Adrian Ambrose
DSCI6007 – Distributed and Scalable Data Engineering
Professor Behzadan
May 11, 2020

K-Means Spark Implementation

1. Motivation

Clustering can be a useful unsupervised learning technique to get a better understanding of a dataset. The general goal of clustering is to group datapoints into similar subsets. This has numerous applications, including improving logistics with geospatial data, finding customers with similar purchase patterns, as well as many others. Clustering is often used in situations with large quantities of data and the goal is to get a grasp on underlying relationships. The large amounts of data can drive a need to efficiently complete the calculations and iterations required for the clustering algorithms. One way to effectively do this is through the use of Apache SPARK to distribute the work on a cluster of machines. In this project, K-means clustering is implemented on an Amazon Web Services (AWS) EMR (Elastic MapReduce) machine cluster to distribute the work. The k-means clustering is applied specifically to geo-spatial data (latitude/longitude coordinates) to find population clusters based on different datasets.

2. Approach

2.1. Storage:

The main resource provider for this project is Amazon Web Services. To keep everything within the same set of services, the first step was to take the raw data of interest and store in AWS S3 storage. Through the EMR, the S3 storage can be accessed as if it were in the Hadoop Distributed File System (HDFS), which makes this approach very convenient.

2.2. Data Preparation:

With the raw data stored in S3, pyspark and jupyter notebooks can be used through the EMR to view and clean the data in preparation for input into the k-means application. This is a critical step because each dataset in this project comes from a different source and can have different fields, missing values, irrelevant values, etc. These issues all need to be addressed before the data is fed to a k-means algorithm. This also allows an opportunity to visualize the input data and get an initial look at the information. Once the data has been cleaned and properly formatted, the results can be stored as a separate file in S3 storage to be accessed SPARK in later steps.

2.3. Spark Application:

To implement the k-means algorithm, a pyspark application is run on the EMR using the "spark submit" command from the terminal. The command information for the application in this project can be seen below (See Table 1).

Command Syntax:

~$ spark-submit application.py s3://bucket-name/filename s3://bucket-name/folder distance_function k max_iterations

spark-submit: terminal command to initiate spark application
application.py: the python application
s3://bucket-name/filename: Path to input file, change names as appropriate
s3://bucket-name/directory: Directory for output
distance_function: 'circle' or 'euclid' to define the distance calculation
k: integer indicating number of clusters
max_iterations: integer indicating the desired max number of iterations (used during testing, should be set to a high number such as 50 or greater)

Example:

~$ spark-submit step3_persistent.py s3a://final-kmeans/clean/mobilenet.csv s3://final-kmeans/results circle 5 50

Table 1: Kmeans Application Command Line Interface

2.4. K-means Algorithm Steps:

1. Initialize Centers:
    a. Select "k" number of points to be the initial centers. In this implementation, the initial points are selected at random.
2. Determine Clusters:
    a. Loop over all the points to determine which center is closest (using the chosen distance metric) and label each point with cluster number
3. New center
    a. Using the created clusters, find the new center points for each cluster
4. Repeat steps 2-3 until the centers converge

2.5. Output Review:

Once the k-means application has been run on the data, the output includes the original datapoints, the associated clusters, and the cluster centers. These are saved to the specified output directory and can then be read into a jupyter notebook for analysis and visualization using common python libraries.

3. System Configuration

3.1. EMR Initialization:

EMR Cluster Configuration Settings:

- General Configuration
    - Logging: Enabled
    - Launch mode: Cluster
- Software Configuration
    - Release: emr-5.29.0
    - Applications: Spark: Spark 2.4.4 on Hadoop 2.8.5 YARN with Ganglia 3.7.2 and Zeppelin 0.8.2
- Hardware Configuration
    - Instance type: m4.xlarge
    - Number of instances: 3

3.2. EMR Configuration:

One of the benefits of working with a preconfigured EMR cluster in AWS is that much of the functionality is available once the cluster begins running. Only the more specific needs require additional set up within the cluster. This project required only minor additions to the EMR cluster and these are explained below.

The first step after connecting to the cluster through ssh is to "sudo yum update" to ensure that everything is up to date on the machine. From there, "sudo pip install" can be used to install the python packages required for the project. These are shown below (Table 2).

Python packages and uses:

- **jupyter** – interactive python notebook, great for data cleaning and visualization
- **pandas** – good for locally viewing and manipulating tabular data
- **matplotlib** – for plotting and visualizing data
- **numpy** – has useful numerical functions

Table 2: Python Packages Installed to EMR

Pyspark is included in the EMR configuration specified during initialization, but it is worth noting that the environment variables need to be modified to have pyspark run with Jupyter Notebooks. This can be done by adding the following lines to the end of the ~/.bashrc file on the machine.

```
export PYSPARK_DRIVER_PYTHON=`which jupyter`
export PYSPARK_DRIVER_PYTHON_OPTS="notebook --port=8888 --no-browser --ip=0.0.0.0 "
```

With the required packages installed and environment variables specified, typing "pyspark" into the terminal will start a Jupyter Notebook server and give the url where the notebook can be accessed. This cluster has all the configuration requirements for completing the data cleaning and kmeans algorithm. It is worth nothing that the **basemap** toolkit (python package) that adds functionality to matplotlib was also installed on a local machine in order to do some of the visualization work. Issues were encountered when installing this package onto the EMR, and the visualization tasks did not need to be completed on the EMR.

4. Application by Dataset

4.1. Dataset 1: Mobilenet

Description:

Mobilenet - information collected from mobile devices on a fictional wireless carrier. This raw data set contains 14 columns and 100,000 rows. Faulty samples and irrelevant columns are removed from this raw dataset. Once the data cleaning and preparation is complete, the cleaned data is condensed to 6 columns and 94,039 rows.

Useful Applications:

Kmeans clustering could be useful for information similar to the Mobilenet dataset in applications such as marketing. The clusters may give insight into local trends, or with some additional study, which local networks warrant greater advertisement investment.

Cleaned Data prior to clustering:

At first glance, the Mobilenet data does not have any obvious clusters (Figure 1). There may be a couple subtle gaps in data points, but it will be interesting to see how the kmeans algorithm breaks up the data.

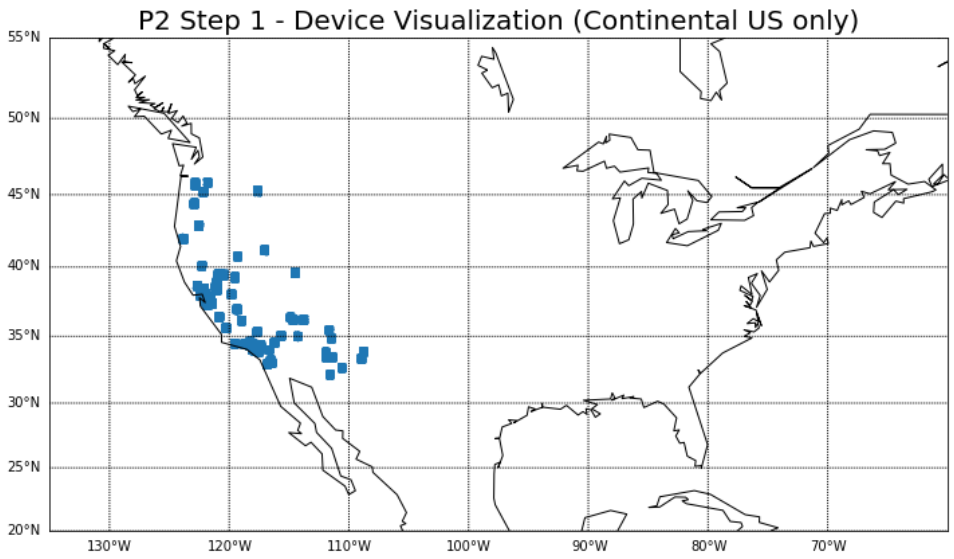

Figure 1: Mobilenet Data Prior to Clustering

Test Plan:

Mobilenet Test Plan

# of Clusters (k)	Distance Metric
5	Great Circle Distance
5	Euclidean Distance

Results:

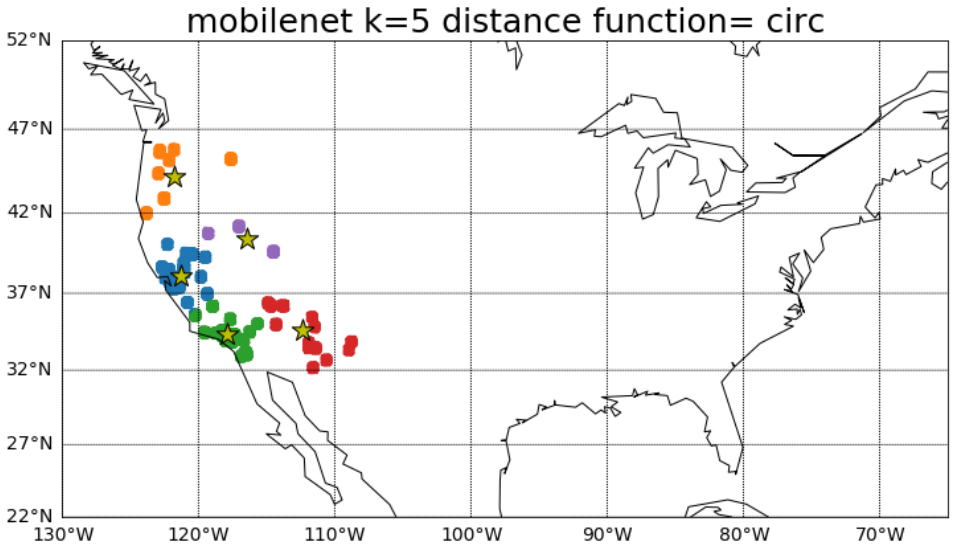

Figure 2: Mobilenet – 5 Clusters using Great Circle Distance Function

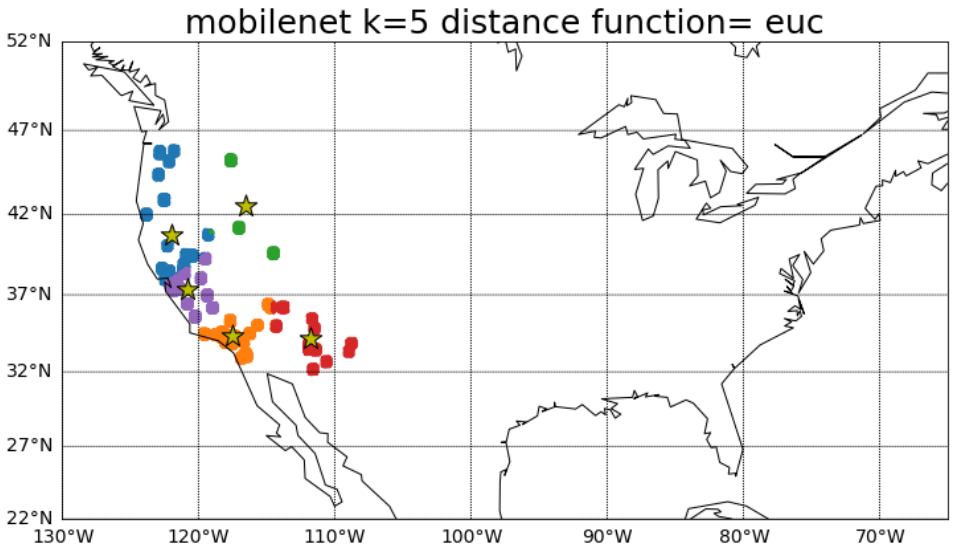

Figure 3: Mobilenet – 5 Clusters using Euclidean Distance Function

Discussion:

With this dataset and number of clusters, the Euclidean Distance and the Great Circle distance resulting clusters are quite similar (See Figure 2 and Figure 3). However, it is worth noting that the centers of the clusters are noticeably different. The same convergence criteria for a full world dataset was also used for this dataset. The convergence criteria should likely have been more restrictive for points that were this close together to begin with.

4.2. Dataset 2: Sample Geographic Data

Description:

Synthetic - Generic sample geometric data. This dataset is 9970 rows of latitude and longitude coordinates. The full dataset is retained and simply formatted for use with the kmeans algorithm.

Useful Applications:

Simple point data such as this can be useful for logistics reasons. An example may be using clustering to determine where warehouses can be most efficient.

Cleaned Data prior to clustering:

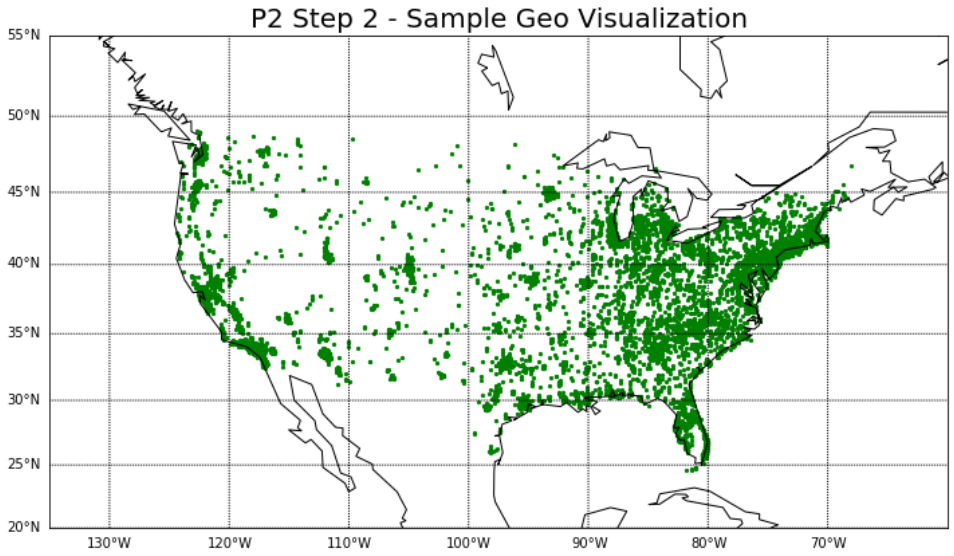

Figure 4: Sample Geo Data Prior to Clustering

Test Plan:

Synthetic Test Plan

# of Clusters (k)	Distance Metric
2	Great Circle Distance
2	Euclidean Distance
4	Great Circle Distance
4	Euclidean Distance

Results:

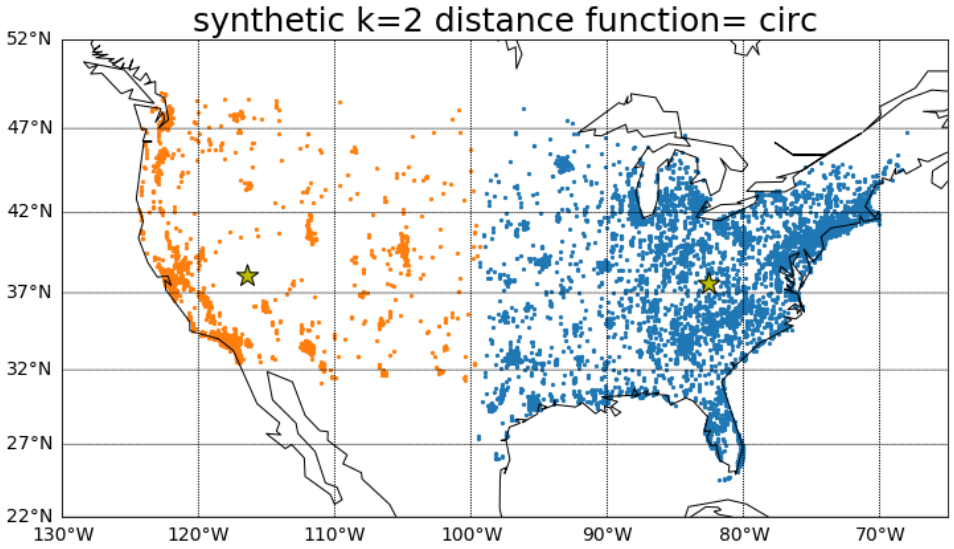

Figure 5: Sample Geo – 2 Clusters using Great Circle Distance Function

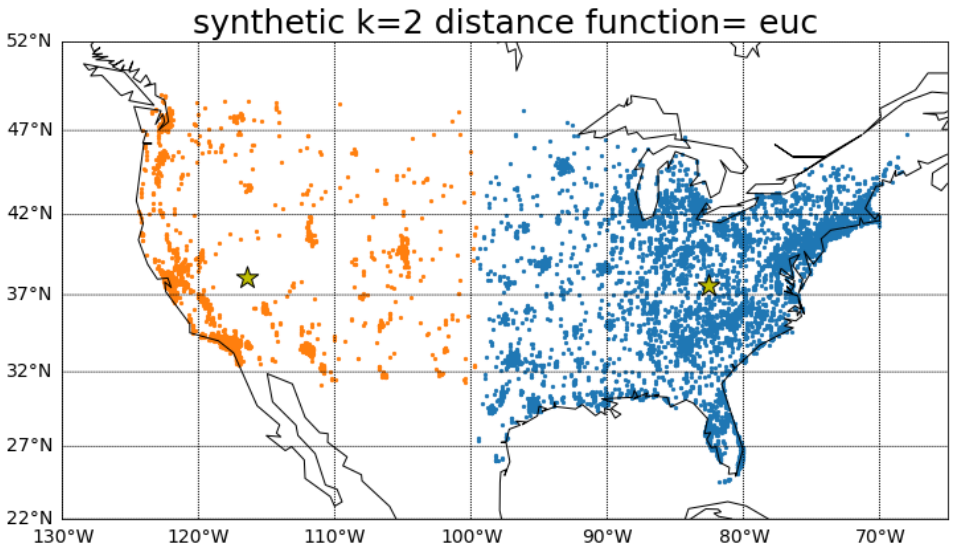

Figure 6: Sample Geo – 2 Clusters using Euclidean Distance Function

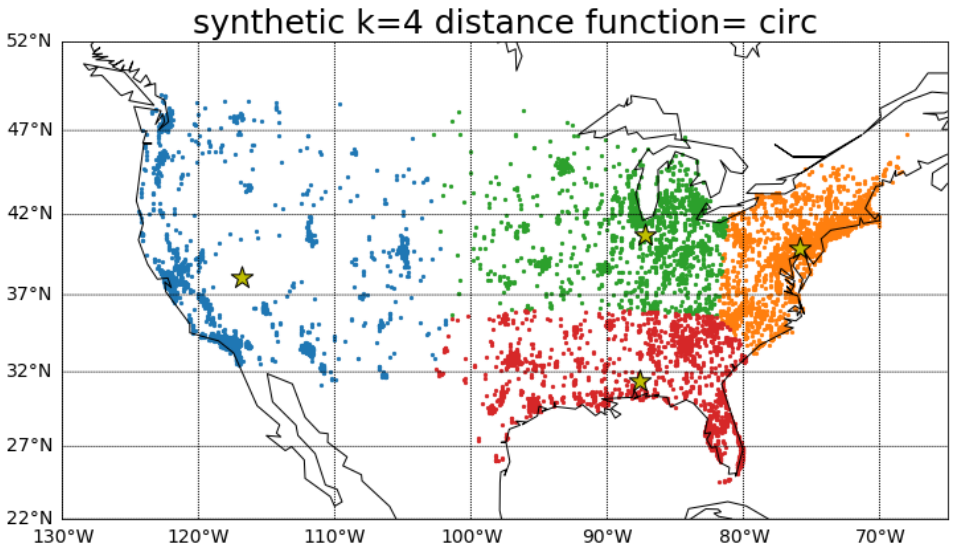

Figure 7: Sample Geo – 4 Clusters using Great Circle Distance Function

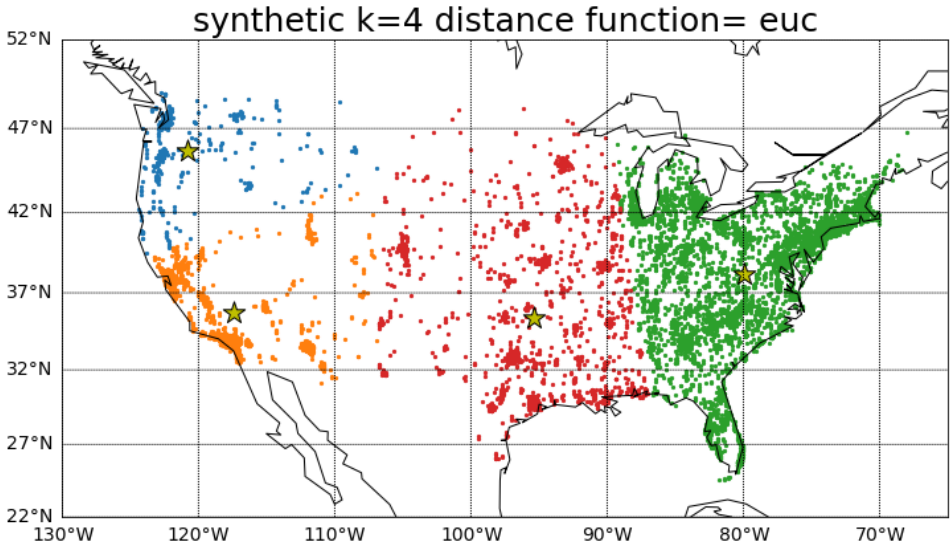

Figure 8: Sample Geo – 4 Clusters using Euclidean Distance Function

Discussion:

With a cluster number of 2, the Great Circle Distance and Euclidean distance resulted in nearly identical clusters. However, with 4 clusters the algorithms result in significantly different clusters. Similar to the Mobilenet data, the convergence criteria may have come into play for the relatively close points in this dataset. Another potential contributor is the center initialization. For this project, the initial cluster centers were chosen at random.

4.3. Dataset 3: DBPedia

Description:

DBPedia - Each record represents a location/place that has a Wikipedia article and latitude/longitude information. The dataset includes 450,151 datapoints which 3 columns - latitude, longitude, and the name of the Wikipedia page.

Useful Applications:

This dataset lends itself to helping with logistics. For global operations logistics, it is important to know which areas are populated. Some assumptions can be made, such as - places with a Wikipedia page are likely to have occupants. Areas where there is a higher density of specified locations are also more likely to have a greater population. The greater density of specified locations is likely an indicator of a more complex societal network (or at least inhabitable land).

Cleaned Data prior to clustering:

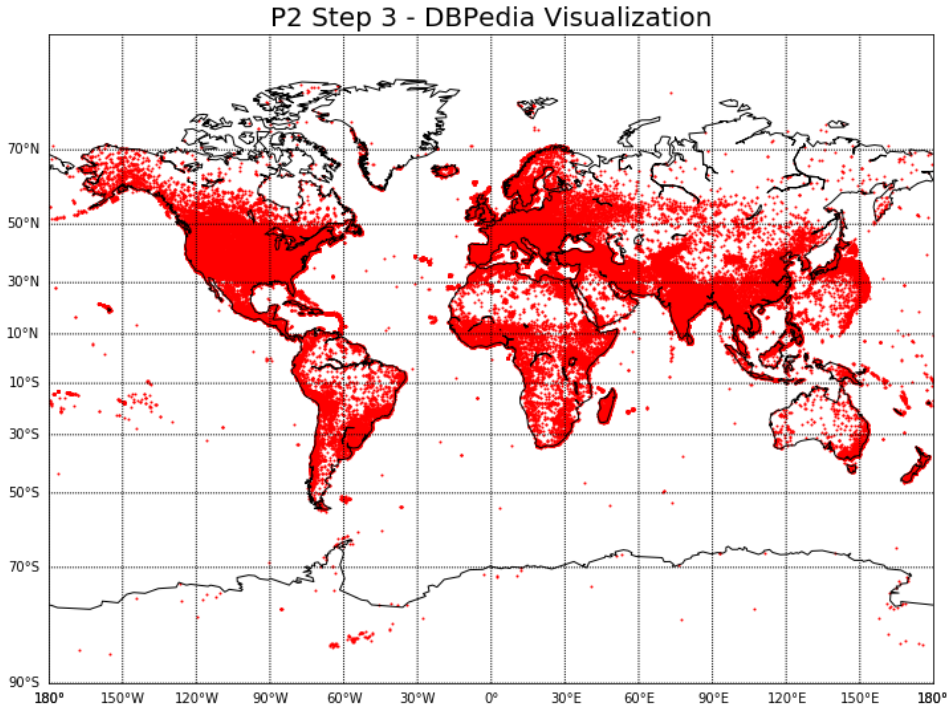

Figure 9: DBPedia Data Prior to Clustering

Test Plan:

DBPedia Test Plan

% of Samples	# of Clusters (k)	Distance Metric
100	6	Great Circle Distance
100	6	Euclidean Distance
20	6	Great Circle Distance
20	6	Euclidean Distance

Results:

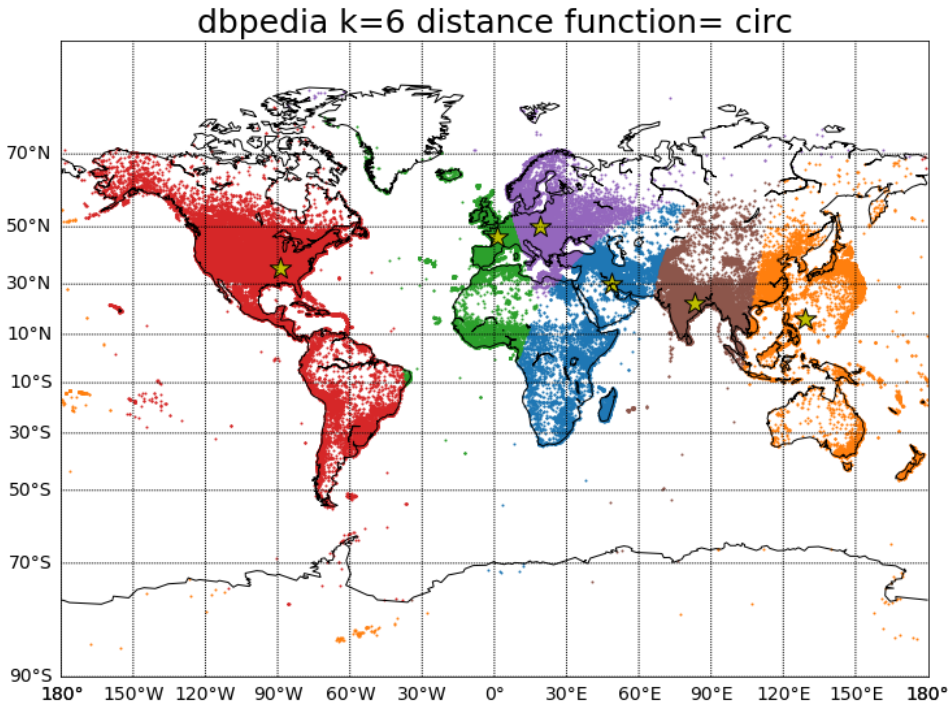

Figure 10: DBPedia – 6 Clusters using Great Circle Distance Function

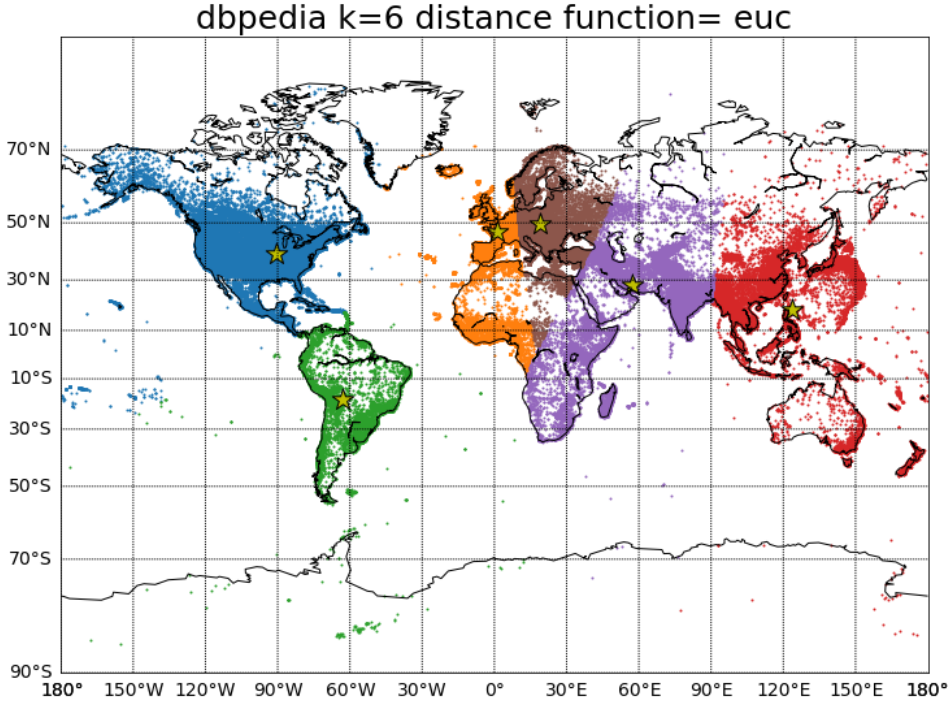

Figure 11: DBPedia – 6 Clusters using Euclidean Distance Function

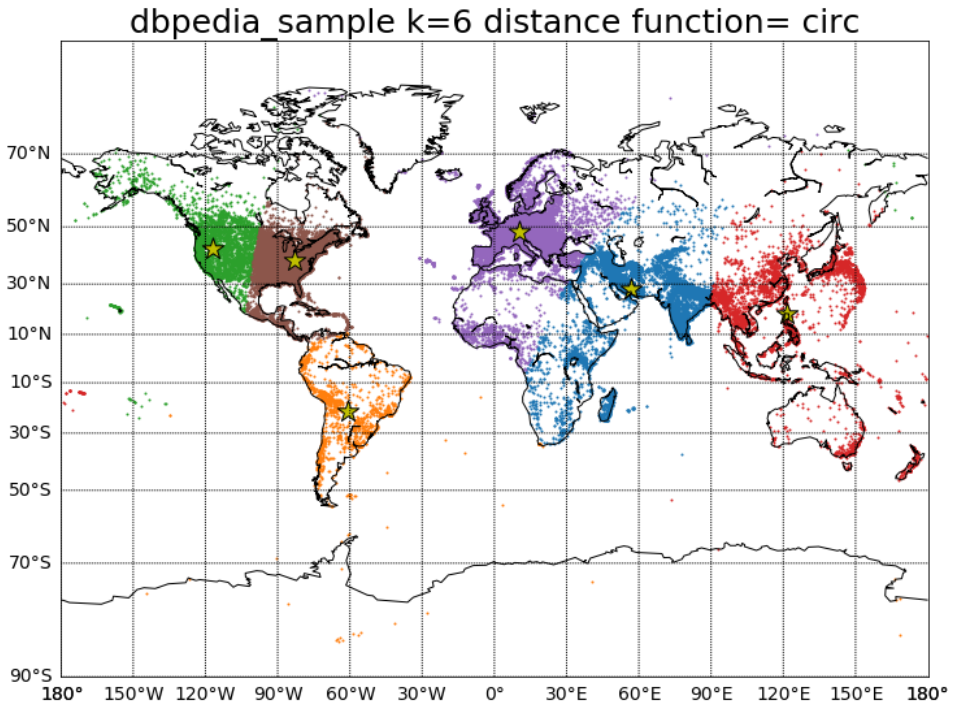

Figure 12: DBPedia Sample – 6 Clusters using Great Circle Distance Function

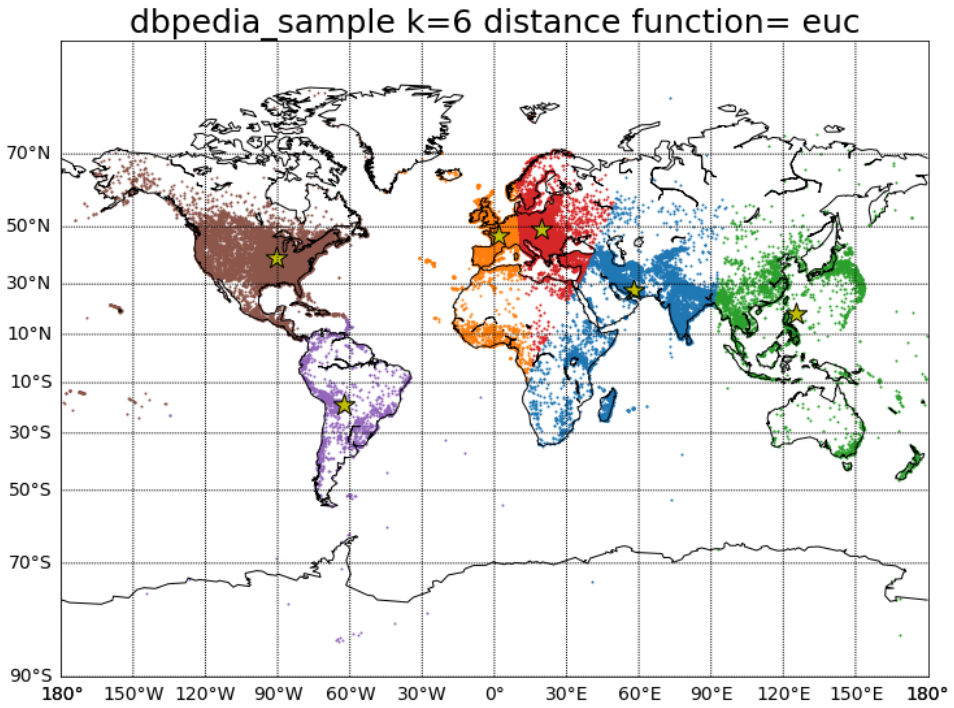

Figure 13: DBPedia Sample – 6 Clusters using Euclidean Distance Function

Discussion:

When comparing the Euclidean distance calculated clusters for the sample and full size, they are found to be similar. This is not surprising because one would expect the same calculation done on a random sample subset to produce similar results. The clusters calculated using the Great Circle distance were

harder to explain. These clusters were not only different from those calculated using Euclidean distance, but also different from each other. The most likely explanation is that the initial cluster points varied enough that the resulting clusters were also significantly different. The number of clusters and distribution of points are also large factors in where the centers reside in the end.

5. Runtime Analysis:

Using the SPARK History logs after conducting the kmeans tests allowed access to useful information regarding the runtimes of the different implementations. Each of the kmeans applications were conducted once with the initially loaded RDD cached into memory, and then repeated without any caching. The results were very telling. Taking the average of all the runtimes grouped by persisted (cached) vs not persisted showed a significant reduction in run time (see Figure 14).

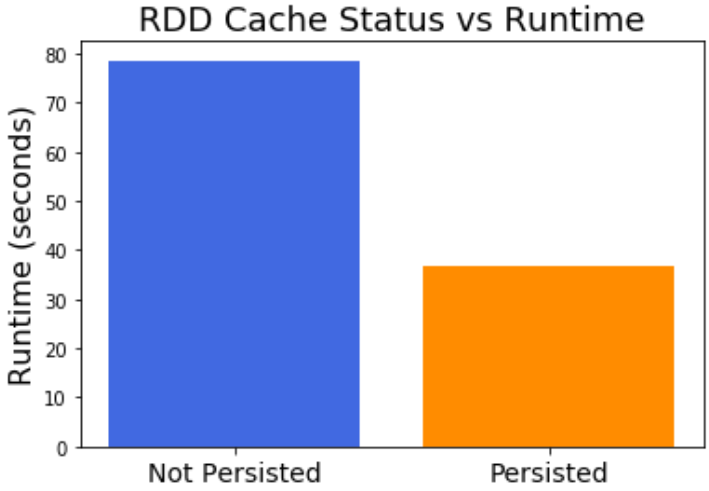

Figure 14: Average Runtime Grouped by Cache Status. The average runtime for the implementations with cached RDD's was less than half that of the nonpersisted.

Breaking this down further by dataset, the difference is much less significant in the smaller datasets. As the datasets and the number of iterations required increases, the runtime difference becomes more pronounced. This is clearly seen in the DBPedia dataset which had the greatest number of datapoints, as well as the highest cluster count (See Figure 15).

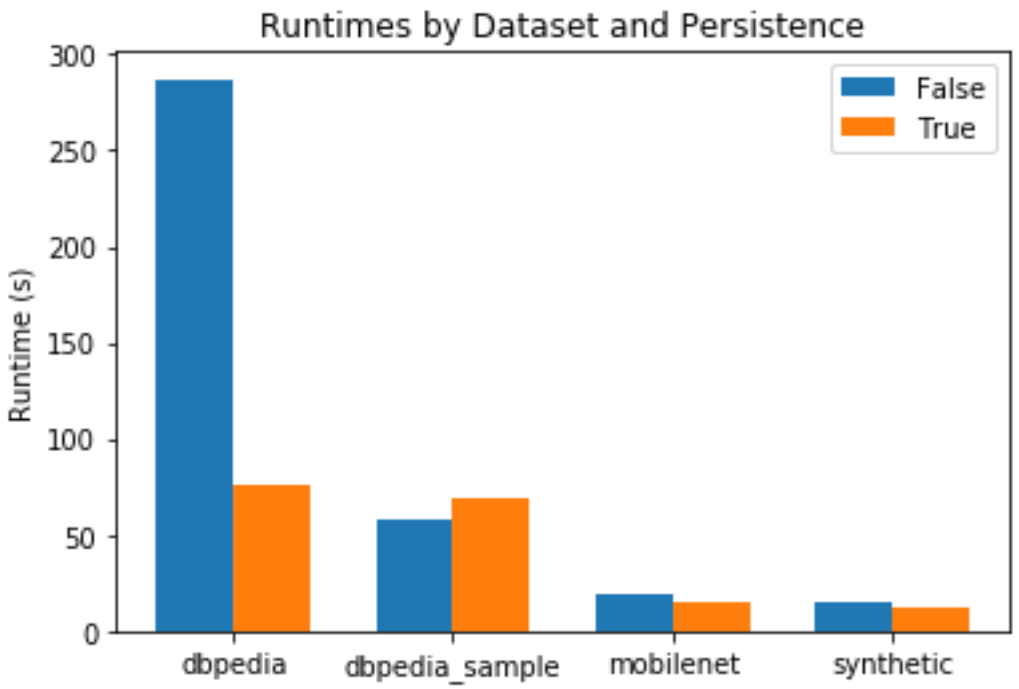

Figure 15: Runtimes Grouped by Dataset and Persistence

Comparing the runtimes as grouped by distance function also revealed that the Great Circle Distance had a higher average runtime than the Euclidean distance function. This could have been due to the number of iterations required (the Euclidean convergence criteria in this project was slightly more relaxed than for the great circle distance), but also due to the computation time required for the Great Circle Distance calculation.

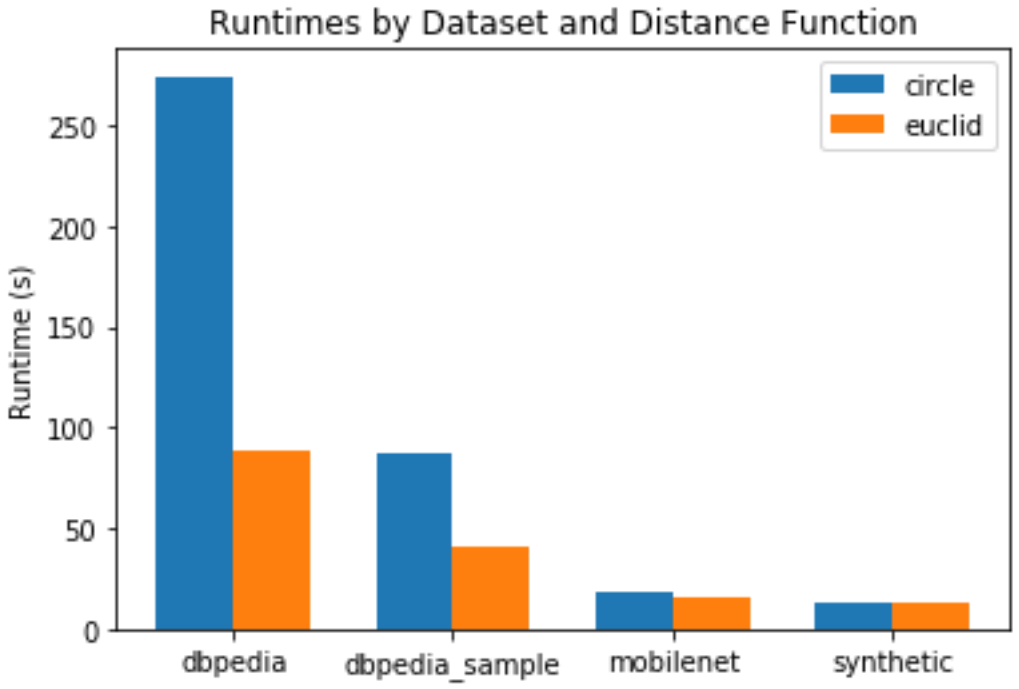

Figure 16: Runtimes Grouped by Dataset and Distance Function

6. Final Conclusions:

6.1. Lessons Learned / Conclusions / Future Work:

After completion of this project, there were several general items worth noting. The first of these was that the initialized cluster centers seem to have a noticeable role in the final cluster results. Cluster centers were chosen randomly in this implementation of kmeans, but repeating these tests with different center initialization functions could provide further insights. It also seems likely that the distribution of the data and the number of clusters likely play a roll in the repeatability of the clusters. The DBPedia dataset was the only dataset to span the full world, and as a result was also the most irregularly dispersed. The DBPedia implementation also had the highest number of clusters calculated. Both of these were likely factors that lead to this dataset producing the most inconsistent cluster calculations.

With the dataset itself playing such a key role, it would make sense in the future to run multiple configurations of kmeans on the same dataset. This would primarily entail trying different numbers of clusters, but also potentially trying different initial centers and distance functions. To build on this project, it would also be interesting to modify the kmeans to account for more than geospatial data. The kmeans algorithm can be expanded to account for more than just the two dimensions for latitude and longitude.

7. Bibliography

1. "Chapter 7: Clustering." *Mining of Massive Datasets*, by Jure Leskovec et al., Cambridge University Press, 2020, pp. 267–268.
2. Adcock, Karina. "How to Plot Latitude and Longitude Co-Ordinates in Basemap." Youtube, 26 Sept. 2018, www.youtube.com/watch?v=XiZbrii49pI.
3. J. D. Hunter, "Matplotlib: A 2D Graphics Environment," in *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90-95, May-June 2007, doi: 10.1109/MCSE.2007.55.
4. Wes McKinney, . " Data Structures for Statistical Computing in Python ." *Proceedings of the 9th Python in Science Conference* .
5. The pandas development team, . "pandas-dev/pandas: Pandas." (2020).
6. http://statistical-research.com/wp-content/uploads/2013/11/sample_geo.txt
7. https://classes.cec.wustl.edu/cse427/lat_longs.zip