Geo-Location Clustering in Spark

1 Introduction and Objectives

1.1 Introduction to Spark

Spark is a fast and general cluster computing system for Big Data. It provides high-level APIs in Scala, Java, Python, and R, and performs as an optimized engine that supports general computation graphs for data analysis. It also supports a rich set of higher-level tools including Spark SQL for SQL and DataFrames, MLlib for machine learning, GraphX for graph processing, and Spark Streaming for stream processing.

Spark completes the full data analytics operations in-memory and in near real-time. It overcomes many limitations of Hadoop MapReduce: Spark has more flexible framework, has higher-level abstraction, supports multiple programming languages, can keep the data in memory, provides interactive shell, and is faster than MapReduce. RDD, Resilient Distributed Datasets, is a fault-tolerant, parallel data structure, and the basic abstraction in Spark. RDD is a relatively flexible structure with a lot of functions to operate data.

1.2 Introduction to k-Means Clustering

k-means is a general leaning algorithm of clustering. It assigns the data points to several centers according to the "distance" or some similarity measurements, and in this geo-location project, geographical distance is the intuitive similarity measurement. The closer two data points are, the more similar; the farther away they are, the more different. So in this geo-location k-means clustering, no matter we use Euclidean Distance or Great Circle Distance, we can get the relatively same result (when the starting centers are identical).

The main idea of k-means is to put all points into k clusters, and each point belongs to a cluster that corresponds to the nearest center. This algorithm is iterative, in that we need to re-calculate the new centers and re-assign the points to the new centers many times until there is no big difference of the change of the centers. In this way, k-means clustering can help find the similar objects and has many useful applications, such as customer clustering, document grouping, spatial data analyzing and so on.

1.3 Project Motivation

For this project, geo-location clustering in spark, we use Spark as the framework to implement k-means. Our motivation for this project are stated below:

(1) Get familiar with Spark. Spark is a useful tool in cloud computing and big data analysis. It would be extremely helpful to know Spark much better by implementing a algorithm to solve a practical problem than finishing labs in the class. As stated above, Spark has lots of advantages and is widely used for fast processing of big data, not limited to interactive queries and data analysis,

but also in graph analyzation and machine learning. Being familiar with Spark will be a basic and important skill for students interested in big data.

(2) Get familiar with k-means and data visualization. k-means is very useful in grouping similar objects and is widely used in marketing, social media analysis, document grouping and so on. This will be a very basic and useful skill. Data visualization, as we used in this project, is another important skill for students interested in big data because visualization will give the most direct feeling and the most intuitive understanding of the results. It is better in presentation of data analysis. Also, the meaning of the results of k-means clustering is important because data interpretation will lead us to better understanding of our datasets and guide us for further analysis and explanations.

(3) This project can be extended to some very interesting and useful analysis. With functionality of stream processing, Spark can be helpful to analyze stream data like Twitter and Foursquare. In stream data analysis, location clustering is useful way to compute the geo-location hot spot. More importantly, we can use these data to monitor what is happening around the world, to give much better event report (or prediction), safety reminder, advertisement, and so on.

2 Implementation

2.1 Data Pre-Processing

There are three datasets used in this project:

- **Device Location Data** is the information collected from mobile devices on Loudacre's network, including device ID, current status, location and so on. This data records have different formats in that they use different delimiters to separate the fields. These data only contain the location information of western USA.
- **Synthetic Location Data** is the dataset of the whole USA, and it only contains the latitude, longitude and location ID. The records are tab delimited.
- **DBpedia Location Data** is the large-scale clustering data extracted from DB-pedia. Each record represents a location or place with a Wikipedia link and latitude, longitude information. The records are space delimited.

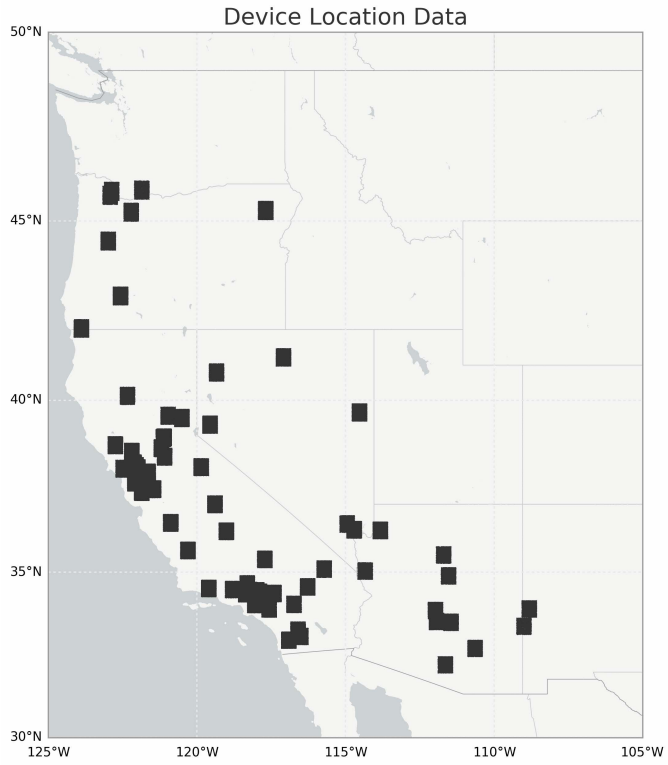
We did not perform any pre-processing on the Synthetic Location Data and DBpedia Location data, but we did pre-processing on the Device Location Data. Each record in this dataset is read and separated by its own delimiter; the bad records (incomplete records or records that have latitude and longitude of 0) are filtered out, and only these information are stored: latitude, longitude, date, device manufacturer, model name and device ID. The output file is saved in comma delimited text files.

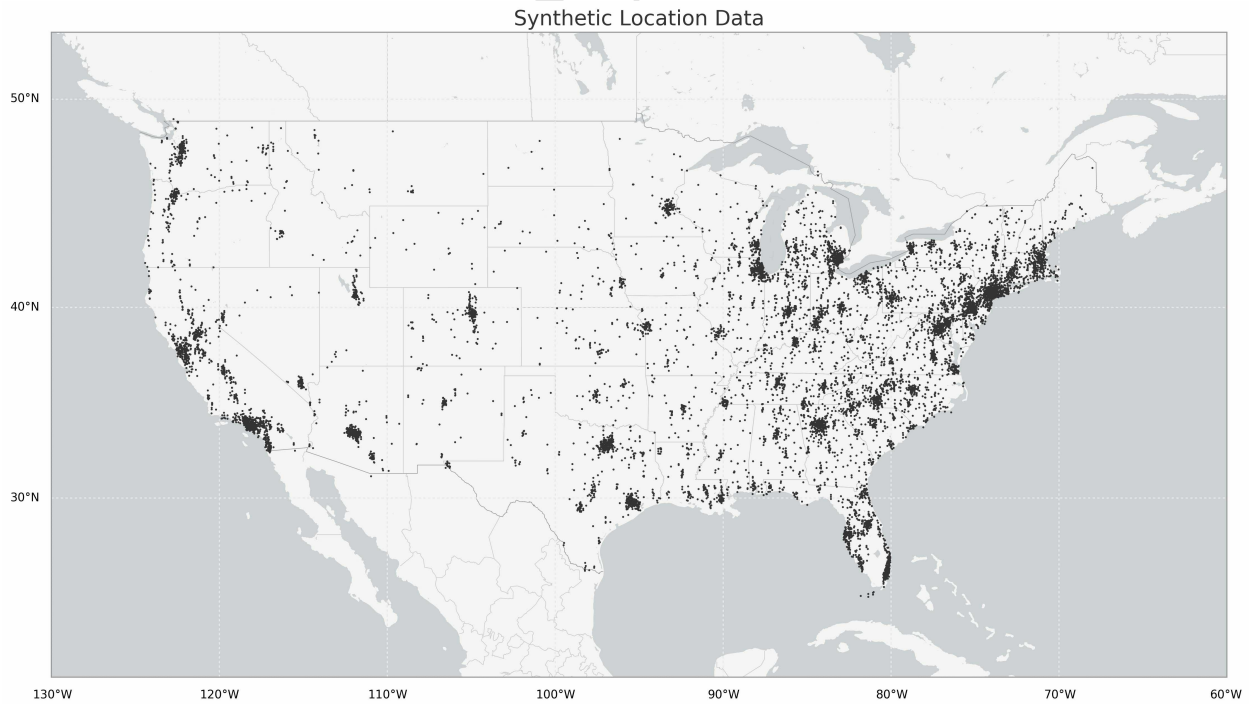
The number of records contained in each file:

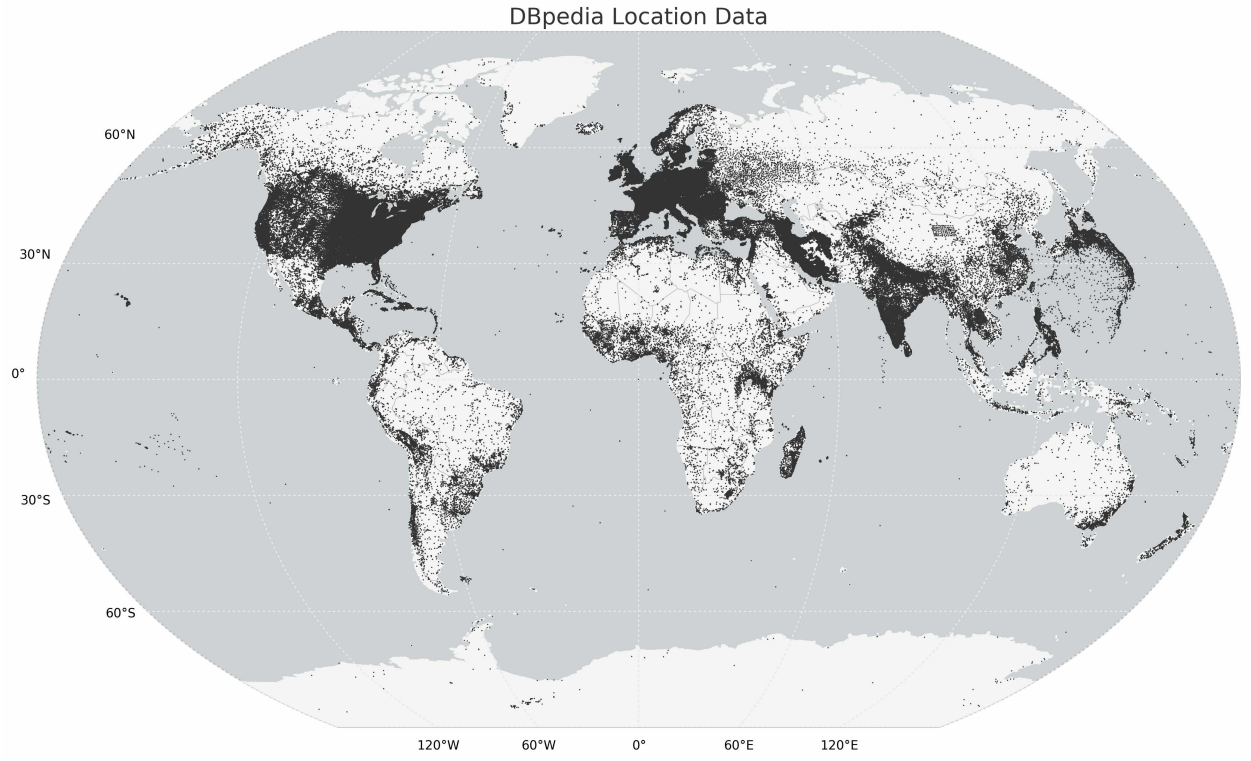
- Device Location Data: 431,857
- Synthetic Location Data: 9,970

- DBpedia Location Data: 450,151

The visualization of the three datasets:







Note that all the **data visualization original files**, with or without clustering, can be found in the `drawmap` folder, together with codes generating these maps.

2.2 Coordinate Representation

The latitude/longitude representation is spherical coordinate system in the three-dimensional space. In this system, radius r represents the radius of the earth (which we used the average radius of the earth, 6371km), inclination θ represents the latitude, and azimuth φ represents the longitude. However, we can also use the Cartesian coordinate system to represent the points.

To convert from spherical coordinates to Cartesian coordinates, we can use the following equations:

$$x = r\ \sin\theta\ \cos\varphi$$

$$y = r\ \sin\theta\ \sin\varphi$$

$$z = r\ \cos\theta$$

To convert from Cartesian coordinates to spherical coordinates, we can use the following equations:

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\theta = \arccos\frac{z}{\sqrt{x^2 + y^2 + z^2}} = \arccos\frac{z}{r}$$

$$\varphi = \arctan\frac{y}{x}$$

2.3 Distance Measurement

We used two kinds of distance measurement methods in this project.

2.3.1 Euclidean Distance

Euclidean distance is the straight-line distance between two points in Euclidean space. In this project, the coordinates should be first converted to Cartesian coordinates in order to calculate the Euclidean distance using the equation below:

$$d(p1, p2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

The Euclidean distance is like the "tunnel distance" between two points in our project.

2.3.2 Great Circle Distance

The great circle distance is the shortest distance between two points on the surface of a sphere. The distance between two points in Euclidean space is the length of a straight line between them, but on the sphere there are no straight lines. We calculate the central angle using the haversine formula:

$$\Delta\sigma = 2 \arcsin \sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cdot \cos\phi_2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)}$$

Where ϕ_1,λ_1 and ϕ_2,λ_2 are the geographical latitude and longitude of two points 1 and 2; $\Delta\phi$ and $\Delta\lambda$ are their absolute differences; $\Delta\sigma$ is the central angle between them.

Then, the great circle distance would be:

$$d = r\,\Delta\sigma$$

2.4 Calculating the Centers

The initial centers are picked randomly with a fixed random seed, in order to generate identical initial centers for a certain dataset, which will be convenient to compare the two distance measurements. The seed can be changed in order to test the performance when choosing different initial centers.

Using the distance measurement specified by the user (either Euclidean or great circle), each points is assigned to its nearest center. The new center will be calculated as the average point of all the points assigned to one center.

To do this, we first used the Cartesian coordinates instead of the latitude/longitude coordinates. Then, all the points of each center are added together, by means of adding x, y and z coordinates respectively. After summing all the points of one center, we can use the formula specified in 2.2 to calculate latitude and longitude back. Note that we do not need to normalize the Cartesian coordinate because the angles will not be affected by the length of the vector.

2.5 Implementation of k-Means

In this program, the input file will be first parsed into the (latitude,longitude) pairs. Then, the Cartesian coordinate of each point will be calculated, generating ((latitude,longitude), (x,y,z)) pairs. This RDD will be persisted because both spherical coordinates and Cartesian coordinates will be used in the following k-means implementations.

The k-means algorithm is implemented using the following schema:

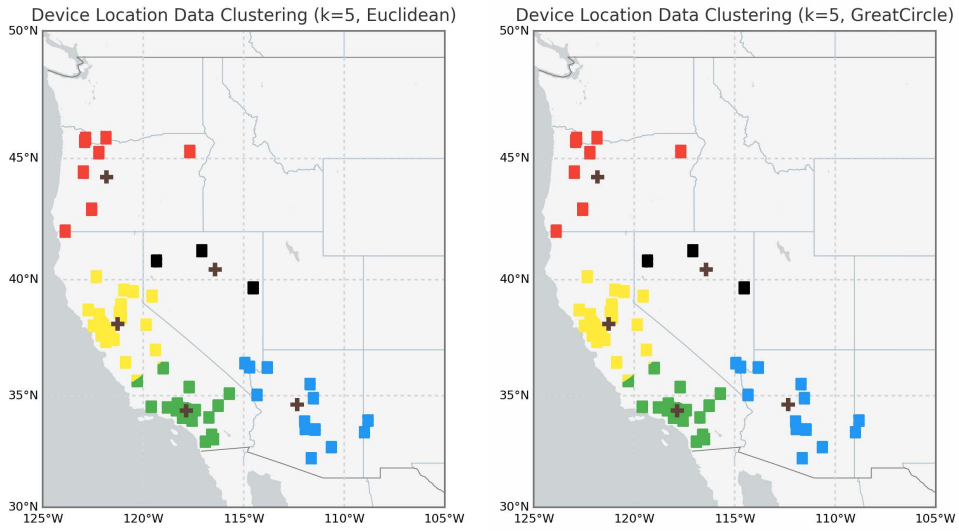
1. Randomly pick k points as start centers
2. Calculate distance between each point to each center, and return the nearest center, using the distance measurement specified by the user
3. Sum all points of each center and calculate the average of all points, which is the new center
4. Calculate the average distance difference between the old centers and the new centers, using the distance measurement specified by the user; if the mean distance difference is larger than converge distance (which is 0.1 in this project), repeat 2-4
5. When the iteration is complete, store all data points and the cluster information, together with all final k centers (the format will be specified below)

The output format of the program will be comma delimited files with records (latitude, longitude, cluster number). The cluster number will be from 0 to $k-1$ for all data points, and will be -1 for all the final centers. This design is for convenience of data visualization.

3 Result and Analysis

3.1 Device Location Data, k=5

The results are shown below:



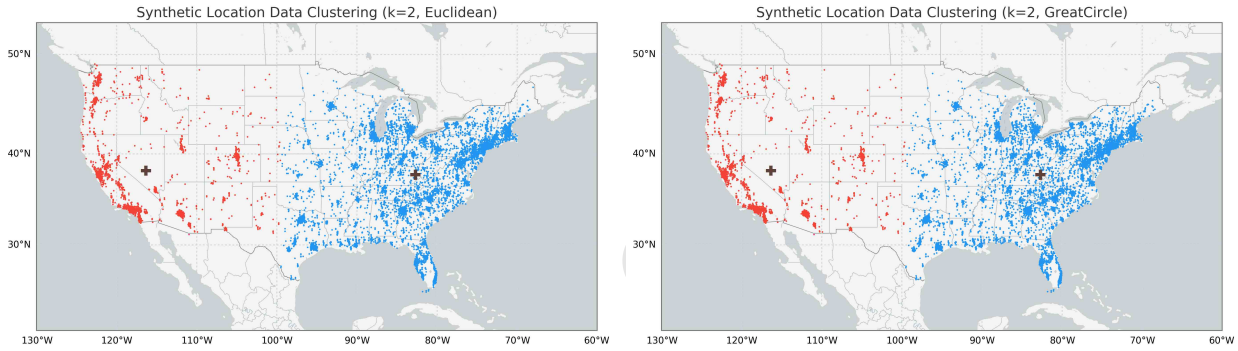
Using device location data with $k = 5$, it seems that the results using Euclidean and GreatCircle are identical. When using `diff` in the terminal the compare the output files, we found that they are totally identical.

The results are meaningful for gathering information about the mobile device distributions. It can guide the company to determine the best locations for customer service to cover as many customers as possible.

3.2 Synthetic Location Data, k=2 or k=4

3.2.1 Synthetic Location Data, k=2

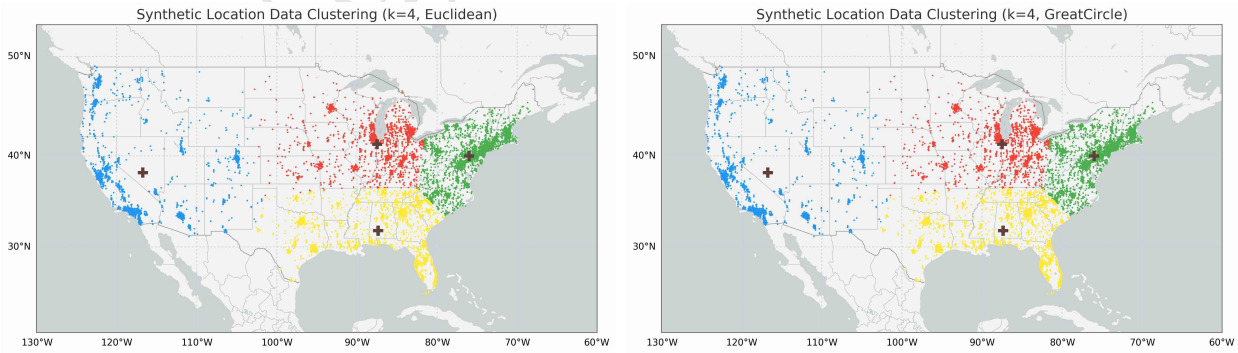
The results are shown below:



For $k = 2$, using Euclidean and GreatCircle does the similar job to geo-location clustering. The cluster centers spread at the east and west closer to the areas having a relatively high density. In the plot, the red cluster centers at State of Nevada, and the second cluster center is at Commonwealth of Kentucky. The result of cluster centers is not at a point where it has a true value, or a density center. Apparently, in the red cluster, the west seaside has more geo-points and has high density. However, the cluster center is not right in there but very near.

3.2.2 Synthetic Location Data, k=4

The results are shown below:



For $k = 4$, using Euclidean and GreatCircle also does the same job to geo-location clustering. The cluster centers spread differently from when $k = 2$. The cluster center at west does not change

dramatically, and it is still in State of Nevada, but the rest three cluster centers are like sub-centers of the east center when $k = 2$. We can see USA are divided into four parts, and the west part is similar when $k = 2$, and the east part is divided into another three parts, each of them has an independent center, but we can still see the three centers' geo-average center is near the east center when $k = 2$. That is because there are more points in east USA than in west USA, so when k-means is processing, the starting points will be more likely to locate in the east part of USA.

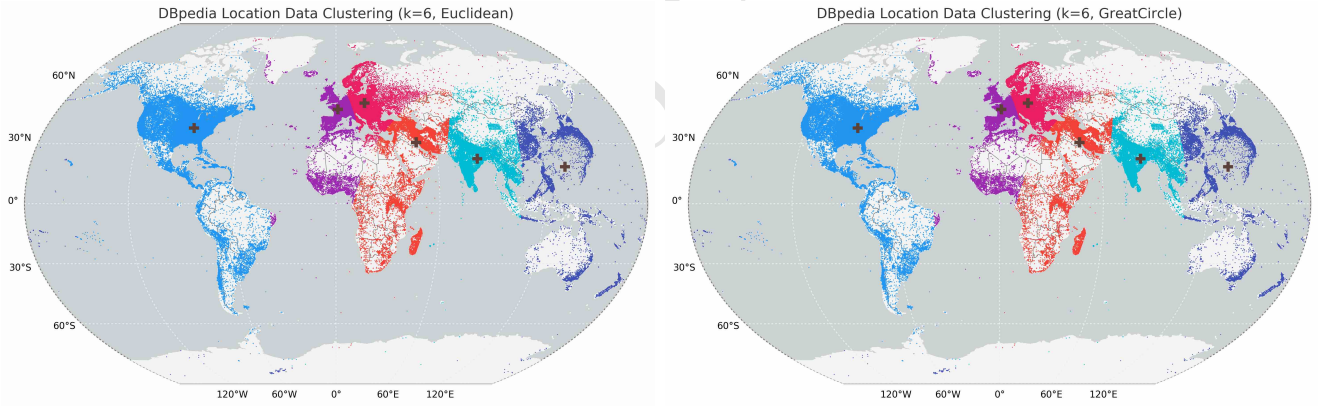
The results for $k = 2$ split the USA into east part and west part, it is a reasonable spliting because east coast and west coast contains the most of the populations. For $k = 4$, it is also reasonable because the mid-west is sparse in points, and the population there is not large. The lake area and the south area have reasonably big populations, which are reasonable to become centers.

3.3 DBpedia Location Data

As we try to determine k, we will try different number of clusters and compare those results. We did not use the US-only data but the whole dataset, and the whole world will be clustered.

3.3.1 DBpedia Location Data, k=6

The results are shown below:

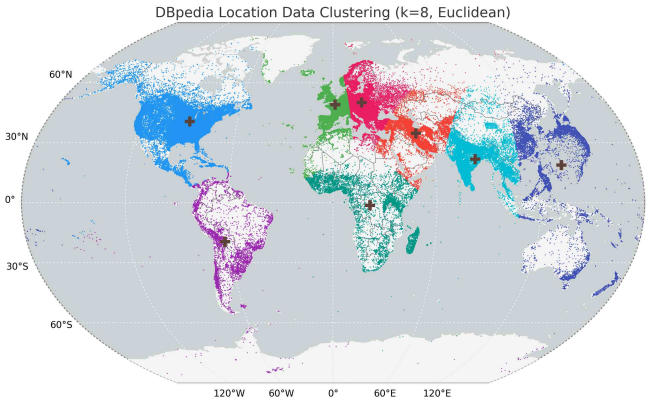


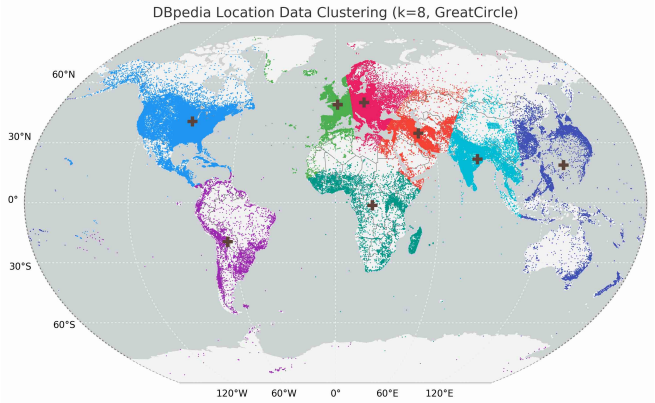
When $k = 6$, all the data points are divided into six clusters. According to the result we have above, we can get: a) North America and most of South America are in the same cluster (in blue); b) Western Europe and northwestern Africa are the main parts of cluster in purple; c) Eastern Europe and parts of Asian are the cluster in magenta; d) Most southeastern Africa is the cluster in red; e) China is split into two parts; western part and part of Asian are in a cluster in lake blue; f) Eastern Asia and Australia are the last cluster in dark blue.

This is a reasonable clustering result, however not perfect.

3.3.2 DBpedia Location Data, k=8

The results are shown below:



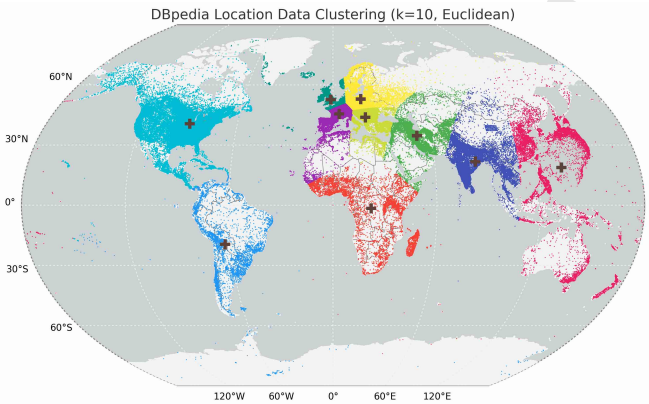


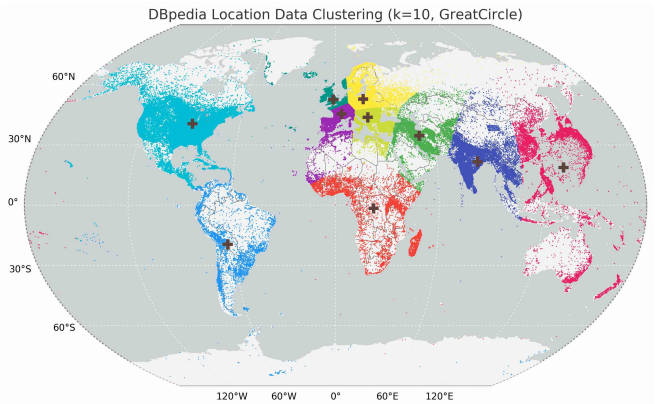
When $k = 8$, geo-location data points are further divided into 8 clusters, besides what we see in when $k = 6$, the North America and South America are in two clusters; eastern Europe, northeastern Africa and western Asia are further divided.

This is a more reasonable clustering result in that North and South America both contain a center; Africa is reasonably distributed; the middle east is also reasonable; however the results in East Asia and Australia is not perfect because the initial points are not chosen here. There are many points around Japan but not many points in China and Australia, leading to the random pick process to be biased.

3.3.3 DBpedia Location Data, k=10

The results are shown below:



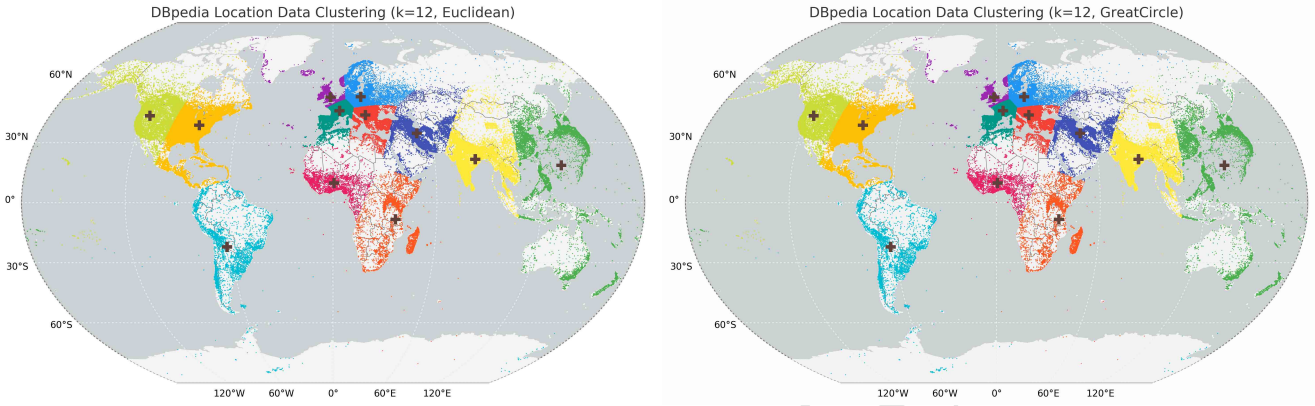


When $k = 10$, there are 10 clusters. Besides what we get when $k = 8$, both eastern and western Europe are further divided into two clusters.

Both two new centers are added to Europe; this is not what is expected.

3.3.4 DBpedia Location Data, k=12

The results are shown below:



When $k = 12$, North America is divided into 2 clusters; Europe is further divided.

The clustering becomes more strange and unreasonable. $k = 8$ gives the most reasonable result up to now.

The results are meaningful in that it can show the distribution of locations in Wikipedia pages. It will give information of the density and classification of current Wikipedia location pages and can give suggestions when building new pages.

3.4 Analysis of Results

1. Results of geo-location k-means based on distance between points reflect that the degree of aggregation from the aspect of geo-location. The closer two points are, the more similar they are, the more probable they are in the same cluster, which is also the first definition of geography.

2. In this project, we only use distance as the only measurement of clustering, without considering border, leading to the split of countries and continents in a cluster result.

3. Using distance as the measurement, the cluster centers are always close to the density center, it is because we take the average summation as the new centers, and where there are more data points, where contributes more on cluster centers.

4. The sampling of the initial centers will be more likely to locate in the places with a larger density of data points. Because choosing the initial centers is random, the places with more points will be more likely to have initial centers. In this way, the clustering result is sensitive to initial condition. Different initial grouping may lead to different results; this is a drawback of k-means algorithm. We will try to prove this in the following section.

5. When $k = 8$, k-means clusters for the DBpedia location data works best. Because it does not take North America and South America as one cluster, and also the clusters in Europe works better. Since there is no magic function to determine k, it is sometimes difficult to determine k before hand. This is another drawback of k-means algorithm.

6. The result is distance-based, therefore the results will be similar to a circular shape. We cannot perform other shapes of clustering using this method.

7. This algorithm may not be very robust in that the point in a cluster which is very far away from the cluster center may be able to shift the cluster center away from its reasonable place.

Despite some drawbacks, k-means clustering algorithm is still a simple and straightforward algorithm which is very easy to implement. It will give reasonable results under most cases, although sometimes not very robust and repeatable. k-medians and k-medoid algorithms are useful substitutions of k-means algorithm and will perform better; k-means++ algorithm can also help to choose the starting points for k-means, which is a very good supplement. Gaussian mixture model will overcome more drawbacks of k-means, in that it can evaluate each attribute of a data point to assign different weights to these attributes, and can overcome the circular shape of the clusters. In the next section, some of the drawbacks of k-means clustering will be discussed.

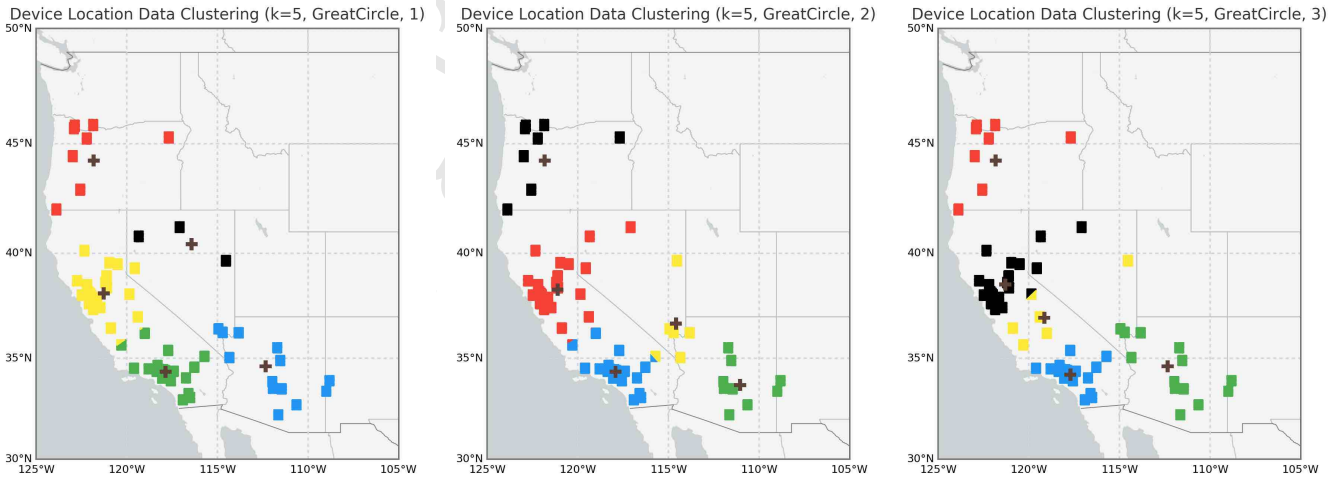
Furthermore, we can see the results are identical for both Euclidean and GreatCircle measurements. It is reasonable because: a) the initial centers are the same for both measurements (we performed random seed to guarantee this); b) both two measurements will give larger distance when two points are physically farther (more differences in latitude or longitude), so when assigning points to the centers, both measurements will yield the same centroid.

3.5 Influence of Changing Random Seed

In this section, we did several different running in each of our datasets and compare the results with different initial centers. For device location data, we use $k = 5$; for synthetic location data, we use $k = 2$ and $k = 4$; for DBpedia location data, since $k = 8$ gives the best result, we will use $k = 8$. Each data will be re-computed for 3 times using different random seeds, and since the results are identical for both distance measurements, we will use great circle distance only.

3.5.1 Device Location Data, k=5

The results are shown below:



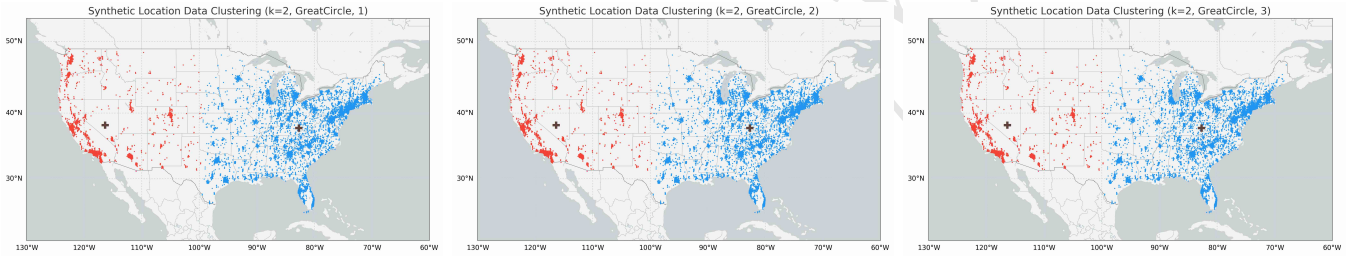
As we can see, when changing seeds, the clustering results are slightly different. The north-est cluster remains the same in all runnings; the other four are different, but the differences are not

dramatic. This is because of the different choice of initial centers.

Also, we can see another drawback of k-means in the third map. The center of the yellow cluster shifted because there are points very far away from the center. These points will drag the center to an unreasonable place. The third map also shows a imperfect distribution of clusters; there are some blocks clearly very far away from any center.

3.5.2 Synthetic Location Data, k=2

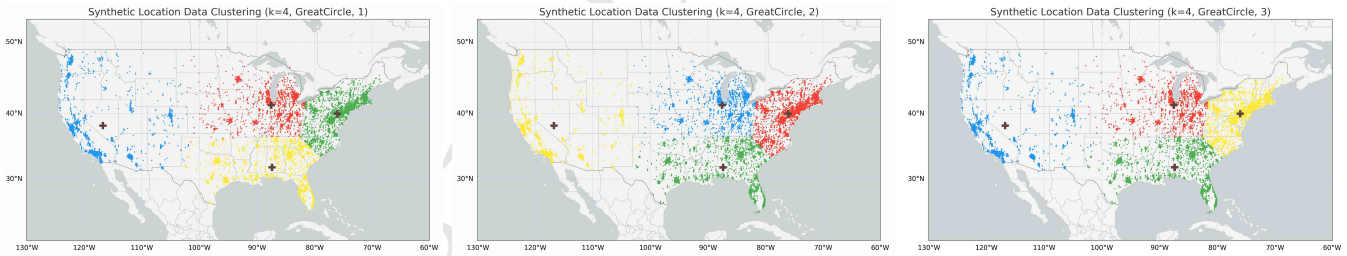
The results are shown below:



All three random seeds yield the same results. This is because the number of clusters is only two; it basically cannot give the different results.

3.5.3 Synthetic Location Data, k=4

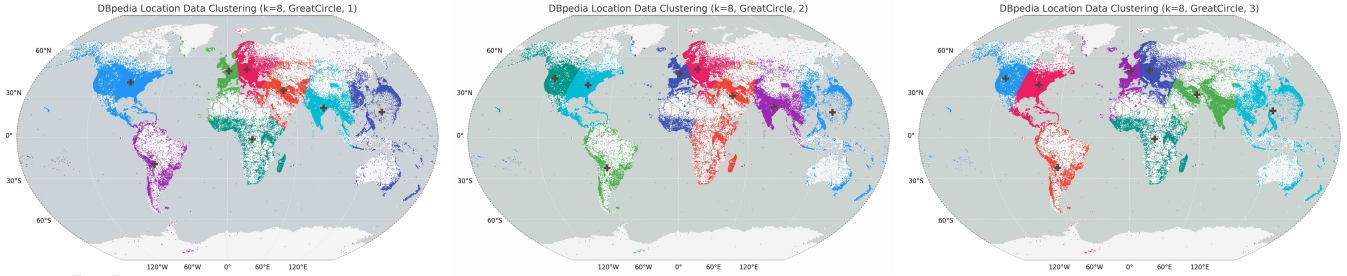
The results are shown below:



All three random seeds yield the same results. This is interesting because the number of clusters is four; however, there are some areas with higher density of points, so the samples are more likely to be chosen there. Therefore, all three runs gives the identical results.

3.5.4 DBpedia Location Data, k=8

The results are shown below:



The results are very different. This is a very good example showing the influence of initial center choices. In the first map, there is only one cluster in North America, but in the second and third map there are two. In the second map. there is no center located in Africa, but in the first and third map there is one. This is because of the choice of initial centers: if there is no center in Africa initially, it is highly unlikely that one center will shift to Africa later; if there is two centers in North America initially, it is highly unlikely that one center will go out of North America. The nature of k-means algorithm indicates that a good choice of starting point is very important for this algorithm, and also a good choice of k is equally important.

3.6 Runtime Analysis

The following runtime analysis is based on the Course VM. The system and hardware information:

```
CentOS
    Release 6.7 (Final)
    Kernel Linux 2.6.32-573.18.1.el6.x86_64
    GNOME 2.28.2
Hardware
    Memory    3.9GiB
    Processor    Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz
```

Also, all the comparisons are based on $k = 5$, local mode, 2 threads, and great circle distance measurement.

The runtime comparison (with and without persistent RDD) is shown below (each case was run 3 times and the average was calculated):

	with persistent RDD			
	1	2	3	average
Device Location Data	2m48.086s	3m3.393s	2m57.698s	2m56.392s
Synthetic Location Data	0m34.950s	0m34.563s	0m34.696s	0m34.736s
DBpedia Location Data	6m8.898s	6m19.170s	5m53.090s	6m7.053s
	without persistent RDD			
	1	2	3	average
Device Location Data	4m7.790s	4m18.229s	4m12.582s	4m12.867s
Synthetic Location Data	0m40.359s	0m43.366s	0m46.620s	0m43.448s
DBpedia Location Data	9m20.167s	9m51.086s	9m24.849s	9m32.034s

As we can see, running time when using persistent RDD is shorter than without using persistent RDD. Persistence will avoid re-computation if an RDD will be used multiple times. Persisting an RDD saves the data in memory by default, and the subsequent operations can used the persisted data. Memory persistence level will have the best performance because it is the fastest. In our program, both the (latitude, longitude) pairs and Cartesian coordinate pairs are highly likely to be reused in the iterative k-means algorithm, therefore, persisting both (latitude, longitude) pairs and

Cartesian coordinate pairs is necessary and will be save a lot of time.

In our example, using the average running time, we can see that using persistent RDD will save 30.2% in device location data, 20.1% in synthetic location data, and 35.8% in DBpedia location data. The bigger the dataset, the more RDD persistence will save.

4 References

1. https://en.wikipedia.org/wiki/Spherical_coordinate_system
2. https://en.wikipedia.org/wiki/Cartesian_coordinate_system
3. https://en.wikipedia.org/wiki/Euclidean_distance
4. https://en.wikipedia.org/wiki/Great-circle_distance

5 Appendix

5.1 About the Program

Programs are located in `src` folder. Since the main parts of k-means are identical for all input files, these parts are separated and written in the `kmeans` folder. In `kmeans` folder, `kmeans.py` contains the command line input parsing function and the k-means function. `tools.py` contains helper functions including coordinate transformation, distance measurement, center assignment and average calculation. The main programs are the `kmeans_xxx.py` files in `src` folder.

All the programs of k-means clustering can be run using the following schema:

```
> spark-submit --master local kmeans_xxx.py <input> <output> <k> <Euclidean|
   GreatCircle>
```

`local` can be replaced by `local[n]` to use n parallel threads, by `yarn-client` to use client mode, or by `yarn-cluster` to use cluster mode. The last two options will be recommended by packing a `*.zip` or `*.egg` file.

k must be an integer and must be greater than 0. The last argument must be one of these: `Euclidean`, `GreatCircle`.

To change the random seed in order to get another initial centers for the k-means algorithm, modify this line in `kmeans.py`:

```
> currentCenters = parsed.takeSample(False, k, 1)
```

Change `1` to other number to change the seed of random sampling.

5.2 Data Visualization

All the data visualization files are located in `drawmap` folder, including all the maps used in this project report. All the `*.py` files are used to either (latitude, longitude) pairs or (latitude, longitude, cluster number) pairs. To use the Python scripts to draw the map, the Python must be version 2.7

or later, together with these packages installed: `numpy`, `pandas`, `matplotlib`, and `Basemap`.

The `drawmap-xxx.py` files are used to draw maps without clustering, and the input must be (latitude, longitude) pairs delimited by commas; the `drawmap-xxx-clustering.py` files are used to draw maps with clustering centers, and the input must be (latitude, longitude, cluster number) pairs delimited by commas, and the cluster center must have a cluster number of -1.

All programs can be run using the following schema:

```
> python drawmap-xxx-clustering.py <input> <output> <title of the map>
```

Note that these programs can only run on a single file. If the output contains more than one files, they should be firstly combined into one file:

```
> cd result
> cat part* > combined
```