CSE 427S FINAL PROJECT

Geo-location Clustering in SPARK

Fall 2017

Instructor: Marion Neumann

Mentor: Jonathan Shieh

Group Member:

Name	WUSTL ID	Student ID	Work
Ruxin Zhang	ruxinzhang	442934	Abs., I., II.
Yutong Sun	yutongsun	451990	III.3.1
Wenshen Song	wenshen.song	437161	III.3.2
Li Xia	li.xia	425031	III.3.3

May 05 2017

Table of Contents

Abstract

This project aims to solve a geo-location clustering problem with big datasets. At beginning, we pre-processed the data to get it into a standardized format. Then we implemented k-means algorithm on different datasets (mobile device from Loudacre's network, DBPedia dataset[1]) with different K- values using Euclidean distance and Great Circle distance respectively. To escape from a local minimum, we improved our K-means algorithm using Monte Carlo method to set initial centroids. It achieved better stability of clustering. Finally, we found and stored a larger dataset (Word Cities Data[2]) on Amazon S3, then executed our Spark program on Amazon EMR. Meanwhile, we visualized clusters and clusters' centroids and compared the running time in different running modes with/without persist.

I. Motivation

Geographical information is playing an increasing important role in many areas of our lives. Taxi company can collect the location of users and taxi then schedule the closest car. Logistics companies may relocate their storage according to the distribution of customs.

K-means clustering is a powerful method to analyze geological information. It is the simplest algorithm which uses unsupervised learning method to solve known clustering issues. It works really well with large datasets.

From this project, we can practice our skill to explore big data via programming in Apache Spark and learn how to using APIs to compute and visualize the data. Meanwhile we can deepen our understanding of K-means algorithm especially how to set initial centrodes to enhance the stability of the algorithm. Besides, our research can be widely utilized in our daily lives as mentioned previously.

II. Introduction

2.1 System Exploration

Apache Spark is especially useful for parallel processing of distributed data with iterative algorithms. It can execute much faster by keeping data in memory. The main abstraction in Spark is that of a resilient distributed dataset (RDD), Represents an immutable, partitioned collection of elements that can be operated on in parallel.

RDD

RDDs are a collection of elements partitioned across the nodes of a cluster and can be operated on in parallel. RDDs can be created from HDFS files and can be cached, allowing reuse across parallel operations. RDDs achieve fault tolerance through a notion of lineage: if a partition of an RDD is lost, the RDD has enough information about how it was derived from other RDDs to be able to rebuild just that partition. RDD we used in this project are listed as follow:

RDD Name	Function
persist()	Persist its values across operations after the first time it is computed.
map()	Return a new RDD by applying a function to each element of this RDD.
filter()	Return a new RDD containing only the elements that satisfy a predicate.
takeSample()	Return a fixed-size sampled subset of this.
collect()	Return a list that contains all of the elements in this RDD.
reduceByKey()	Merge the values for each key using an associative reduce function.
saveAsTextFile()	Save this RDD as a text file, using string representations of elements.

Table 1-1 Spark RDD used in project

Run Spark in Different Mode

Once Spark is up and running with Hadoop, we can launch it via one of three modes: local, yarn-client or yarn-cluster.

Local mode: This launches a single Spark shell with all Spark components running within the same JVM. This is good for debugging on your laptop or on a workbench.

Yarn-cluster: The Spark driver runs within the Hadoop cluster as a YARN Application Master and spins up Spark executors within YARN containers. This allows Spark applications to run within the Hadoop cluster and be completely decoupled from the workbench, which is used only for job submission.

Yarn-client: The Spark driver runs on the workbench itself with the Application Master operating in a reduced role. It only requests resources from YARN to ensure the Spark workers reside in the Hadoop cluster within YARN containers. This provides an interactive environment with distributed operations.

2.2 K-means Clustering

Given a set of observations$(x_1, x_2, \ldots, x_n)$, where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into $k\ (\leq\ n)$ sets $S = \{S_1, S_2, \ldots, S_k\}$ so as to minimize the within-cluster sum of squares.Formally, the objective is to find:

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

where $\boldsymbol{\mu}_i$ is the mean of points in S_i. $\|\mathbf{x} - \boldsymbol{\mu}_i\|$is distance between point and centroid.
The algorithm is composed of the following steps:

1. Set $\mu_1, \ldots, \mu_k$ chosen from $x_1, \ldots, x_n$ as initial group centroids;
2. Assign each object to the group that has the closest centroid.
3. When all objects have been assigned, recalculate the positions of the K centroids $\mu_1, \ldots, \mu_k$, here, $\mu_i$is set to the average of all points assigned to cluster j.
4. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

Distance Definition

Euclidian distance

$$D_{euclidean} = \sqrt{(\Delta lat)^2 + (\Delta long)^2}$$

Great circle distance

The spherical law of cosines formula can have large rounding errors if the distance is small, thus we adapt the haversine formula [2] which is numerically better-conditioned for small distances:

$$D_{great\_circle} = 2 \arcsin \sqrt{sin^2\left(\frac{\Delta lat}{2}\right) + \cos(lat_1)\cos(lat_2)sin^2\left(\frac{\Delta long}{2}\right)}$$

Estimating the Number of Clusters *k*

The Elbow method [4] is a method of interpretation and validation of consistency within cluster analysis designed to help finding the appropriate number of clusters in a dataset. This method looks at the percentage of variance explained as a function of the number of clusters: One should choose a number of clusters so that adding another cluster doesn't give much better modeling of the data.

Conclusion

Although it can be proved that the procedure will always terminate, the k-means algorithm does not necessarily find the most optimal configuration, corresponding to the global objective function minimum. The algorithm is also significantly sensitive to the initial randomly selected cluster centroids.

In this project, we use Great Circle Distance (GCD). Because the dataset used in this project is the coordinate of different locations (longitude and latitude). Considering the Earth is a sphere. GCD is much proper than other distance types.

III Implementation

3.1 Development using Small Data

Before implementing K-means clustering algorithm, we preprocessed the data into a standardized format.

Part I. K-means Clustering

In this K-means algorithm, we set a converge threshold, if the sum of distance between old centers and new centers less than 0.1 than we think K-means is converge. The pseudo-code of our algorithm is showing below:

PSEUDO-CODE OF K-MEANS

```
Random select k cluster centers μ_j⃗ from the dataset.
tempDist=infinite, convergeDist=0.1
        while tempDist>convergeDist
                for all data points x_i⃗
                        find the closest cluster μ_j⃗ center for the x_i⃗
                set γ_ij = 1 and generate (cluster index,(p,1)) key-value pair
                        for each cluster index calculate the mean of points, which is the new cluster
center
                        set temDist be the sum of distance of old centers to new centers
                end for
Plot data points and cluster to the map
```

Part II. Results and Analysis

In this part, this project realize k-means and try to use different datasets to test the efficiency and accuracy of this algorithm. The results for different data sets are showing below. (Both of them use four cores and set 0.1 as the convergence distance).

1. Device Location Data

Using k=5 to calculate the k-means clusters for device location data. And the running time of it is 9.1332s(0.002537h).

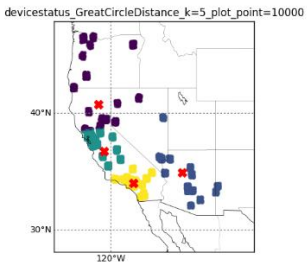

Figure 3.1-1 K-means for devices status data (k=5)

2. Synthetic Location Data"

Using k=2 and K=5 to calculate the k-means clusters for dataset named sample_geo.txt. When k=2, the running time is 15.48s (0.0043h). When k=4, the running time is 16.38s (0.00455h). Thus, there is a tendency that with the k increase, the time increase too.

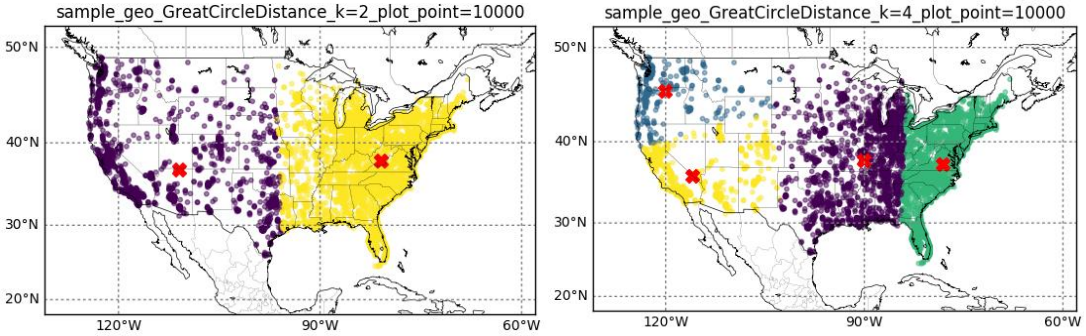

Figure 3.1-2 K-means for sample_geo data (k=2 and k=4)

3. DBpedia Location Data

The results are showing below:

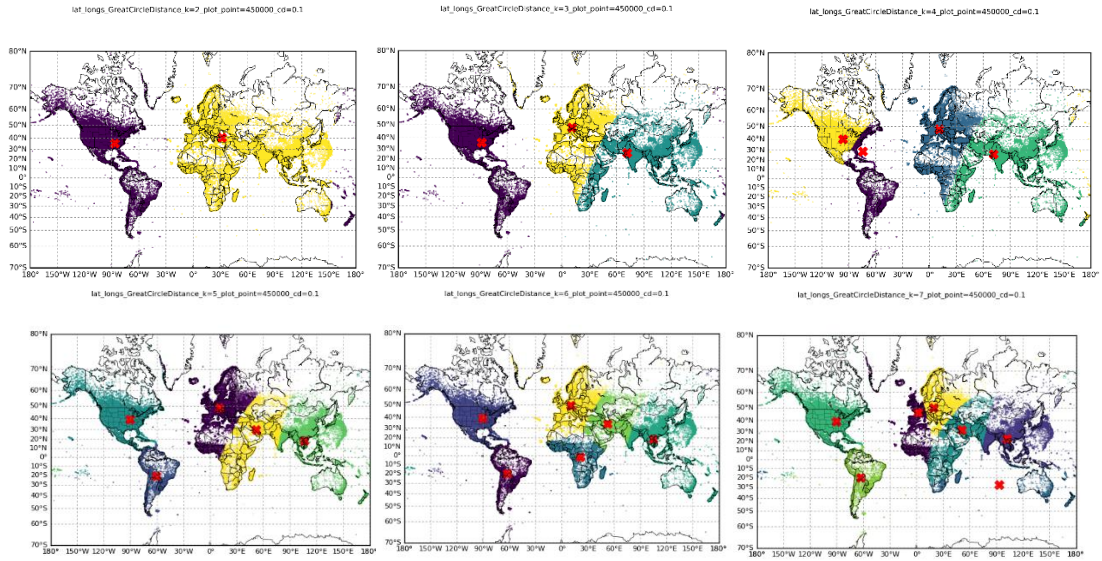

Figure 3.1-3 Different k=2 to 7 k-means in GCD for DBpedia location data.

The running time of this dataset is showing below:

Number of cluster	K=2	K=3	K=4	K=5	K=6	K=7
Running time	0.02635	0.03092	0.03615	0.04267	0.07497	0.06466

Table 3.1-1 Running time for random select k-means

In this part, we use "DBpedia location data" as an example to analyze running time. Because the start points are random selected, it is easy for this simple algorithm being trapped in local minima. Thus, the running time for this method is influenced by start points a lot. Thus, it is hard to tell an accurate tendency of running time when the number of cluster increase. But from the table showing above, we can see the with the number increase, there is a high chance that the running time will also increase.

Meanwhile, because the initial cluster centers are random selected, some clustering results is quite different from common sense. The following are some results which has big sum of distances.

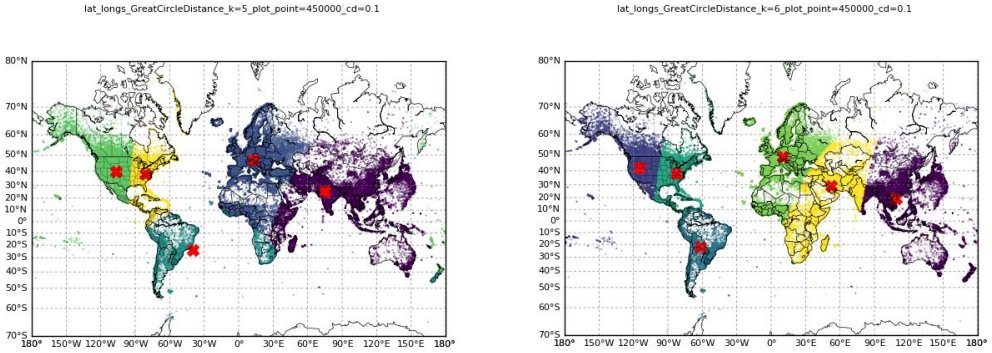

Figure 3.1-4 Results which has big sum of distances.

In order to avoid this situation and trapped in local minima, we try to introduce Monte Carlo (see 2.2)into K-means to improve the performance of this algorithm.

4. Free Cities

Last but not least, we find a larger dataset from Free World Cities Database and the dataset has a name of worldcitiespop. It includes attributes like Country, City, AccentCity, Region, Population, Latitude, Longitude, and we preprocess it by extracting Latitude and Longitude. After that the original 144.1 MB data decreases to 57.5 MB with 3173959 records. It is still larger compared to 33.5 MB DBpedia dataset with 450151 records.

It is used to describe the location of different cities around the world. The results are showing below:

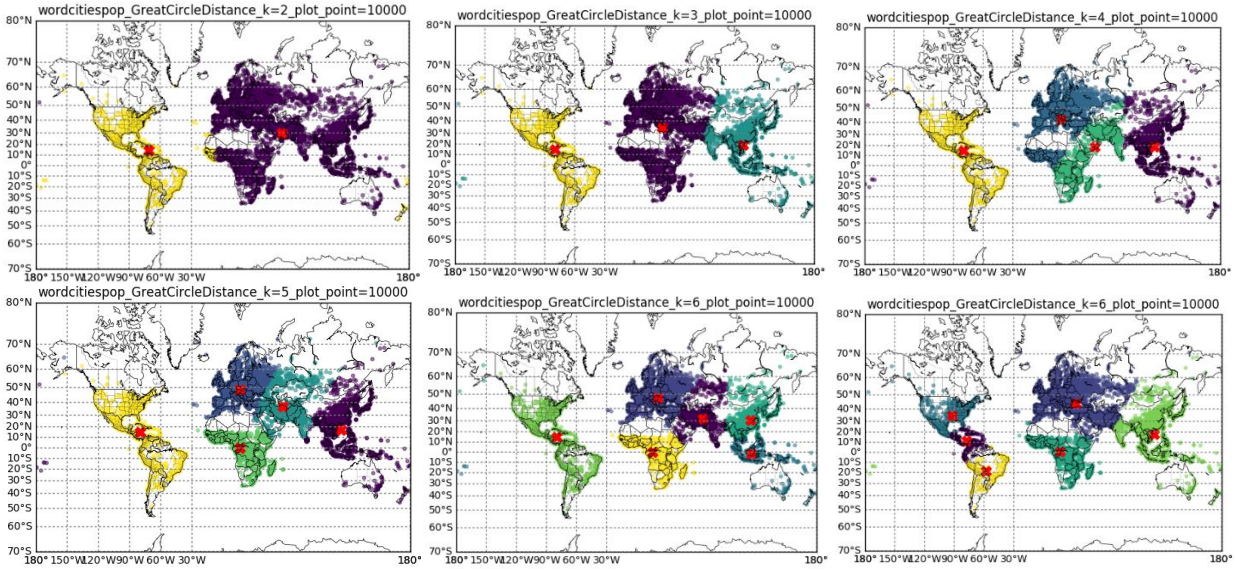

Figure 3.1-3 Different k=2 to 7 k-means in GCD for WordCities data.

Number of cluster	K=2	K=3	K=4	K=5	K=6
Running time	0.12889	0.18665	0.19084	0.21618	0.23405

Table 3.1-2 Running time for random select k-means

Part III. Conclusion

From above results, we can see it is easily for random select method to cluster North and South America together and cluster Asia and Australia together. The reason of this situation may be there are too few points in Australia, in other words the data in Australia is sparse and there is a small chance to build an independent cluster for Australia, which means for all local minima we may find, many of them cluster Asia and Australia together.

3.2 Monte Carlo Method in Initial Centers Determination of K-means

Part I. Introduction and Methods

As known, k-means method has an objective function (measured by the squared Euclidean distance):

$$O = \sum_{i=1}^{k} \sum_{x \in G_i} ||x - \mu_i||^2$$

For a general case, objective function may have a different form, $\sum_{i=1}^{k} \sum_{x \in G_i} dist(x, \mu_i)^2$. Nevertheless, k-means should have a global minimum for objective function. However, the existence of global minimum doesn't guarantee that our k-means algorithm will converge to that value. In original k-means, it takes initial centers randomly and updates the index of each points by finding the minimum distance with different centers then updates the centers by averaging points in same clusters. A simple instance can explain its limitations. Suppose we have 4 points (black ones) at four vertexes of a rectangle, showing in Figure 3.2-1, and we want to cluster them use k-means with k=2:

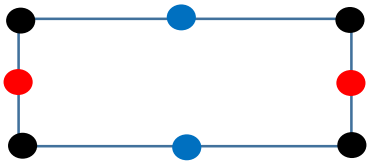

Figure 3.2-1. a toy dataset proven that k-means cannot always converge to global minimum

According to the original algorithms, both blue centers and red centers are converged results, but only the red ones converge to global minimum. Such a problem also exists in gradient descent method, which could be trapped by local minimum.

If we don't make drastic changes to original k-means iteration algorithm, the best way to solve the problem is playing with initialization. Now the weakness of original k-means is mainly about that, random selecting of initial centers may be too close to a certain local minimum and easily get trapped in that well. To deal with this, we can implement Monte Carlo methods for initialization.

Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results[1]. It was developed to handle with nuclear reaction simulation in WWII. Here we illustrate Monte Carlo method in a simple way:

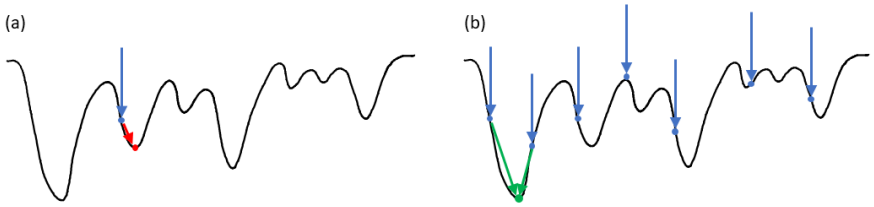

Figure 3.2-2. (left) One-time random initialization and (right) Monte Carlo initialization

As Figure 3.2-2 shows, if using one-time initialization (left), it is highly possible that we eventually trapped at a local minimum, as the red point shows; However, Monte Carlo method (right) can enable us to throw several random starting points and drives them to nearest minimum points. As long as the starting points distribution is even and dense enough, it is likely that we could at least find one starting point leading to the global minimum, as the green point shows. What's more, considering the large volume of our data, it is quite expensive and time-consuming to perform Monte Carlo on the whole dataset. Thus, we come up with another method to take a small random sample of the whole dataset, and perform the Monte Carlo method on this small dataset. Although the small dataset is less likely to generate the same centers for objective function minimum as the whole dataset, it can still reflect the properties and patterns and thus give centers not far away from the realistic ones. Consequently, we first perform Monte Carlo method on the small sample dataset and record the minimum objective functions and corresponding coordinates of centers after k-means iterations; then we simply use these centers as initial centers for the whole dataset to perform k-means. Ideally, these centers should be close to the optimized centers leading to the global minimum and the later process should be fast. In a sense, this algorithm can both improve the accuracy and efficiency for k-means on whole dataset. The pseudo code is as follows:

```
for i =1:numMC: #here numMC is the times of monte carlo method, here we generally use 100
{
        sample_data=data.take_sample()  #take samples from data, here we use n/100 or n/1000 points, n = of whole dataset
        kcenters=random(sample_data, k)#pick random k centers
        for p in sample_data:
        {
                do{
                        p.index=closestpoint(p, kcenters) #assign p with closest centers index
                        kcenters=(mean(p) in same p.index) #kcenters = average of p in each cluster
                } until(kcenters doesn't change )
        }
}

for p in data:
{
        do{
                p.index=closestpoint(p, kcenters)
                kcenters=(mean(p) in same p.index)
        } until(kcenters don't change ) #practically we use 0.1 as a convergedist
}
```

[1] https://en.wikipedia.org/wiki/Monte_Carlo_method

In practice, we need to deal with much more complex situations such as the convergence under various size of sample data and iteration details like stopping criterion. All details are included in pyspark script and suitable parameters are declared with specific outputs.

Part II. Results and Analysis

We implement Monte-Carlo enhanced algorithm to device location data (devicedata), synthetic location data (sample_geo) and DBpedia location data (DBpedia) with different sets of parameters, then visualize the final results with the assistance of Matplotlib Basemap toolkit[2] to plot the world maps. All local implementation are running on 4 nodes.

1. Device location data

We are asked to using k=5 to calculate the k-means clusters for device location data. Herein sample data size is 1/1000 of data size. Monte Carlo steps are 50 and iteration limitation for each k-means is 20 times in sample data. Convergence distance is 0.0 for sample data and 1.0 for the whole data. To verify the effectiveness of Monte Carlo method, we first plot before and after Monte Carlo k centers for sample data using GreatCircleDistance:

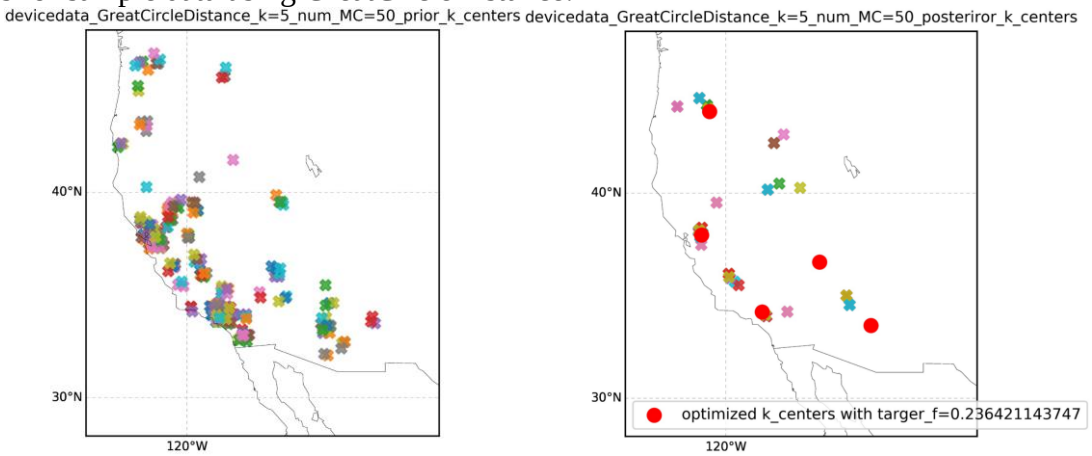

Figure 1.2-3. k centers before (left) and after (right) Monte Carlo k-means in sample data

Initially, random assignments of k centers are distributed evenly through the sample data, and different colors represent one assignment of 5 centers at each Monte Carlo step. In the right plot we can see that after k-means clustering most centers are gathered at certain region, indicating k-means method has the ability to find the minimum. Meanwhile the dispersed points also show that traditional k-means cannot guarantee the absolute convergence to a global minimum, thus we pick up the centers (red circle) with the smallest object function (noted as target_f in the plot) in this group of centers as the starting centers for the whole dataset. Here, number of Monte Carlo method=50 and sample size=100 are tested to guarantee a stable output of optimized k-centers.
After sample data initialization, we obtain the final results for the whole dataset.

[2] https://matplotlib.org/basemap/

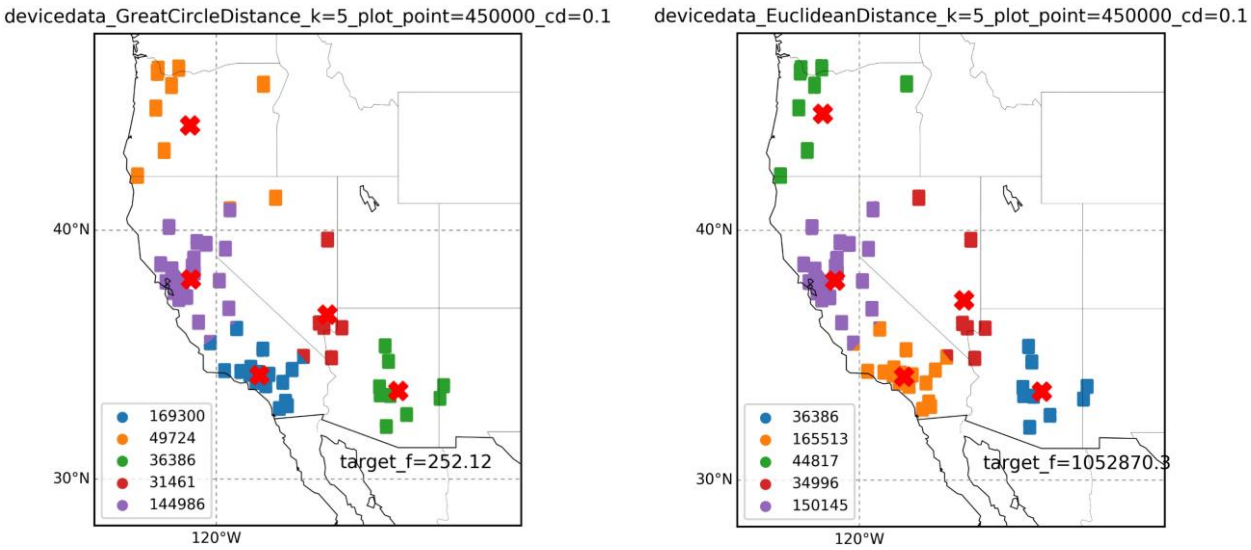

Figure 3.2-2. devicedata k=5 k-means result with GreatCircleDistance (left) and EuclideanDistance (right)

First, we noticed that all data points are gathered as small rectangle clusters and distributed in the Western US. Our clustering mainly deal with those small rectangles. In a sense, for this specific situation, we could renormalize each small rectangle to its centroid to simplify the data map. However, to keep consistency and generality, we still stick to our method without preprocessing the data in this way. We can see that data are clustered in a reasonable pattern and both GreatCircleDistance (GCD) and EuclideanDistance (ED) have similar results. The reason is that GCD degenerates to ED when the region scale is fairly small compared to the earth. Besides, the large difference of objective function(target_f) is due to we use different distance measure, where GCD keeps the radius to be 1 and ED has latitude and longitude degree as values to perform the norm.

2. Synthetic location data

Here we use dataset named sample_geo.txt to do k-means clustering. For k=2 case,

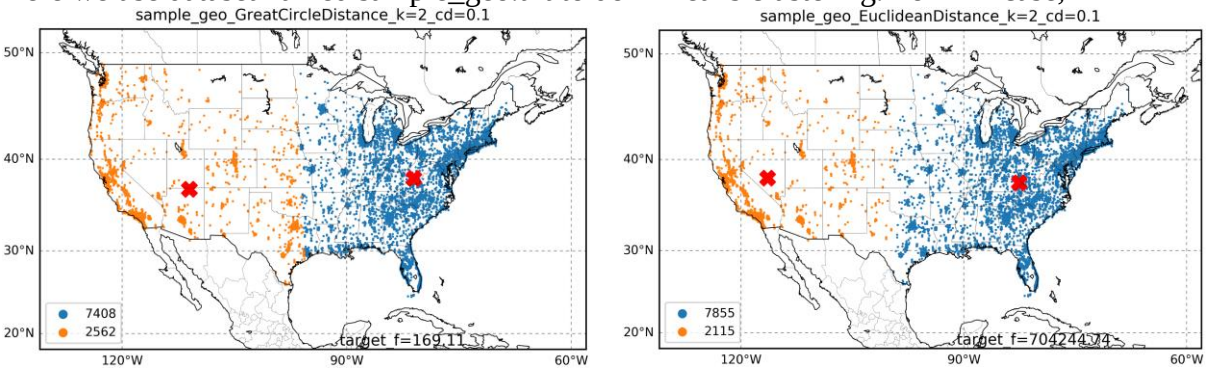

Figure 3.2-3. synthetic location data k=2 k-means result with GreatCircleDistance (left) and EuclideanDistance (right)

There is no significant difference in two distance measurements due to the simple reason that k=2 make k-means a binary clustering method, and in U.S. region both GCD and ED give a north-south boundary. We can imagine that we expand a ring region on earth which contains the whole U.S. to a 2D flat. This operation will not lead to a tremendous change in closeness measurements of a west-to-east binary clustering.

However, things get a little different for k=4 situation:

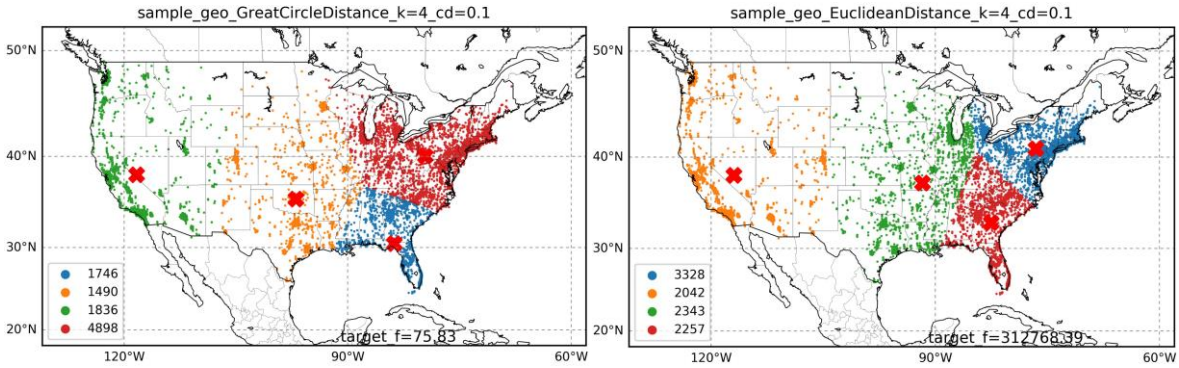

Figure 3.2-4. synthetic location data k=4 k-means result with GreatCircleDistance (left) and EuclideanDistance (right)

It can be observed that in eastern U.S, the red and blue parts are divided by distinct shapes of boundaries in GCD and ED. As far as I am concerned, it is caused by the essence of GCD and ED definition. Recall the formula:

$$D_{euclidean} = \sqrt{(\Delta lat)^2 + (\Delta long)^2}$$

$$D_{great\_circle} = 2 \arcsin \sqrt{sin^2\left(\frac{\Delta lat}{2}\right) + \cos(lat_1)\cos(lat_2)sin^2\left(\frac{\Delta long}{2}\right)}$$

Suppose there are 2 sets of geo-points (M1, M2) and (N1, N2), in which M1 and M2 have the same latitude or longitude difference as N1 and N2. The only difference is N1 and N2 have larger latitude than M1 and M2 (which means M sets are closer to the equator).

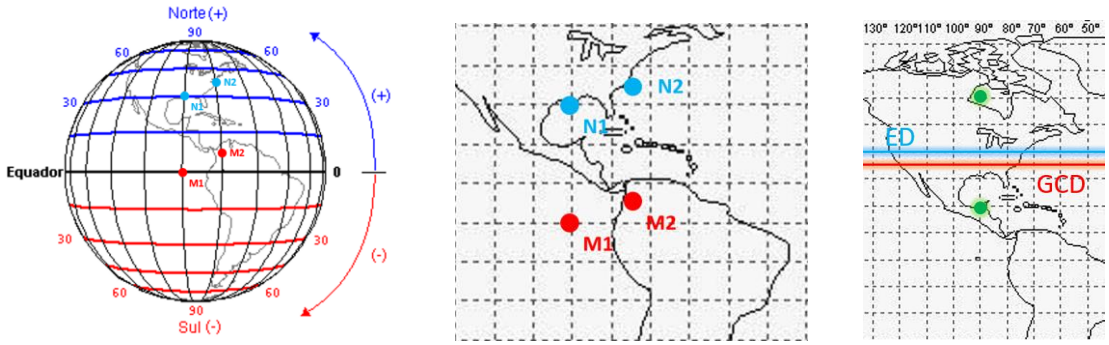

Figure 3.2-5. sketch map for M and N sets of points in 3D (left) and 2D (middle) world map and decision boundary difference (right).

In Euclidean distance measurement, since $(\Delta lat, \Delta long)$ for (M1, M2) is equal to (N1, N2), ED of M1M2 and N1N2 should be same. But for Great Circle distance measurement, $\cos(\text{lat}_{N1})\cos(\text{lat}_{N2})$ is smaller than $\cos(\text{lat}_{M1})\cos(\text{lat}_{M2})$, leading to GCD of N1N2 is smaller than M1M2. This could explain what we found in eastern U.S. boundary difference for GCD and ED. For 2 centers distributed along latitude direction, as Figure 3.2-5 right shows, the blue ED boundary should be exactly the midnormal line of 2 green points. However, in GCD, distance from upper point to blue ED boundary is less than that of lower point, because of our previous analysis. Therefore, GCD boundary should be shifted to south compared to ED boundary. That is exactly what we observed in Figure 3.2-4, in which GCD boundary in eastern U.S. is much closer to equator than ED boundary.

It is a general conclusion that, in large enough scale, GCD will lead to distinguished difference dealing with north-south clustering compared to ED, meanwhile for east-west clustering the difference should not be notable (GCD calculation doesn't contain absolute longitude value), as we observed in Figure 3.2-4 western U.S.part.

3. DBpedia location data

Here we use dataset named lat_longs.txt to k-means clustering. For DBpedia dataset, we need to deal with much more complicated situations due to its world-wide scale entangled distribution.

Also, the generous size of dataset makes it obliged to test the convergence of parameters. Since now we are dealing with world-wide scale data, our goal is to find objective function as small as possible for GCD.

Let's start from k=5. First, we test number of Monte Carlo steps, as we can see:

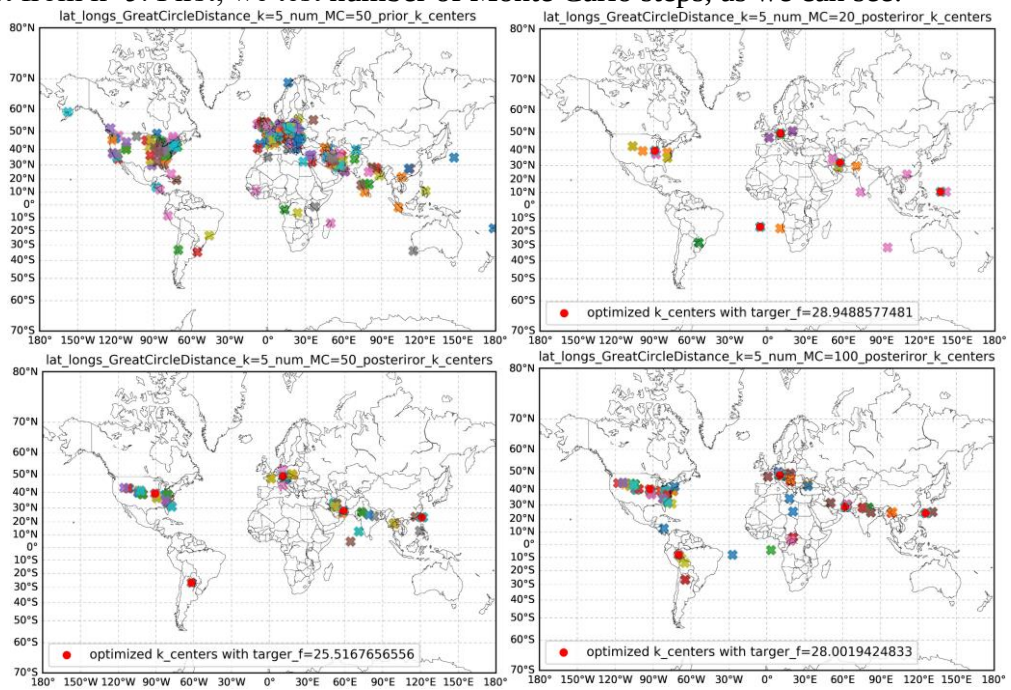

Figure 3.2-6. k centers before (upper-left) and after (rest) Monte Carlo k-means in sample data with size=1/1000 total size

As for sample_size = total_size/1000 case, when increasing the number of Monte Carlo from 20 to 100, the output centers are gradually converged. In practice, we implement a much more rigorous criterion that sample_size=total_size/100 and number of MC=100 for k>4 cases. After the determination of parameters, we are ready to run different k in GCD.

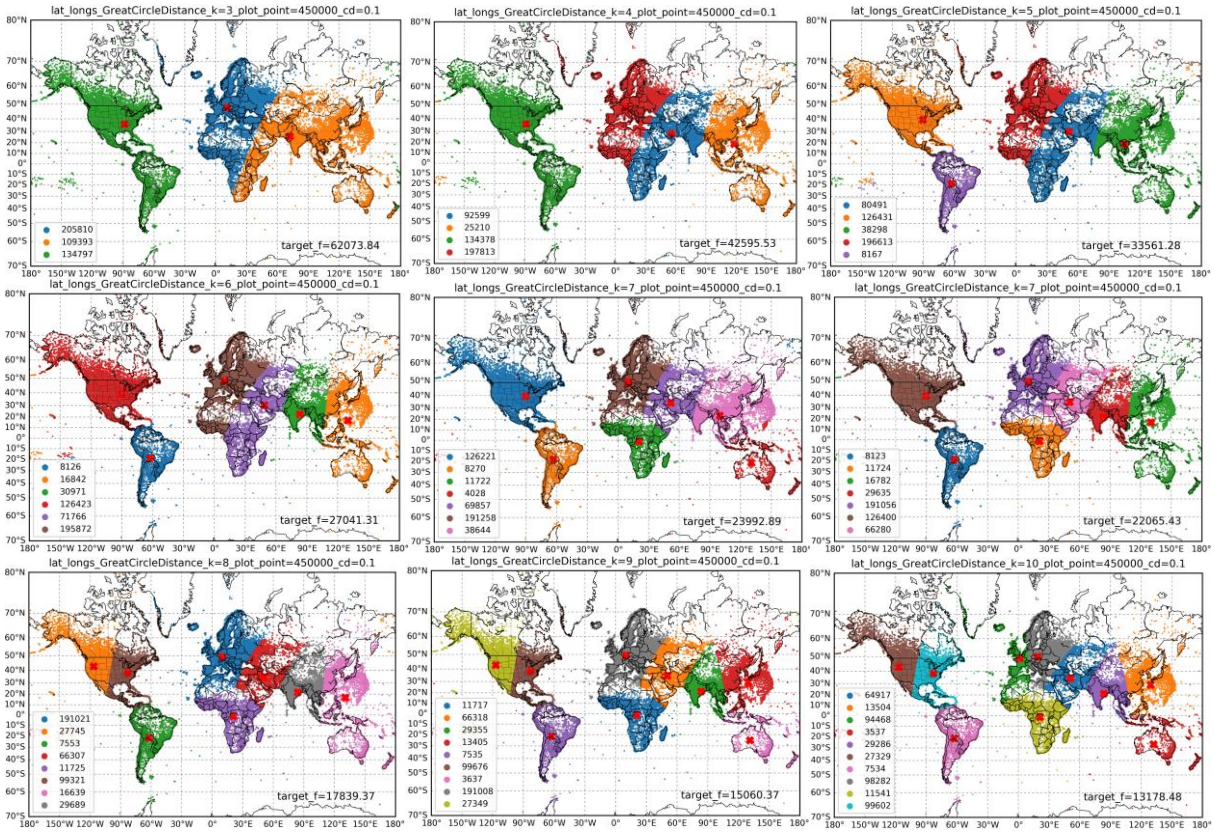

Figure 3.2-7. Different k=3 to 10 k-means in GCD for DBpedia location data. We use sample_size=total_size/100 and number of MC=100 for k<7 case and number of MC=200 for k>7 case. For k=7 we present 2 figs here, left for numMC=100 and right for numMC=200

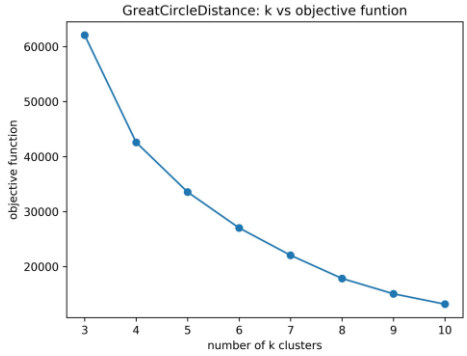

Figure 3.2-8. objective function vs number of k clusters for GCD

From Figure 3.2-7 we can see the change of clustering as k increasing from 3 to 10. And Figure 3.2-8 shows that objective function gradually decreases with not abrupt change. It is hard to choose an optimized k since we don't have a criterion here, needless to say the explanation for different clustering could be various according to different views. As far as I am concerned, DBpedia collects the geo location from Wikipedia, and the frequency of each location largely depends on political, economic, cultural, military affairs, and so on. This could be concluded as a summary of Geopolitics, which is a study of the effects of geography (human and physical) on international politics and international relations3.

[3] https://en.wikipedia.org/wiki/Geopolitics

Based on this concept, I pick up k=7 as a crucial point. First, k=7 gives a clear clustering pattern: Successfully clustering North and South America. For other parts of the World, Europe and north Africa are assigned to one cluster. Central and southern Africa are separated from them by the Sahara. Europe and Asia continent are basically separated along the Ural Mountains, which is also the realistic geo-boundary of Europe and Asia. Middle eastern area, including Egypt and Afghanistan as well as some central Asian republics of Russia are in the same cluster, which is reasonable since these areas are the focus of news in these days. Forward to east, we can see that boundary mainly lies at Chinese and Indian boundary. These boundaries are not only political, but are geographical (Thar Desert of India, Tian Shan of China). After that a problem occurs. Since we have 2 clustering result for k=7, one is separating Australia and Asia apart, and the other one separate China apart, half China with Australia and half with Southeast Asia and India.

We must realize that k-means can only deal with convex data, and here the pattern is far more complicated than that. As result, we compared the objective function and obviously, the later clustering has a smaller value. Again, here we cannot tell good from bad, because of the limitations of k-means cluster, as well as the realistic conditions are hard to judge (for example, geographically Asia and Australia should be two clusters, but actually in World Cup they are in the same division). If we keep increasing k to 8, then the north America are separated into western and eastern ones. It is not strange since north America has a large volume of points and large areas, making it the most preferred one for separating to lower the objective function. But here we consider it as a disadvantage of k-means and stop k at 7. Looking back at smaller k<7, k=5 and 6 basically clusters the world in a reasonable way, and as I put previously, it all depends on personal preference of how to explain that. k=3, 4 are obviously performing badly so we neglect them. As far as I am concerned, I prefer k=7 clustering with lower objective function, it is both easy to analyze and effective to implement, although it costs more time than smaller k number.

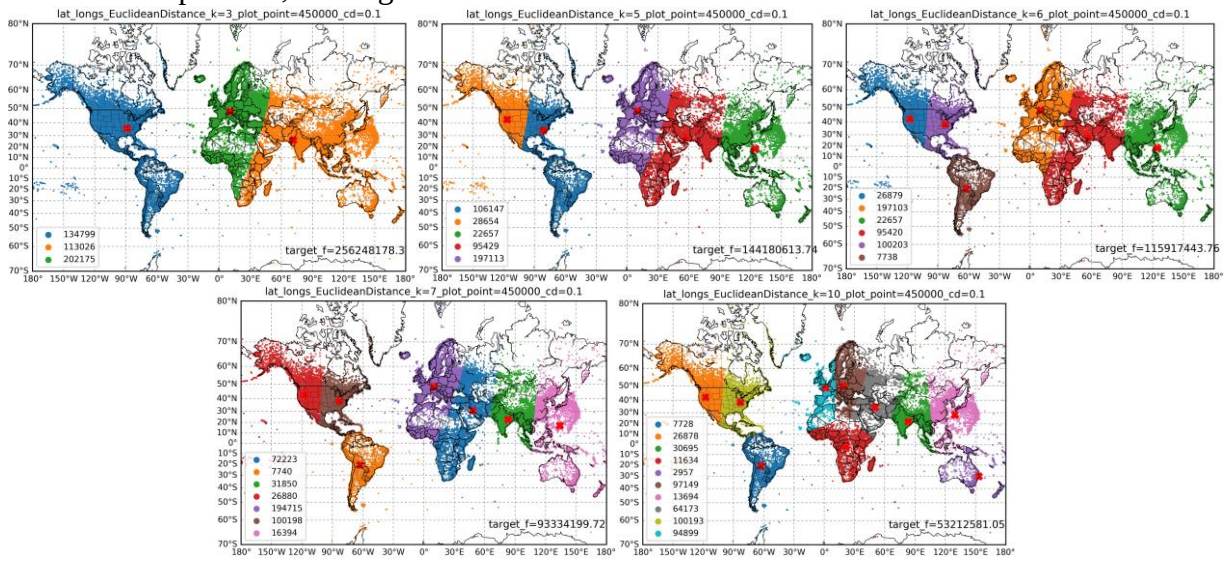

Figure 3.2-9. Different k=3, 5, 6, 7, 10 k-means in ED for DBpedia location data.

k=3 performs as bad as GCD because k is too small. The problem is ED starts to separate north America too early. For large k it may not be such a big issue since every large cluster are divided into small parts. But for k=5, 6, 7, it is unreasonable to do that. ED lacks overview for big-scale data, since in geo-data it is just an approximation of GCD in small scale.

In the last section of this part, we discuss runtime analysis. Lat_longs data are running on numMC=100 and sample_size=data_size/100, while others are running on numMC=50 and sample_size= data_size /1000. For each configuration, we run 3 times and take average time. It worth to note that Monte Carlo implementation makes running time very robust, due to its determined numMC and iteration upper bound for each k-means. It is obvious to see that Monte Carlo sample run consumes the most time and k-means on the whole dataset is very fast. In a sense, Monte Carlo increase the efficiency of k-means under the same accuracy. Generally, persist of RDD could decrease the whole data runtime, but for sample runtime it is complicated. From the table, we can find that persist RDD largely increase the efficiency of device data with k=2. However, for other data and configuration the improvement of speed seems insignificant. The probable reason is that device data has quite large a size as well as easy-to-cluster essence. Under that case, the persistence could be the vital for efficiency. However, in other cases, the data are either small or complicated to deal with, so persist or unpersist methods will not make too much difference.

	persist			unpersist		
	sample runtime (h)	whole data runtime(h)	total time(h)	sample runtime (h)	whole data runtime(h)	total time(h)
lat_longs_GreatCircleDistance_k=6_plot_point=450000_cd=0.2	0.181088	0.029614	0.2107	0.17414	0.03068	0.20482
lat_longs_EuclideanDistance_k=6_plot_point=450000_cd=0.2	0.175292	0.061788	0.2371	0.17586	0.08016	0.25602
devicedata_GreatCircleDistance_k=5_plot_point=450000_cd=0.2	0.067346	0.013136	0.0805	0.14181	0.0191	0.16091
devicedata_EuclideanDistance_k=5_plot_point=450000_cd=0.2	0.0646725	0.02513	0.0898	0.13996	0.04088	0.18084
sample_geo_GreatCircleDistance_k=2_cd=0.2	0.0424125	0.00056	0.043	0.05169	0.00067	0.05235
sample_geo_EuclideanDistance_k=2_cd=0.2	0.05293	0.00065	0.0536	0.0527	0.00083	0.05353
sample_geo_GreatCircleDistance_k=4_cd=0.2	0.04917833	0.0009175	0.0501	0.05173	0.00075	0.05248
sample_geo_EuclideanDistance_k=4_cd=0.2	0.0521925	0.0015425	0.0537	0.05217	0.002	0.05418

Table 3.2-1. runtime analysis of local implemetation

4. Free World Cities Database[4]

Since worldcitiespop dataset is still a geo-dataset and its distribution doesn't differ much from DBpedia, it is reasonable to treat it with Great Circle Distance. We implement with same settings of environments, and set numMC=50 and sample_size= 1/1000*data_size. Then we try different k=2-10:

[4] https://www.maxmind.com/en/free-world-cities-database

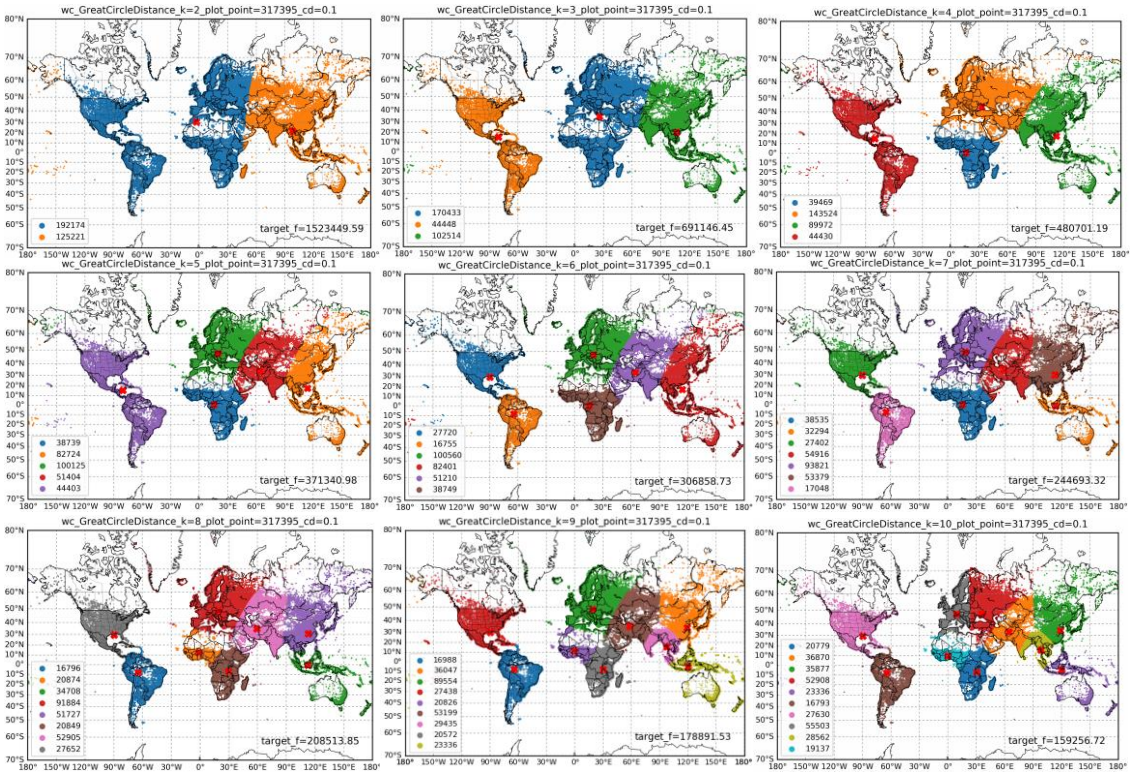

Figure 3.2- 10. Different k=2 to 10 k-means in GCD for World Cities data. We use sample_size=total_size/1000 and number of MC=50 all cases.

We can see that it shows similar pattern as DBpedia data, but obviously here k-means have better performances. In the middle and south Africa, points gather in a larger density and the Sahara becomes a distinct boundary. Cities location is much dependent on geography rather than news value, therefore this dataset should be less dispersed than DBpedia.

Even for a small k=4, Africa is clustered as an individual part. For k=7, distinct parts of world are clustered satisfactorily. We can see the north and south America, Africa continent, Europe, Middle Asia, East Asia and Australia. Still, k-means does a bad job with Europe-Asia Continent, which is an inevitable shortcoming due to its failure at non-convex situation, as we proved previously.

Nevertheless, k=7 already meets our expectation. Another interesting thing about this dataset is that when k increases to large value like 8, 9, k-means starts to touch Africa rather than America. For k=9 k-means extract western Africa and surprisingly, India and Indochina from China. It indicates worldcities data are more competent for detailed clustering than DBpedia. Herein we pick up k=9 as our preference.

configuration	sample runtime(h)	data runtime(h)	total time(h)	target_f
wc_GreatCircleDistance_k=2_plot_point=317395_cd=0.1	0.13853	0.05584	0.19436	1523450
wc_GreatCircleDistance_k=3_plot_point=317395_cd=0.1	0.15331	0.06965	0.22296	691146.5
wc_GreatCircleDistance_k=4_plot_point=317395_cd=0.1	0.14325	0.08226	0.22552	480701.2
wc_GreatCircleDistance_k=5_plot_point=317395_cd=0.1	0.15414	0.09514	0.24928	371380.2
wc_GreatCircleDistance_k=6_plot_point=317395_cd=0.1	0.16425	0.1088	0.27306	306858.7
wc_GreatCircleDistance_k=7_plot_point=317395_cd=0.2	0.16829	0.14545	0.31374	244396.6
wc_GreatCircleDistance_k=8_plot_point=317395_cd=0.1	0.18568	0.13277	0.31844	208513.8
wc_GreatCircleDistance_k=9_plot_point=317395_cd=0.1	0.18007	0.17222	0.35229	179222.2
wc_GreatCircleDistance_k=10_plot_point=317395_cd=0.1	0.19307	0.19074	0.38381	159256.7

Table 3.2-2. runtime analysis of local implemetation on world cities dataset

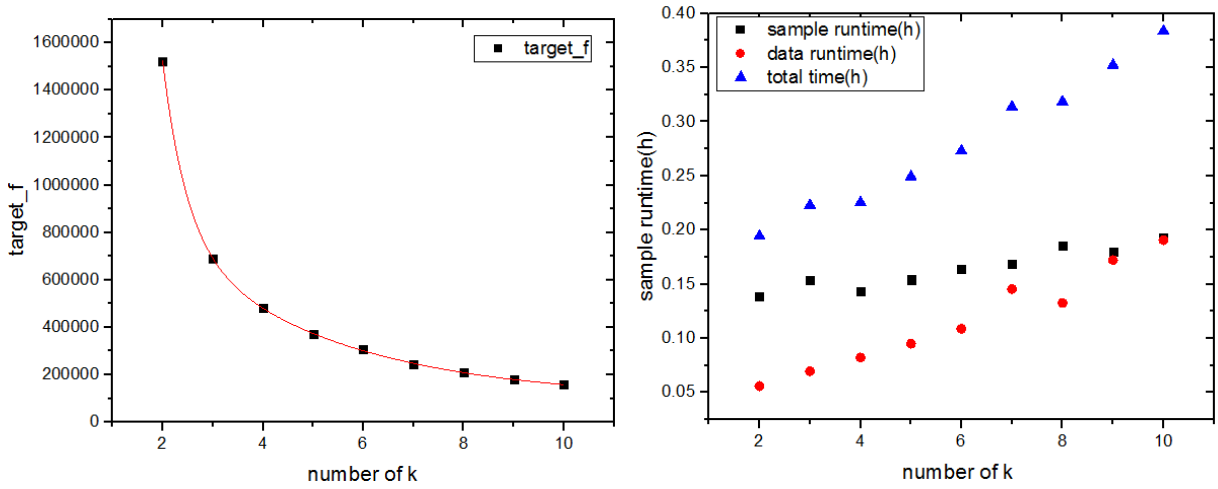

Figure 3.2- 11. k dependence of objective function and running time of World Cities dataset

As k increases, we can see that objective function still smoothly decrease like that in DBpedia, while both sample and whole data runtime are increasing.

Part III. Conclusion

As summary, we implement Monte-Carlo method on sampled data to find a proper starting point for k-means method to overcome k-means' shortcoming of trapping in local minimum, meanwhile it greatly reduces the runtime for k-means on the whole dataset under the same accuracy. Our practice on three datasets as well as an extra one are satisfying, and we find k=7 is best for DBpedia data while k=9 is best for World Cities data and both are reasonable to explain. Also, some details of methods have been introduced, such as different effects of great circle distance and Euclidean distance, why k-means cannot deal with non-convex data, and so on. Overall, our Monte-Carlo enhanced k-means is proved to be effective, and there is still ample room for further improvement of its accuracy and speed.

3.3 EMR Cloud Computing Implantation

Amazon offers on-demand cloud computing platforms from 16 geographical regions across world. The mostly used part is the EC2 as elastic compute cloud, S3 as Amazon Simple Storage Service,

and EMR as the Amazon Elastic MapReduce. The operate process is pretty straightforward and Amazon has configuring everything.

Part I. Big Data Problem and Cloud Execution Process

Although both of Lat-long data and the free world city dataset do not have a extreme big size, the running time for high K is significantly large. When K=7, it fails in some of our computer to converge in hours. It is necessary for people to considering using the powerful cluster and cloud computing platform such as Amazon AWS. Not only it applies huge data storage S3, and also it configures all kind of cloud computing tools, work flows, clusters and so on.

First we logged in Amazon AWS account, and get on EC2 to create a key pair. Then we upload the dataset into S3 bucket. Then we create a new resource with the same target. Then we go EMR to create a new cluster with the same tag. In EMR configuration we can choose compute node (here we choose 7 including 6 work-nodes and 1 master node. Also we choose spark as the running script tool. Finally we use the Zeppelin notebook to write and run the scripts.

Part II. Result and Analysis

In this part, we used two dataset, including the lat-longs dataset given by the course, and the worldcitiespop.txt. The size and instance numbers comparison is in the following table:

Dataset/property	Lat-longs	Worldcitiespop.txt
size	33.5 MB	57.5MB
records number	450151	3173959

Table 3.3-1 Size of Big Dataset

And we tried the original k-means algorithm with randomly initial center selection and Monte Carlo initial center selection. Then we showed the results and compared the running time for each method with the local machine.

The cluster results for K = 2-7 for the Lat-Long dataset are showing in the following figure:

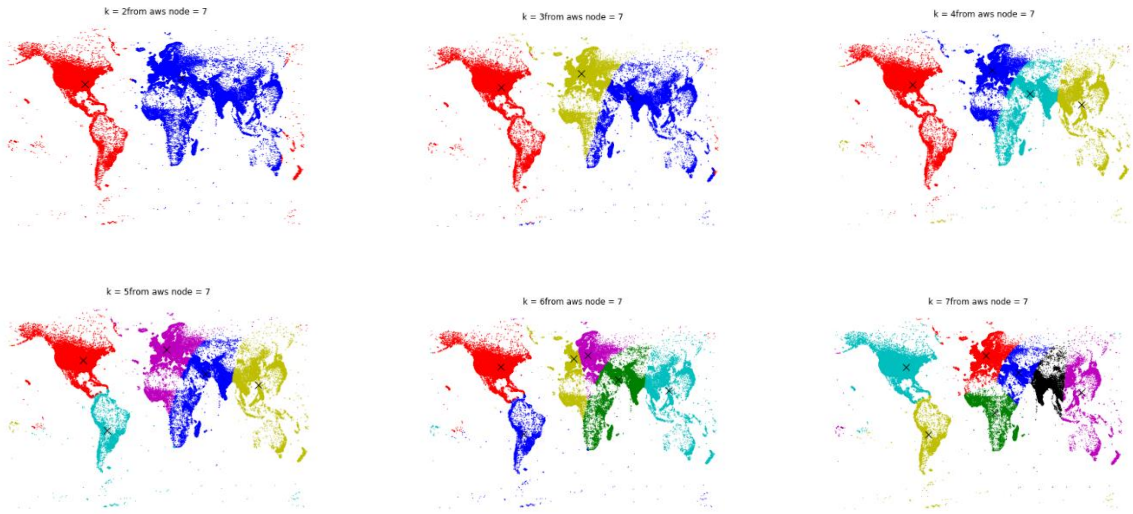

Figure 3.3-1 Different k=2 to 7 k-means for Lat-Long dataset data.

The running timing comparison between local and cluster for this part is showing in the following table (time(hr)):

	K=2	K=3	K=4	K=5	K=6	K=7
Local(4)	0.026	0.031	0.036	0.043	0.075	0.065
Cluster(7)	0.010	0.010	0.016	0.034	0.035	0.031

Table 2.3-2 The running timing comparison between local and cluster

The cluster results for K = 2-7 for the Worldpopcities.txt dataset are showing in the following figure:

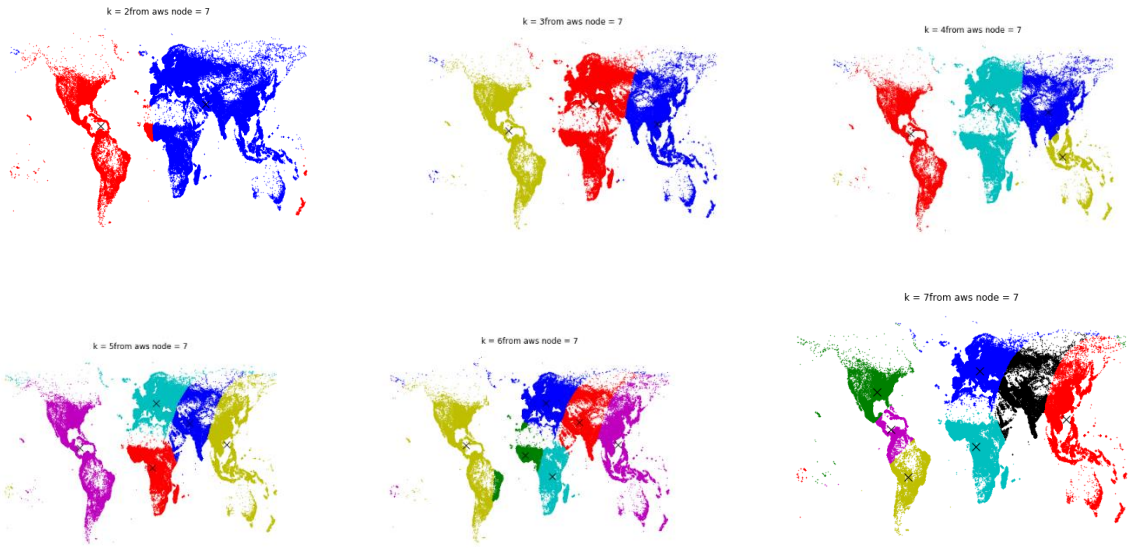

Figure 3.3-2 Different k=2 to 7 k-means for Worldpopcities dataset data.

The running timing comparison between local and cluster for this part is showing in the following table (time(hr)):

	K=2	K=3	K=4	K=5	K=6	K=7
Local(4)	0.12	0.18	0.19	0.21	0.23	
Cluster(7)	0.11	0.083	0.12	0.16	0.21	0.23

Table 2.3-3 The running timing comparison between local and cluster of Worldpopcities data

The cluster results for K = 2-7 using Monte Carlo methods for the Lat-Long dataset are showing in the following figure:

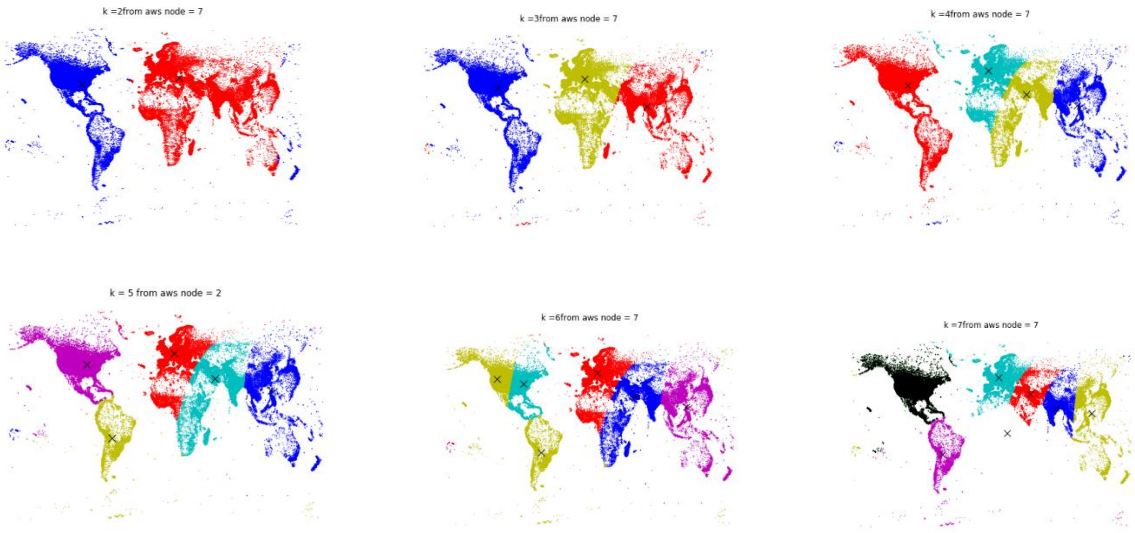

Figure 3.3-5 The cluster results for K = 2-7 using Monte Carlo methods for Lat-Long

The running timing comparison between local and cluster for this part is showing in the following table (time(hr)):

	K=2	K=3	K=5	K=4	K=6	K=7
Local(4)	0.05		0.16	0.05	0.25	
Cluster(7)	0.08	0.11	0.12	0.12	0.11	0.12

Table 3.3-4 The running timing comparison between local and cluster of Worldpopcities data using Monte Carlo

The cluster results for K = 2-7 using MC methods for the Worldpopcities.txt dataset are showing in the following figure:

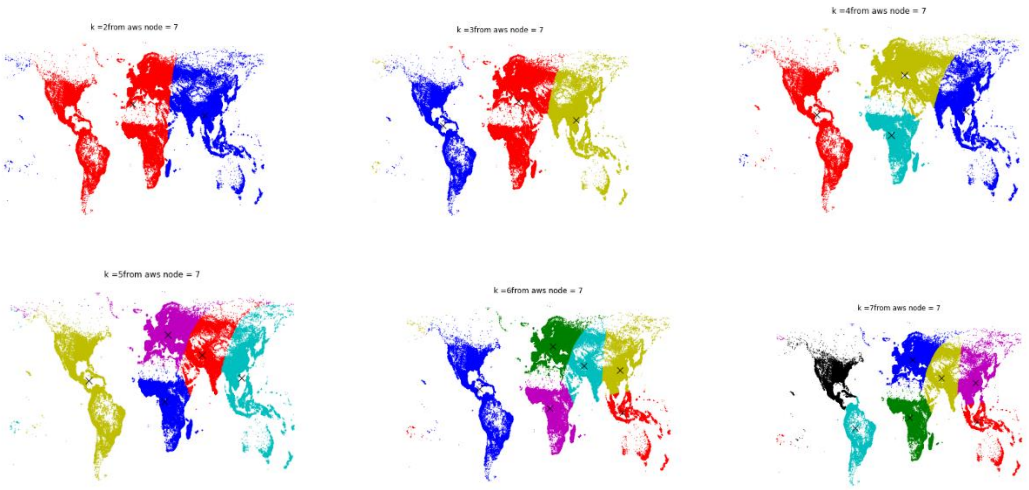

Figure 3.3-5 The cluster results for K = 2-7 using Monte Carlo methods for Worldpopcities

The running timing comparison between local and cluster for this part is showing in the following table (time(hr)):

	K=2	K=3	K=4	K=5	K=6	K=7
Local(4)	0.19	0.22	0.22	0.24	0.27	0.31
Cluster(7)	0.16	0.183	0.191	0.22	0.25	0.23

Table 3.3-5 The running timing comparison between local and cluster of Worldpopcities data using Monte Carlo

Part III. Conclusion

From the results and table we can see, at most time the EMR cluster with 7 nodes (6 work node and 1 master node) has a shorter running time with local machine with 4 cores. However there are some cases they have pretty close running time, and even the local machine can run faster than the cluster. It may be caused by the couple reasons. From the k-means algorithm itself has a very random initial center selection and converging process, so in some case the data gets converged extremely quick because of the better initial centers. In the other hand, the cluster requires some transferring time between work nodes and master nodes, but the local machine requires less of this between different cores.

IV. Discussion

Furthermore, we want to talk about the limitations of k-means. The biggest limitation is k-means can only deal with convex data. Here is my proof:

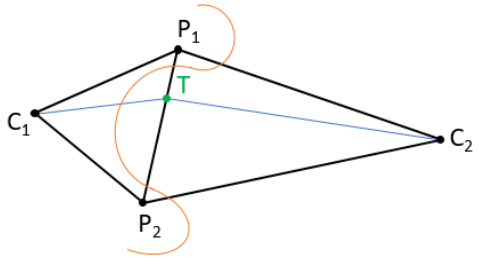

Figure3.2- 12. sketch plot

Let's suppose that k-means can handle with non-convex problem. If we have 2 centers on xy plane called C_1, C_2 and there are 2 points P_1, P_2, $s.t.$ $\left|\overrightarrow{C_1P_1}\right| < \left|\overrightarrow{C_2P_1}\right|$, $\left|\overrightarrow{C_1P_2}\right| < \left|\overrightarrow{C_2P_2}\right|$. Meanwhile according to non-convex definition, we must have at least one point on P_1P_2, called T, $s.t.$ $\left|\overrightarrow{C_1T}\right| > \left|\overrightarrow{C_2T}\right|$. Now we can have: $(0 < k < 1)$

$$\overrightarrow{C_1T} = \overrightarrow{C_1P_1} + \overrightarrow{P_1T} = \overrightarrow{C_1P_1} + k\overrightarrow{P_1P_2} = \overrightarrow{C_1P_1} + k\left(\overrightarrow{C_1P_2} - \overrightarrow{C_1P_1}\right) = k\overrightarrow{C_1P_2} + (1-k)\overrightarrow{C_1P_1}$$

$$\overrightarrow{C_2T} = \overrightarrow{C_2P_1} + \overrightarrow{P_1T} = \overrightarrow{C_2P_1} + k\overrightarrow{P_1P_2} = \overrightarrow{C_2P_1} + k\left(\overrightarrow{C_2P_2} - \overrightarrow{C_2\square_1}\right) = k\overrightarrow{C_2P_2} + (1-k)\overrightarrow{C_2P_1}$$

$$\Rightarrow \left|\overrightarrow{C_1T}\right|^2 = k^2\left|\overrightarrow{C_1P_2}\right|^2 + (1-k)^2\left|\overrightarrow{C_1P_1}\right|^2 + k(1-k)\overrightarrow{C_1P_2}\cdot\overrightarrow{C_1P_1} \qquad (1)$$

$$\Rightarrow \left|\overrightarrow{C_2T}\right|^2 = k^2\left|\overrightarrow{C_2P_2}\right|^2 + (1-k)^2\left|\overrightarrow{C_2P_1}\right|^2 + k(1-k)\,\overrightarrow{C_2P_2}\cdot\overrightarrow{C_2P_1} \qquad (2)$$

By definition, for first 2 terms (2)>(1); for the third term, since $\left|\overrightarrow{C_2P_1}\right| > \left|\overrightarrow{C_1P_1}\right|, \left|\overrightarrow{C_2P_2}\right| > \left|\overrightarrow{C_1P_2}\right|$, $cos\angle P_1C_2P_2 > cos\angle P_1C_1P_2$, so (2)>(1) $\Rightarrow \left|\overrightarrow{C_1T}\right| < \left|\overrightarrow{C_2T}\right|$. But this contradictory to non-convex assumption, in which T is closer to C_2. So, we can say that k-means cannot handle with non-convex problem.

This weakness of k-means in fact causes plenty of problems in our clustering. For instance, clusters on Europe-Asia Continent is like strip by strip, which looks quite unrealistic. For better details, the only way is increasing k, but this could make the whole picture drastically change. In a word, it is almost impossible for k-means to specify on locality and still hold a globality.

Reference

[1] https://classes.cec.wustl.edu/cse427/lat_longs.zip
[2] https://www.maxmind.com/zh/free-world-cities-database
[3] https://en.wikipedia.org/wiki/Haversine_formula
[4] Yan M, Ye K. Determining the number of clusters using the weighted gap statistic[J]. Biometrics, 2007, 63(4): 1031-1037.