Geo-location Clustering using the k-means Algorithm

Motivation:

Geolocation is the identification or estimation of the real-world geographic location of an object, such as a radar source, mobile phone, or Internet-connected computer terminal (Wikipedia). Today's world is a digital world and every person is connected with some kind of devices. With geolocation, we can easily identify the location of a person of device. This can be useful in many ways, government and police uses geolocation to track people. Some companies use it for implementing augmented reality in games such as Pokémon go. If company wants to know the user or website visitor location, it can use geolocation of that user.

Clustering creates a group of data with common features. Geo-location clustering can be useful for analyzing the user data of the company based on user locations and their buying patterns. Based on the geo-location clusters, company can decide marketing strategies, warehouse locations etc., for individual cluster.

Approach:

1) Getting data and Analyzing data:
   There are 3 data files given for this task – DeviceStatus, lat_longs, Sample_geo.
   i) DeviceStatus file contains the information collected from devices of a private network. It has features such as device ID, current status, location etc.
   ii) Lat_longs file has the data of latitude and longitude co-ordinates from the entire world.
   iii) Sample_geo file contains the subset of latitude-longitude co-ordinates

2) Using AWS service – S3 bucket for storing the data files:
   Amazon simple storage service is an object storage service which can accommodate large dataset. It provides scalability, data availability, security and performance.

3) Using AWS service – EMR (Elastic MapReduce) for processing the data:
   Amazon Elastic MapReduce is a web services tool for big data processing and analysis. It supports workloads based on Apache Spark, Presto, Hive, Pig functionality.

4) Preprocessing the data using Pyspark techniques:
   i) Extraction: Data files are uploaded to S3 bucket. I extracted the data files from s3 bucket.
   ii) Transformation: the file extracted from S3 bucket is a RDD object, so in this step I transformed RDD object to spark data frame with appropriate delimiters for each file.
   iii) Selection: As per the requirement of a task, I have selected few features from file. Also for one file, instead of taking the entire world data, I have used only USA data.

5) Creating the clusters for each data file:
   Using K-means clustering algorithm, I have tried to cluster the data of latitude and longitude.

6) Visualizing the clusters using Basemap:
   Basemap is a python library which is mostly used for visualizing latitude-longitude for particular region or entire world. I have used same library for visualizing my analysis.

System Configuration:

Below are the steps I performed as a part of system configuration

1) **Create EC2 instance:**
   (1) Login to AWS educate account and open EC2 dashboard
   (2) Click on 'Launch Instance' and select 'Ubuntu 18.04' AMI
   (3) Click 'Review and Launch' button
   (4) When pop up appears- click on create new key-pair file and give name of key-pair file and download it.
   (5) Launch the instance
   (6) For this assignment I have used port# 8888 so in security groups of EC2 instance, I have added this port

2) **Create S3 bucket:**
   (1) Open AWS S3 service console
   (2) Click on create bucket link
   (3) Give the name of bucket and hit 'Create' button

3) **Setup and configure EMR cluster:**
   (1) Open EMR console and click on create cluster option
   (2) Give the name of cluster

(3) In Application section, select ‘Spark’ instead of ‘Hadoop’
(4) Select ‘m4.xlarge’ for instance type
(5) Give EC2 key-pair name which was created earlier and click create option
(6) In EC2 instance dashboard, when Hadoop master and slave instances are in running state, edit Security rules for mater EC2 instance to add port# 8888

4) **Opening a pyspark jupyter notebook in a browser:**
   1) SSH into a Hadoop master node
   2) Install all required packages using pip (I have installed boto3, ipyparallel, jupyter, basemap, pyyaml, pandas and ipython)
   3) Set the pyspark driver path in ‘.bashrc’ file
   4) In terminal, give command ‘pyspark’
   5) Open a notebook using IPV4 address of master node and token generated in terminal

Big data application/dataset:

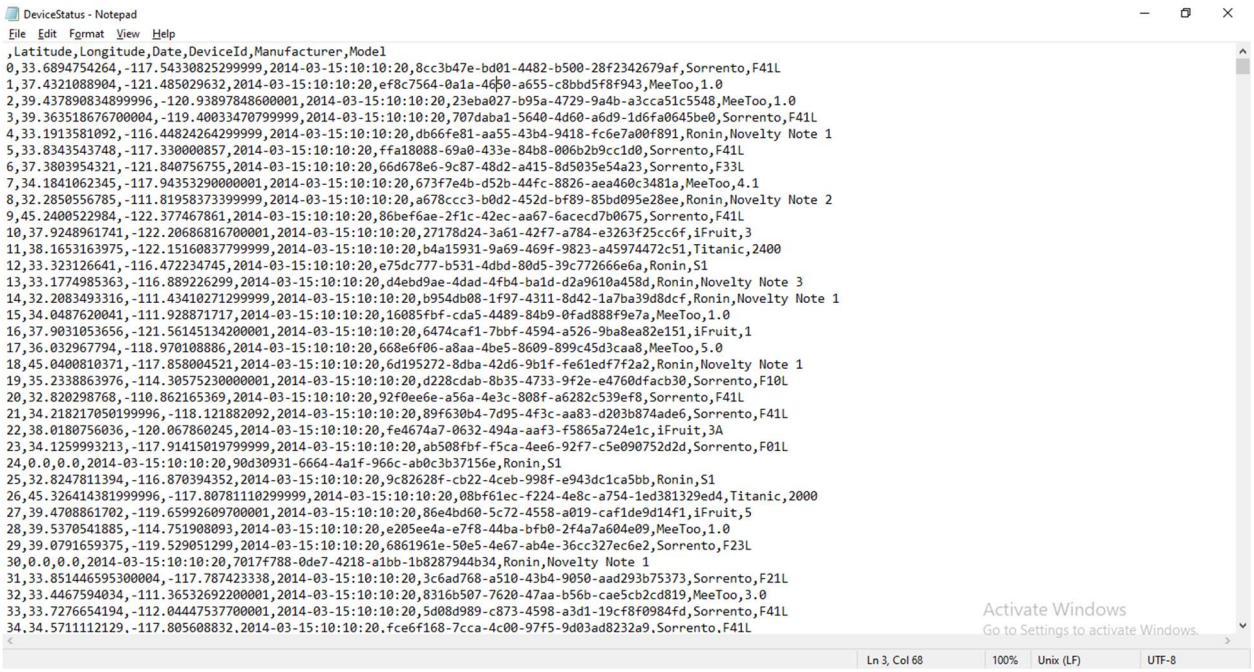
1) **Description of data processing**
   (1) For this task 3 files are given. I have uploaded all 3 files to S3 bucket. Then using boto3, I am reading each file from bucket to spark dataframe.

```
In [8]: s3 = boto3.client('s3',aws_access_key_id='ASIA6DHH57K7XGORCC4N',aws_secret_access_key='sKcdUKdqMNQVpnjKGPnEmR66Fps73hWCm92fj2Ig'
        bucket = 'finalprojectdsde'
        key = 'devicestatus.txt'

        obj = s3.get_object(Bucket=bucket, Key=key)
        #df = pd.read_csv(obj['Body'], sep=r'[\|\/\,]', header=None, engine='python')

In [9]: spark = SparkSession \
            .builder \
            .appName("Python Spark SQL basic example") \
            .config("spark.some.config.option", "some-value") \
            .getOrCreate()

In [5]: print(sqlContext)

        <pyspark.sql.context.SQLContext object at 0x7f7ab70c2490>

In [10]: body = obj['Body']
         csv_file = body.read().decode('utf-8')

         #df = spark.read.csv(StringIO(csv_file),header=None, sep=r'[\|\/\,]')

In [11]: df = pd.read_csv(StringIO(csv_file),header=None, sep=r'[\|\/\,]',  engine='python')

In [12]: df1 = spark.createDataFrame(df)
```


   (2) After reading file into spark dataframe, I have dropped and renamed columns. And also filtered out the data with latitude and longitude values as 0.0

(3) Then I have performed few more preprocessing steps like rearranging dataframe columns, splitting the columns etc.
(4) As a final step of data processing, I have uploaded the processed dataframe as a CSV file back to the S3 bucket.

```
In [20]: pandas_df = df3.select("*").toPandas()

In [19]: file_name = "MobileNetData_spark.csv"
         df3.to_csv(file_name)
         s3.upload_file(file_name, 'finalprojectdsde', 'devicedata1')
```



2) Implementation of k-means algorithm:

(1) For this task, I am reading preprocessed file from S3 bucket in pyspark
(2) For calculating distance of point from centroids, I have used Euclidean distance and Great Circle distance
(3) The input parameters – distance function and number of clusters are taken from Command line

```
take_input = raw_input("Enter The Distance function to use and number of clusters : \n")
take_input = take_input.split(" ")

if take_input[0] == "eu":
    dist_funct = EuclideanDistance
elif take_input[0] == "gc":
    dist_funct = GreatCircleDistance
else:
    print("wrong function shoule be 'eu' or 'gc'")
    exit(-1)

centers, clusters = kmeans(filtered_data, dist_funct, int(take_input[1]))
```

(4) As a part of k-means cluster algorithm, I am arbitrarily choosing k centers. Then using method given as input from CLI, I am calculating distance of each point with center and creating clusters

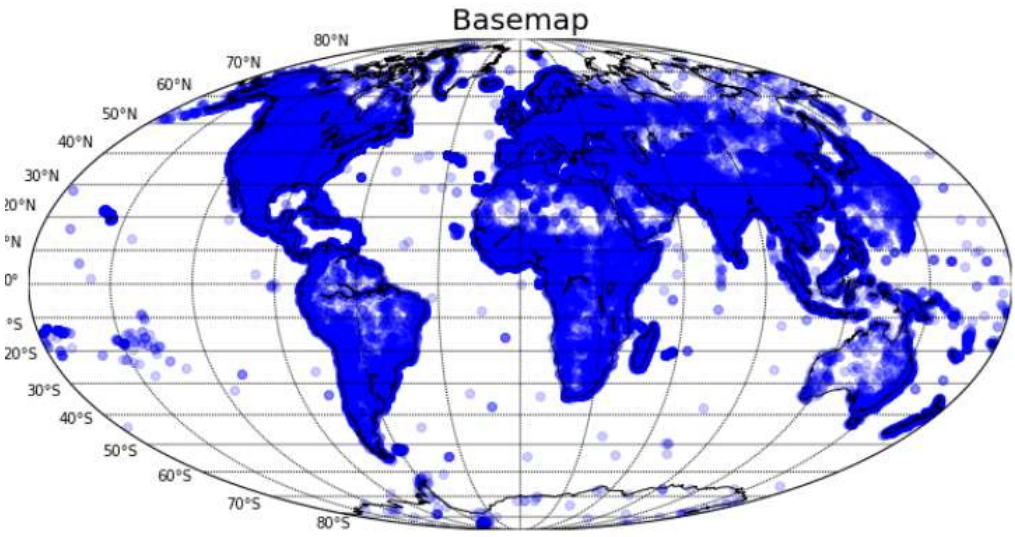
(5) In next step, I am recursively updating center points and clusters. I am repeating this process until the difference between old center point and new center point is not as small as 0.01 (convergence criterion)

```
In [34]: def kmeans(filtered_data, dist_funct, k):
             clusters = []
             centers = filtered_data.takeSample(False, k)
             diff = 1.0
             conv_criteria = 0.01
             kMean_df = pd.DataFrame(columns = ['Latitude' , 'Longitude', 'cluster_id' ])

             while diff > conv_criteria:
                 cluster = filtered_data.map(lambda p: (closestPoint(p, centers, dist_funct), [p, 1]))
                 newCenters = cluster.reduceByKey(lambda p1,p2: (addPoints(p1[0],p2[0]), p1[1]+p2[1])).map(lambda l: (l[0], np.array(l[1]
                 diff = getDiff(newCenters, centers)
                 for (i, newCt) in newCenters:
                     centers[i] = newCt

             print(centers)
             return centers, kMean_df
```

3) Visualizing Result of k-means:

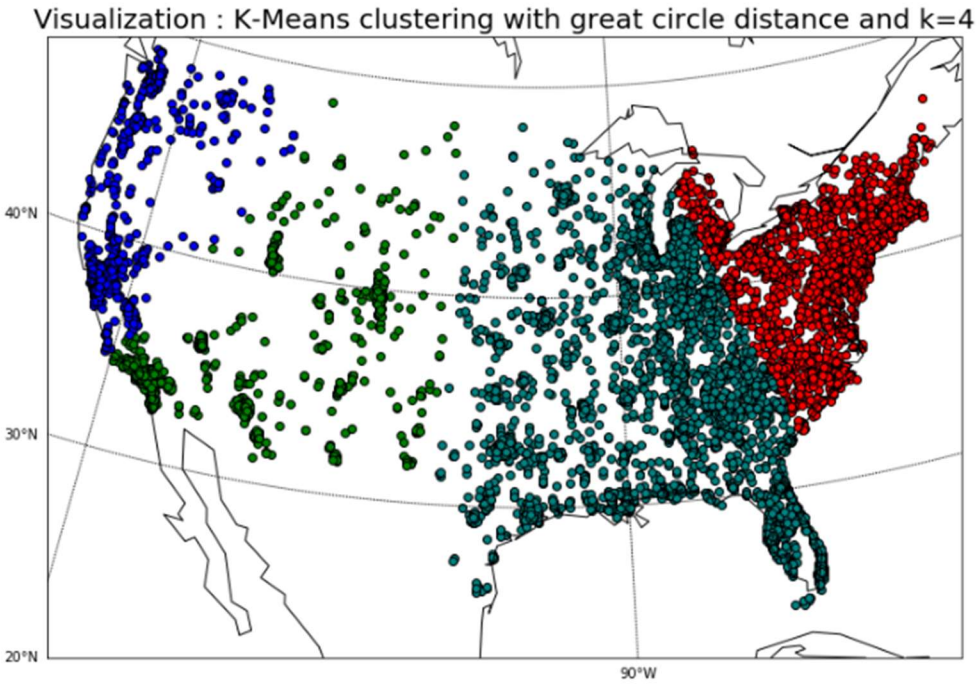
For visualizing all the graphs, I have used basemap library.

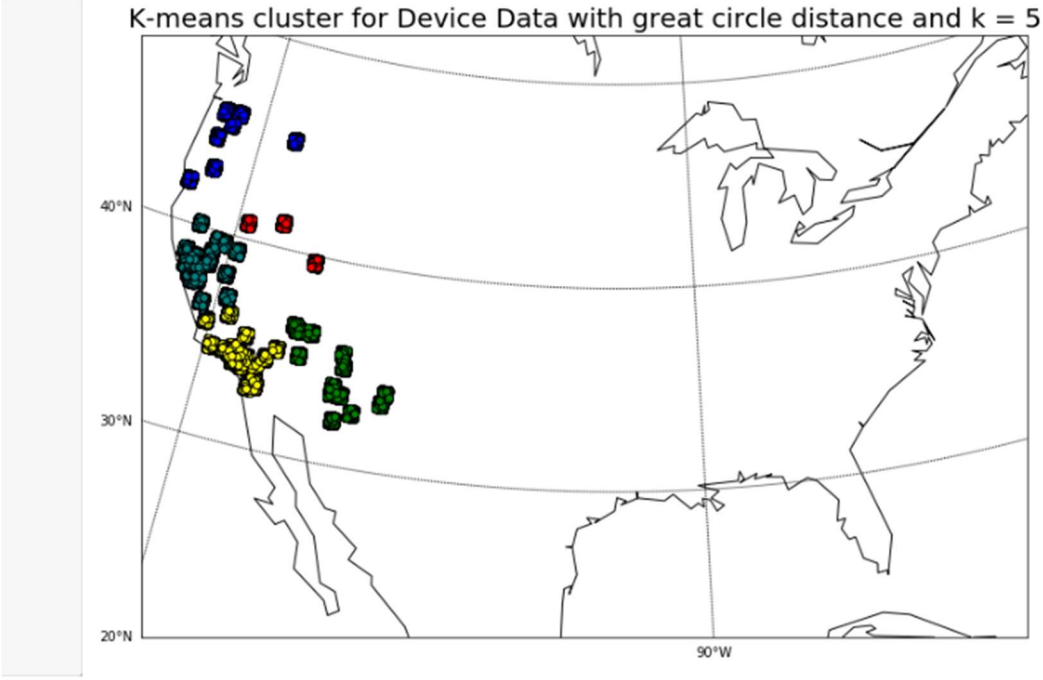
1) Visualization of latitude-longitude data before implementing k-means clusters:



This plot is plotted using all the data of entire world latitude-longitude coordinates.

2) Visualization of data after implementing k-means clusters:





Conclusion or Future Work:

The latitude-longitude dataset was big enough but with AWS spark, it was easy to do the processing. The process of creating clusters using parallelism was less time consuming.

I have done clustering only for US data. But it can be done on even larger data and data of multiple companies instead of only one company.