Title: Project 3 - Geo-Location Clustering in Spark
Names: Bob Skowron, Kevin Kim, Jason Walker
Keys: rskowron, keonshik.kim, jwalker
SVN: jwalker: https://svn.seas.wustl.edu/repositories/jwalker/cse427s fl17/

Introduction and Motivation

Why K-Means

Clustering, specifically k-means, is a frequently used approach to identify patterns in data. The k-means algorithm is used to identify patterns in data related to the position or distance of data points from one another. While the most common measure of distance is Euclidean distance, a simple two-dimensional distance of 2 data points, accounting for the spherical nature of the geographical coordinate system used in this project requires Great-circle distance, a measure of distance of 2 data points along a sphere. Therefore, Great-circle distance is also part of our implementation. A Spark implementation, written in Pyspark, utilizes the distributed computing and storage features of Hadoop Map/Reduce and Hadoop Distributed File System (HDFS). Spark is the preferred implementation framework because of its ease of use when dealing with iterative and interactive data sets. The results of our local and cloud-enabled executions are discussed on several sample data sets and real-world data sets. This analysis includes an overview of the Spark runtime as well as conclusions and lessons learned from our approach.

When to Use K-Means

The k-means algorithms has many uses and is frequently used as an initial approach to data mining to determine general patterns or trends in large data sets. Certainly, there are more accurate algorithms while there also are specialized use-cases that likely require either a custom implementation or an entirely different algorithmic approach. However, the simplicity of a k-means algorithm and its accepted status as a work-horse in data mining peaked the group's interest in this algorithm. One real-world example would be clustering of gene expression and variation to identify the genetic underpinnings of a disease or the effect of an abnormality (It's widely accepted that many cancers result from multiple genetic mutations often driven by a single founding event, https://doi.org/10.1038/nature10762). The evolution of these multiple sub-clones can often be recapitulated using k-means clustering on the variant allele frequencies to determine the number of cancer genomes present in a patient, https://doi.org/10.1371/journal.pcbi.1003665.

Our Usage of K-Means

During this project we will demonstrate the real-world use of the k-means algorithm on a weather data set. Weather data was chosen due to its extremely large volume that could push the limits of our limitation if so desired. In addition, the nature of the data has potential financial and social implications. For example, an insurance firm may use historical weather trends to identify regions of the United States that are susceptible to severe weather patterns, including hail and tornadoes. Those events most likely to result in property damage and claims may impact the premiums paid by customers that live or travel often in such regions. Alternatively clustering of severe weather events might be used socially as a tool for people to plan special events, vacations, or even selection of a new residence or job location.

Approach and Implementation

Code Implementation

Our Pyspark code allows the user to choose input file path, output file path, choice of distance measure (Euclidean or Great-Circle), and k value. Our code first pre-processes the input file using a custom snippet of code specific to each data set. An RDD of data points is created by transforming the lines into a pair RDD of (latitude, longitude) pairs (let us call this data_RDD). Then the code runs our implementation of the k-means algorithm, which consists of 2 main parts.

The first part starts with picking k number of initial centroids randomly from data_RDD and storing them in a list and creating a separate RDD of centroid (call this centroid_RDD) to output. Next, using this list of initial centroids, a new RDD (call this cluster_RDD) is created as a pair RDD of format (K,V) = (cluster number , (latitude, longitude)). At this point, cluster_RDD is persisted. After this initial mapping of data points to the clusters, a loop is started, under which new centroids are found and then data points are mapped to clusters based on the locations of the new centroids. The loop is a conditional loop (while loop) that runs until convergence, which is checked by testing if the all the new centroids are at most 0.1 distance away from the old centroids. If all the centroids are within 0.1 distance of old centroids, then the program exits the loop (the centroid list and centroid_RDD overwritten after checking for convergence at each iteration, so the new centroids are stored first in a separate RDD of latitude and longitude format and then a list that is the result of collecting this RDD of new centroids). Additionally, at the end of each iteration, centroid_RDD and the cluster_RDD are persisted. Cluster_RDD is still a pair RDD of format (K,V) = (cluster number , (latitude, longitude)) at this point.

The second part starts after exiting the loop. Upon exiting the loop, the purpose is to transform the cluster_RDD into a format (k, data type, cluster number, latitude, longitude, distance from the centroid) where the data type is either centroid or map (meaning that the data point is a mapped value under a certain cluster). The distance from the centroid is calculated by using a distance measure of choice, which implementation will be elaborated below.

The use of latitude and longitude pairs in calculating distances, both Euclidean and Great-circle, is not appropriate since latitude and longitude pairs are x,y pairs with finite ranges for x and y. This would be problematic in calculating the distance between 2 points along the opposite edges of this finite coordinate system since the distance calculated may not actually be the shortest distance. This is so since Earth resembles a sphere: there are 2 arcs going from any set points A to B when the arcs and the points A and B are all on a single plane (2 arcs combined form a circle whose radius is the radius of Earth). Calling these 2 arcs L1 and L2, the length of L1 will be shorter, equal to, or larger than the length of L2 and vice versa. It is because of this discrepancy in lengths of arcs, that it is required to convert latitude and longitude pairs to Cartesian coordinate pairs. Using the converted Cartesian coordinate pairs, calculations of both Euclidean and Great-circle distance (GCD) can be done correctly (GCD requires conversion from Cartesian to Spherical coordinates).

Cloud Execution Approach

The dataset that we chose for our cloud execution was NOAA's Severe Weather Data Inventory. Specifically, we focused on the Hail Signatures from NEXRAD. This dataset met many of the requirements for us of

- Size
- Quantity
- History
- Relevance

The data is stored in CSV files, bucketed by year and compressed. Each decompressed file is approximately 500MB and contains approximately 10 million lines. While there is data back to 1995, we focused on the past 10 years giving us over 5GB and nearly 100 million lines to process in the entire dataset. The data contained within the files includes the following fields:

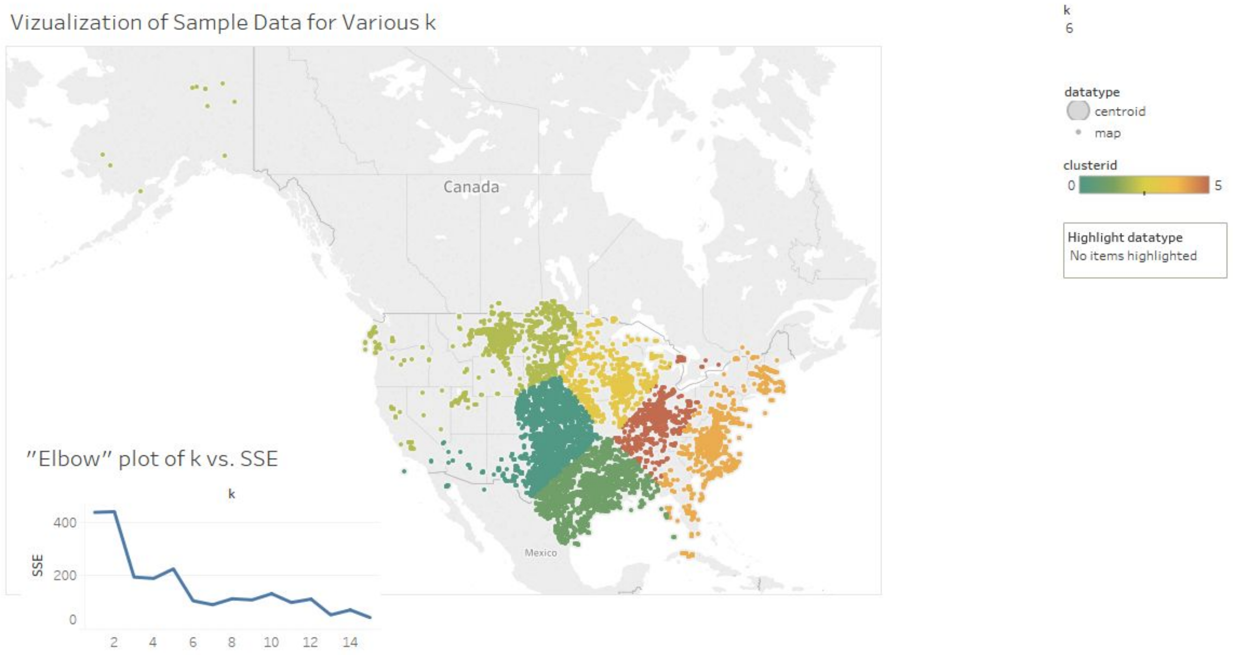
ZTIME - Date and time of observation
LON - Longitude
LAT - Latitude
WSR_ID - NEXRAD or TDWR SITE ID
CELL_ID - CELL ID UNIQUE TO RADAR SITE
PROB - PROBABILITY OF HAIL [PERCENT]
SEVPROB - PROB. OF SEVERE HAIL [PERCENT]
MAXSIZE - MAX SIZE [IN]

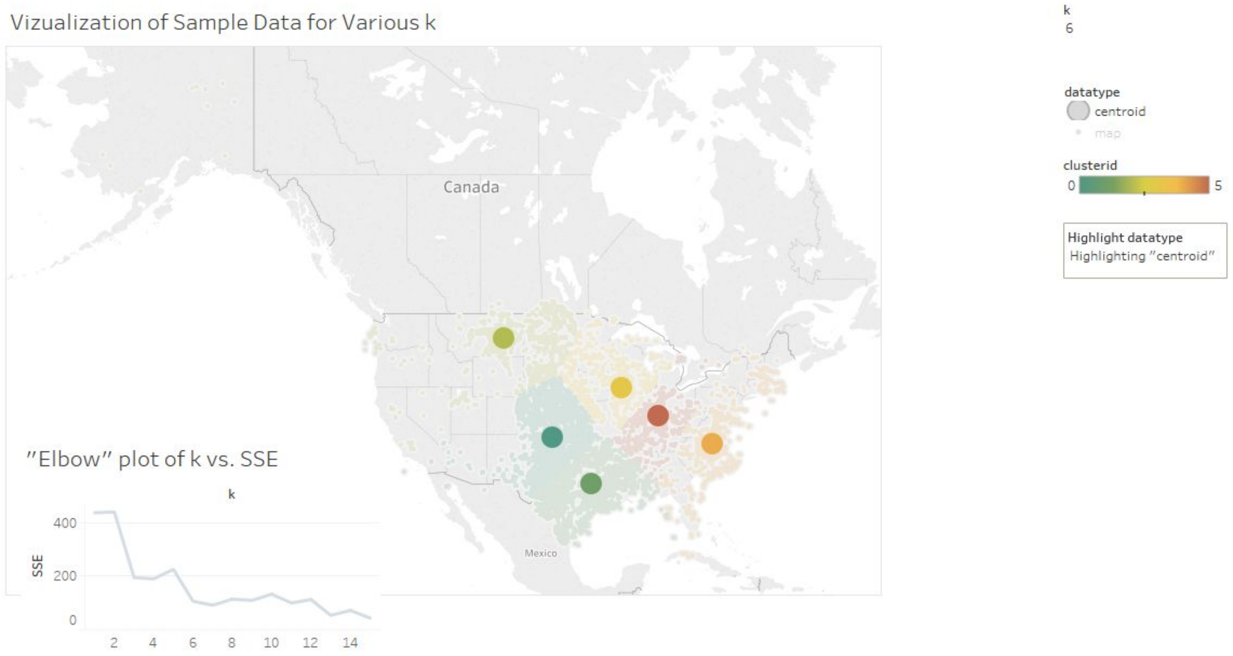
When thinking about datasets, we tried to find ones that had some business relevance. In this case, we thought about how a clustering algorithm could be utilized in the insurance industry. For hail, we hypothesized that by utilizing a clustering algorithm, an insurance company could more accurately assess the risk of loss, and thus any associated premium charges, due to severe weather, specifically hail, in a given area. A simple k-means clustering by itself will not provide much information, but in the conclusions we lay out a couple enhancements to our model that could prove fruitful.

The data itself is fairly clean and easy to parse. Provided in CSV files, the overhead of parsing is limited to removing the headers and filtering records where any of SEVPROB, PROB, or

MAXSIZE are -999. After removing these values, we can simply retain the latitude and longitude of the record. All of this filtering is handled in the initial load in SPARK. In an extension, we would consider also keeping SEVPROB, PROB and/or MAXSIZE for additional analysis.

Once we know how to parse and filter our full dataset, we set out to run some analysis locally first. We started with a very small subset of the data, taking the first 50 lines (including headers) and running the k-means algorithm on this subset. While not necessarily a representative sample, it was small enough for us to visualize the dataset and evaluate the accuracy of the k-means algorithm. After we were satisfied with this analysis, we proceeded to use a larger subset of the data. In this case, we focused on records where SEVPROB (the probability of severe hail) was greater than 90%. This resulted in approximately 20,000 records for the year of 2016. The visualization of this data is below. The data gave a good representation of the dispersion within the full dataset and thus we decided to use this as a proxy to help determine an optimal k value. As displayed in the visualization is an “elbow” plot detailing the sum of squared distances from the centroids. This analysis led us to choose k for a larger dataset in the range of 4 to 6.





The next step was to apply this analysis on AWS. The first step was getting the data into S3. While not necessarily an extremely large dataset by Big Data standards, uploading 5GB of data using a home internet connection doesn't necessarily run quickly. Reviewing recommendations on AWS, for file sizes larger than 100MB they suggest using their API and a multi-part upload call. This splits the file into smaller chunks and parallelizes the upload. In order to do so, we utilized the python package boto3 and a simple script to upload the files to S3. Again, while not a huge dataset, rough estimates would put this mechanism at more than twice as fast as the basic GUI upload. In addition to loading the 10 years of files, we also uploaded our sample file for testing purposes.

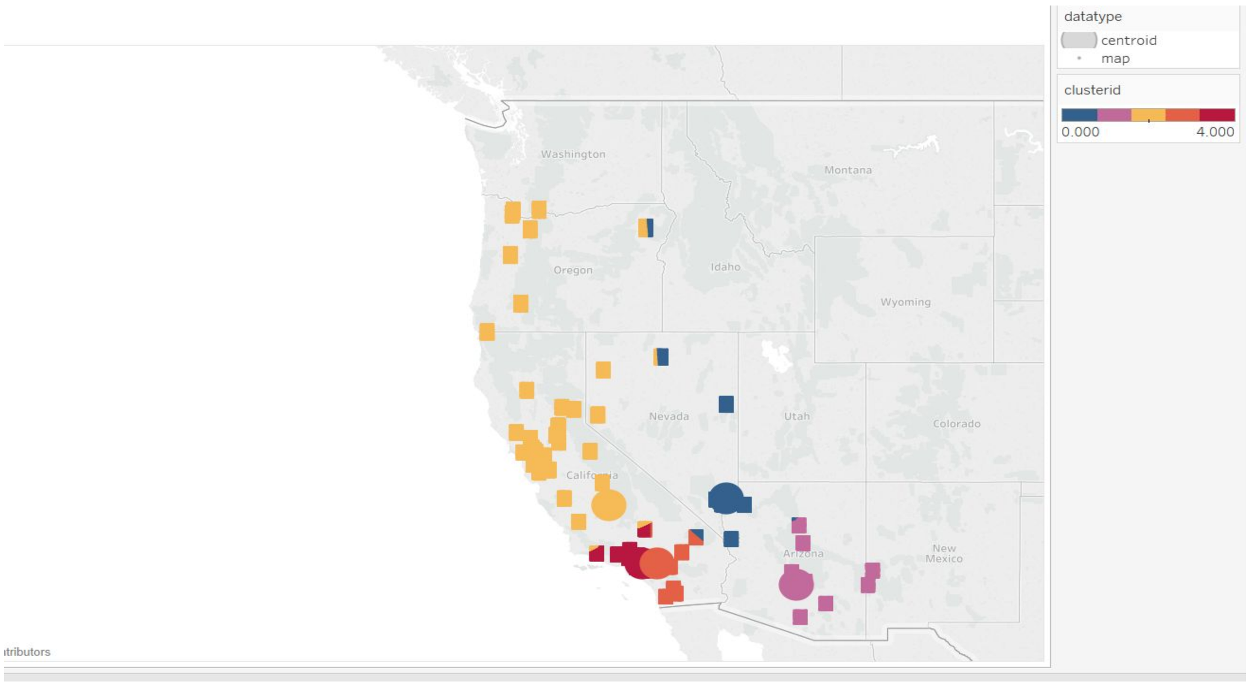
Once the data had been loaded, we set up the cluster and ran our script against the sample file. This allowed us to first test that we had the job set up correctly and secondly that our script would succeed in the cloud. There was a lot of trial and error to get the correct locations and settings in place to run. Once we had ensured that this was working correctly, we again ran on the larger filtered subset to validate outputs and get a rough indication of timing. We were now ready to run our analysis on the larger datasets.

Results

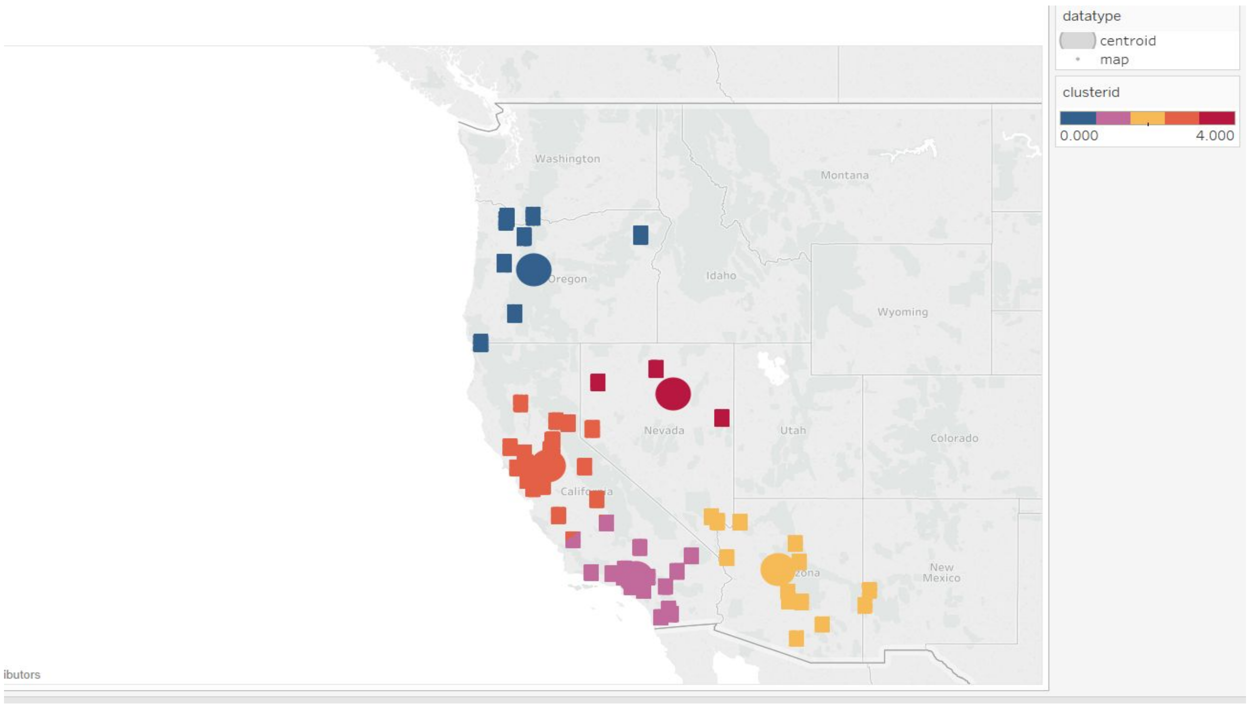
Local Results

- Device Status Data:

Device Status Data clustered at k = 5 using Euclidean Distance



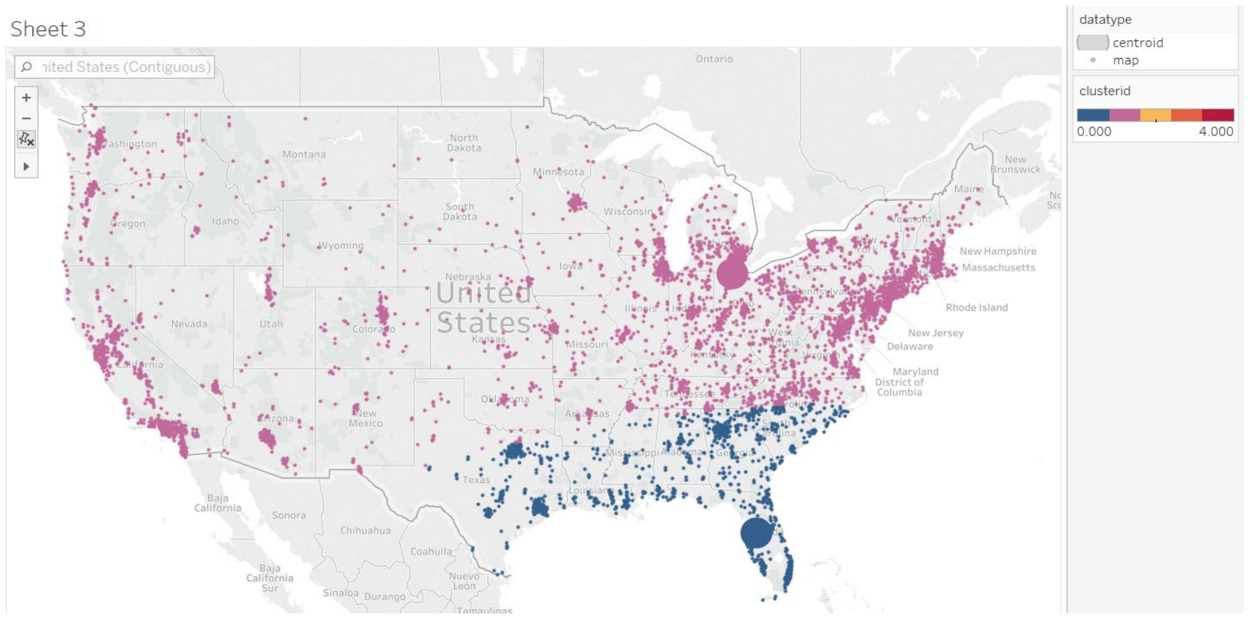
Device Status Data clustered at k = 5 using Great Circle Distance



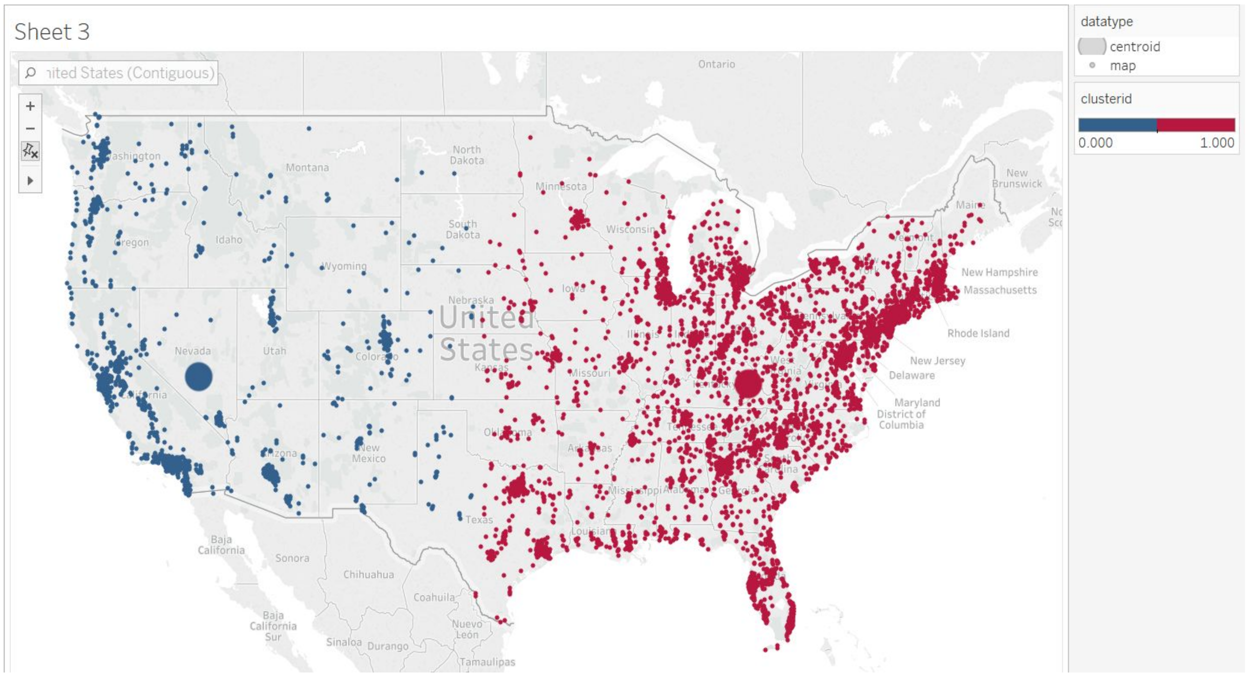
The main difference between above 2 visualizations is that clustering done with Great-circle distance (GCD) shows clusters that are more evenly distributed in geographical regions than does one done with Euclidean distance. In other words, using GCD seems to construct clusters that are more circular than oval (as in Euclidean distance), meaning that the radius of the cluster in a GCD implementation would likely to be more uniform to any directions from the centroid of each given cluster.

- Synthetic Data:

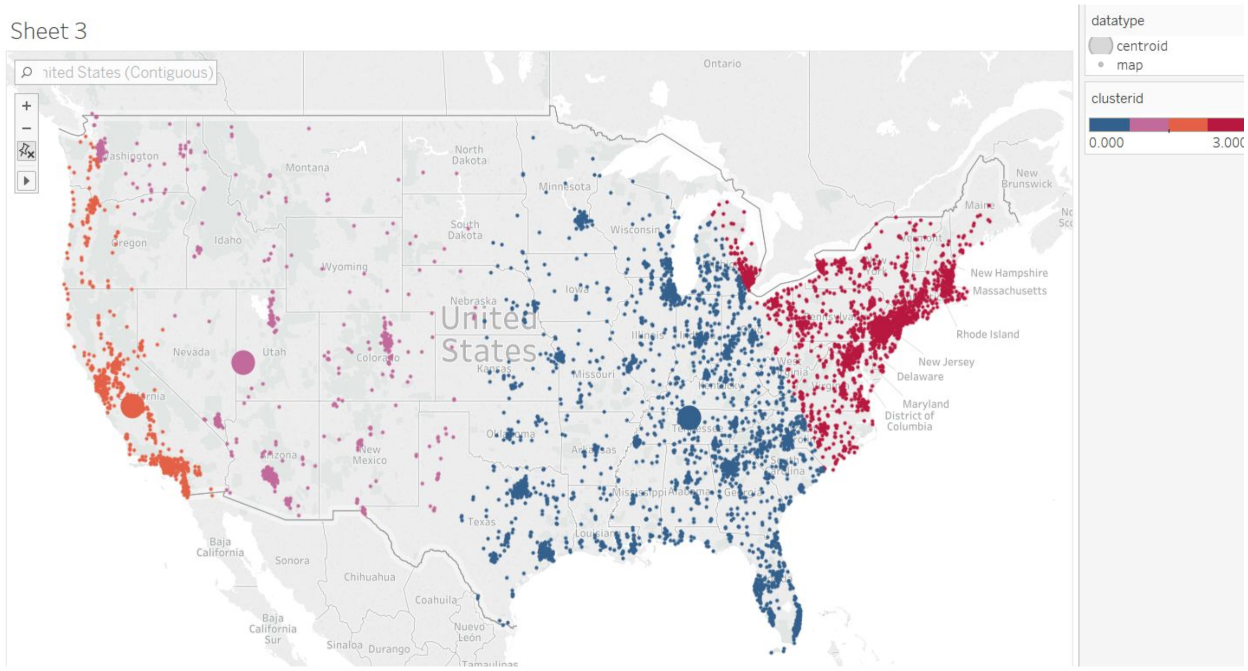
Synthetic Data clustered at k = 2 using Euclidean Distance



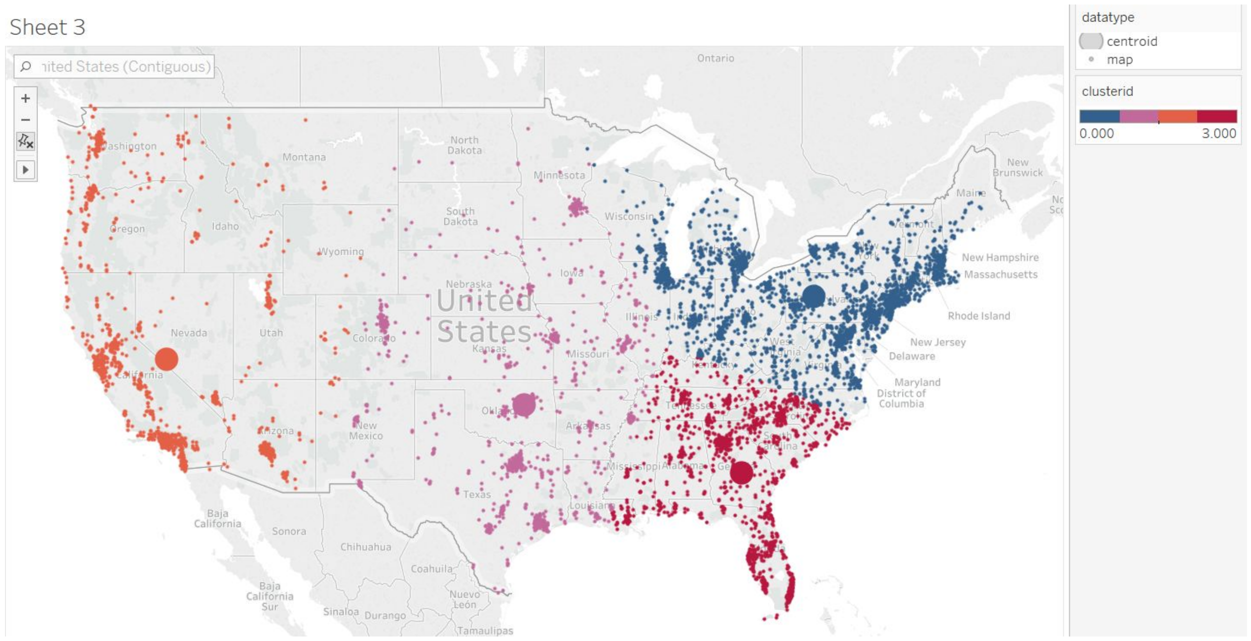
Synthetic Data clustered at k = 2 using Great Circle Distance



Synthetic Data clustered at k = 4 using Euclidean Distance



Synthetic Data clustered at k = 4 using Great Circle Distance



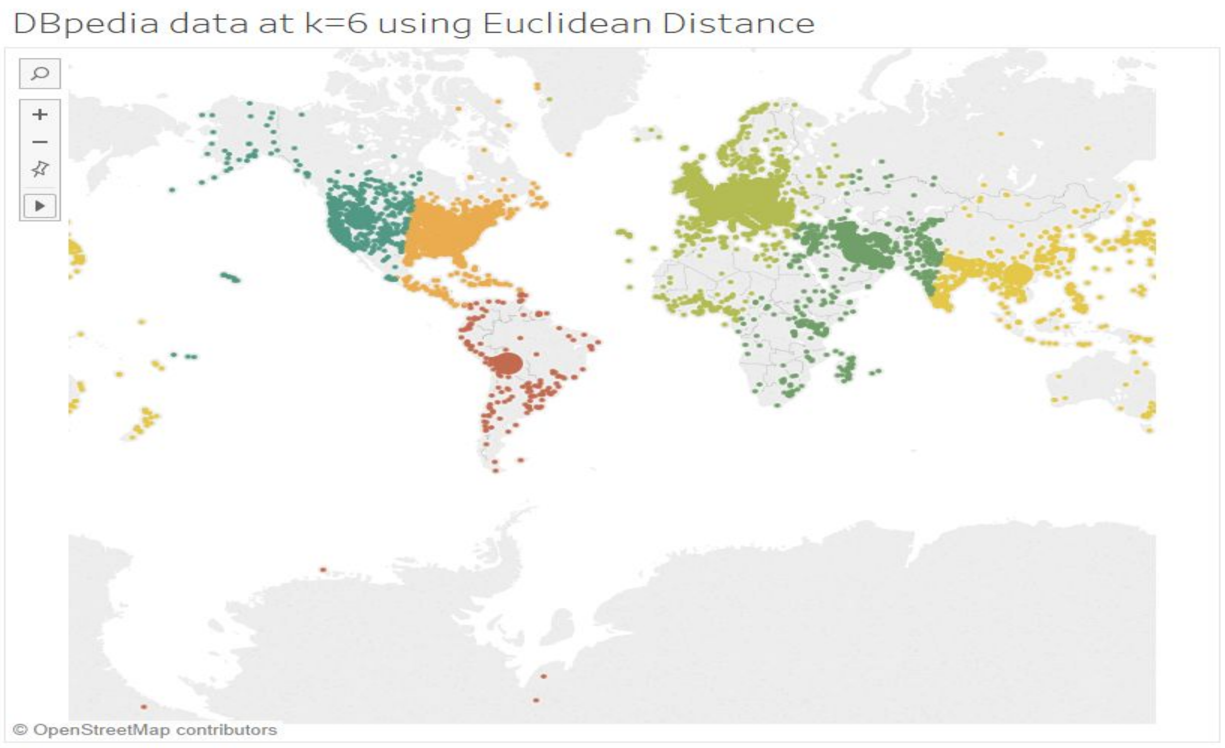
Given a data that is more geographically spread out, it seems that GCD implementation provides centroids at a more appropriately "central" location of the cluster (the distance from the centroid to edge points of the cluster seem more uniform using GCD than using Euclidean distance).

Choosing a smaller k (=2) vs. a larger k (=4) showed some interesting results. For this particular set of data, using k=2 divided up the United States into 2 clusters that are usually considered to be 2 distinct regions within the United states (for Euclidean distance, the South and the rest of the continental US and for GCD, the East and the West). It is noticeable that using GCD evenly split up the US into 2 clusters while using Euclidean distance provided 2 clusters with very different sizes. As k increased to 4, it seems that the clusters represent what seems to be more specific geographical regions of the US that are often tied to a certain culture as well (for Euclidean implementation they are - the West Coast, the Rockies, the Midwest & the South, and the East Coast; for GCD - The West Coast & the Rockies, The Midwest & Texas, the South, and the Northeast). It is an interesting observation that with a bigger k value, the clusters with denser data points seem to be smaller than others for the GCD implementation.

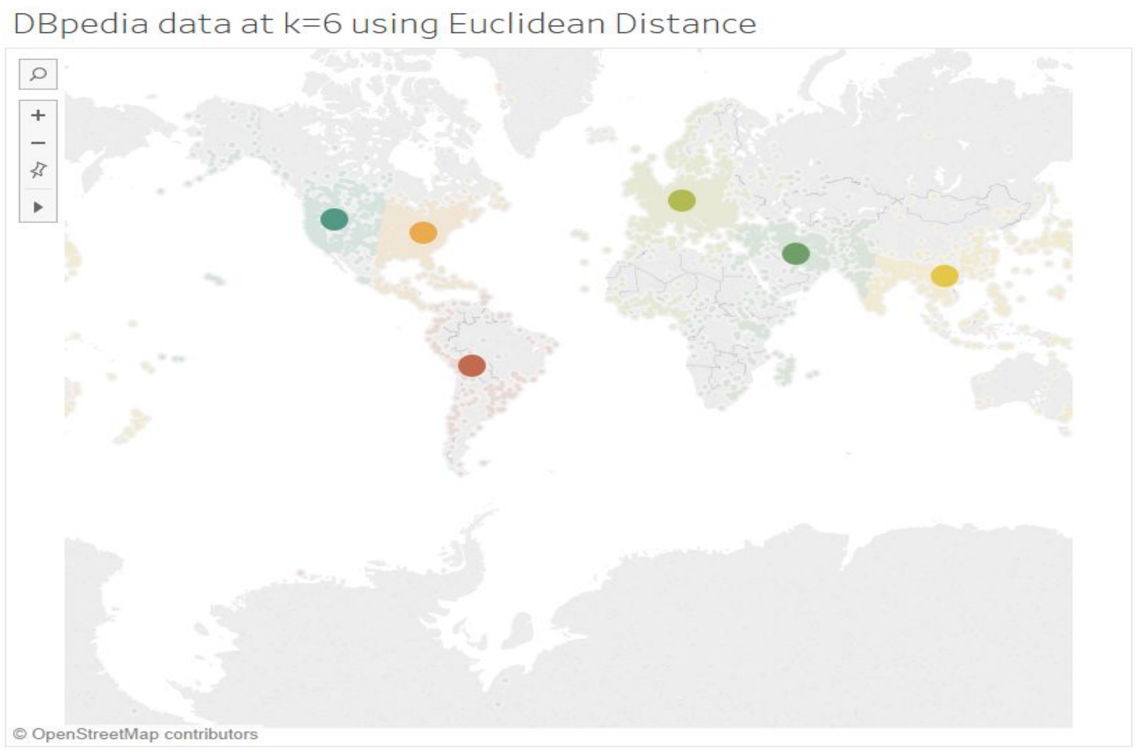
The difference in where the clusters are located depending on the implementation of distance calculations, is quite significant. Given a set of data spread across a bigger region than the device status data, the clusters are located in very different places under different means of calculating distance, especially with a lower k. Mathematically, Great-circle distance is more accurate for clustering geographic data points than it is for Euclidean distance, however for small regions, the error could be minimal enough for using Euclidean distance implementation. It is generally more difficult to implement GCD because of the requirement to convert from Cartesian coordinates to Spherical coordinates and a more complicated equation. Nonetheless, with a big region, it could be observed that using GCD would be considered to be a better practice than using Euclidean distance if accuracy is of the matter.

- DBpedia Data:

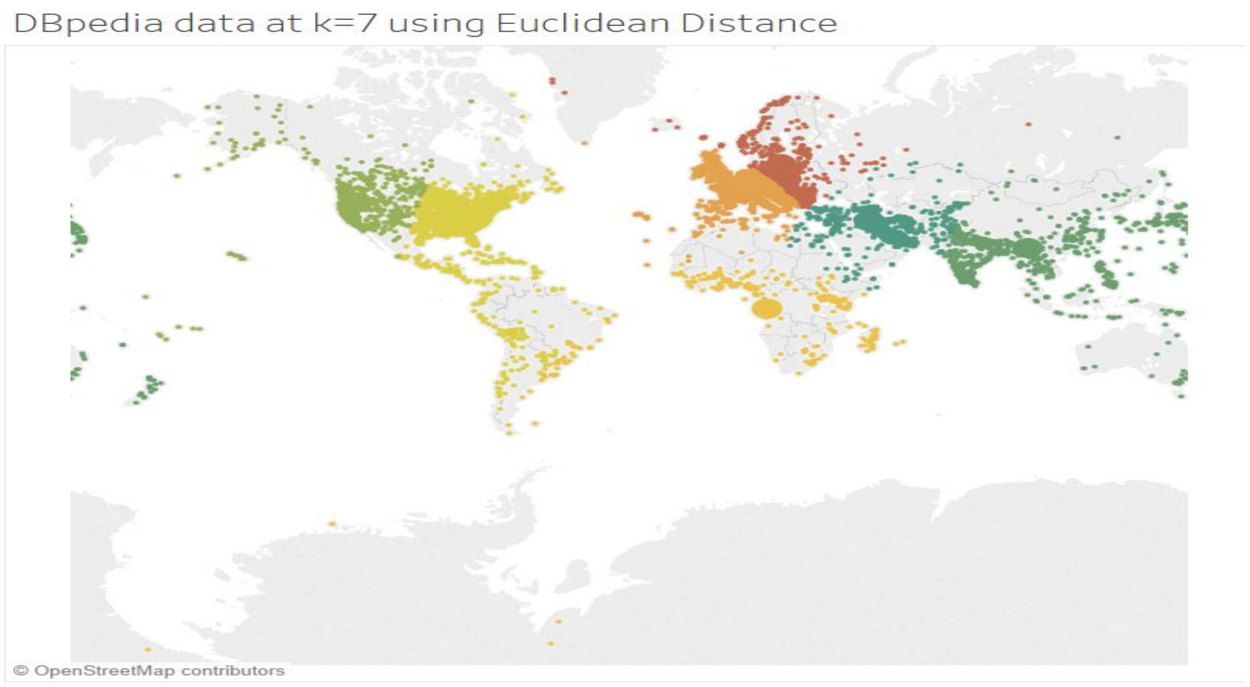
*NOTE: 10,000 entries of DBpedia data were sampled randomly at all regions

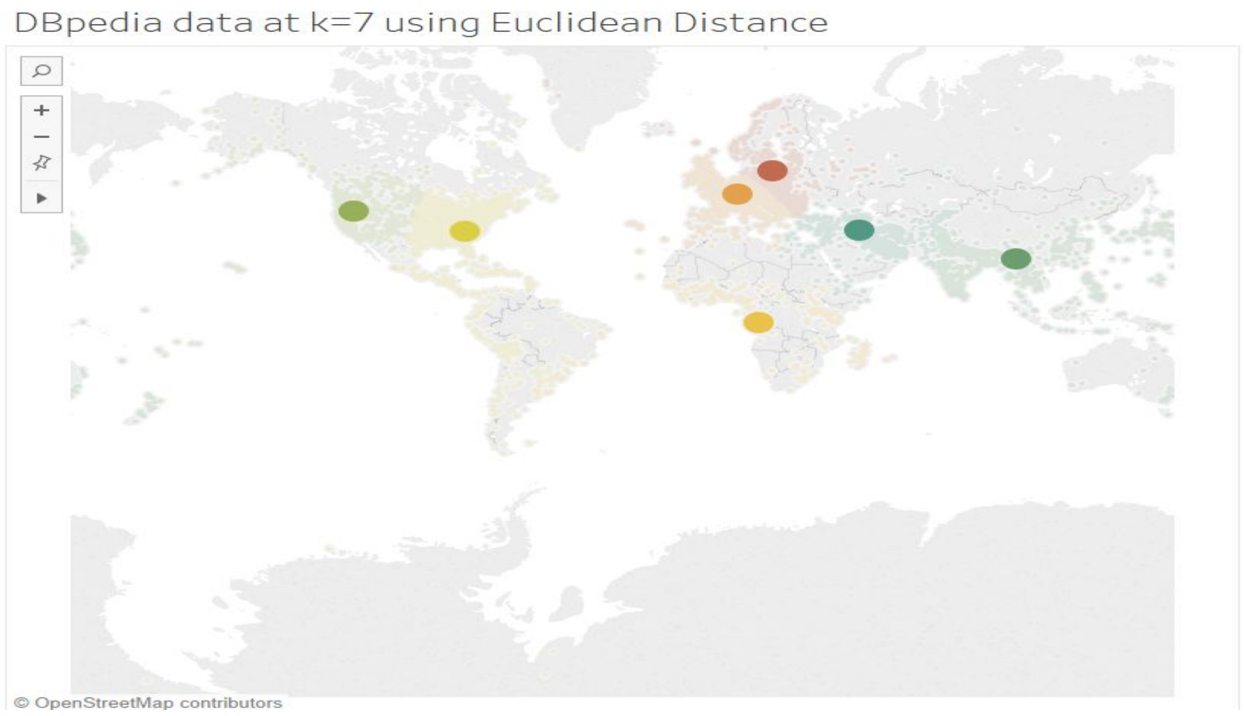

DBpedia data at k=6 using Euclidean Distance
© OpenStreetMap contributors

f


DBpedia data at k=6 using Euclidean Distance
© OpenStreetMap contributors

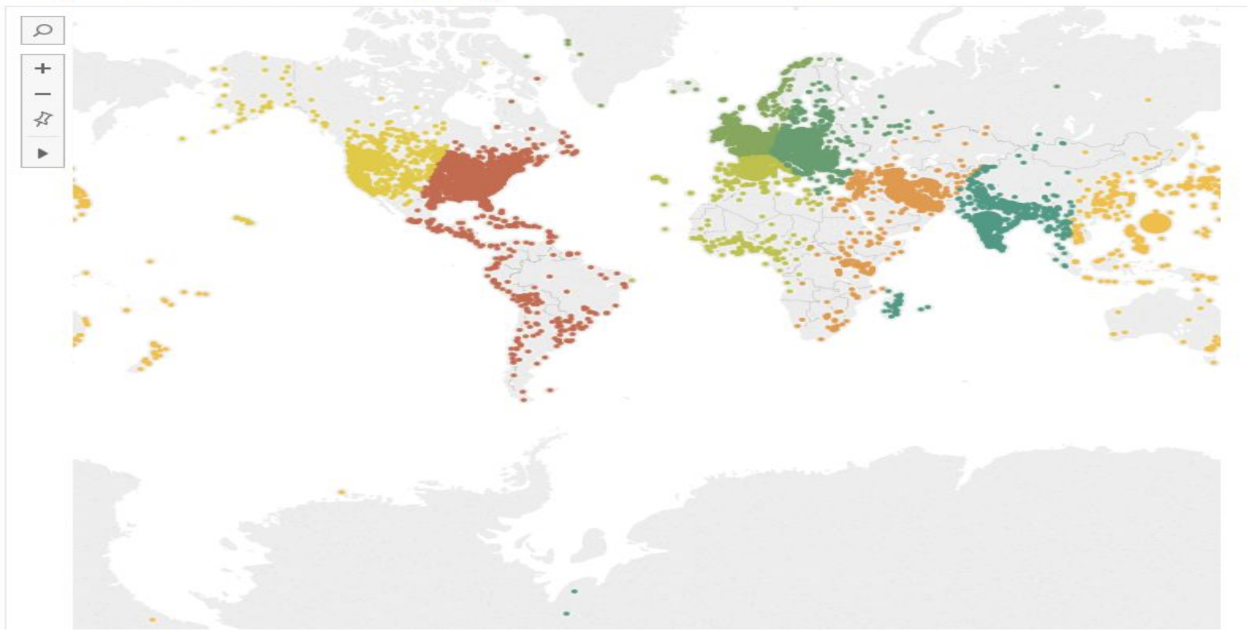
At k=6, the clusters seem to represent the following regions - East of North America, West of North America, South America, Europe & Eastern Africa, West Africa & Middle East & Western India, and Eastern India & Asia & Australia & Pacific islands. These clusters do not quite seem to be a correct representation of geographical regions that are often considered together.


DBpedia data at k=7 using Euclidean Distance
© OpenStreetMap contributors

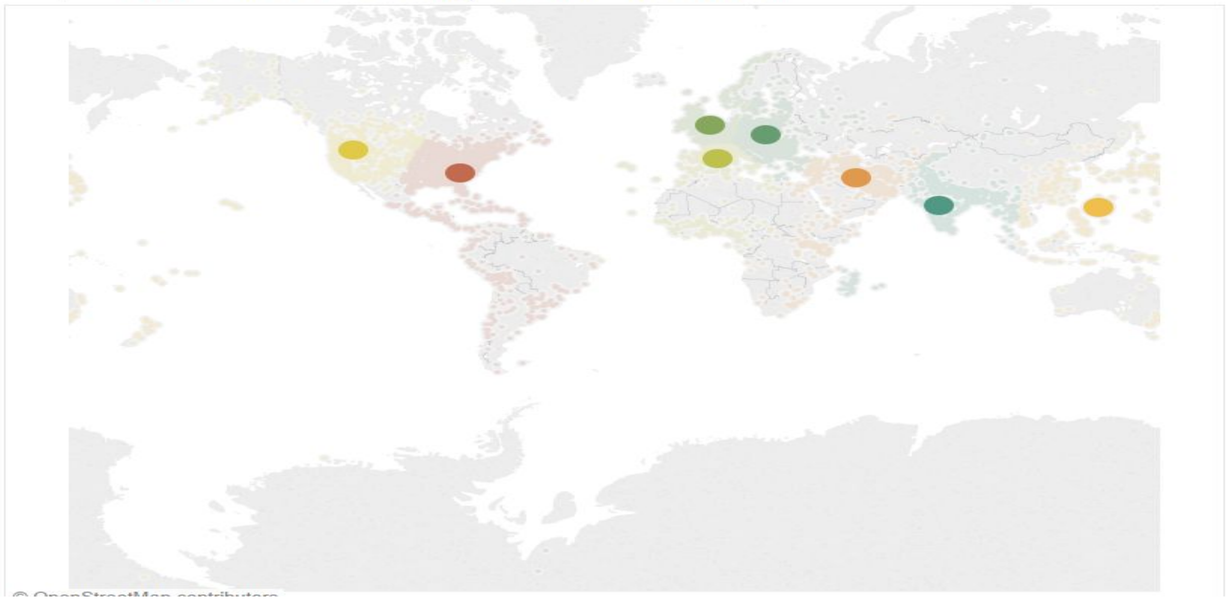

DBpedia data at k=7 using Euclidean Distance
© OpenStreetMap contributors

With k=7, some clusters still do not seem so fitting to actual geographical regions that are often considered together - East N. AMerica, West N. America & South America, Western and Central Europe, Northern & Eastern Europe, Africa, Middle East, Asia & India & Australia & Pacific Islands. Yet, with k=7, the regions seem more fitting than in k=6 with a few exceptions. In fact, there are some significant shifts in clusters as k increased to 7 from 6, especially the merging of South America and West of North America into 1 cluster.

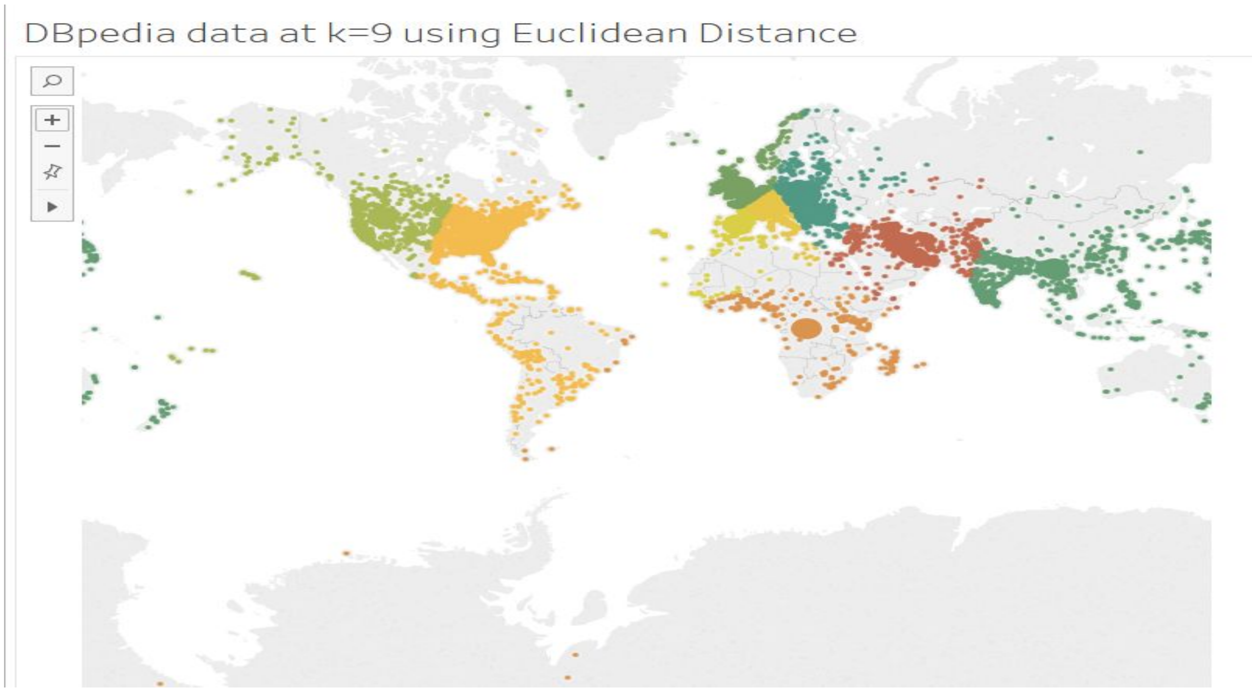
DBpedia data at k=8 using Euclidean Distance

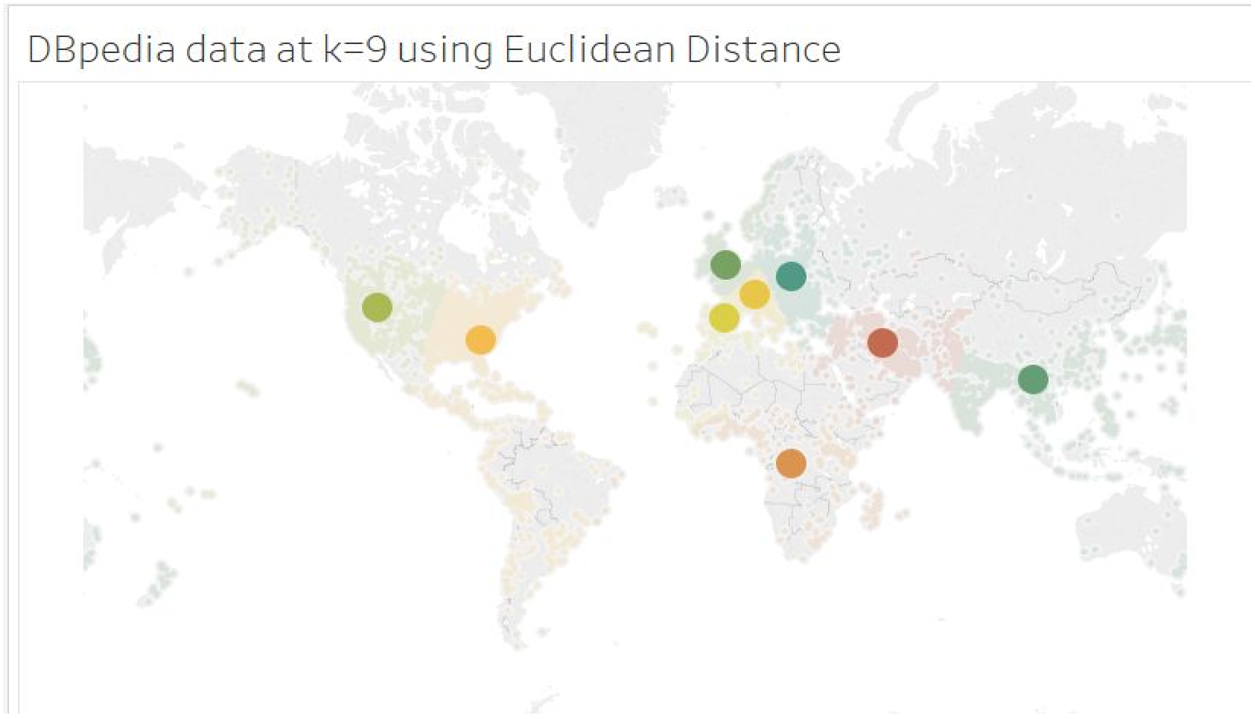


DBpedia data at k=8 using Euclidean Distance



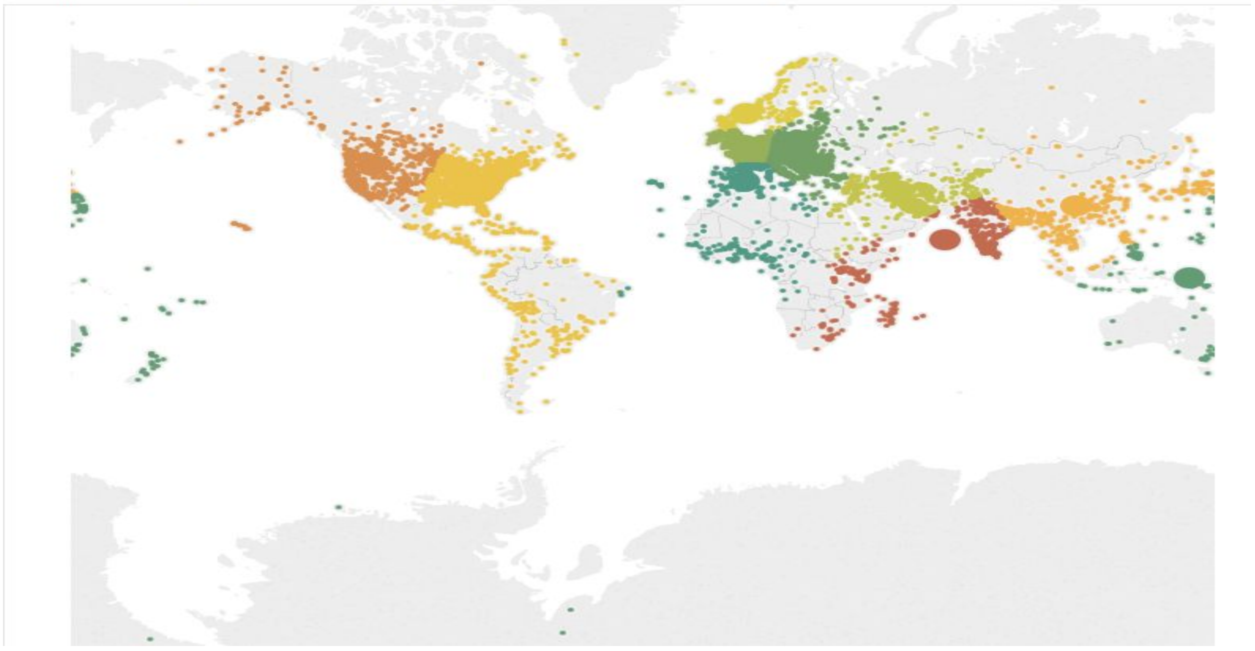
At k=8, the regions are becoming more specified - East N. AMerica, West N. America & South America, Western and Northern Europe, Central and Eastern Europe, Southern Europe & West Africa, Middle East and East Africa, Indian subcontinent, Asia & Australia & Pacific Islands. Again, there are some significant changes in where the clusters are located at.



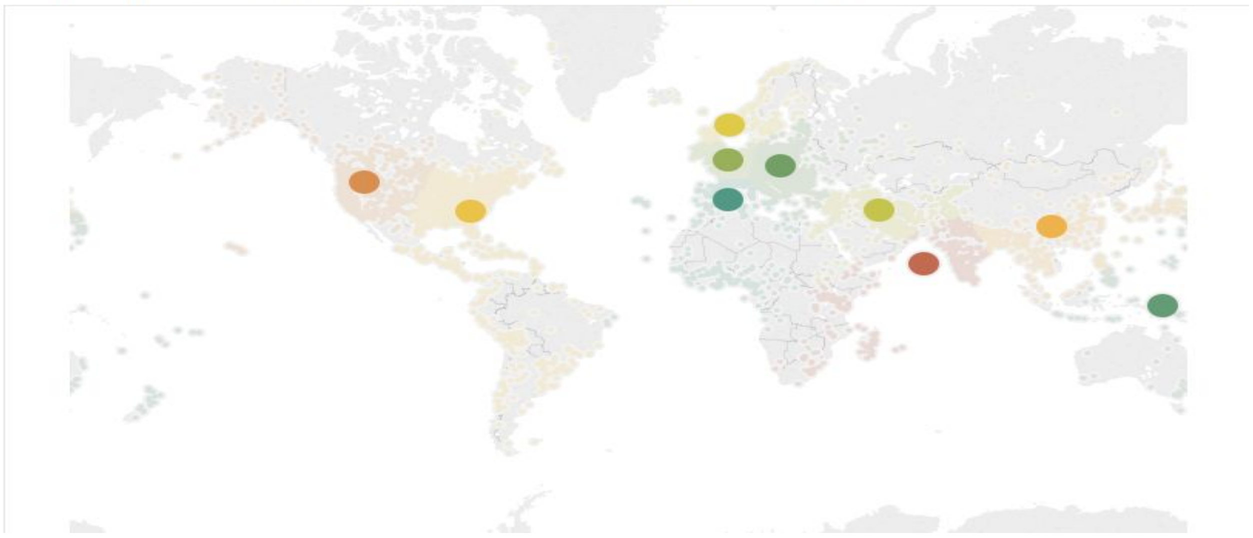


At k=9 - East N. AMerica, West N. America & South America, Western and Northern Europe, Central Europe, Mediterranean, Eastern Europe, Sub-Saharan Africa, Middle East, India & Asia & Australia & Pacific Islands. At k=9, the most of the regions with very dense data plots (Europe) is divided up into specific regions, however, less dense regions such as Asia and Australia, and the Americas are not quite divided up into specific regions. (Note, the term regions is being used interchangeably with clusters). Again, there are some significant changes to where the cluster are located at, especially with the merging of Asia and India into 1 cluster.

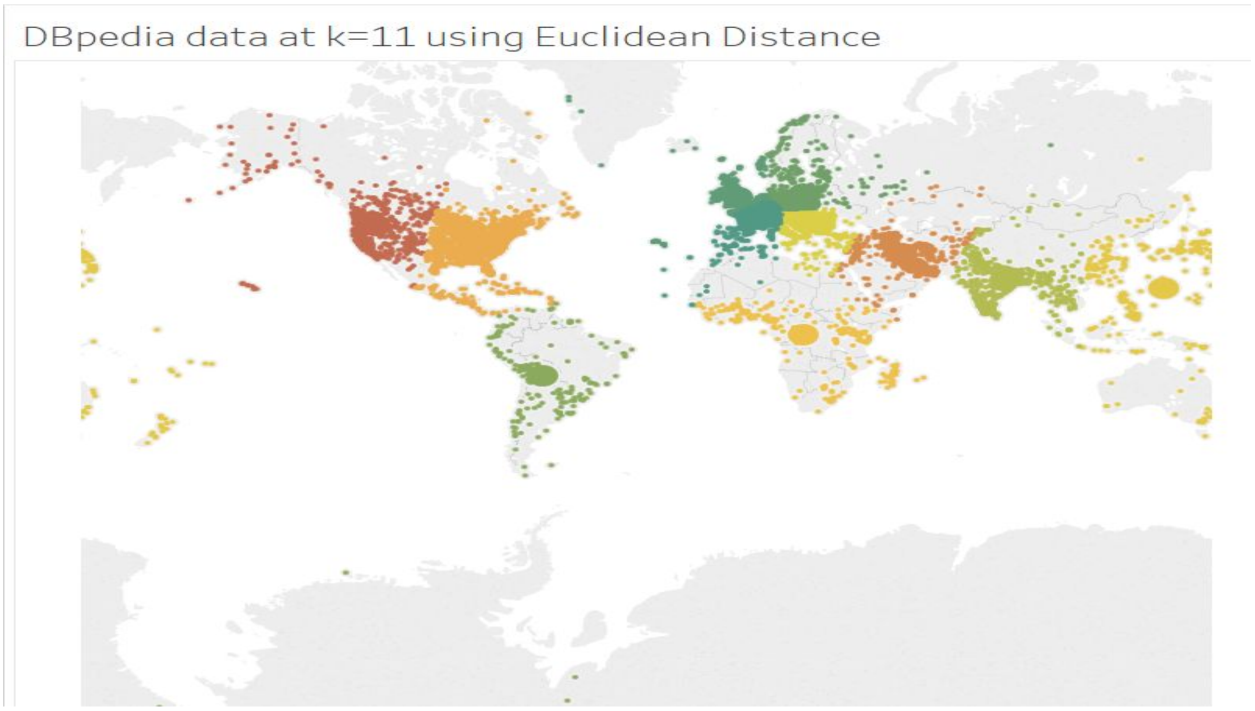
DBpedia data at k=10 using Euclidean Distance



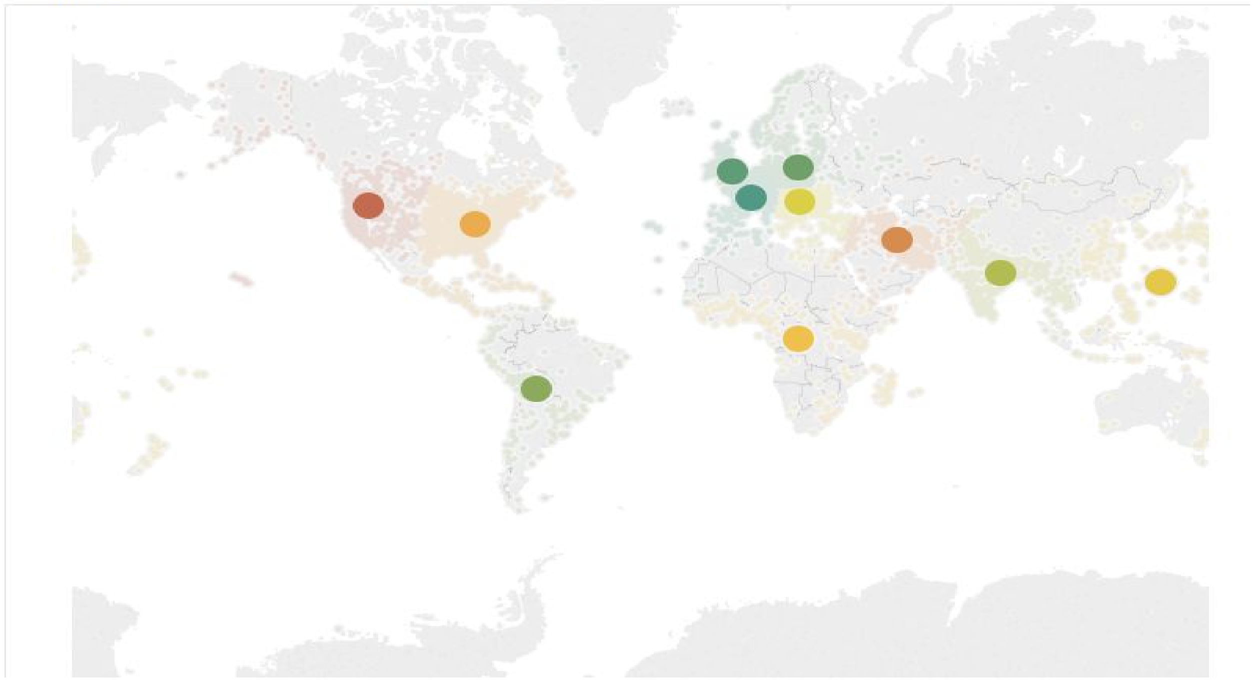
DBpedia data at k=10 using Euclidean Distance



At k=10 - East N. AMerica, West N. America & South America, Western Europe, Northern Europe, Central and Eastern Europe, Mediterranean and East Africa, Middle East, India and West Africa, Asia, Australia & Pacific Islands. As k grows, each clusters are starting to represent more specific geographical regions. However, because the clusters are being determined by distances, some clusters often represent specific segments of a region instead of a whole. Significant changes of where clusters are located is again seen as parts of Africa have moved to other clusters.



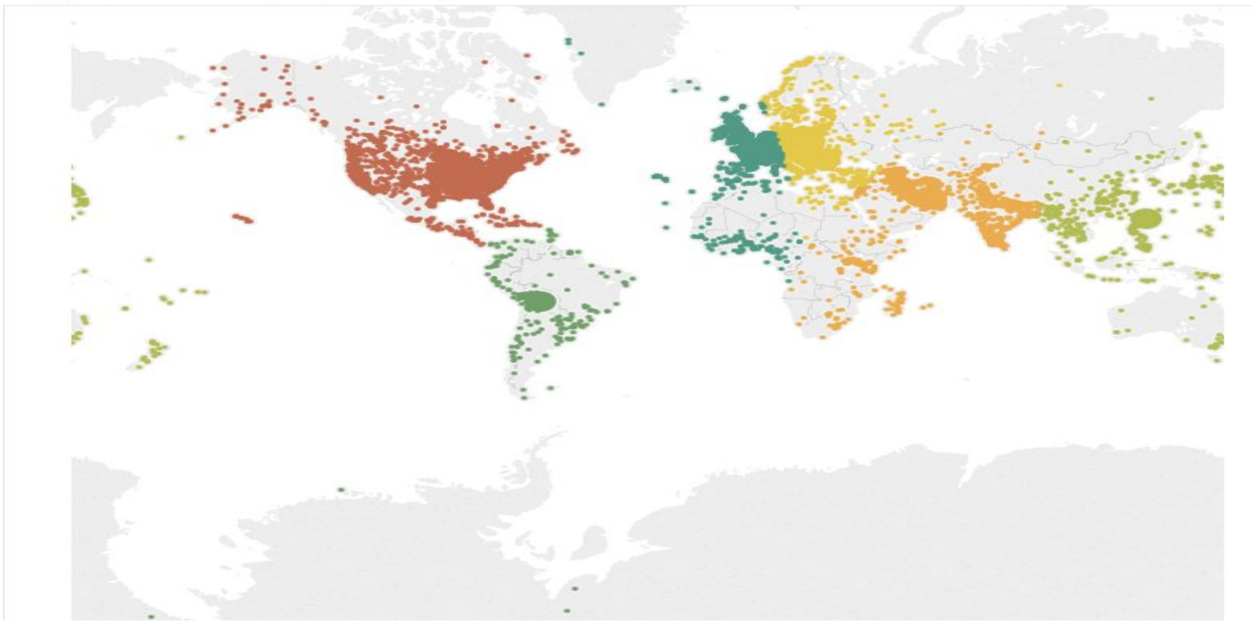
DBpedia data at k=11 using Euclidean Distance



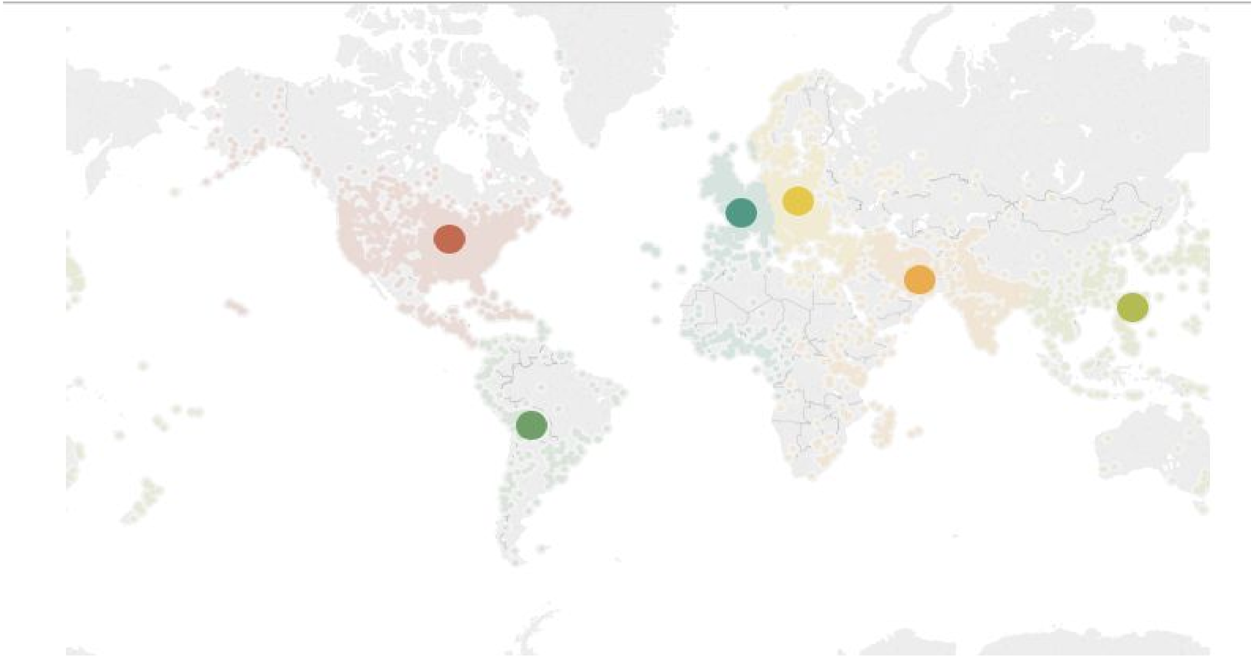
At k=11 - East of N. America, West of N. America, South America, Northern Europe, Western Europe, Central Europe and Mediterranean, Eastern Europe, Middle East, Sub-Saharan Africa, Indian subcontinent, Asia and Australia. This distribution of clusters seem to be the most accurate in terms specific geographical regions that corresponds to the clusters.

However, true clusters might not be represented by these geographical regions. DBpedia data may be more complicatedly related then commonly considered geographical regions. This notion suggests that finding the true value of k and determining initial centroids would be important to clustering the DBpedia data. Also, since the data is spread throughout the entire globe, the use of Euclidean distance is likely not appropriate in determining where the true clusters are at. Therefore, what the appropriate k value is cannot be argued properly until the discussion of changes in clusters using the GCD implementation.

DBpedia data at k=6 using GCD

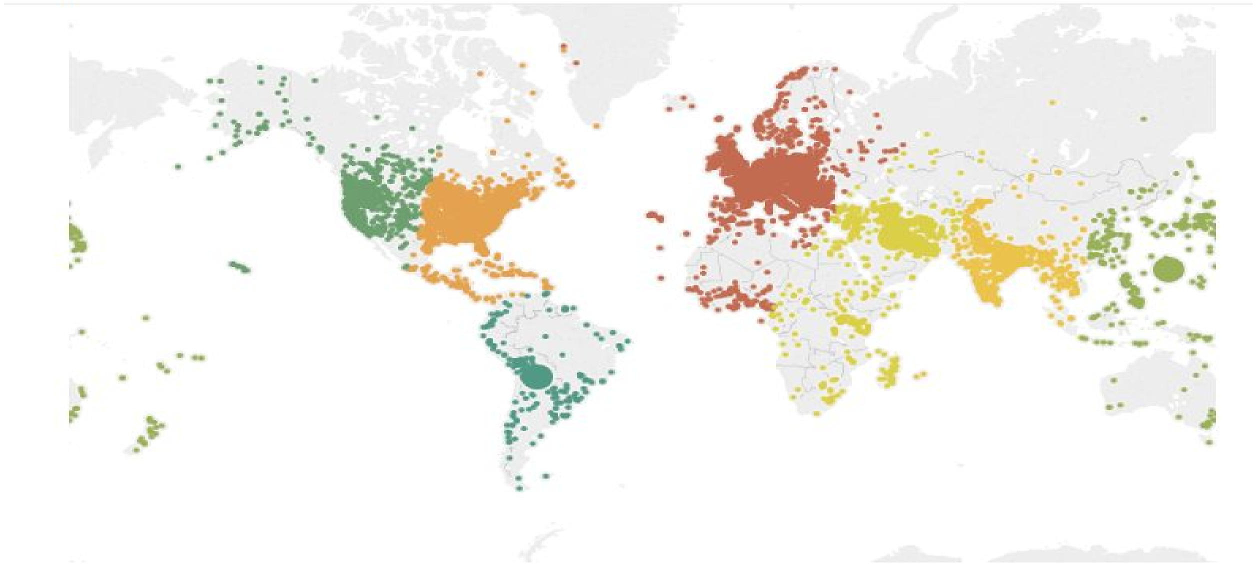


DBpedia data at k=6 using GCD

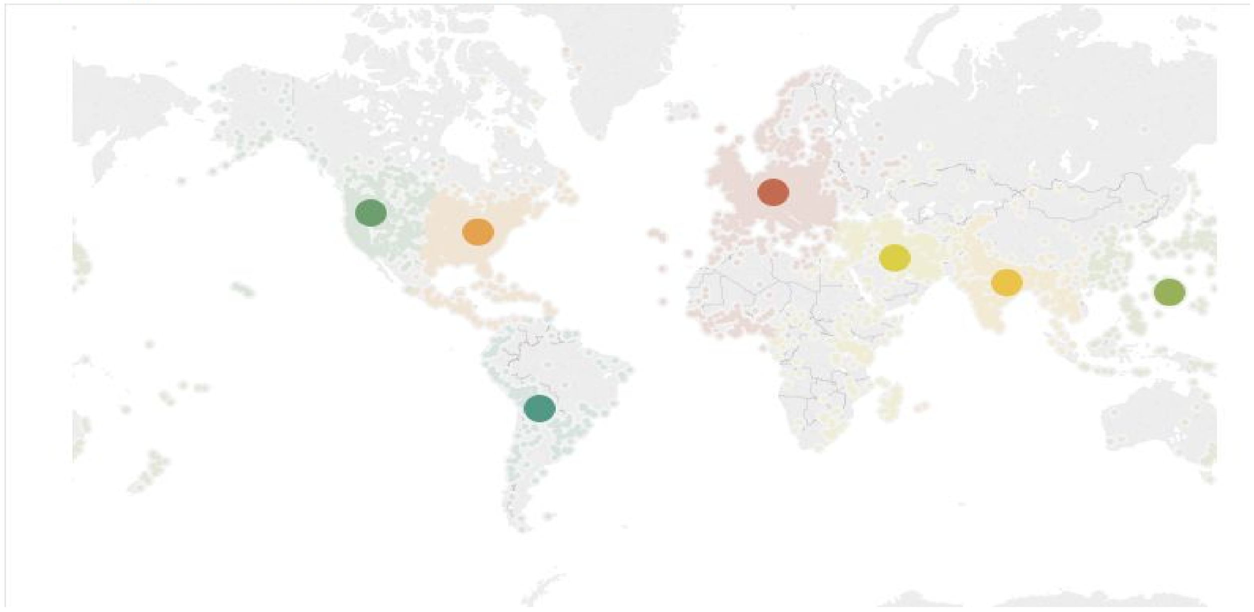


For GCD implementation: at k=6, the clusters seem to be represent regions that are typically represented as common geographical regions we often consider while Africa is divided up and clustered together with either Western Europe or Middle East. - North America, South America, Western Europe and East Africa, Eastern and Northern Europe, Middle East and West Africa and India, and Asia and Australia.

DBpedia data at k=7 using GCD

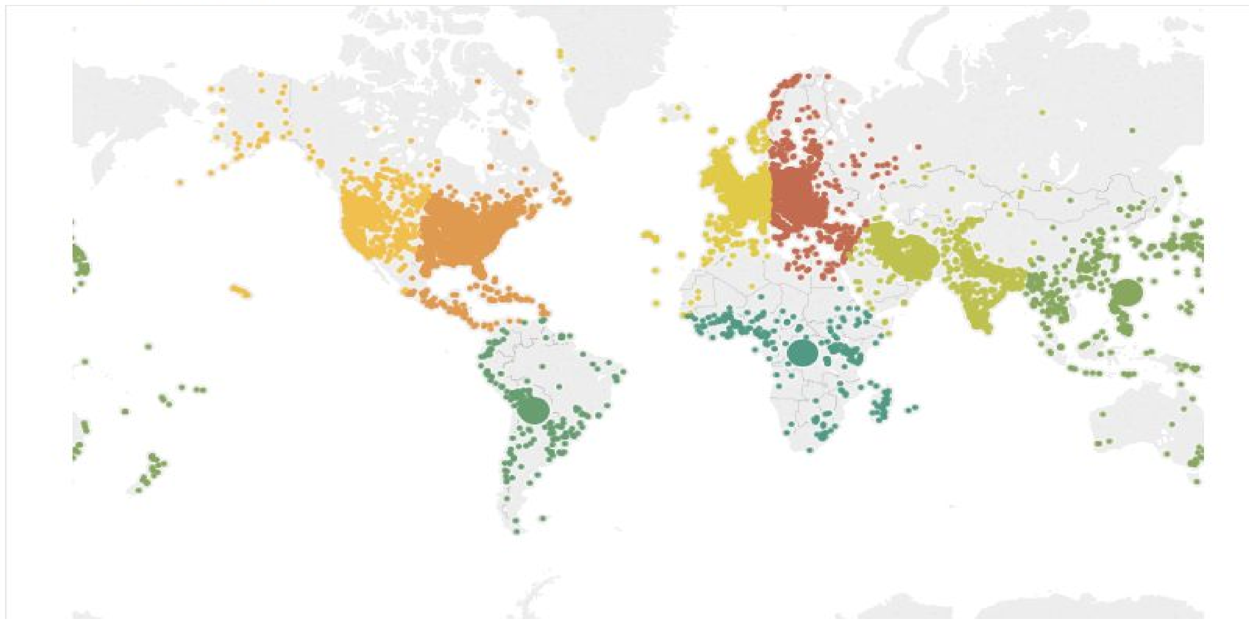


DBpedia data at k=7 using GCD

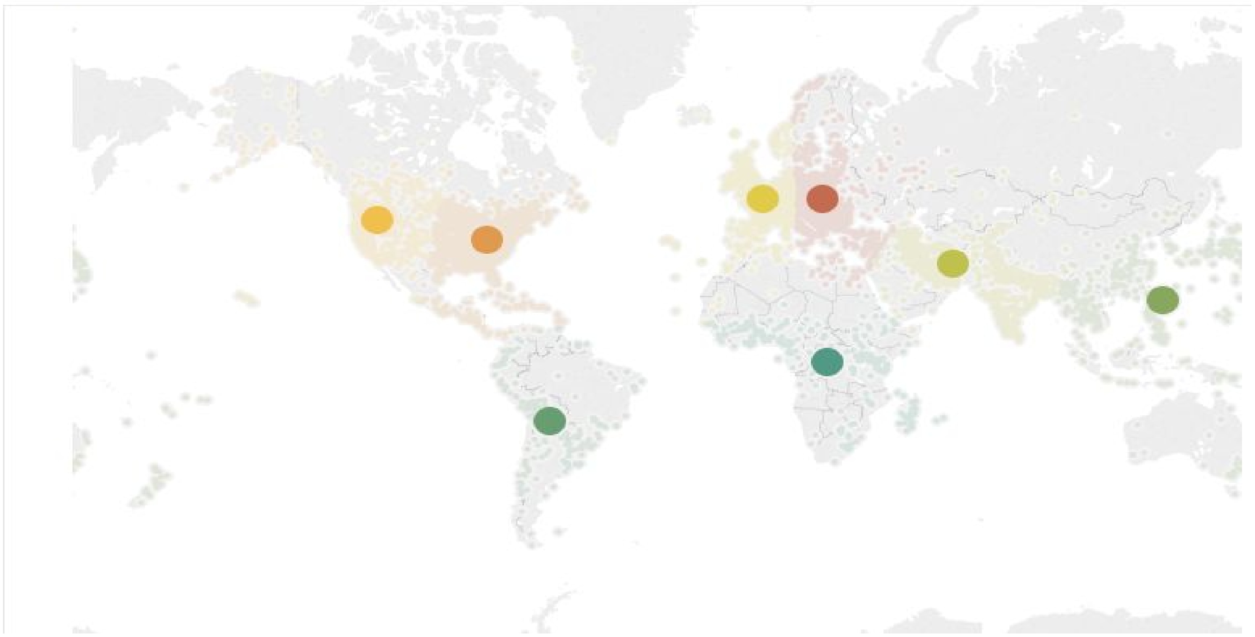


At k=7, North American cluster is divided up into 2 separate clusters (likely due to high density of data points in North America) while europe has been combined into 1 cluster and the Indian subcontinent formed a new cluster.

DBpedia data at k=8 using GCD

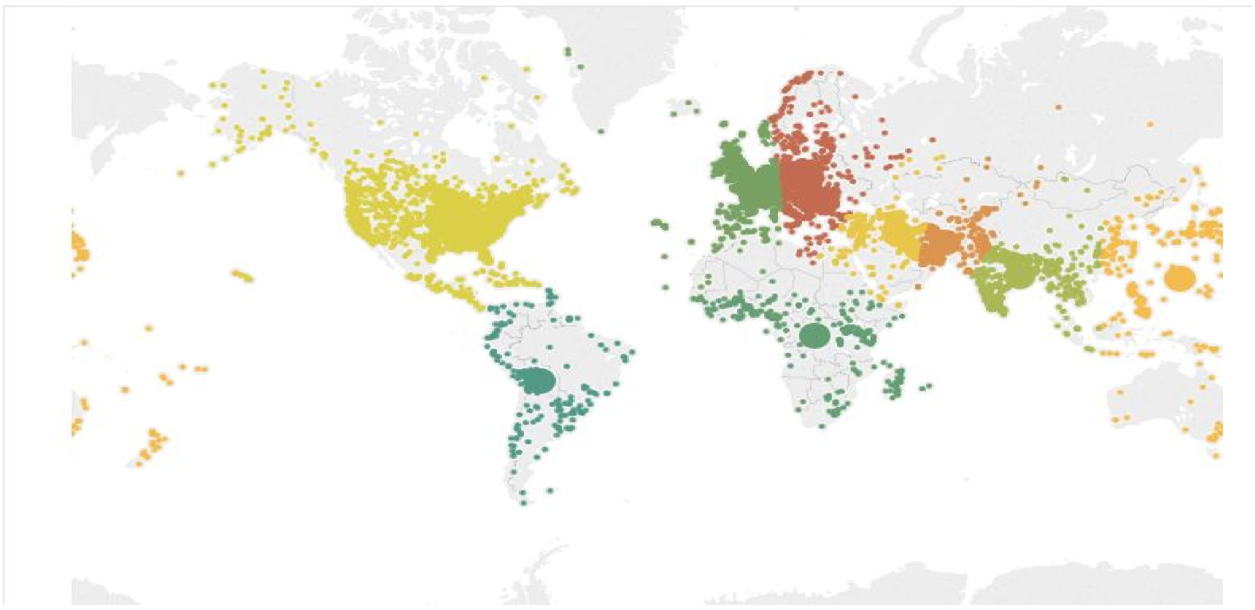


DBpedia data at k=8 using GCD

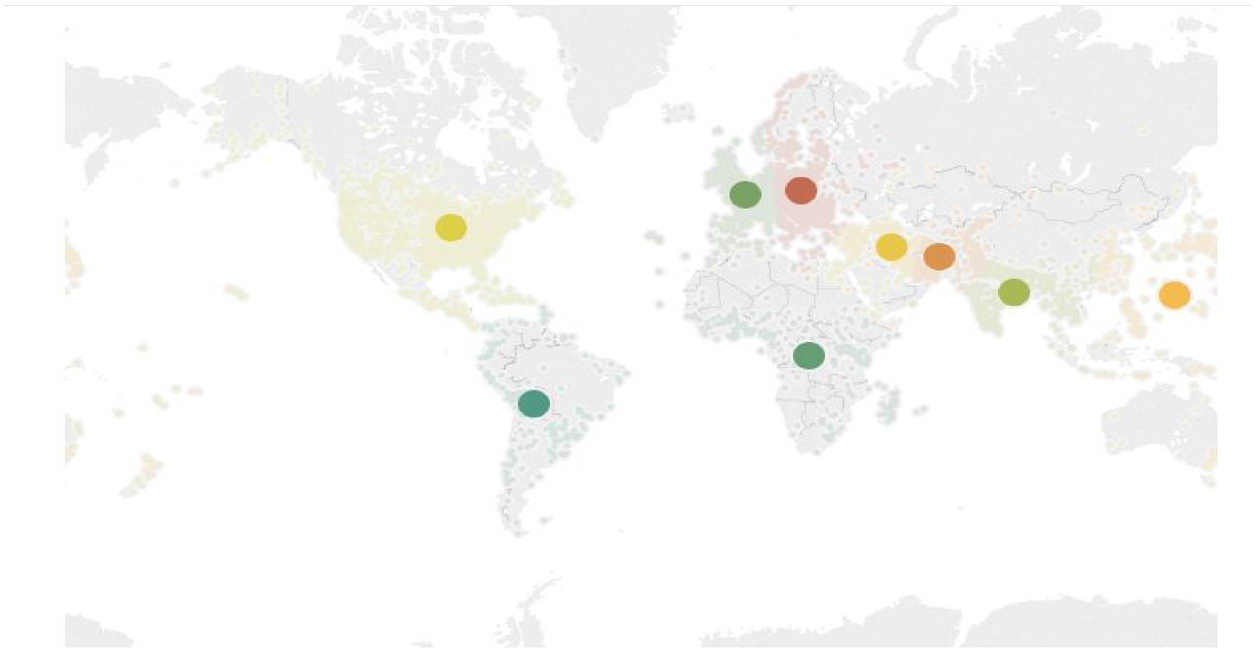


At k=8, Europe is divided up into 2 separate clusters and a new cluster for Africa emerged while the clusters that represented Middle East and the Indian subcontinent at k=7 are now merged into 1 cluster. Also, it can be observed that clusters representing South America, Africa, and Asia & Australia are larger and sparser than other clusters, which is consistent with previous observations of clusters with GCD implementation of k-means algorithm. Additionally, from k=7 to k=8 there have been some significant changes to where the clusters are located at, especially with Africa, which did not have its own cluster before.

DBpedia data at k=9 using GCD

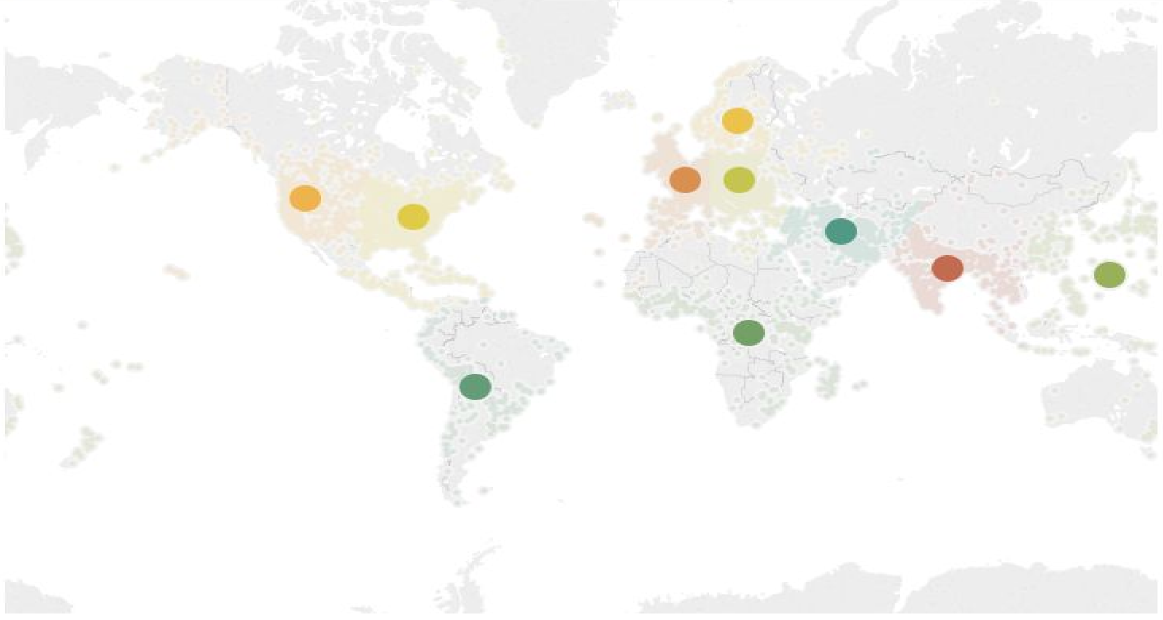


DBpedia data at k=9 using GCD

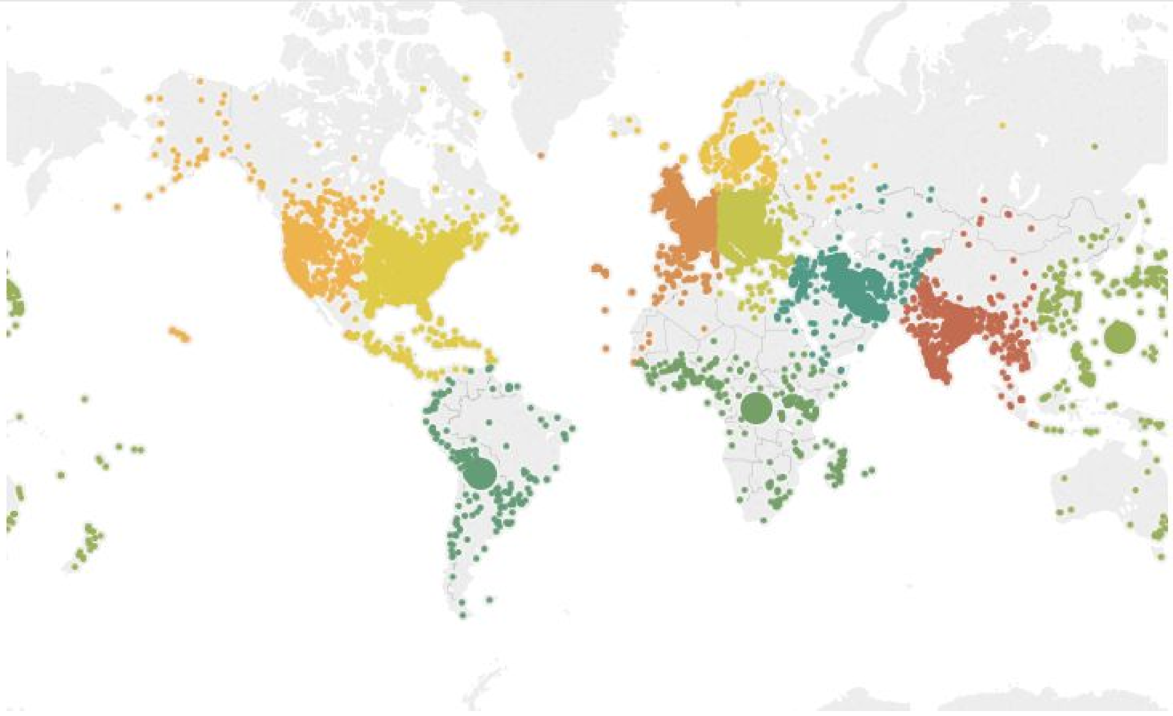


At k=9, the 2 clusters in North America merged while the cluster that previously represented Middle East is divided up into the Arabian side and Persian side, forming 2 clusters. This is an interesting observation that may suggest there may be a difference in the density of data points is between the Middle East and North America. Also, starting at k=9, all the clusters seem to represent geographical regions commonly considered together - North America, South America, Western Europe, Eastern Europe, Africa, Arabia, Persia, Indian Subcontinent, Asia & Australia. Furthermore, the where the clusters are located at do not seem to have changed much from clusters at k=8.

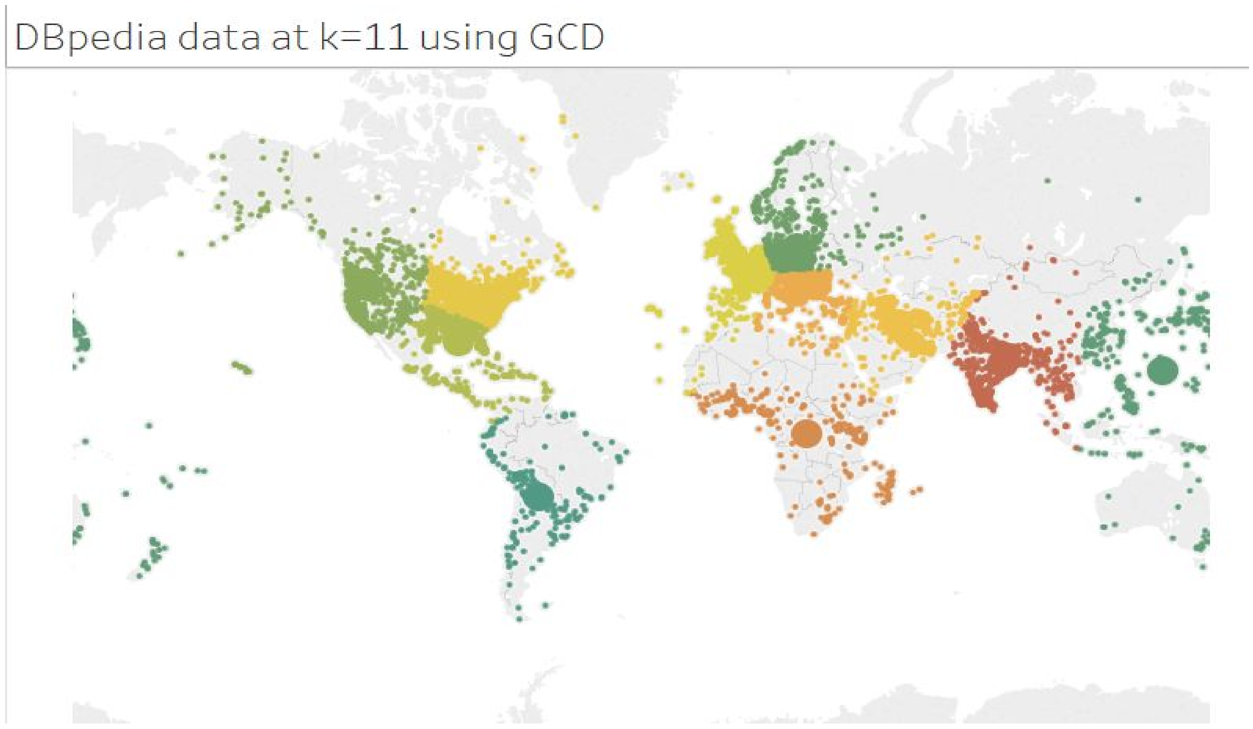
DBpedia data at k=10 using GCD

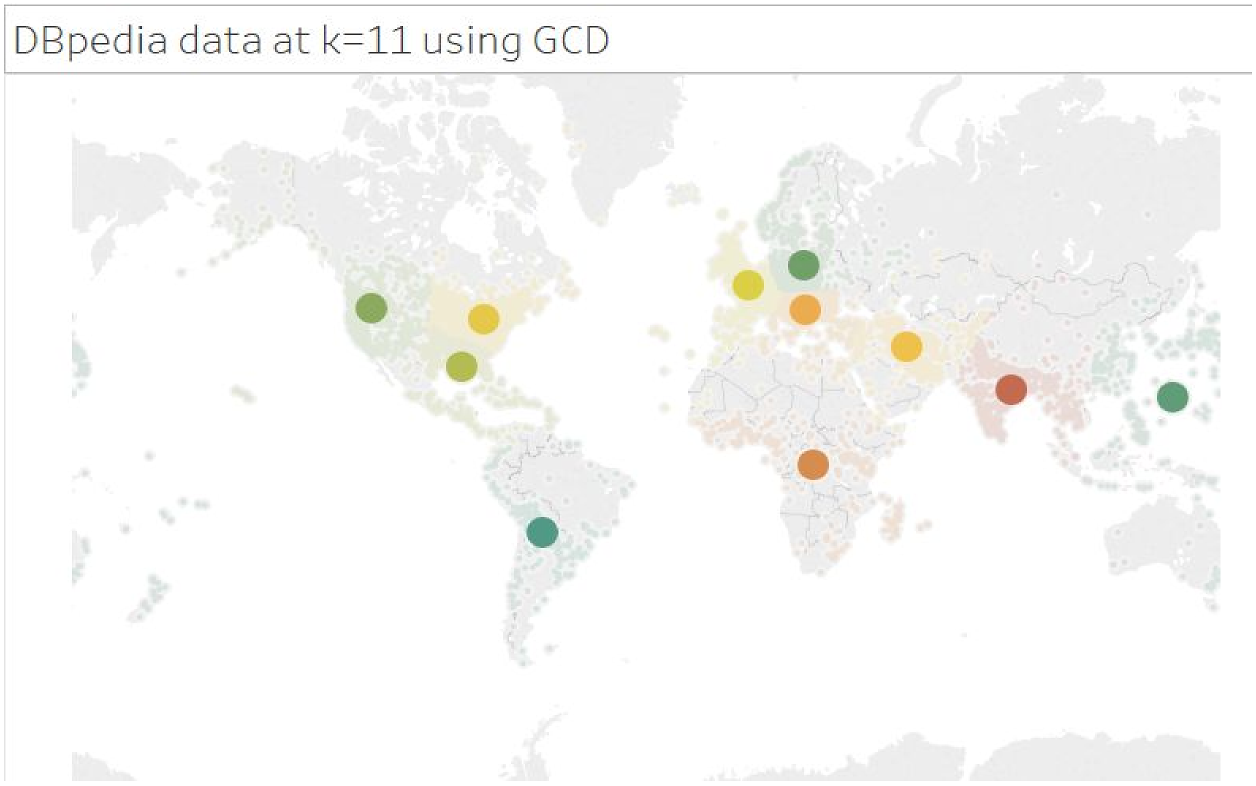


DBpedia data at k=10 using GCD



At k=10, North American cluster is again divided up into 2 clusters while the 2 Middle Eastern cluster merged and a new Northern Europe cluster has emerged. Again this suggests possible difference in density of data points between Northern Europe and the Middle East. As well as difference in density within Europe. Additionally, the locations of where the clusters are located did not change so much from k=8. Some clusters seem to just have been broken up into more clusters.





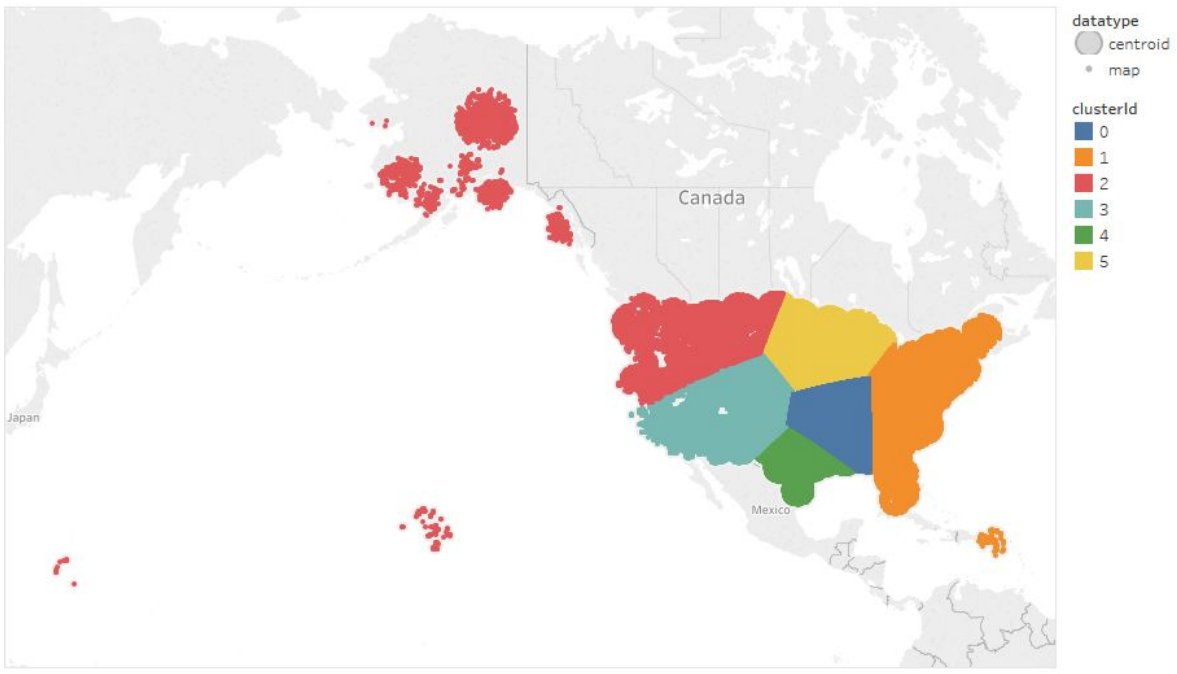
At k=11, some interesting results are observed. The clusters seem to correspond to very distinct regions of the world - South America, Northeast of North America, South East of North America, West of North America, Western Europe, Central and Northern Europe, Eastern Europe, Africa, Middle East, Indian Subcontinent, Asia and Australia. These regions are commonly considered to be the regions that are often divided up and grouped together in terms of cultural similarities.

It should be noted that starting from k=8, with the emergence of a new cluster in Africa, the change in clusters seem to be merely divisions of pre-existing clusters being broken up apart into smaller clusters. In a mathematical view, it can be observed that the clusters converge starting at k=8 since there are no major changes to where the centroids are located at. However, considering the geographical nature of the data set, any k could be appropriate starting at k=9, where the clusters seem to each represent a single geographical area often considered by itself. Of course, since the locations of the centroids seem to have converged at k=8, any k>=8 would be okay to use depending on what to extract from the data set. For example, if analysis of very specific regions of the world was required, a high k value would be of need.

Cloud Results

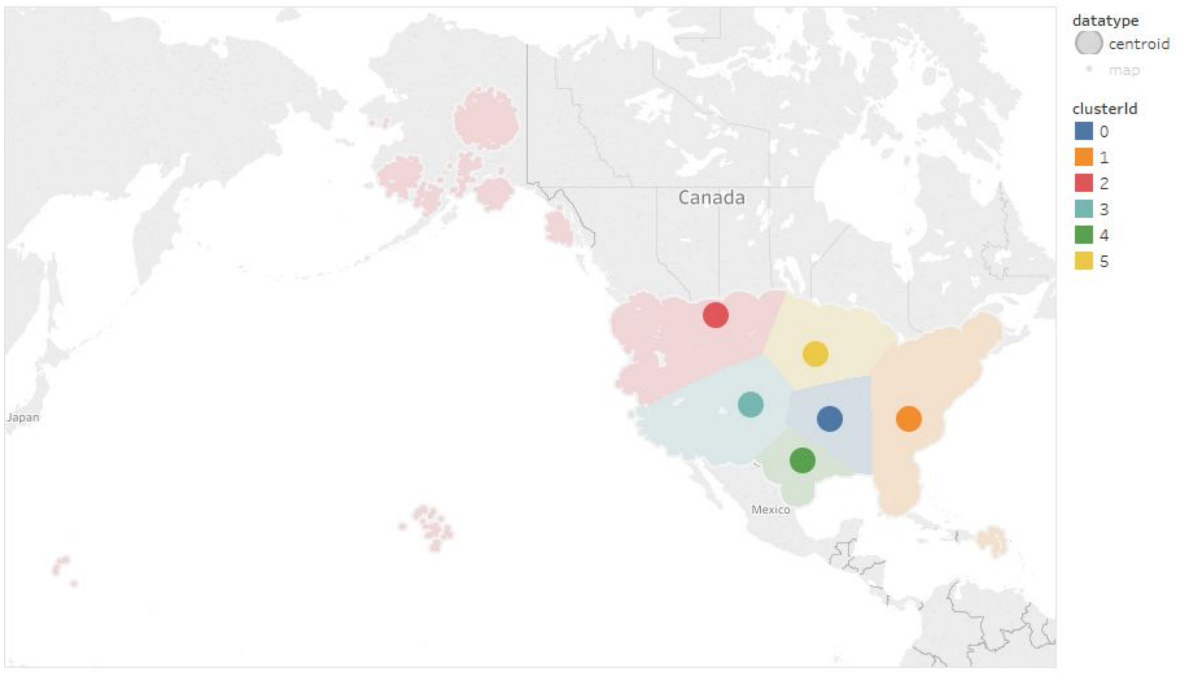
Once on the cloud, we ran a few sample cases to ensure our setup was working properly. The first sample case was running the clustering on the smallest sample of only 50 records. Once we had the setup working for this, we ran the analysis on the sample of 20,000 data points as in the above to validate that the clustering was producing relevant and comparable results. Finally, the first large dataset that we ran our analysis on was a single year with no filters and k=6. Although the k=6 was predicated on severe hail probabilities in testing, we believed that it would be a good comparison case. This dataset began as approximately 10.6 million records and filtered into approximately 6.8 million after the removal of invalid records. The results of this follow.

2016 Full Dataset; k=6



Map based on lon and lat. Color shows details about clusterId. Size shows details about datatype.

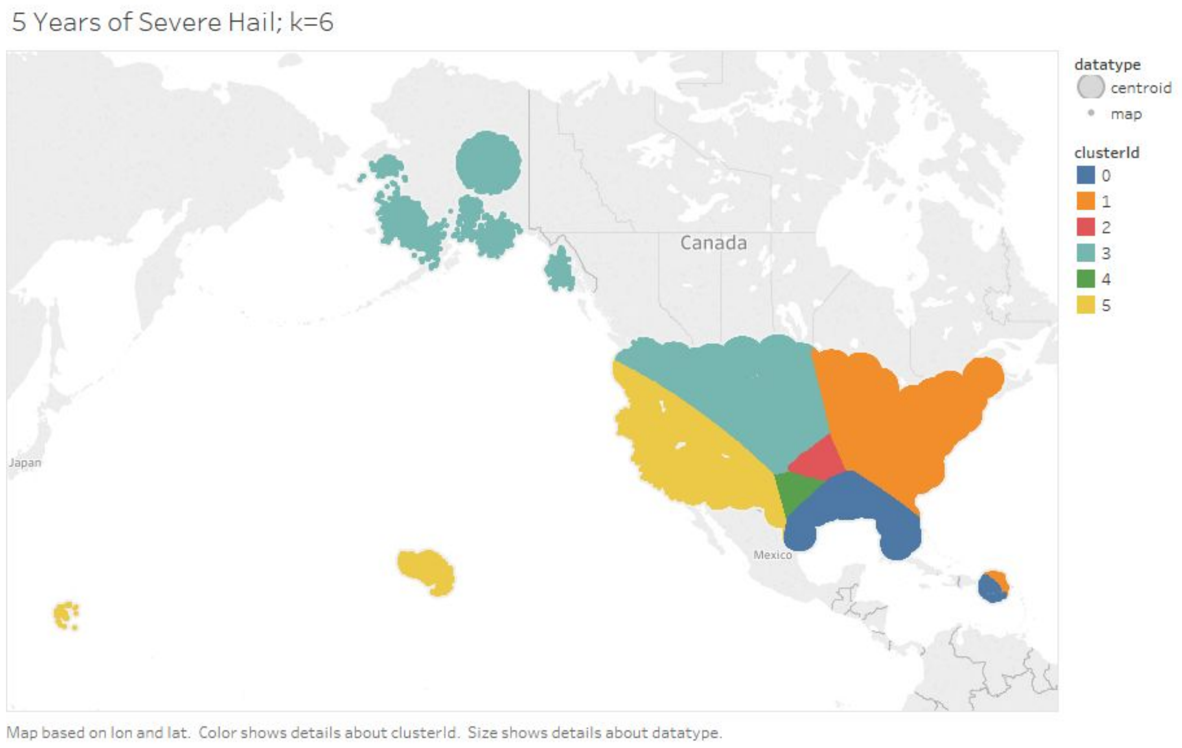
2016 Full Dataset; k=6

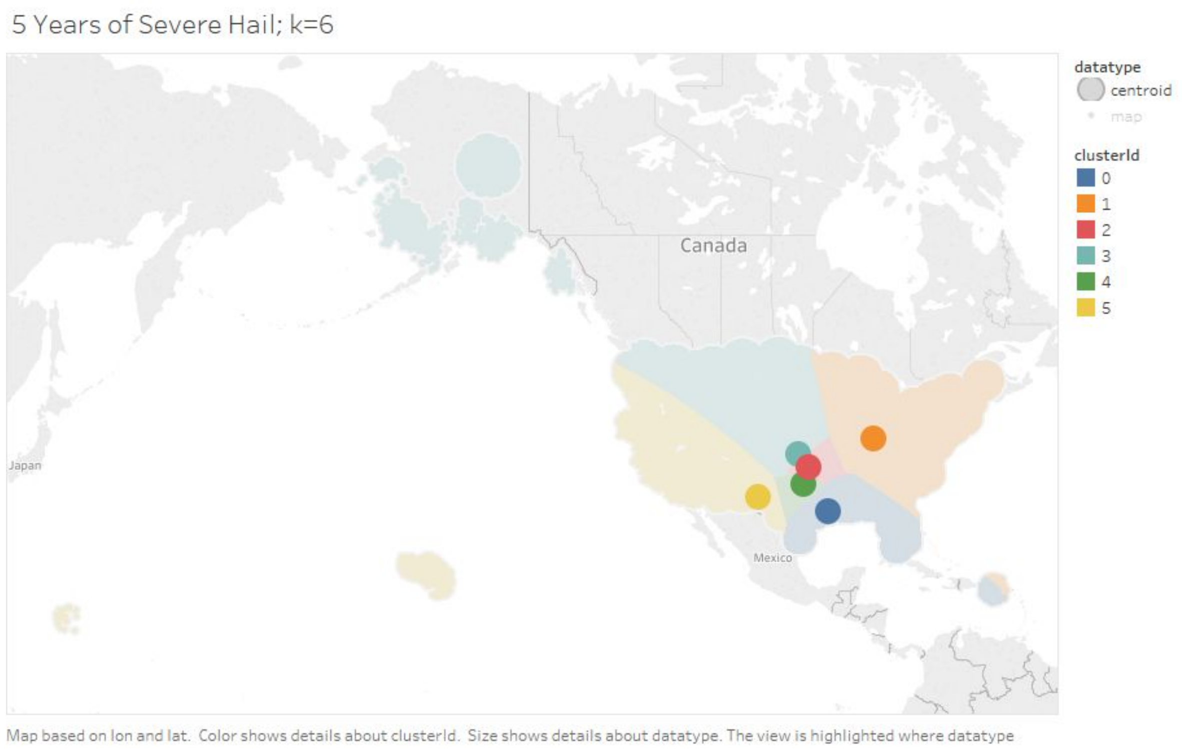


Map based on lon and lat. Color shows details about clusterId. Size shows details about datatype. The view is highlighted where datatype

The main analysis we ran was to take 5 years of data with SEVPROB > 0 and cluster it into 6 clusters as that appeared to be a good starting place based on our testing. The dataset began with approximately 50 million records and once filtered for SEVPROB > 0 it reduced the set down to approximately 7 million. The visualization of the clustering analysis follows below.

First, we wanted to make one small comment on the post processing of the data and visualizations. Similar to when we needed to use the API for uploading the files, when retrieving the output for visualization it became apparent that downloading via the GUI was not efficient. We decided to use the AWSCLI to download and it made post-processing of the data immensely easier. We had to do the visualizations locally, as the Tableau connector to EMR requires Impala or Hive to also be set up on the cluster.


5 Years of Severe Hail; k=6
datatype
centroid
map
clusterId
0
1
2
3
4
5
Canada
Japan
Mexico
Map based on lon and lat. Color shows details about clusterId. Size shows details about datatype.


5 Years of Severe Hail; k=6
datatype
centroid
map
clusterId
0
1
2
3
4
5
Canada
Japan
Mexico
Map based on lon and lat. Color shows details about clusterId. Size shows details about datatype. The view is highlighted where datatype

Our result set was interesting as it identified some clear areas of interest. Namely, tornado alley seems to be a clear cluster as does the northern and southern parts of Texas. Additionally, those clusters are notably smaller than the others. You can also see the pull of the region in the location of the other centroids.

Comparing the results of the full 2016 dataset to that of only the severe hail we can make some interesting remarks. First, in the 2016 you can see the clusters tornado alley and the southern parts of Texas forming even with the non-severe data included. You can also see that the clusters in the 2016 data are further apart than those in the severe data. This would imply that not only is there a higher density of hail recordings in the tornado alley/northern Texas region, but also that is it more severe than the rest of the country. This seems logical as that region is anecdotally more prone to hail storms, especially severe hail.

Runtime Analysis

Local Runtime

*NOTE: ALL Cases run with k =5 and 2 threads, so Tasks = 2 * Stages*
It seems that given x= number of jobs, and y = number of stages: y = (5/3)(x-5) +7*

Table 1. Local Runtime Analysis with RDD Persistence

With Persistence								
	Loudacre_Euclidean				Loudacre_GCD			
Runs	Run Time (minutes)	Jobs	Stages	Tasks	Run Time (minutes)	Jobs	Stages	Tasks
1	0.85	5	7	14	2.5	29	47	94
2	0.86667	5	7	14	3.7	44	72	144
3	0.81667	5	7	14	4.6	59	97	194
4	0.83333	5	7	14	3.4	44	72	144
5	0.81667	5	7	14	4	50	82	164
Avg	0.83667	5	7	14	3.64	45.2	74	148
	Synthetic_Euclidean				Synthetic_GCD			
Runs	Run Time (minutes)	Jobs	Stages	Tasks	Run Time (minutes)	Jobs	Stages	Tasks
1	0.51667	5	7	14	0.75	44	72	144
2	0.53333	5	7	14	0.7	32	52	104
3	0.56667	8	12	24	0.8	53	87	174
4	0.53333	5	7	14	0.83333	56	92	184
5	0.55	5	7	14	1	92	152	304
Avg	0.54	5.6	8	16	0.81667	55.4	91	182
	Dbpedia_Euclidean				Dbpedia_GCD			
Runs	Run Time (minutes)	Jobs	Stages	Tasks	Run Time (minutes)	Jobs	Stages	Tasks
1	1.9	20	32	64	4.4	59	97	194
2	1.1	8	12	24	6.6	92	152	304
3	1.6	14	22	44	6.7	95	157	314
4	1.6	14	22	44	4.8	65	107	214
5	1.1	8	12	24	6	71	117	234
Avg	1.46	12.8	20	40	5.7	76.4	126	252

Table 2. Local Runtime Analysis without RDD Persistence

Without Persistence								
	Loudacre_Euclidean				Loudacre_GCD			
Runs	Run Time (minutes)	Jobs	Stages	Tasks	Run Time (minutes)	Jobs	Stages	Tasks
1	0.93333	5	7	14	4.2	56	92	184
2	0.83333	5	7	14	5.4	80	132	264
3	0.86667	5	7	14	3.2	44	72	144
4	0.85	5	7	14	2.7	32	52	104
5	0.96667	5	7	14	3.9	53	87	174
Avg	0.89	5	7	14	3.88	53	87	174
	Synthetic_Euclidean				Synthetic_GCD			
Runs	Run Time (minutes)	Jobs	Stages	Tasks	Run Time (minutes)	Jobs	Stages	Tasks
1	0.56667	5	7	14	1.1	125	207	414
2	0.53333	8	12	24	1	110	182	364
3	0.51667	5	7	14	0.91667	80	132	264
4	0.61667	8	12	24	0.98333	104	172	344
5	0.53333	8	12	24	0.8	53	87	174
Avg	0.55333	6.8	10	20	0.96	94.4	156	312
	Dbpedia_Euclidean				Dbpedia_GCD			
Runs	Run Time (minutes)	Jobs	Stages	Tasks	Run Time (minutes)	Jobs	Stages	Tasks
1	2.2	23	37	74	5.2	71	117	234
2	1.3	11	17	34	5.6	77	127	254
3	1.6	14	22	44	6.3	89	147	294
4	1.5	14	22	44	7.8	113	187	374
5	1.3	11	17	34	8.6	125	207	414
Avg	1.58	14.6	23	46	6.7	95	157	314

The longer run time of GCD implementation compared to Euclidean implementation is expected since GCD implementation requires more operations than does the Euclidean implementation. It seems that running the algorithm without persisting increases the runtime on average regardless of the way the distances are being calculated. It is also noteworthy that ranges of the run times with GCD implementation tends to be wider. Also, it seems that as the the size of datasets get larger, the range of runtimes seem to get wider (Synthetic being smallest, then loudacre, and then DBpedia). Perhaps the more important observation is that running the algorithm without persistence, the range of runtimes is wider than running with persistence regardless of the size of the datasets.

Cloud Runtime

Surprisingly, we found that even on the larger dataset, processing was quite quick running in AWS. Running the filtered sample (approximately 20,000 records) across 15 k values finished in just over a minute. Running a k-means clustering on the full set of data for one year only took 5 to 6 minutes on average. The result set was approximately 6.8 million records, filtered down from 10.6 million due to invalid data.

The analysis above took approximately 9 minutes to run with the Euclidean distance measure.

The comparable runs using the great circle distance took approximately three to four times as long on average, in line with our expectations given the timing seen during the local testing.

Conclusions

Lessons Learned

- Divide and Conquer

Using the suggested roles and responsibilities, we divided the work into logical units. The Problems provided a guide that was really useful, but additionally we divided the work into pieces that each of us could tackle at one time with defined responsibilities. For example, it was imperative that the local developer handle the execution for evaluation of runtimes (same compute host, etc.) but the results were shared in an online drive for visualization and evaluation by all members of the group. From a project management perspective, GitHub issue tracking, pull requests and code review helped facilitate the collaborative effort and efficient communication required for this project.

- Scaling up from a local implementation to the cloud

Given the size of the data we were working with, there was almost no way we would have been able to perform the analysis on a full dataset locally. The procedure of test locally -> test in pseudo-distributed -> test on cluster -> run on cluster definitely helped us. The two key elements in the procedure were testing locally and testing on the cluster. The former allowed us to jump right into specifying a k value on the large dataset. This eliminated the potentially very costly, both time and financially, process of determining k by running over a range of possibilities. The latter helped to ensure that we had the correct setup on the cluster before running the full job. As we went through a few iterations to get the job configuration, code location and output location correct, it would have again been very costly to run the full dataset only to find out we had an incorrect output location.

- Cloud costs can add up fast

In only one evening of processing in AWS we managed to exhaust our free tier resources. Commonly, the cloud is perceived as cheap and compared to locally hosting servers, it definitely is. However, when working with Big Data, between the large dataset sizes and the compute scaling necessary to perform tasks in an efficient manner, it cannot be ignored how quickly those costs can add up.

Future Work

In our implementation, in order to seed the initial centroid points we resorted to a random sample of the data. While on smaller datasets (e.g. loudacre, sample_geo) this did not cause any issues and the centroids consistently converged, on the larger hail dataset we did notice some subjectivity to initial conditions when running with the Euclidean distance measure. This is the result of two things likely. First, the convergence requirement of .1 is actually equivalent to nearly 700 kilometers. This would more quickly converge and not allow the centroids adjust completely to the dataset. Secondly, the hail data often had subclusters with extremely high densities, for example the areas surrounding tornado alley. If the random point was chosen on one side or the other, given the previous comment, it would be difficult to move the centroid out of the gravity of that subcluster. In a future extension, we would like to rerun with a lower convergence criteria to as to promote stability and accuracy.

There are also definitely a few efficiencies to be gained in the code itself. As an example, storing the cartesian coordinates and radian equivalents upfront would eliminate the overhead of calculating those every time we evaluated a distance metric. This could improve our runtimes and allow us to perform some of the larger data set analysis more efficiently.

Sadly, we did not have enough time (nor financial resources) to explore additional extensions to the hail dataset analysis focus on how to improve the usefulness of the output. The dataset also contains two probability statistics (Probability of hail and Probability of Severe hail) and a maximum size field. These fields can be used to glean more information about the areas in the cluster. For example, rather than computing the centroid assuming equal weights, we could weight by either of the probability measures. This could elicit additional information about the areas that see the most severe hail. Likewise, we could explore the use of a “severity score” calculated as the max hail size * probability. This score could serve as a proxy to the expected loss for an insurance company.

Also, as mentioned above, the hail data almost has a sub-clustering nature to it. Even within the larger clusters, there are definitely areas where there is a considerably higher density of points than others. Running a secondary k-means clustering on the already clustered data, especially with one of the other enhancements, could prove fruitful in identifying key sub-clusters.

Lastly, we would have hoped to run a time series analysis of the clustering. We would be interested in looking at whether or not the clusters, both location and size, are consistent or varying across time. Movements in the clusters or increases in the probability of severe hail in areas could provide valuable information to changing climate patterns.