K-means Clustering Algorithm and Application

Jane Han, Huilin Xu, Alan Zhang, Zubin Zhang

{hanzheyu, huilinlynn, xinyuzhang, zhangzubin}@wustl.edu

1 Introduction

The goal of our project is to study the k-means clustering algorithm and its applications to clustering geolocation dataset in real life. The clustering of data is to group data by how similar they are to each other. For example, we could cluster students into groups by their major, or group customers by their rating on movies. The knowledge of which group an object belongs could be useful in many cases in real life. Grouping customers based on their purchasing transactions could help develop a more targeted marketing such as recommending items based on the purchasing behaviors of other customers in their group. Given the locations of all warehouses, we could arrange a more efficient shipments to save time and money. Or we could classify documents based on their topics to have a more organized database.

K-means clustering algorithms [1] is one of the multiple methods to cluster dataset. It clusters data into k groups based on their distance to each other. It needs a predefined k for the number of clusters and a distance function. The algorithm first initializes k centroids. Then it iteratively assigns all data points to their closest clusters and update the position of the centroids until it hits a certain stopping criterion. Because this is an unsupervised learning algorithm, there is no deterministic answer of setting k, and it is the programmer's job to choose the initial centroids and to define the stopping criterion.

In this project, we developed a k-means algorithm in Spark [2]. At the beginning, three small datasets containing the geological locations, were used to interpret the runtime, best choice of k, etc. To further study the algorithm's application to the real world, we used the dataset of the New York yellow cab pickup and drop off information in hope to learn more information hidden in the data such as the hot spots for cab services. As the data size greatly increased, we also experienced the cloud storage service using the Amazon Simple Storage Service (Amazon S3) and the data processing and analysis tool on the Amazon Elastic MapReduce (Amazon EMR). The k-means algorithm gave us surprisingly good results and provided us with many insights to understand this real world big dataset.

2 Approach

2.1 Algorithm Description

Our k-means clustering program takes 4 parameters: input directory, output directory, number of clusters k, and a distance function (Euclidean or great circle distance). An RDD is created to read data from input directory and to hold the preprocessed data which is in the format of (latitude, longitude). This RDD is persisted so that it is accessible for later iterative use. The initial k centroids are chosen randomly from the dataset. Then the program starts to iteratively assign data points to clusters and update the centroids until the positions of the centroids converge.

A spark job is created for each iteration. Each job has two stages. The first stage is to map each data entry from the persisted RDD [3] to a pair of (cluster index, data location, 1). Each data point is assigned to the cluster with the closest centroid. The second stage is to reduce data by key so that each entry is a cluster index with its corresponding sum and total count of all data points assigned to it. A map function is followed by this reduce to get the mean value of all data points in each cluster. And this becomes the updated centroids of the current iteration.

After each iteration, we calculate the sum of distance changed in centroids locations compared to their previous locations. If the change is less than 0.001, we claim that the centroids positions have converged. Finally, all data points from persisted RDD are assigned to the final clusters. The position of each centroid and data points in its group are saved to the output directory.

We applied this k-means algorithm with different distance functions and k to datasets and evaluated the algorithm by direct visualization and run-time comparison. The results and analysis are given in section 3.2 and 4.2.

2.2 Distance Function

Our program implements two measures of distance: Euclidean and great circle [4]. Euclidean distance between (p_1, p_2) and (q_1, q_2) is defined as

$$\sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$

This equation does not consider the spherical nature of the Earth. When two locations are very close to each other, this should be a good measure. However, considering two points are relatively far away, their actual distance should be longer than their Euclidean distance. Thus, the Great circle measure should be considered when points are geologically far away.

Great circle distance d between two spherical coordinates $(\text{lon}_1, \text{lat}_1)$ and $(\text{lon}_2, \text{lat}_2)$ is defined as

$$\begin{aligned} \text{dlon} &= \text{lon}_2 - \text{lon}_1 \\ \text{dlat} &= \text{lat}_2 - \text{lat}_1 \\ a &= sin^2(\frac{\text{dlat}}{2}) + cos(\text{lat}_1) \\ &\quad * cos(\text{lat}_2) * sin^2(\frac{\text{dlon}}{2}) \\ c &= 2 * arcsin(min(1, \sqrt{a})) \\ d &= R * c \end{aligned}$$

where R is the radius of the Earth. The distance between any two points is now along the surface of the ground, which is a more realistic and meaningful measurement.

3 Part I. Small Data

3.1 Data

Three small datasets were used to test our program on pseudo cluster. The first dataset (device_status) was information about mobile devices on the Loudacre's network which covers the western USA. Only the location of devices was used for the project. After filtering, there were 431857 data entries left in this dataset. The second dataset (sample_geo) was synthetic location data within the USA. It had 4985 entries. The third dataset (DBpedia) was the locations of all Wikipedia articles worldwide [5]. It had 450151 entries. For all datasets, the location of each data point was in the format of latitude and longitude.

3.2 Result and Discussion

Figure 1 and 2 are the visualization for the 5 clusters of device_status dataset using Euclidean distance and great circle distance. The clustering results look mostly the same. The slight difference in the central most likely was due to the randomness of the initial centroids selection. This result makes sense since California is a relatively small area comparing to the whole world. We can assume the surface of California is flat so that the Euclidean and great circle distance between two points in California are basically the same.

Figure 3 to 6 show the clustering on the sample_geo dataset. Figure 3 and 4 show that when k = 2, there was no difference in the clustering between two distance measures. They both split the data into western US and eastern US. When k was set to 4, the clustering based on Euclidean function (Fig. 5) looked like cutting the data vertically into 4 parts, which is not the same when using great circle function (Fig. 6). In this case, using great circle resulted in a slightly better clustering since it is more reasonable to group Florida to the east coast instead of mid-east with Tennessee. The territory of the US is a large enough so that the curvature of its surface cannot be ignored. And this is the reason that there's a significant difference between two points distance based on the two distance measures. Especially because the US is in the Northern Hemisphere, the difference between two measures is bigger for the data points in the Northern part of the US.

Figure 7 to 24 present the clustering of DBpedia dataset for k from 6 to 14 using both Euclidean distance and great cicle distance. From visual inspection, clustering with great circle distance function and setting k equal to 10 seem to be the optimal way to group the data points as shown in figure 16. There are 3 centroids in North America, 3 centroids in Europe, one in South America, one in Africa, one in western Asia, and one in eastern Asia and Australia. The k = 12 clustering with Euclidean distance resulted in similar centroids only with some difference in the range of each cluster especially around the borders of our map. Note that the x and y axes of our visualization are longitude and latitude. The top and the bottom of our map are the north and south poles. The left and right edges of our map should be viewed as connected. Since the great circle distance between two geo-locations takes it into consideration that the Earth is globe while Euclidean distance doesn't, we can see that clusters based on great circle functions are more realistic. For example, in figure 14, data points along the top edge of our map are in the same cyan-colored cluster, and data points along the bottom edge are in the dark pink cluster. Although they seem to be spread out on this map, they are actually very close as they are in the poles. This is something we don't see in the Euclidean distance clusters.

Another thing to notice is that, as k increases, there are much more centroids crowded in Europe comparing to other continents. This phenomenon suggests that data points in Europe are really dense. We can infer that although Europe is a small continent, they have more contributions to wikipedia articles than people from other continent such as South America and east Asia. More generally, we could claim that people from different area of the world are not equally involved in wikipedia based on our observation of this dataset's clustering. This claim aligns with the report on wikipedia use across the world from 2014 [5] which states that Europe have the most users, North America have second most users, and the other continents had much fewer.

Table 1 compared the average (from 3 runs) local run-time of clustering the three datasets for implementations with and without persisting the RDD. As a control across trials, we set k = 10, number of threads to be 4, and the distance function to be great circle for all runs. Across the datasets, we noticed that it took much more time to cluster device_status and DBpedia dataset than to cluster the geo_sample dataset. This makes sense since device_status and DBpedia dataset have more data entries as discussed in section 3.1. For the two larger datasets, implementation without persisting the initial raw data RDD took more time than with persist. Without persisting the RDD, each iteration in the k-means algorithm repeatedly read data from file and filtered data to the desired form. This could take significantly more time for a large dataset.

	Persist	No Persist
Device_status	64	110
Geo_sample	14.33	14.33
DBpedia	104	142

Table 1: Local Run-time Comparison in seconds.

4 Part II. Big Data

4.1 Data

We take the yellow car trip data in one month in New York from Kaggle [6] as our original dataset to see the popular pick-up region and analyze whether it is in accordance with the realistic conditions in New York. Its original size is 1.99 GB with 12,748,986 records and 19 different features, such as VendorID, trip_distance, passenger_count, pickup_datetime, pickup_longitude, pickup_latitude, tip_amount, etc.

This dataset was pre-processed using the Pandas package to keep the "pickup_latitude" and "pickup_longitude" as our main interest. Specifically, when processing the data, we dropped records with some missing values and filtered the latitude out of range [39,41] and longitude out of range [-75,-73]. This is the map range for the New York area. Due to the quality of the dataset, we have seen wrong values more than 400, which is absolutely out of New York. After filtering, we left with 12,504,845 records. And gave each data an unique id and saved the processed data into a tsv file.

4.2 Cloud Configuration

Firstly, we created an AWS EMR cluster with an m5.xlarge machine for the Master Node and two m5.xlarge machines for the Core Workers. After identifying the public IP of master node, we connected computer to the master node via SSH and transferred the PySpark program to the master node. Also, we uploaded our dataset to the S3, and set the the output path to S3 so that we can run the job with the command line and checked the results in S3 directly.

4.3 Visualization

To better interpret the result, we wish to plot the data point on the actual New York map. Therefore, we use Folium [7] to better visualize our result. Folium makes it easy to visualize data that has been manipulated in Python on an interactive Leaflet map. For each cluster, we assign it an unique color to make the clustering result more vivid.

4.4 Result and Discussion

Figure 25 to 31 are the clustering of yellow-cab pickup geo-locations in NYC with k from 2 to 8. We used the Euclidean distance function only because based on our previous discussion the great circle distance and the Euclidean distance performs similarly on small area. As shown in figure 25, when clustering the data into two groups, there is a very condensed cluster around Manhattan (green) and a wider and sparser one in Queens (blue). As we increased the number of k, The huge cluster around Manhattan broke down into smaller clusters while the sparse cluster in Queens stayed the same. When k was set to 7 (figure 30), the clusters outside Manhattan started to change. Based on our knowledge about New York, we decided that $k = 7$ was the best choice. To help understand the clustering results, table 2 describes the locations of the centroid of each cluster. The pick up locations are centered around airports, and iconic and popular sites within Manhattan such as the Met and SOHO.

	Color	Nearest famous spot to centroid
1	Dark blue	The Metropolitan Museum of Art
2	Pink	Rockfeller Center
3	Gray	Madison Square Park
4	Light blue	SOHO
5	Dark Green	Manhatten Bridge
6	Yellow	LaGuardia Airport
7	Green	John F. Kennedy International Airport

Table 2: Locations of centroids for the 7 clusters.

From Table 2, these locations of centroids corresponding to 7 clusters are transportation hubs, Tourist attractions and popular detinations, i.e. the busiest points in New York. Thus, the cab company could send much more cabs around these areas to satisfy larger demand and achieve greater profits. Also, government could consider setting more pick-up stop in those locations, offering more convenience to both taxi drivers and passengers.

We also recorded average run-time after running our algorithms on this dataset on cloud for three rounds with each different k. As Table 3 shows, the average run-time increases when k goes up. In our implementation, to calculate the sum of the centroids difference between two consecutive rounds. This sum will increase as k goes up while our converge condition is set a fixed value, 0.001. In another word, it becomes harder and harder to make this total difference lower than the threshold, which directly results in a longer run-time.

K	Average running time(min)
2	2.1
3	2.3
4	4.9
5	4.1
6	7
7	10
8	6.6

Table 3: Cloud runtime of clustering NYC cab pickup geo-locations for k from 2 to 8.

5 Conclusion & Future Work

In this project, we developed a k-means clustering algorithm and investigated its applications to both small and large datasets. During this process, we looked into the effect of k and distance function on the clustering, and analyzed the run-time across different k, data size, and implementation. When clustering geo-location data, the performance of the distance functions depends on the scope of data. When the scope is small like California or New York City, the Euclidean distance and the great circle distance are very similar since we can ignore the curvature of the small surface. One exception is the two poles. Due to our measure system of longitude and latitude, geo-locations in the North/South pole could have very different longitude and latitude values even though they are in a close area. Therefore, if the dataset we are interested in are mostly located in the poles, we would prefer great circle distance to Euclidean distance. When the scope is very large such as the U.S. or the globe, great circle distance gives a much better clustering since it considers the radius of the Earth.

As for determining the optimal k for clustering, we only evaluated the resulted clustering through visual inspection with references to the maps. We found that in general, the preferred k is larger for larger scope (wider in range) datasets. If we want a more mathematical way of choosing the optimal k, we could try the Elbow-test by drawing the average diameters of the clusters versus number of k. The average diameter will decrease as k increases. The k corresponding to the turning point in that slope will be the optimal k we are looking for.

We found that runtime of our clustering algorithm was longer for larger sized dataset, higher k, and implementation without persisting data points RDD. It makes sense that more records requires more computations when assigning them into clusters. The fact that it took longer without persisting RDD also agreed with our hypothesis. Without persist, the program repeatedly load and filter the dataset during each iteration. This could take significantly more time if the dataset was large or hard to converge. The number of k mattered due to out implementation of calculating the converge condition. As discussed in section 4.4, our converge condition didn't scale with k, which makes the algorithm harder to converge as k goes up. We can try other implementations such as averaging the changes in distance of all centroids to avoid this problem.

During our analysis of big dataset, we faced some difficulty in visualization. Since our NYC cab dataset contained over 12 million records, it was impossible to draw all data points on the NYC basemap. We tried the visualization with multiple numbers of sampled data points and chose the number of data to sample that was both possible to draw and representative of the clusters. We didn't have difficulties regarding the clusters' converging. This is probably due to the fact we were using clusters on AWS and also the characteristics of our dataset. It was very condensed in a small area. Although the data size was huge, the area it covered was small. Especially most data are in Manhattan which is rectangular in shape. This could be the reason that it wasn't so hard to cluster our data points.

Using k-means algorithm to group dataset can reveal a lot of hidden information regarding the data. For geo-locations, the clustering shows the closeness between data points and the density of data in different areas. Combined with other information outside the dataset, even more things can be implied. For example, the clustering of DBpedia data suggested the unequal participation on Wikipedia across different parts of the world as we discussed in section 3.2. Since we knew that Asia have more population than Europe. The fact that there were more clusters in Europe than in Asia implied that the Europeans were more likely to contribute Wikipedia articles than the Asians. For our NYC cab pick up location dataset, the clustering gave information about areas in NYC with high cab demands. The position of centroids were the most popular spots among that cluster area. These information could be beneficial to the cab company in their management and marketing. In short, with the great power of cloud computing and advanced data processing framework and tools, we are able to implement and run the k-means algorithm in an effective and efficient way. This greatly help us reveal and infer many geological features and human behaviors, which are hidden in the dataset.

References

[1] A. Trevino, "Introduction to k-means clustering," 2016.

[2] J. VanderPlas, "Python data science handbook."

[3] D. TEAM, "Rdd persistence and caching mechanism in apache spark."

[4] L. W. Nyman, "Geographic information systems faq," 1997.

[5] M. Graham, "Geotagging reveals wikipedia is not quite so equal after all," 2014.

[6] Kaggle, "aaic yellowtaxi demand prediction," 2015.

[7] A. Ivan, "Spatial visualizations and analysis in python with folium," 2018.

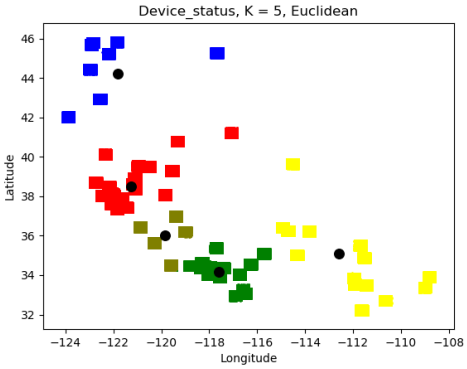

Figure 1: 5 clusters of device_status dataset using Euclidean distance

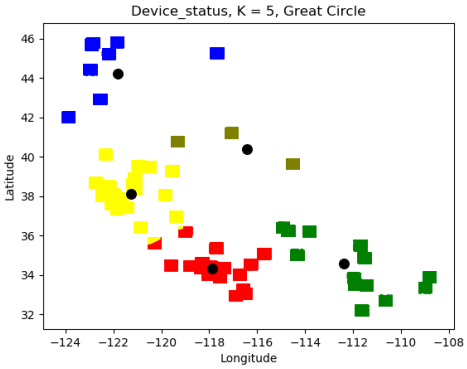

Figure 2: 5 clusters of device_status dataset using Great Circle distance

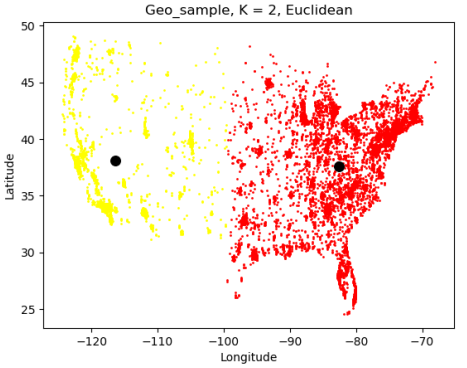

Figure 3: 2 clusters of geo_sample dataset using Euclidean distance

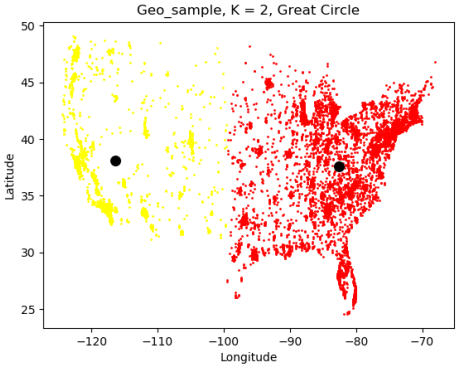

Figure 4: 2 clusters of geo_sample dataset using Great Circle distance

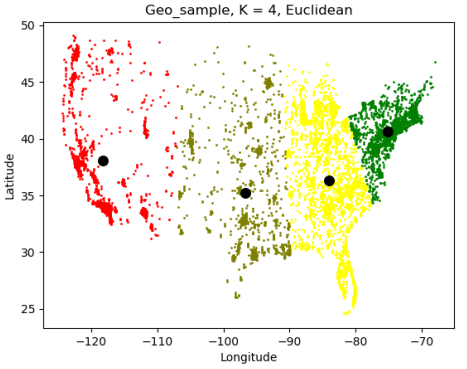

Figure 5: 4 clusters of geo_sample dataset using Euclidean distance

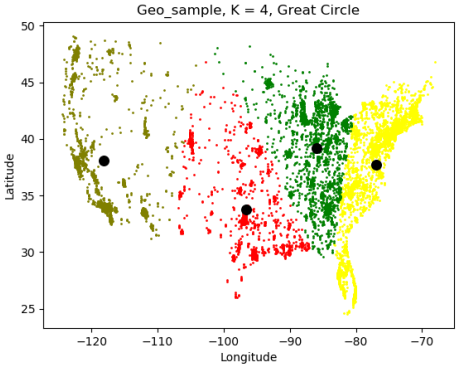

Figure 6: 4 clusters of geo_sample dataset using Great Circle distance

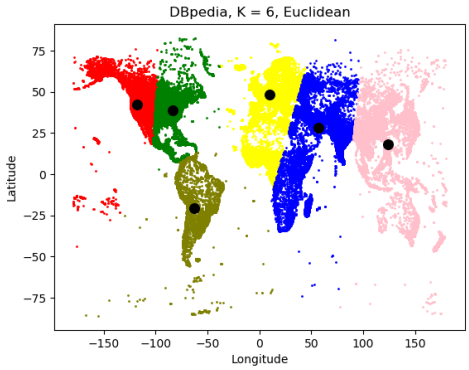

Figure 7: 6 clusters of DBpedia dataset using Euclidean distance

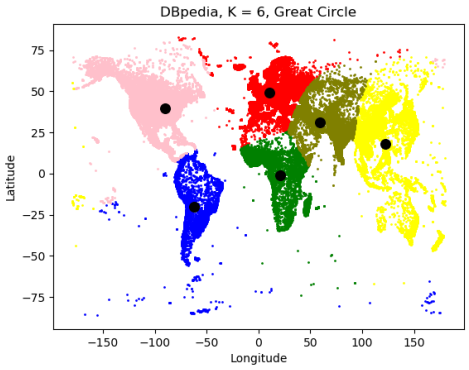

Figure 8: 6 clusters of DBpedia dataset using Great Circle distance

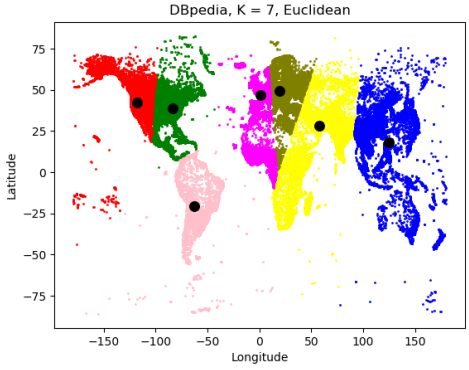

Figure 9: 7 clusters of DBpedia dataset using Euclidean distance

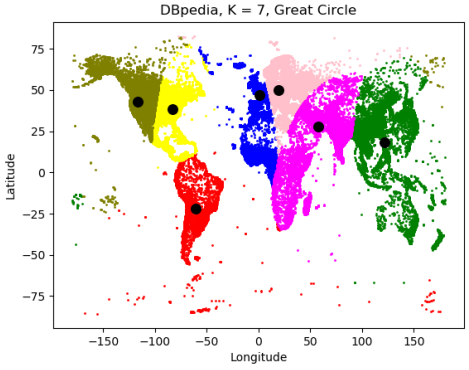

Figure 10: 7 clusters of DBpedia dataset using Great Circle distance

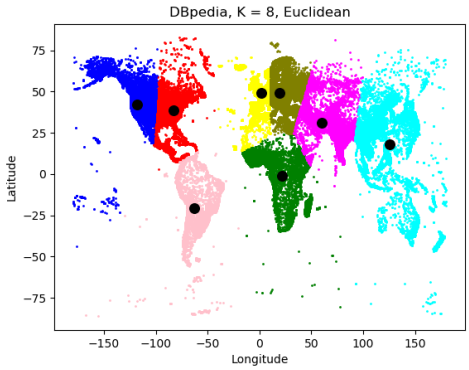

Figure 11: 8 clusters of DBpedia dataset using Euclidean distance

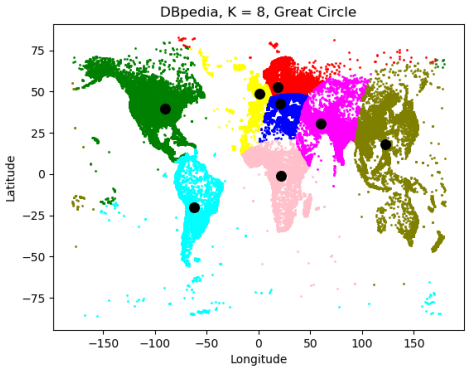

Figure 12: 8 clusters of DBpedia dataset using Great Circle distance

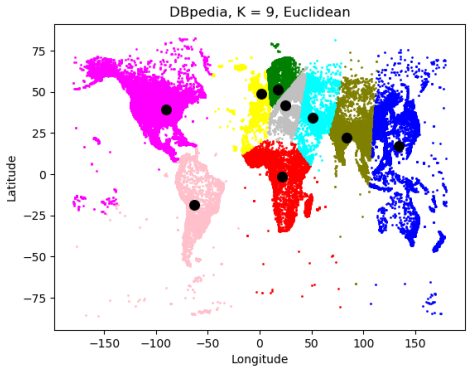

Figure 13: 9 clusters of DBpedia dataset using Euclidean distance

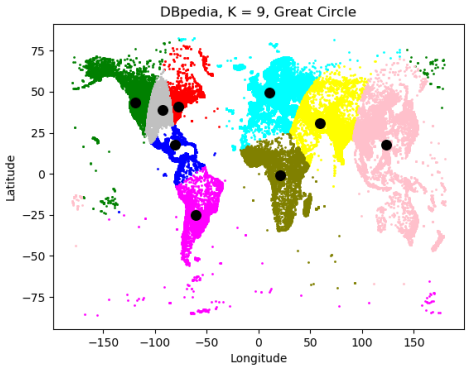

Figure 14: 9 clusters of DBpedia dataset using Great Circle distance

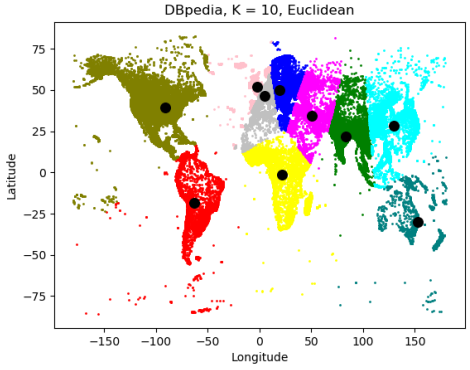

Figure 15: 10 clusters of DBpedia dataset using Euclidean distance

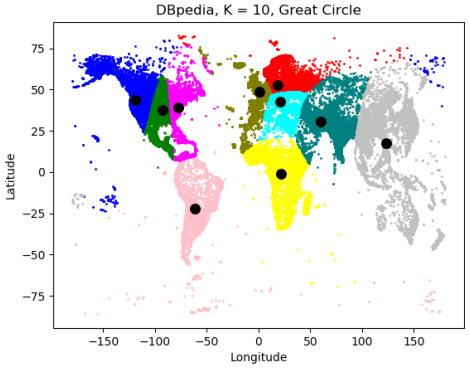

Figure 16: 10 clusters of DBpedia dataset using Great Circle distance

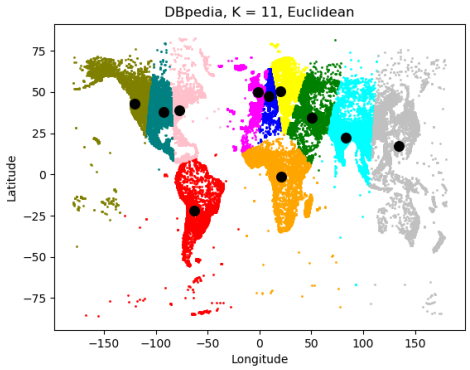

Figure 17: 11 clusters of DBpedia dataset using Euclidean distance

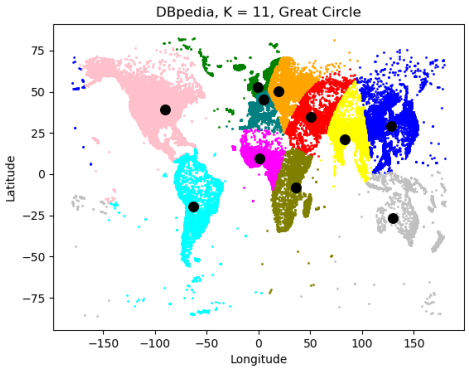

Figure 18: 11 clusters of DBpedia dataset using Great Circle distance

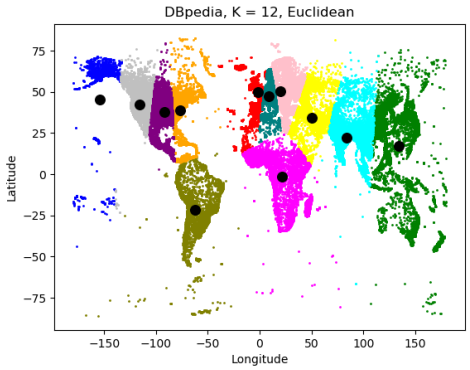

Figure 19: 12 clusters of DBpedia dataset using Euclidean distance

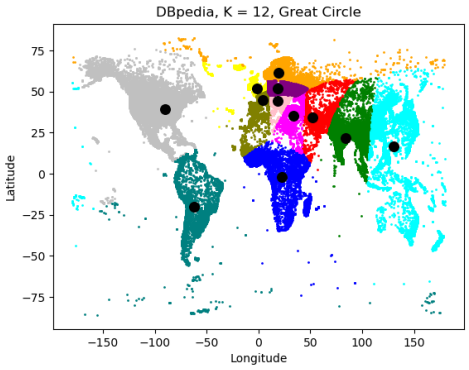

Figure 20: 12 clusters of DBpedia dataset using Great Circle distance

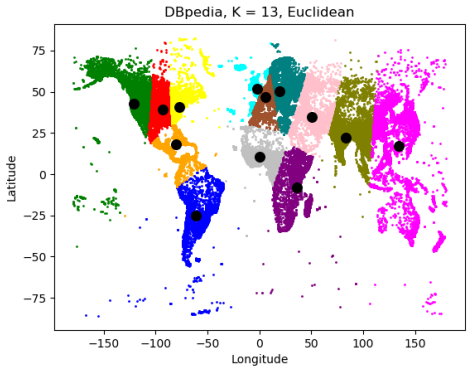

Figure 21: 13 clusters of DBpedia dataset using Euclidean distance

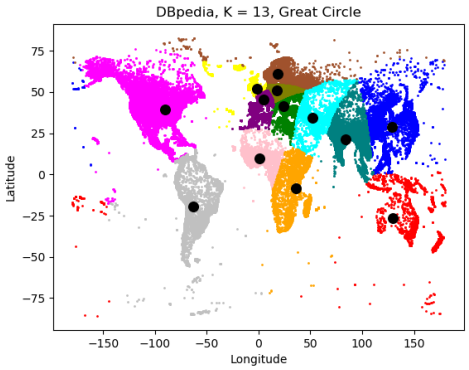

Figure 22: 13 clusters of DBpedia dataset using Great Circle distance

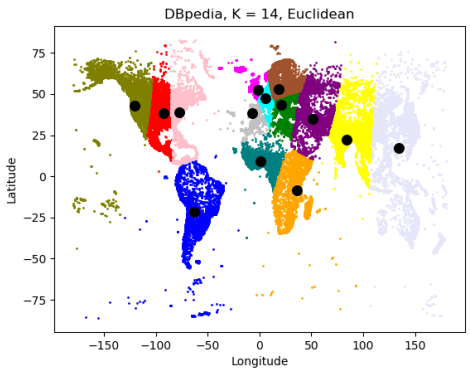

Figure 23: 14 clusters of DBpedia dataset using Euclidean distance

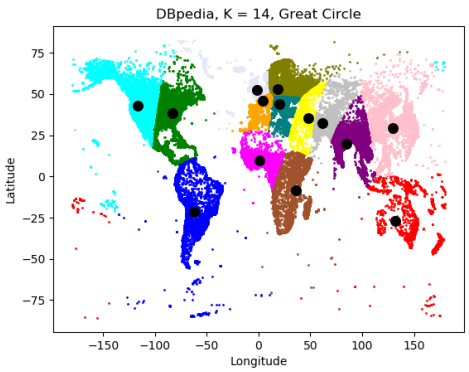

Figure 24: 14 clusters of DBpedia dataset using Great Circle distance

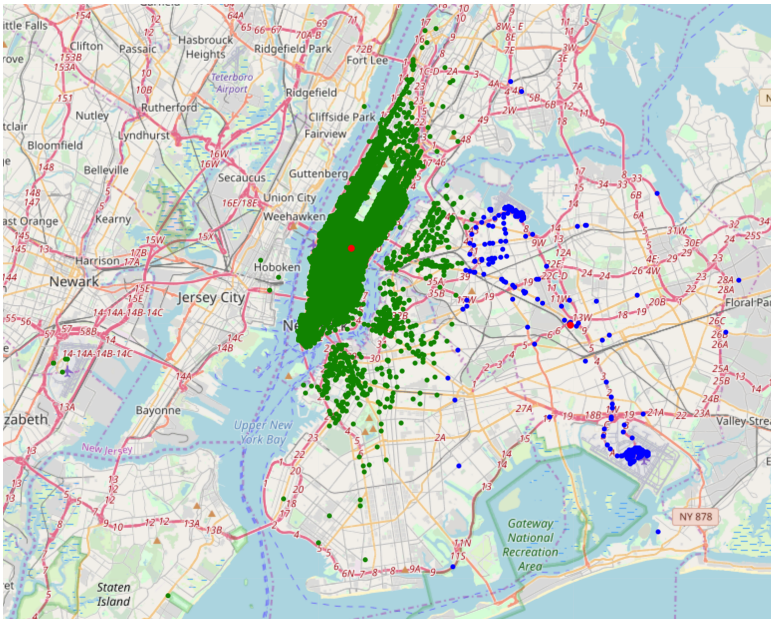

Figure 25: 2 clusters of yellow car trip data in New York

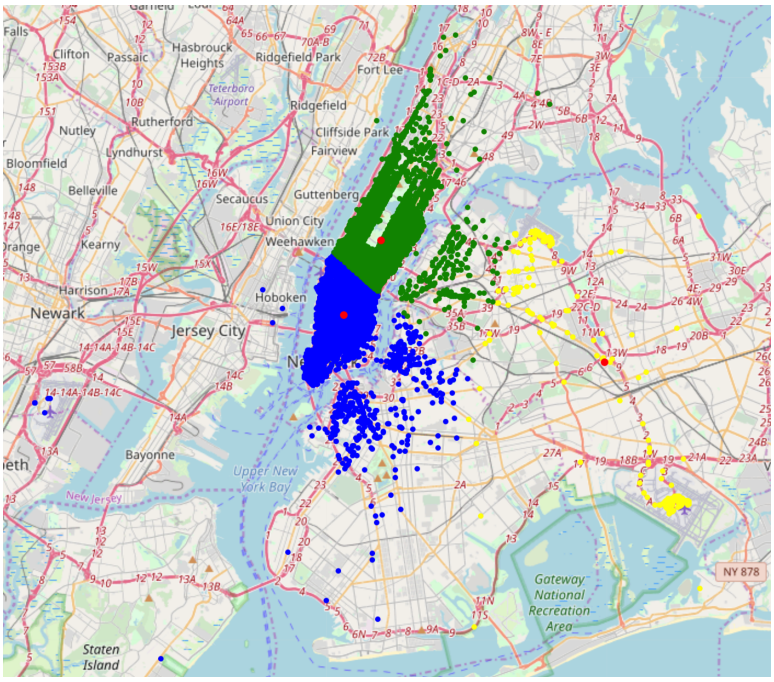

Figure 26: 3 clusters of yellow car trip data in New York

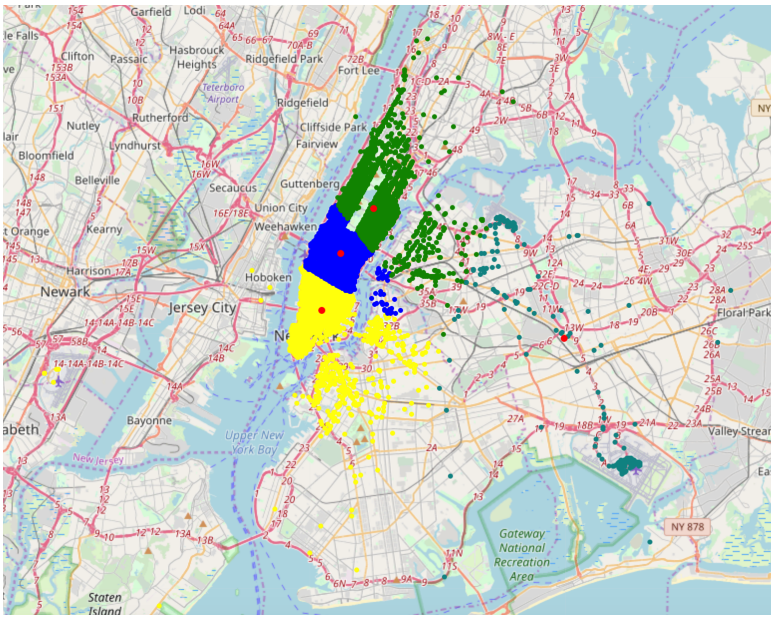

Figure 27: 4 clusters of yellow car trip data in New York

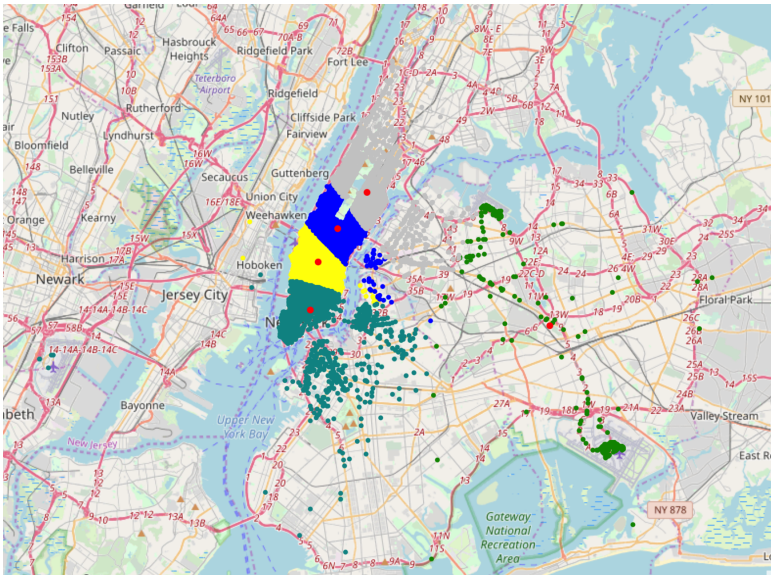

Figure 28: 5 clusters of yellow car trip data in New York

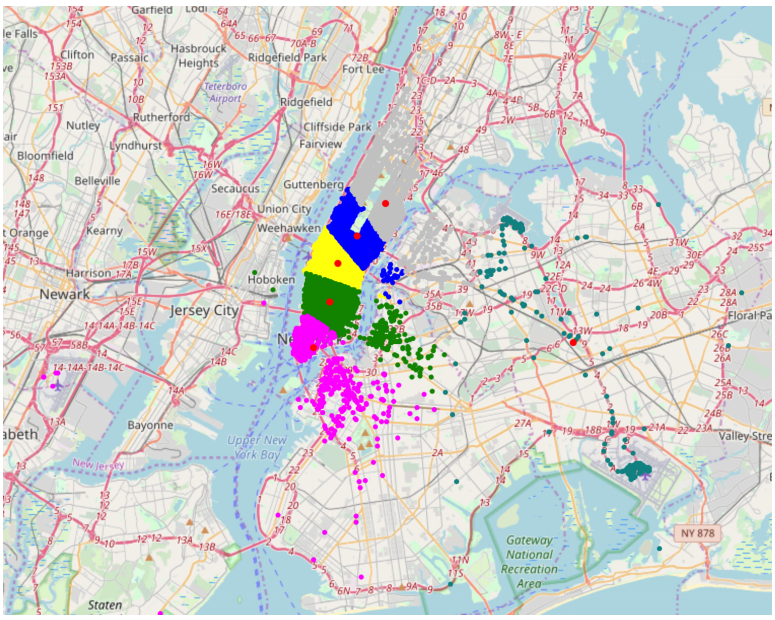

Figure 29: 6 clusters of yellow car trip data in New York

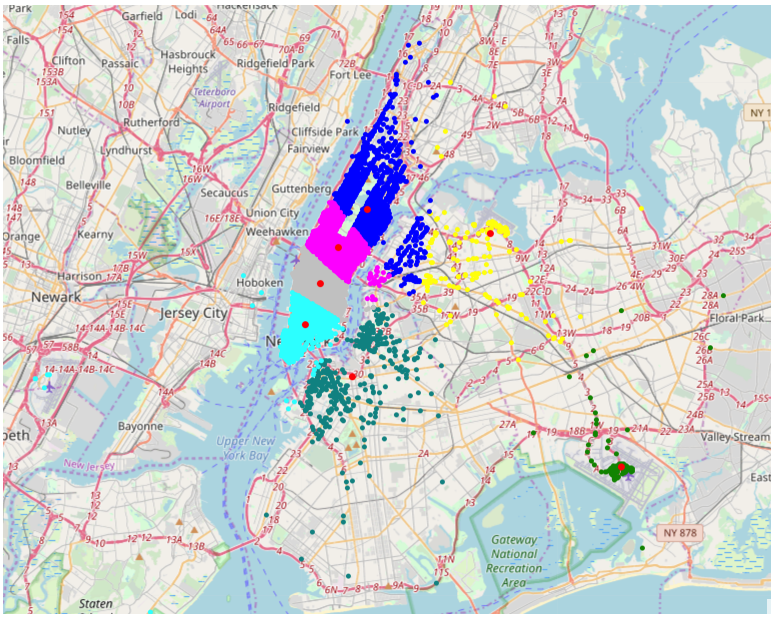

Figure 30: 7 clusters of yellow car trip data in New York

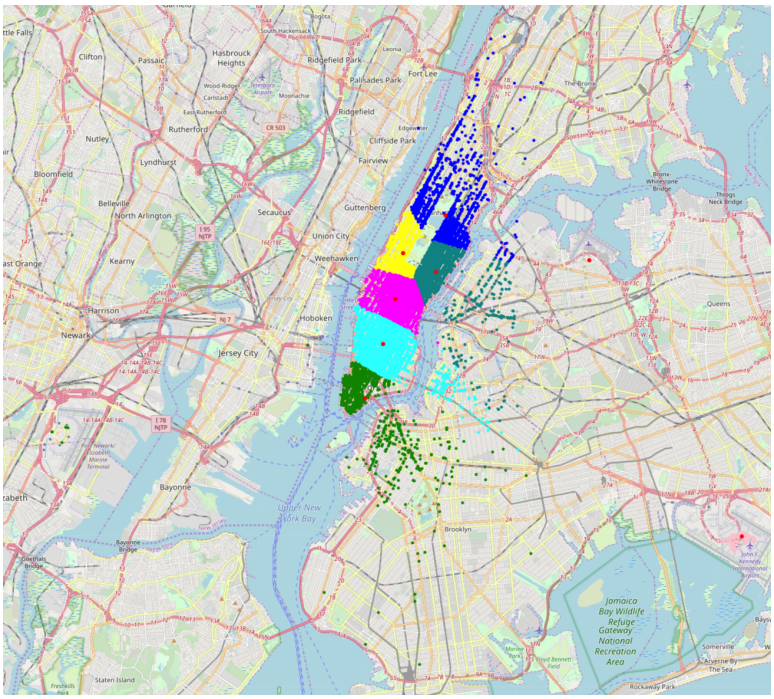

Figure 31: 8 clusters of yellow car trip data in New York