K-Means for Geo-Location Clustering in Spark, with extension into Yelp Reviews and UFO Sighting

December 13, 2017

Hongjin Lin

Shuyu Qin

Shuo Wu

Alan Zhang

Washington University in St. Louis

Problem 1: System Exploration

We begin with an exploration of Spark. In this section, we describe how we developed our familiarity with the tools that we will rely on later. First, we read the Spark Programming Guide, browsed through the Hue File Browser, and gained comfort with the Python Documentation, as Python will be our language of choice. Then, we practiced with two datasets: “purplecow.txt” and “frostroad.txt”. Specifically, we used these datasets as input to a toy application. We developed a simple Word Count application for this purpose. With it, we 1) monitored the jobs and stages (in Spark Application UI), 2) manipulated File System settings (in command line: see Table 1), and 3) assessed Execution (in YARN Resource Manager). Because this “practice” is not relevant to this paper, we conducted it for milestone 1 but omit the details here for clarity-sake. Parts of “wordcount” results are shown in Figure 1.

```
('all', 1)
('just', 1)
('less', 1)
('both', 2)
('Had', 1)
('yellow', 1)
('leads', 1)
('not', 1)
('trodden', 1)
('difference', 1)
('had', 1)
('should', 1)
('better', 1)
('to', 1)
('wood', 2)
('black', 1)
('sorry', 1)
('has', 1)
('them', 1)
('Oh', 1)
('far', 1)
```

```
('a', 1)
('be', 1)
('I', 4)
('ve', 1)
('cow', 1)
('rather', 1)
('never', 2)
('But', 1)
('one', 2)
('to', 1)
('see', 2)
('purple', 1)
('tell', 1)
('can', 1)
('you', 1)
('seen', 1)
('anyhow', 1)
('than', 1)
('hope', 1)
('d', 1)
```

Figure 1: “Wordcount” results (left: frostroad.txt, right: purplecow.txt)

Cluster Mode	spark-submit –master yarn-cluster file.py output.txt
Client Mode	spark-submit –master yarn-client file.py output.txt
Local Mode	spark-submit –master local file.py output.txt
N-parallel Threads	spark-submit –master local[N] file.py output.txt

Table 1: Commands to change execution mode (local, local multi-threaded, cluster). P1 Step 4

Problem 2: Data Preparation

<u>Prepare data</u>

Now familiar with distributed processing using Spark, we transition into a specific use case. The focus of this paper is to apply K-Means algorithms to cluster location data. More on K-Means clustering will be explained in the next section. For now, we describe our preparation for this objective. We practiced with exercises on 3 different location-based datasets titled: 1) Synthetic Location, 2) Loudacre Mobile Device Location (see Table 3 and Figure 2 for Data Scrubbing example), and 3) DB-pedia Location. See Table 2 for descriptions of all location datasets.

Dataset	Location Subject	Location Format	Size of Dataset	Why Use?
Synthetic Location	Random selection of geographic coordinates in 48 states (statistical-research.com)		5000 records	Small dataset. No extraneous data columns. Good for practice.
Loudacre Mobile Device Location	mobile device product details and manufacturer location	Latitude and Longitude Coordinates (same for all)	130,000+ records	Moderate size. Requires pre-processing of data fields. Good for practice .
DB-pedia Location	Geotagged Wikipedia articles (ex: Fig 1)		450,000+ records	Large size. Good for full-scale testing.

Table 2: Three Location Datasets used

Figure 2: Example of Geotagged Wikipedia Article: Washington University in St. Louis

Action to Practice	**Command (example)**
Load	Data=sc.textFile(“file:///home/training/training_materials/data/devicestatus.txt”)
Replace Delimiter	.map(lambda line:line.replace(“|” , ”,”)
Filter Records	.filter(lambda fields:len(fields)==14)
Separate Fields	.map(lambda fields:fields[0]+”,”+fields[1]+”,”+fields[2]+”,**manufacturer**”+fields[3]+”,**model**”+fields[4]+”,”+fields[5])
Save New File Format	Data.saveAsTextFile(“loudacre/devicestatus_etl”)

Table 3: Pre-processing (Scrubbing) Practice Example: Loudacre Device Location Data

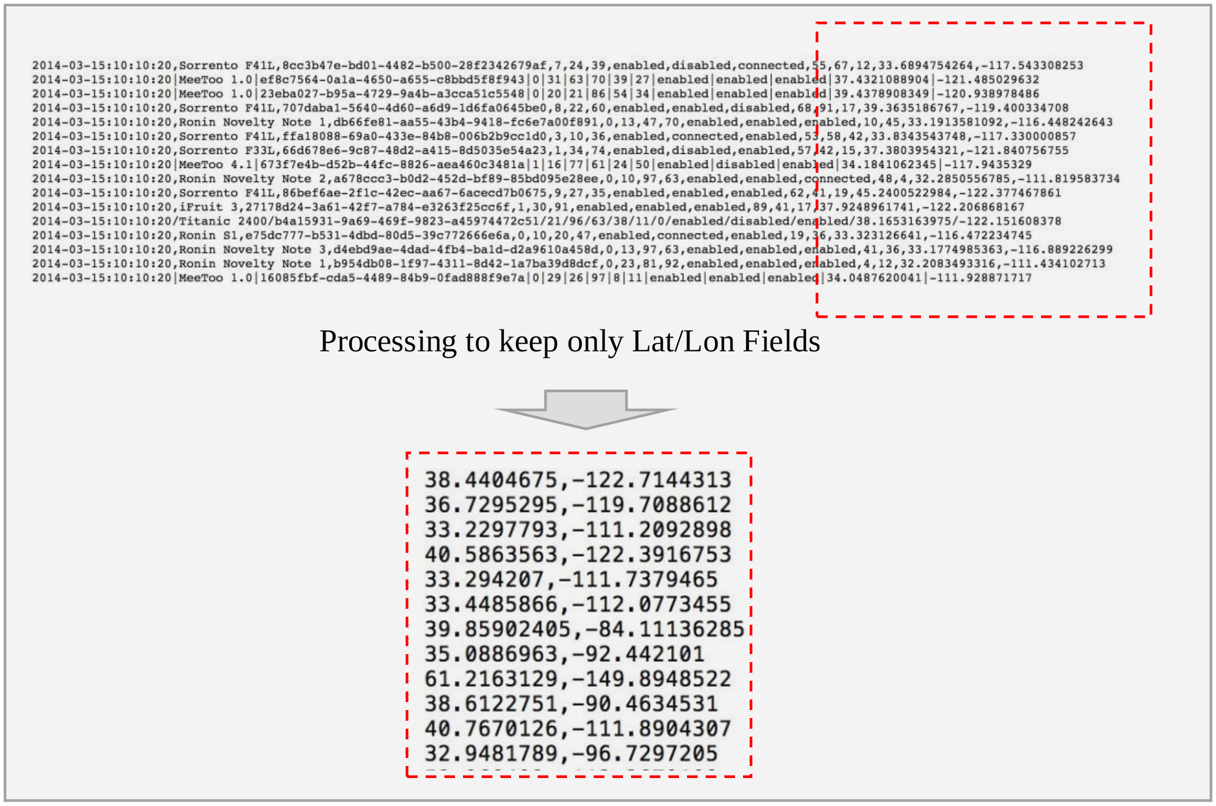


Figure 3: Pre-processing Practice Example: Loudacre Device Location Data

Visualize data

After getting the latitude and longitude of each set of data, we can visualize the location data in the global map using “ploty”. Figure 3 shows the visualization of device status data; Figure 4 shows the synthetic location data, and Figure 5 shows the DB-pedia location data in US region. In order to get the US region data, we filter the raw data by the bounding box, the bounding box is: [latitude: 25~49, longitude: -130~-70].

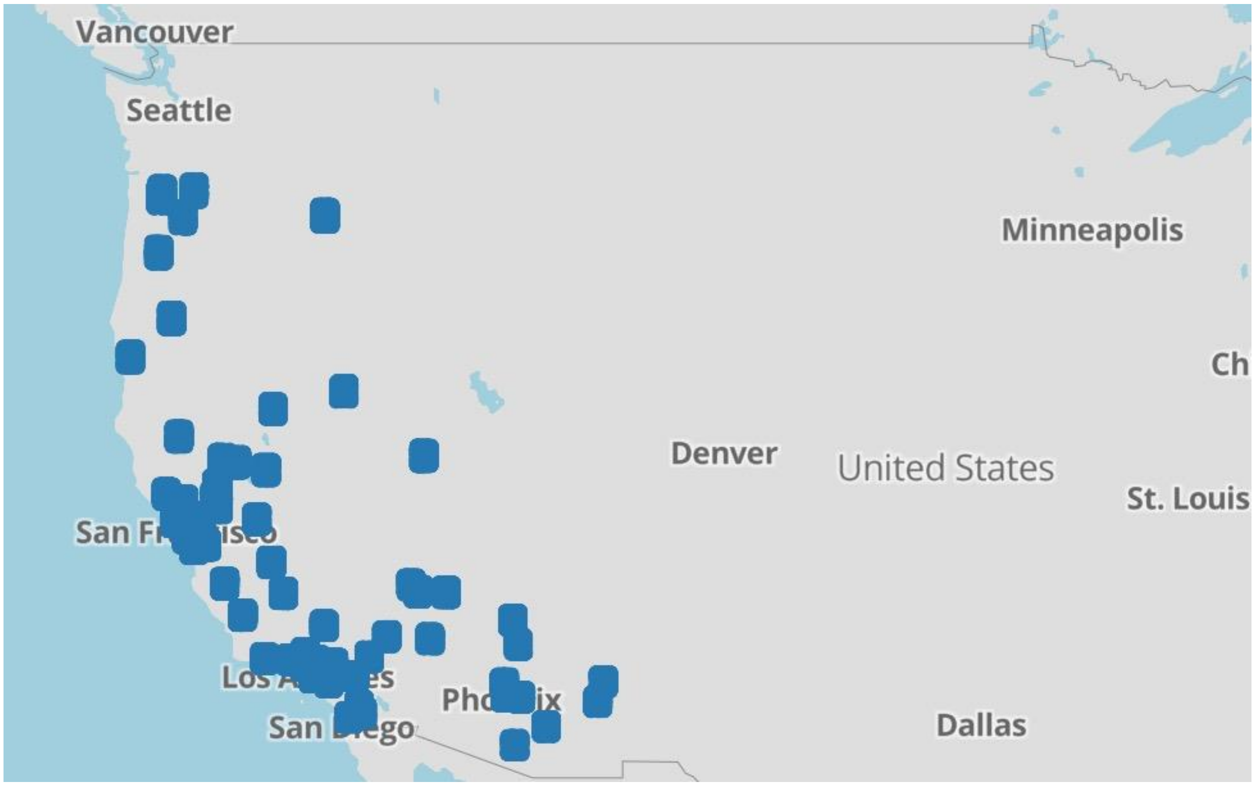


Figure 4: visualization of device location data

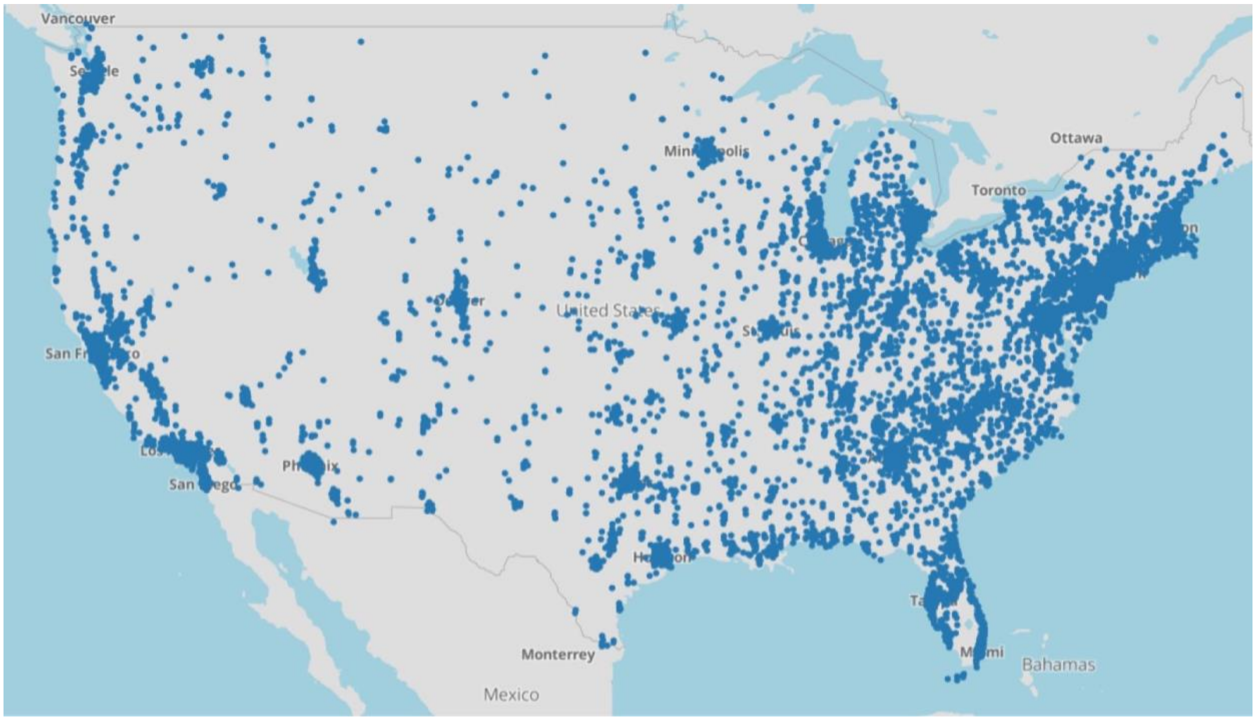


Figure 5: visualization of synthetic data

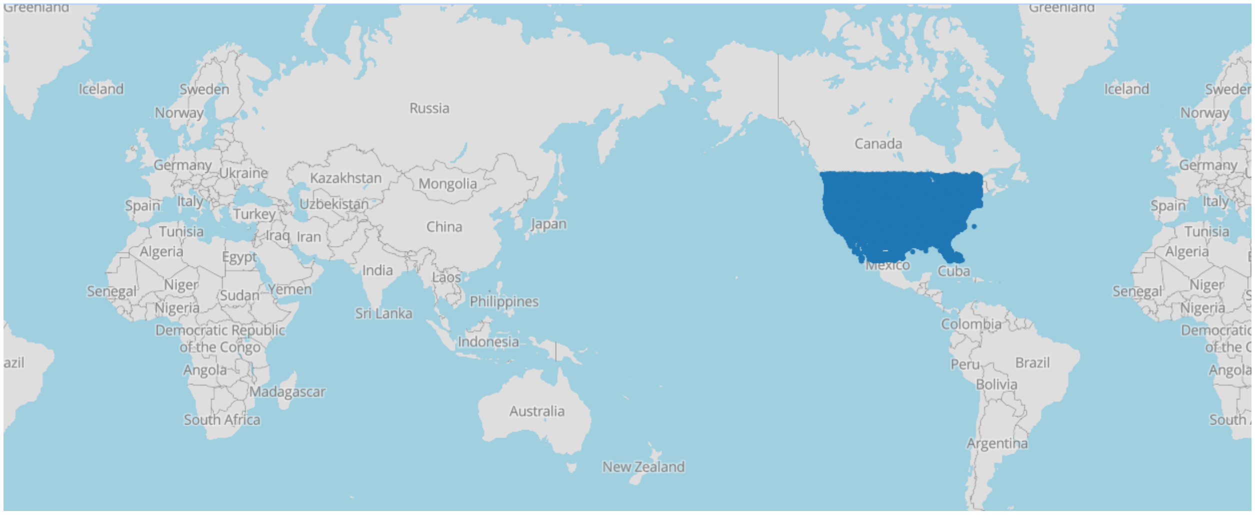


Figure 6: visualization of DB-pedia data (US region)

Problem 3: Clustering Big Data – k-means in Spark

Understanding Parallel Data Processing and Persisting RDDs

Before we move on, it is important to note why Spark is useful for calculating K-Means Clusters. The distance between data-points and their cluster's centroid is calculated iteratively, regardless of the number of K or the metric of distance. It is updated continually until convergence, at which point the centroid is at the minimal distance from all data-points assigned to the cluster. On large datasets, this iterative computation can be extremely costly and slow. Spark allows us to perform our calculations over a distributed dataset, crunching the computations in piece-meal sizes (RDDs). Importantly, Spark also has the ability to cache or persist data, which frees us from the need to re-run iterations over and over again unnecessarily. With RDD persistence, we do not have to re-invent the wheel every-time we need to work with an iterated outcome.

Understanding and implementing k-means

Researchers are interested in identifying patterns in social groups, whether they use the data in academia, government or industry. There are many ways to identify and describe a social group, that is, to distinguish one group from another, but we focus here on one approach: clustering. By clustering a seemingly homogenous population into smaller sub-populations, and resolving the blurred whole into heterogeneous factions, researchers can identify important commonalities that characterize and distinguish one group from another. Geolocation clustering is one notable example. Through geographical analysis, one can determine for instance which geographical areas are commonly “visited”, “checked into”, “activated”, “converted”. These results then enable a wide range of location-based services such as targeted-marketing, fraud prevention, content filtering, and financing. Determining the boundaries of these “geographical areas” is not trivial. Two notable methods include DBSCAN clustering and K-Means clustering algorithms, and it is the latter that we report on here.

The K-Means algorithm clusters data by minimizing the distance between data-points within a cluster and maximizing the distance between clusters, for K clusters. Each cluster has a centroid, and it is from this central point that each data-point’s distance is minimized. In short, k-means groups data by minimizing the sum of squared distances between the data points and their cluster’s centroid. The K number of centroids/clusters is a parameter set by us, and the rationale driving our decision is two-fold. Firstly, the number of clusters should be sensible and relevant to the researcher’s task at hand (e.g., clustering professional athletes with k=2 makes sense if one is interested in Gender Sub-populations but not if one is interested in seeing Conference Divisions). Secondly, the clusters, once determined, should be of roughly comparable size (e.g., k=3 does not make sense if one of the clusters contains negligible amount of the total data). In our project, we visualized the clusters that result from alternative K's by using http://plot.ly/, a free online data visualization platform.

Another discretionary element, aside from K, is the metric by which we define "distance". In our project, we compare two distance-measurement-algorithms: Euclidean vs Great Circle. Euclidean distance is a linear measurement of distance, which describes the length of a straight line between data-points. Great Circle distance is relevant in geolocation data analytics because the earth is spherical, and this distance measurement takes into account the curvature of the planet. The difference between the two measurements would be most pronounced in cases where the effects of curvature are non-negligible. For instance, when clustering geolocations over small distances, curvature is not too important, but when clustering across continents and hemispheres, curvature is crucial. Of course, the straight-line Euclidean distance through the earth's core would be shorter, but for practical purposes, the actual travelling distance along the earth's curved surface, is more relevant. The complete code for both distance algorithms is submitted to SVN. And the main code calculating those two kinds of distance is displayed in Figure 3. When we are executing our K-Means algorithm, we also use the function of closestPoint, addPoints and convergeDist. closestPoint is given a (latitude/longitude) point and an array of current center points, it returns the index in the array of the center closest to the given point. addPoints is given two points, it returns a point which is the sum of the two points. convergeDist is used to decide when the K-Means calculation is done, i.e. when the amount the locations of the means changes between iterations is less than 0.1. These functions are shown in Figure 4. In the next section, we will show when this difference is important, and when it is not.

```
# return the Euclidean distance of two points
def EuclideanDistance(p1, p2):
        d1=p1[0]-p2[0]
        d2=p1[1]-p2[1]
        d=d1**2+d2**2
        dis = d**0.5
        return dis

def haversine(point1,point2):
        """
        Calculate the great circle distance between two points
        on the earth (specified in decimal degrees)
        """
        # convert decimal degrees to radians
        earth = 6378137
        p11=point1[0]*pi/180
        p12=point1[1]*pi/180
        p21=point2[0]*pi/180
        p22=point2[1]*pi/180
        dis=cos(p12)*cos(p22)*cos(p21-p11)+sin(p12)*sin(p22)
        if dis>-1 and dis<1:
           gcd=earth*(acos(cos(p12)*cos(p22)*cos(p21-p11)+sin(p12)*sin(p22)))
           return gcd
        else:
           return EuclideanDistance(point1, point2)

# return the Great Circle Distance of two points
def GreatCircleDistance(p1, p2):
        return haversine(p1, p2)
```

Figure 7: main code of calculating Euclidean vs Great Circle

```
# return the index within centroids that is the closest point to p
def closestPoint(p1, centers, type1):
        index=0
        if type1 ==0:
                mindis = EuclideanDistance(p1,centers[0])
        else:
                mindis = haversine(p1,centers[0])
        for i in range(len(centers)):
                if type1 ==0:
                        if(EuclideanDistance(p1,centers[i])<mindis):
                                mindis = EuclideanDistance(p1,centers[i])
                                index = i
                else:
                        if(haversine(p1,centers[i])<mindis):
                                mindis = haversine(p1,centers[i])
                                index = i
        return index

# return a point that is the sum of the two points
def addPoints(p1, p2):
        return p1 + p2
def Covergent(previous,now):
        de=0 # original different is 0
        for i in range(len(previous)):
                de = de+EuclideanDistance(previous[i],now[i])
        final = float(de/len(previous))
        return  final
```

Figure 8: main code of three main functions

Compute and Visualize Clusters

In the Loudacre Mobile Device data, both distance measures produce similar clustering. The main difference is observed in the western-most locations. Using Euclidean distance: The Pacific Northwest, Northern & Central, and Southern California clusters span similar geographical distances as observed on the flat map. Using Great Circle distance however, the Pacific Northwest and Northern California cluster combines to form the green (see Figure 3). In Figure 5, we show also that the two difference distance measures yield similar clusters in the DB-pedia data.

The difference between a Euclidean-based and Great Circle-based distance measure is most obvious in the Synthetic Location data (see Figure 4). When the data is divided into two clusters, the two approaches yield similar clusters, one in the western half of the country and the other on the eastern. However, when we create 4 clusters, the approaches diverge in their results. Most notably, when going from K=2 to K=4 using Euclidean Distance, the eastern cluster seems to divides into 3 clusters with centroids that form a triangle. The 4 clusters in the Great Circle approach are distributed more evenly along the horizontal, with the Western cluster splitting into a West and East sub-cluster, and the Eastern cluster also splitting into a West and East sub-cluster, forming in effect a Western, Mid-Western, Mid-Eastern, and Eastern cluster, where the centroids are arranged linearly from east to west, as opposed to triangularly. This could have implications for supply-chain. However, we note that geographical features like mountain ranges and bodies of water are not factored into the distance measurement, so a different measure would be needed to determine clusters based on actual ground travelling-distance.

Figure 9: Loudacre Mobile Device Location

a) 

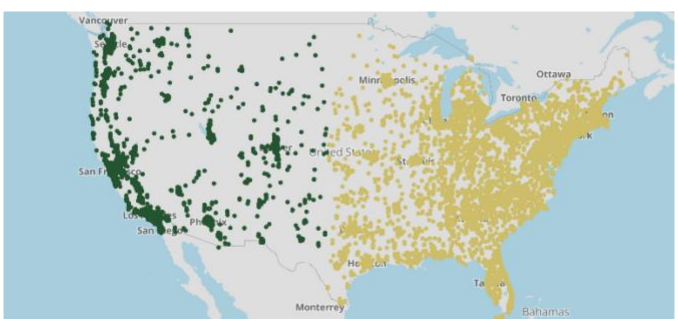
k=5; Euclidean distance

b) 

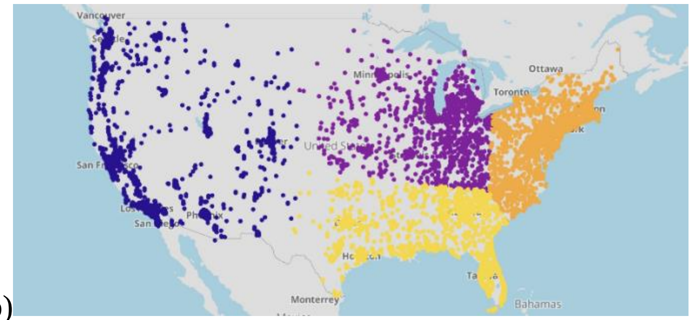
k=5; Great Circle distance

Figure 9: Visualization of Clusters in Loudacre Mobile Device Location Data with k=5; (a) using Euclidean distance and (b) using Great Circle distance

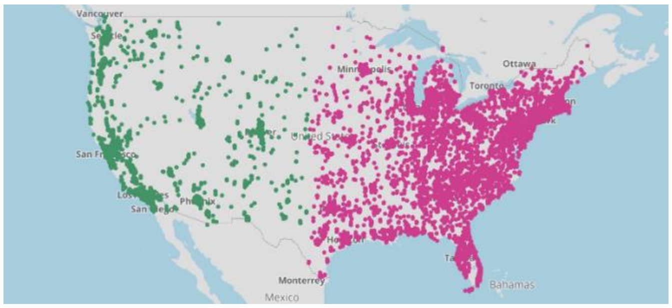
Figure 10: Synthetic Location

a) 

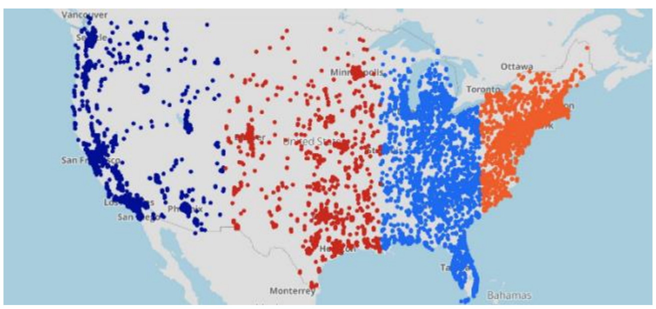
Euclidean distance with K=2

b) 

Euclidean distance with K=4

c) 

Great Circle distance with K=2

d) 

Great Circle distance with K=4

Figure 10: Visualization of Clusters in Synthetic Location Data using Euclidean distance; (a) with K=2 (b) with K=4, and using Great Circle distance; (c) with K=2 (d) with K=4

Figure 11: DB-pedia Location

a) 
k=6 using Euclidean distance

b)  k=6 using Great Circle distance

c)  k=4 using Euclidean distance

d)  k=4 using Great Circle distance

Figure 11: Visualization of Clusters in DB-pedia Location Data with k=6; (a) using Euclidean distance and (b) using Great Circle distance, and with k=4; (c) using Euclidean distance and (d) using Great Circle distance

Run Time Analysis

After we finished the geolocation clustering with Spark, we want to do some researches about the run time, mainly by comparing the runtime of your k-means implementation (using the same value for k) for all three datasets with/without using persistent RDDs. When we want to persist our RDDs, we simply add a “.cache ()” or “.persist ()” to each RDD.

The comparison is in table 4 below.

	DB-pedia Euclidean k = 6	Dbpedia gcd k = 6	Synthetic Euclidean k=4	Synthetic gcd k=4	Device Euclidean k=5	Device gcd k=5
With persist	141s	176s	18s	19s	172s	212s
Without persist	167s	207s	18s	19s	225s	235s

Table 4: Run time analysis of 3 datasets

From the table above, we can see that RDD persist will decrease the run time a lot actually, however, when the dataset is too small it can’t be seen from the run time(such as Synthetic dataset).

Problem 4: Big Data and Cloud Execution

Yelp Reviews

We wanted to compare the distance algorithms on another extension example, this time with a commercially relevant dataset. We obtained from yelp.com/dataset a subset of yelp reviews data (the full version required special permissions, and included 4.7 million reviews and 12 metropolitan areas). The data fields include the city and state (as well as latitude and longitude coordinates) of each restaurant reviewed, as well as a description of the review and pictures submitted with the review. The screenshot of part of yelp dataset is shown in Figure 8. (One line)

```
{"business_id": "YDf95gJZaq05wvo7hTQbbQ", "name": "Richmond Town Square",
"neighborhood": "", "address": "691 Richmond Rd", "city": "Richmond
Heights", "state": "OH", "postal_code": "44143", "latitude": 41.5417162,
"longitude": -81.4931165, "stars": 2.0, "review_count": 17, "is_open": 1,
"attributes": {"RestaurantsPriceRange2": 2, "BusinessParking": {"garage":
false, "street": false, "validated": false, "lot": true, "valet": false},
"BikeParking": true, "WheelchairAccessible": true}, "categories":
["Shopping", "Shopping Centers"], "hours": {"Monday": "10:00-21:00",
"Tuesday": "10:00-21:00", "Friday": "10:00-21:00", "Wednesday":
"10:00-21:00", "Thursday": "10:00-21:00", "Sunday": "11:00-18:00",
"Saturday": "10:00-21:00"}}
```


Figure 12: yelp dataset example

We run the K-Means algorithm for this dataset in Amazon AWS. See Figure 6 for our cluster groupings using K=5. We chose K=5 because we thought it would distinguish North America from South America. However, it did not, and it seems that 4 clusters may be reasonable. There seems to be a diffuse cluster in the east (purple), a diffuse Midwest cluster (yellow and light purple), a compact West cluster (orange), and a European cluster (salmon). Since the analysis is concentrated on primarily northwestern hemisphere data, the curvature of the planet is not a significant factor in distance calculations. As a result, Euclidean and Great Circle methods produce similar clusters. We expected slight differences, because South America and Europe data are included with North America data, but did not see any, perhaps because the sample size is small.

In future analysis, it would be interesting, to use the complete dataset, and to run a semantic analysis or even an image-content analysis. The analysis could code for restaurant descriptions (e.g., cuisine, price), or for emotional descriptions (e.g., enjoyable, terrible, etc). The image analysis could include building or environment characteristics, or food/plate attributes. From there, one could see how the different clusters vary by these coded descriptors. This type of analysis could reveal for instance if certain cuisines are perceived as more tasty in American vs

European reviewers, or if stand-alone restaurants are more expensive than within-mall restaurants in one region vs another.

Figure 13: Yelp

a)  K=5 using Euclidean distance

b) 

K=5 using Great Circle distance

Figure 13: Visualization of Clusters in Yelp Location Data with k=5; (a) using Euclidean distance and (b) using Great Circle distance

UFO Sightings

We wanted to compare the distance algorithms on another extension example, this time with a bigger dataset. We obtained from data.world a collection of 5178 reported UFO-sightings from US and Canada in 2016. The data fields include the city and state (as well as latitude and longitude coordinates) of each sighting, as well as a description of the sighting (e.g., "I observed a orange light hovering over near or fireworks were being “ and “Two balls of orange light seen above the trees; one split into two!”). The screen shot of this data set is in Figure 10 below.

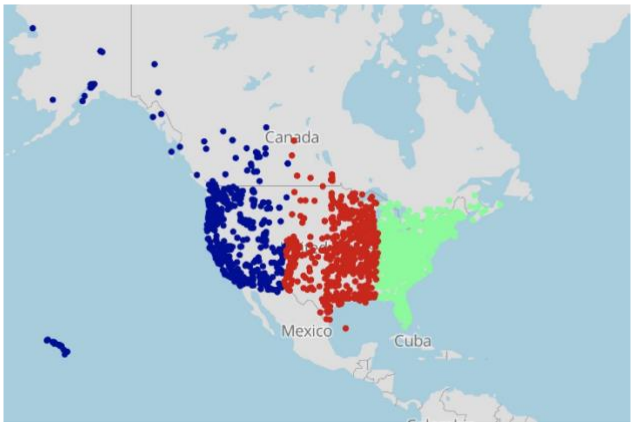
	A	B	C	D	E	F	G	H
1	Date / Time	Country	City	State	Shape	Summary	lat	lng
2	12/21/16 19:15	USA	Waynesboro	VA	Sphere	Bright round obj	38.0652286	-78.9058876
3	12/21/16 00:00	USA	Louisville	KY	Unknown	Multiple craft a	38.2542376	-85.7594069
4	12/20/16 22:30	USA	Santa Rosa	CA	Sphere	Bright orange pl	38.4404675	-122.714431
5	12/20/16 19:00	USA	Fresno	CA	Circle	Twelve orange gl	36.7295295	-119.708861
6	12/19/16 21:53	USA	Reymert	AZ	Circle	While camping in	33.2297793	-111.20929
7	12/19/16 21:50	USA	Redding	CA	Light	Stargazing I saw	40.5863563	-122.391675
8	12/19/16 20:02	USA	Gilbert	AZ	Light	Undulating swarm	33.294207	-111.737947
9	12/19/16 19:02	USA	Phoenix	AZ	Circle	Bright light in	33.4485866	-112.077346
10	12/19/16 18:30	USA	Huber Heights	OH	Cylinder	Entire family se	39.8590241	-84.1113629
11	12/19/16 18:00	USA	Conway	AR	Chevron	((NUFORC Note: N	35.0886963	-92.442101
12	12/19/16 10:30	USA	Anchorage	AK	Circle	4 red silent cra	61.2163129	-149.894852
13	12/19/16 06:50	USA	Town and Country	MO	Light	I was heading N	38.6122751	-90.4634531
14	12/19/16 03:30	USA	Salt Lake City	UT	Triangle	Triangular shape	40.7670126	-111.890431
15	12/19/16 01:27	USA	Richardson	TX	Triangle	3 triangular sha	32.9481789	-96.7297205
16	12/18/16 22:00	CANADA	Wetaskiwin	AB	Light	Bright Red Light	52.968492	-113.36792
17	12/18/16 21:50	USA	Berryville	AR	Formatior	I saw my dog loo	36.364792	-93.5679666
18	12/18/16 21:15	USA	Honolulu	HI	Light	I observed what	21.304547	-157.855676
19	12/18/16 21:00	USA	St. George	UT	Chevron	V-shaped lights	37.104153	-113.584131
20	12/18/16 20:40	USA	Bend	OR	Other	Large fleet of 1	44.0581728	-121.31531
21	12/18/16 19:30	USA	Mission	KS	Light	Too yellow to be	39.0277832	-94.6557913
22	12/18/16 19:00	USA	Lancaster	OH	Light	I thought they w	39.7136754	-82.5993293
23	12/17/16 21:00	USA	San Pedro	CA	Other	Object changing	33.7358518	-118.292293
24	12/17/16 20:00	USA	Kahana	HI	Fireball	Fireball 4 secon	21.5543942	-157.873405

Figure 14: example of UFO dataset

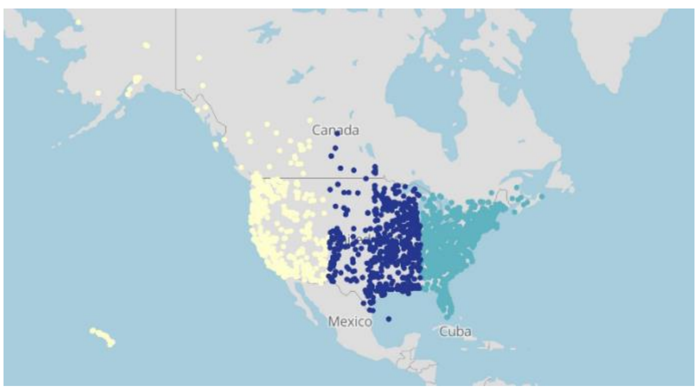
See Figure 11 for our cluster groupings using K=3. We chose K=3 because the distribution was uniform and made geographic sense. There seems to be a Western, Midwestern, and Eastern cluster, with the Easter cluster being most compact and the Western cluster most disperse. Since the analysis is confined to USA sightings, the curvature of the planet is not a significant factor in distance calculations. As a result, Euclidean and Great Circle methods produce similar clusters.

Because our time was limited, further analysis was not conducted. It would be interesting, in future analysis, to run a semantic analysis on the descriptions. The analysis could code for physical descriptions (e.g., shape, color, number, etc), or for emotional descriptions (e.g., scary, ominous, glorious, etc). From there, one could see how the different clusters vary by these coded descriptors. This type of analysis could reveal for instance if eastern sightings are perceived as scarier than western sightings, or if eastern sightings are predominantly of large circular objects while western sightings are predominantly of smaller triangular objects.

Figure 14: UFO Sightings

a) 

K=3 using Euclidean Distance

b) 

K=3 using Great Circle Distance

<u>Run time analysis</u>

The run time in Amazon AWS of both datasets are document in table 5 below.

	Yelp data Euclidean K=5	Yelp data gcd K=5	UFO data Euclidean K=3	UFO data gcd K=3
Run time	344 s	372 s	16 s	19 s
Data size	675130 line	-	5178 line	-

Table 5: run time analysis Yelp vs. UFO

From the table above, we can see that the bigger yelp data will cost more time in AWS to process, and gcd algorithm takes more times than Euclidean because it is more complex to execute.

References

https://www.oreilly.com/ideas/clustering-geolocated-data-using-spark-and-dbscan

http://wiki.dbpedia.org/services-resources/datasets/previous-releases/dataset-38#h336-2

https://statistical-research.com/index.php/2013/11/04/spatial-clustering-with-equal-sizes/

yelp.com/datasets