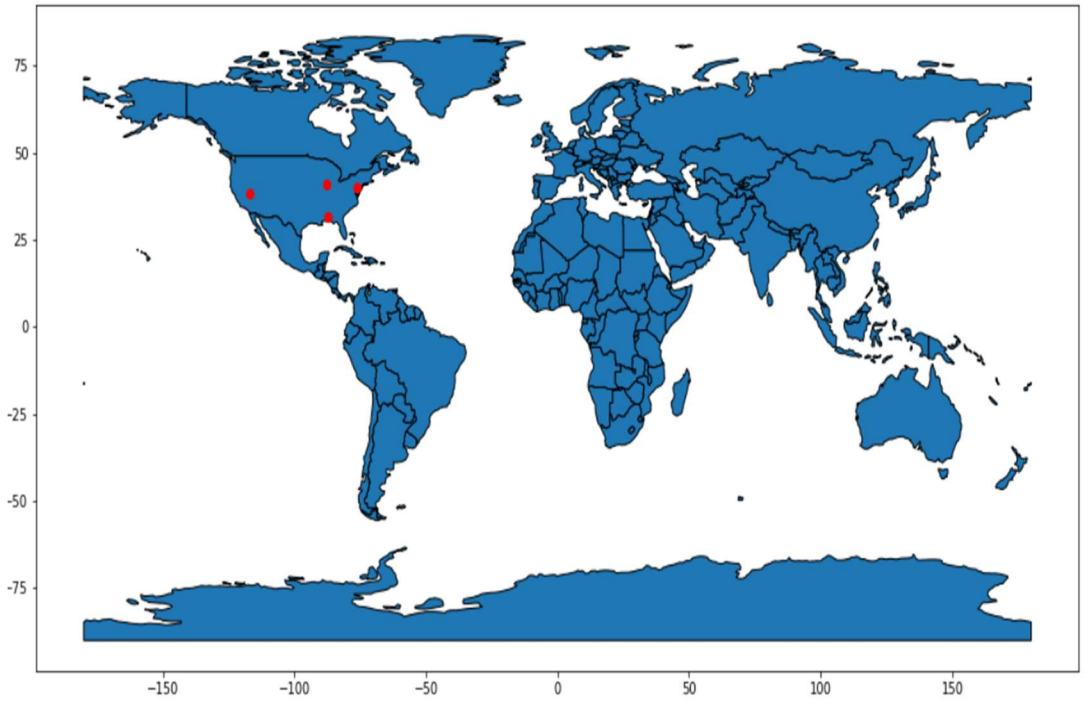

## 3) For k=2 , Euclidean Distance , Synthetic Data



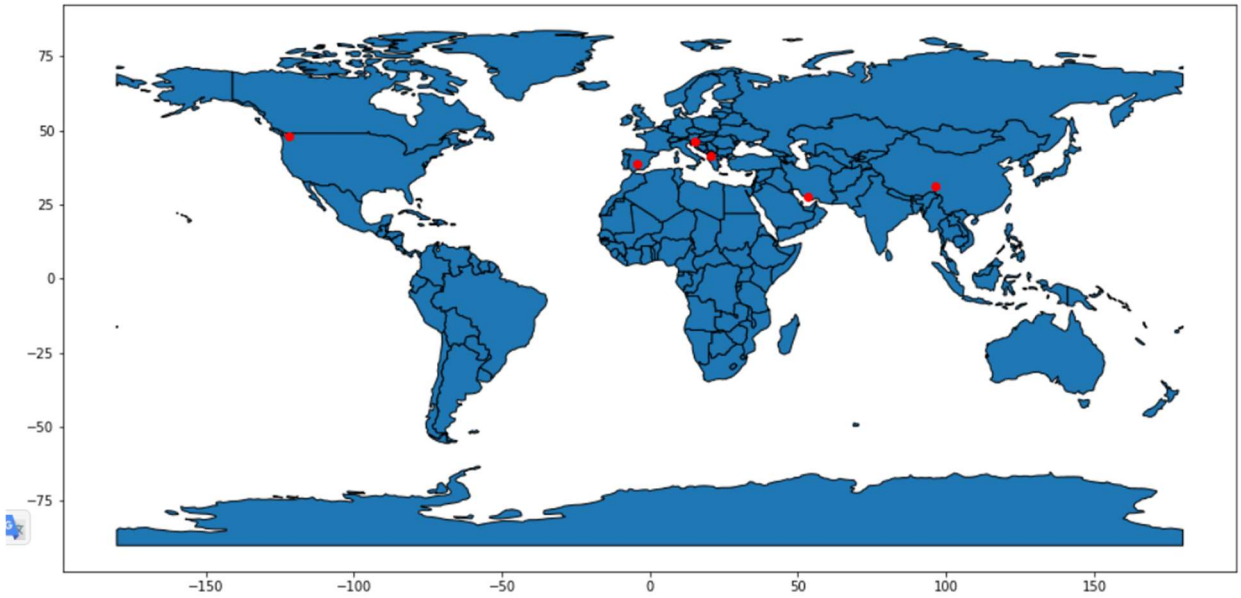
## 4)For k=4, Eucliden Distance, Synthetic Data



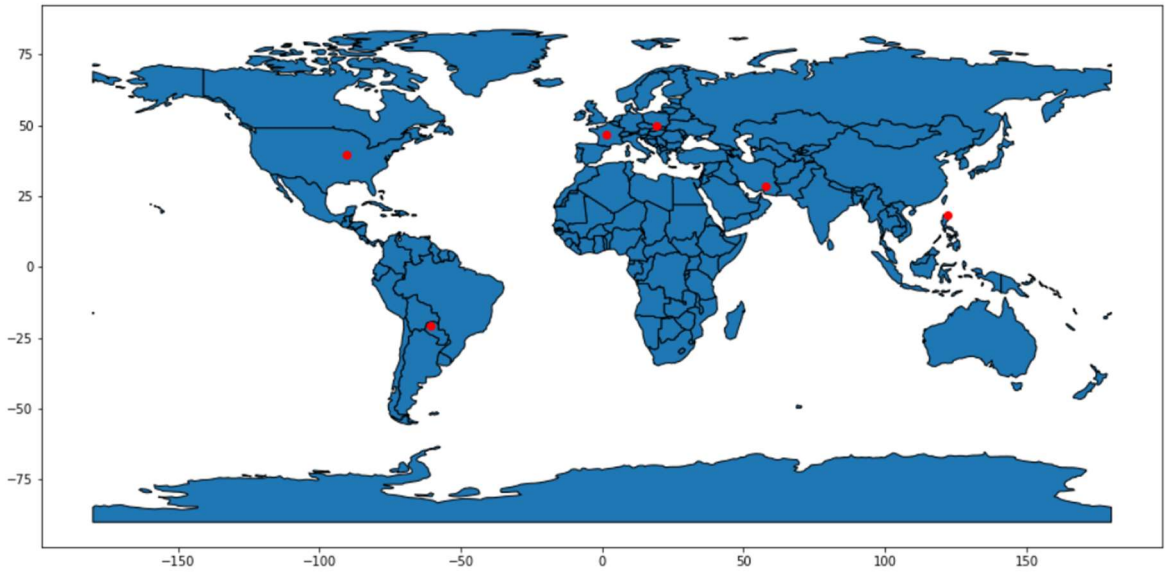

## 5)For k=4, Great Circle, Synthetic Data



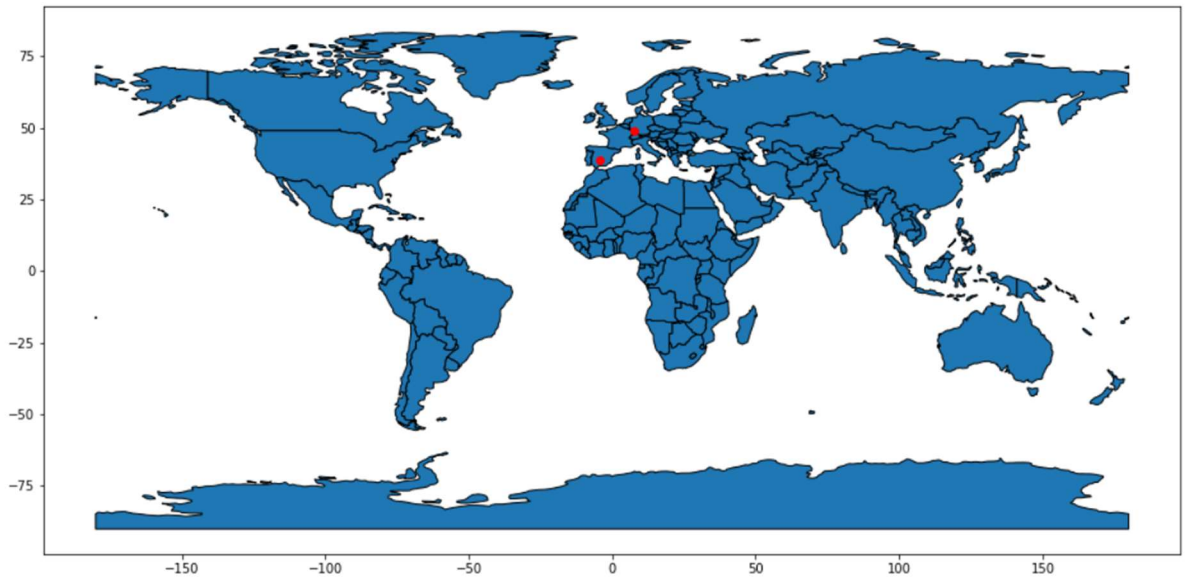
## 6) K=6, Euclidean Distance, DBPedia Data



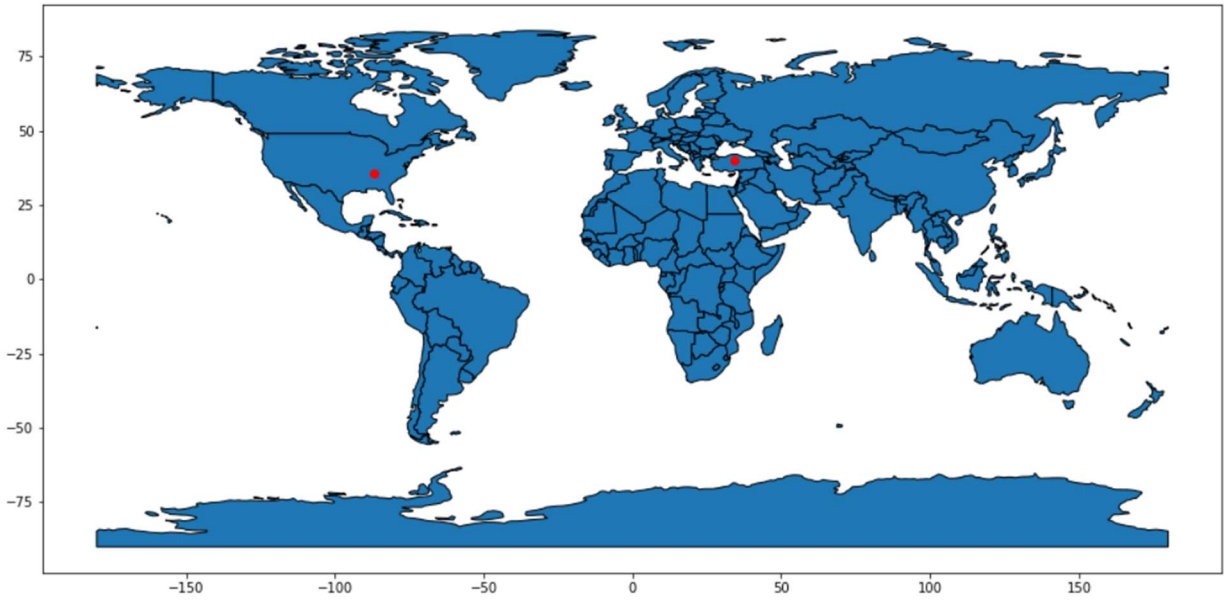

## 7)K=6, Great Circle Distance, DBPedia Data



## 8)K=2, Euclidean Distance, DBPedia Data




## 9)K=2, Great Circle Distance, DBPedia Data



## OBSERVATIONS:

- Euclidean Distance Visualization centroids were not perfect
- Some of the Euclidean Distance Centroids were merged with other centroids which is weird.
- There is not enough Distance between two Euclidean distance centers
- Great circle visualizations were looking great.
- Their clusters were well spaced.
- There's large distance between one centroid to other centroid.

## RUNTIME ANALYSIS:

The best advantage of doing kmeans calculation with aws emr spark is, it shows the run time for every task. To see this run time and history of our tasks open spark history which is running in 18080 port number. For evry task you run on your machine get the id of that and you can find that task runtime in the history server. The spark context stores the id of task. sc.applicationId gives the applicationId and use this Id to find run time of your tasks

**For k=4, Great Circle Distance**

| Threads | Data | Cached | Runtime |
|---|---|---|---|
| 4 | Synthetic | Yes | 1 min |
| 3 | Synthetic | Yes | 1.2 min |
| 2 | Synthetic | No | 1.5 min |
| 4 | DBPedia | Yes | 1.5 min |
| 3 | DBPedia | Yes | 1.7 min |
| 2 | DBPedia | No | 2.7 min |
| 1 | DBPedia | No | 3.2 min |
| 4 | DeviceLocation | Yes | 1 min |
| 3 | DeviceLocation | No | 1.2 min |
| 2 | DeviceLocation | No | 1.7 min |

## Future Enhancements of the model/ Enhancements:

The same model could be used for every other problem which deals with geolocation data. I have identified a huge dataset and applied this kmeans clustering model on it, It did give good results. The dataset I have identified is world cities with latitudes and longitudes. Having cities with latitudes and longitudes we can identify clusters of cities and with good distances between them and develop the near by places on par with those clusters.

**URL**: https://www.kaggle.com/max-mind/world-cities-database

# REFERENCES


- http://mathforum.org/library/drmath/view/51879.html
- https://jonisalonen.com/2014/computing-distance-between-coordinates-can-be-simple-and-fast/
- https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a
- Course era
- Stack Overflow
- Wikipedia

# CONCLUSION

Implementing a machine learning on spark is an amalgamation of two verticals. By Implementing this project I have figured out that Not always Euclidean distance comes to the rescue, we may sometimes have to choose our metrics according to the domain. Solving this in pyspark reduced my time drastically because of rdd's.

Applying the same algorithm on an completely out of context dataset also yielded some fruitful results.

The results for this model were tested multiple times, for every time I never saw any drastic change in results.