Geo-location Clustering for Fund Raising

Haoran Li, Zhichao Li, Hai Le

February 27, 2018

Abstract–: Traditionally, non-profit institution ran fund raising campaign by sending mails to potential domestic donors. Donors gave gifts, i.e., cash and check to the institution. Due to the low efficiency of running such a fund raising campaign by the institution itself, nowadays, another third-party fund raising agency will be employed. The mail creation and dispatching jobs are delegated to the fund raising agency. Meanwhile, the agency will study the behavior of the donors and update their campaign strategy. In this project, we get the REAL data of a gift list from a fund raising agency. Base on the raw data, we first leverage Spark to transform the donor address to coordinates data via Google Geocode system. By transforming the coordinates to the GEOID via US Census Bureau, we also join the gift data with the demographic data in FFIEC. Then, based-on K-mean algorithm, we cluster the donors by their distance. Meanwhile, statistics are performed for the gift data and demographic data. The data are processed on AWS EMR cluster using Spark.

1 Introduction

Traditionally, a non-profit institute sends mails to their potential donors to ask them for *Gifts*. This procedure is usually to be considered as low efficient as donors seldom reply for a fund raising mail or send the Gift.

In order to run a fund raising event more efficiently, a non-profit institutes tends to ask a third-party agency for helping them running a fund raising event, which usually referred as a *Campaign*. In other words, a third party agency will help this non-profit institution for creating and mailing fund raising mails to potential donors.

The non-profit institute (also referred as *Client*) will be benefit from a third-party campaigns service by:

1. The institution will initial a fund raising campaign by ask the *Fund Raising agency*. Usually, a raw *mail list* will provided to the fund raising company. The donors in such a mail list is considered as historical donors. The fund raising company will process this raw mail list and yet create an advanced version, say, exploring more potential donors by using supplemental strategy (a trivial strategy is just following the raw mail list).

2. A Fund Raising agency usually serves for several non-profit institutions. They will have a lot of Printing-Email transactions with different envelop providers. That means, a fund raising agency could get a relatively low price per envelop when they quote the envelop providers. As a result, it helps the institution to cut down the cost.

In this project, **we get an updated gift list file from a fund raising agency**. We are going to help the fund raising agency to clustering the donors based on different criteria. A basic version is based on their geo-location via a k-means algorithm implemented on Spark. Joining the data in FFIEC, we are still able to find the relationship between gift revenue and tract income level.

2 Background

2.1 Fund Raising

Here is a simplified procedure of how the institution, fund raising agency and donors interact with each other in a campaign, as shown in Fig. **??**.

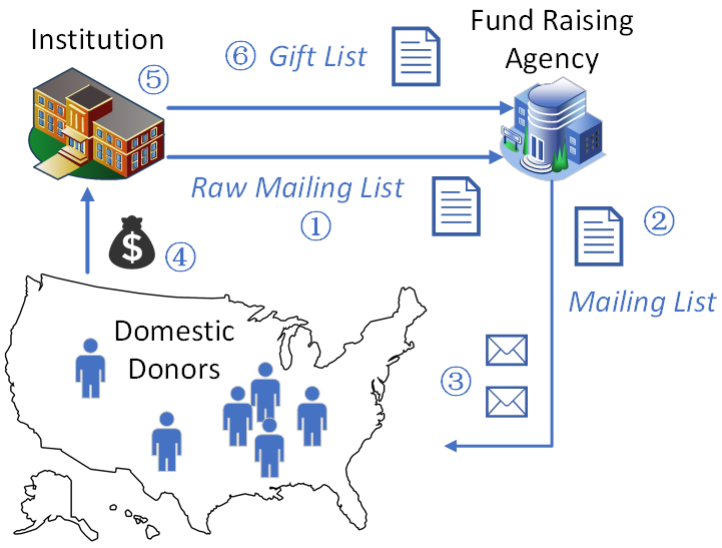

Figure 1: Fund Raising Conceptual Diagram

- 1. The institution creates the raw mailing list

  An institution will request the fund raising agency to a running campaign. In order to do this, the institution will provide some *raw mailing lists*. Usually, these raw mailing lists are created based the donors historical information in institutions own database. The fund raising agency will treat these raw mailing lists as raw input data.

- 2. The fund raising agency creates a campaign mailing list based on raw mailing list.

  Usually, the Raw Mailing List is not a clean dataset. That means, there are some address errors, typos, mis-alignment and missing fragments. The Fund Raising Agency will clean that data. A typical solution is to ask a Data Vendor to do so. Then, the Fund Raising Agency will get a clean Campaign Mailing List.

- 3. The fund raising agency prints and send the mails to domestic donors.

  Fund Raising Agency will create the mails. Specifically, mails in different packages are with different envelops and different contents and styles. Besides, a fund raising mail will also contain a Business Reply Mail envelop, which are prepaid, in that mail for donor to mail their gifts. (You can find the similar thing in your mailbox when you are invited to apply for a new credit card or to attend a survey.) The important thing is, the *Full Code* will appear at the envelop of the business reply mail. That enables the fund raising agency to evaluate the *return of interest (ROI)* and do further data analysis in the future.

- 4. Donors Receive Fund Raising Mail and Send Gift

  Typically, a donor who donates a gift will send their cash or check in the Business Reply Mail envelop, which is contained in fund raising mail. The gift mail will be sent to the *institution*.

- 5. The institution receives gifts and update Gift List

  After receiving the gift mail from donors, the institution will update the gift information into their own database system. Again, since the code can be find on the gift envelop, it enables fund raising agency to evaluate their campaign performance by querying base on the gift code.

- 6. The fund raising agency receives Update

  Unfortunately, the fund raising agency cannot get the *Gift List* at real-time. It cannot directly access its clients internal database. Instead, the Fund Raising Agency will ask the institution to update Gift

List in a CSV Sheet separately a few month after the last Campaign. A database manager of the institution will dump the gift list from their database. Note that, although unnecessary, the database manager usually dumped all the historical gift records in their entire database to a single file (Well, it sounds insane but that's the true story in real world).

2.2 Demographic Census

GeoID. GEOIDs are numeric codes that uniquely identify all administrative/legal and statistical geographic areas for which the Census Bureau tabulates data [**?**]. Data users rely on GEOIDs to join the appropriate demographic data from censuses and surveys, interpretation and mapping. With the GEOIDs, data users can relatively easily pair the appropriate demographic data with the appropriate geographic data.

Converting Coordinates to GeoID. Converting the universal geographic data (i.e., GeoCode, or coordinates) to a US government geographic data (i.e., GEOID) is very important. Luckily, the United States Census Bureau has assigned GEOIDs, to geographic entities to facilitate the organization, presentation, and exchange of geographic and statistical data. The US Census Bureau provide an on-line Geocoder tool [**?**]. It is possible for public users to leverage this Geocoder tool to convert a GeoCode (i.e., coordinates) into **GEOID**, as Fig. **??** and Fig. **??** shown.

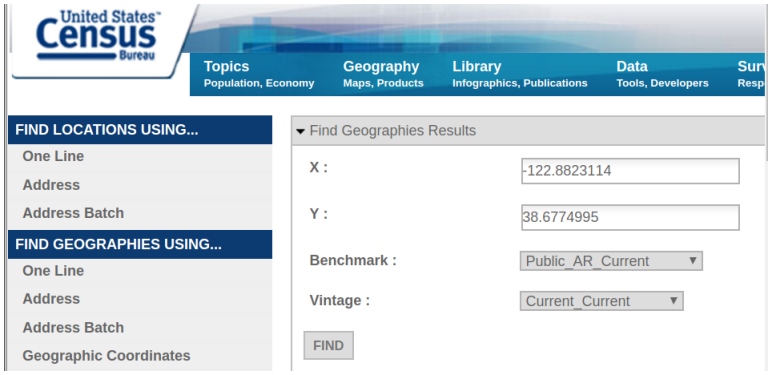

Figure 2: US Census Bureau Geocoder

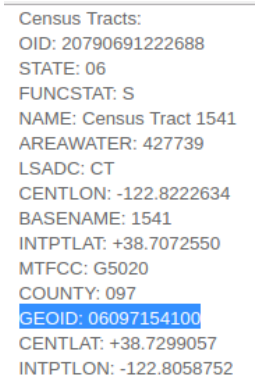

Figure 3: Converted GEOID

Demographic Data. The **Federal Financial Institutions Examination Council (FFIEC)** is a formal inter-agency body empowered to prescribe uniform principles, standards, and report forms for the federal examination of financial institutions [**?**].

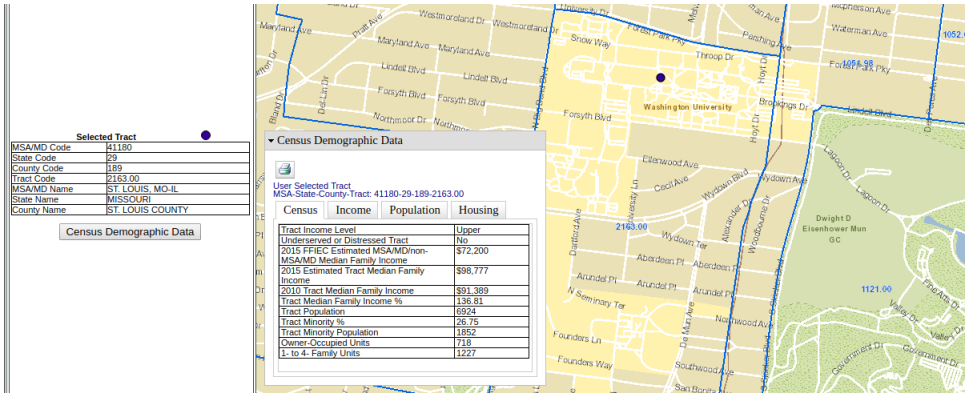

Figure 4: FFIEC Web UI

As shown in Fig. **??**, when specified a GeoID (comprised by StateID, CountyID and TractID) to FFIEC Geocoding system [**?**], we can get the demographic data, such as population, income level and tract median

income.

3 System Architecture

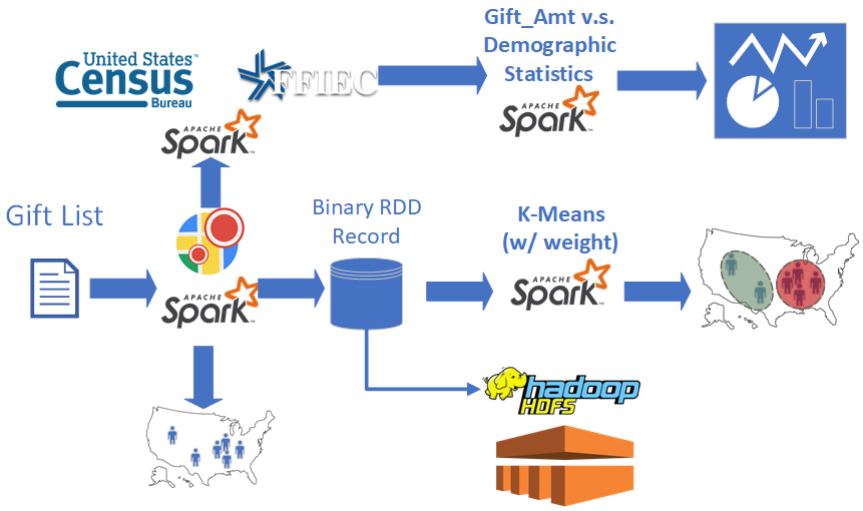

Figure 5: System Architecture

The system is comprised of four major part as shown in Fig. **??**.
Preprocessing: Coordinates The bottom-left showed the data preprocessing for coordinates. Based on the Gift List, we get the coordinates data for geo-clustering via Google Geocode API.
Preprocessing: Demographic Data The top-left showed the data preprocessing for getting demographic data. Based on the coordinates, we get the **GEOID** data via US Census Bureau. Then, based on the GEOID, we can get the demographic data via FFIEC.
Local Processing The top-right and top-middle showed the local processing for data. We used K-mean to clustering the donors. And we use the demographic data to show some statistics.
Cluster Processing The bottom-right showed the cluster based testbed for our system. We processed our data on a EMR cluster.

4 System Implementation

4.1 Data Preprocessing

Getting Coordinates for Geo-Clustering.

In order to using k-means clustering algorithm to processing the coordinates, the key point is to transform the address of donor in *Gift List* to coordinates.

The *gift list* we get from the fund raising company is a CSV file dumped from an institution's database: We are going to use a subset of the gift record from 1985 - 2005. This file include **54458** records. Each record has multiple fields such as *DonorID, Name, Gift Amount, Date, Street, City and State.*, as shown in Fig. **??**.

The first transformation is truncate the data: We don't want to disclose the donors' name, and some columns are unnecessary for our analysis. We just discard those column. The *Gift Amount* in original file is a string formatted like "$10.00". We will process the string and transform it to an integer, where the unit is cent. Further more, the separated addresses were concatenated to a full address. The processed record contains four columns which are *DonorID, Gift Amount, Date, and Full address*, as shown in Fig. **??**.

Thanks to google Map, we are able to leverage the GoogleGeocode system to transform the full address to a (Latitude, Longitude) pair. This procedure is referred as **Geocoding**.

An interesting part is, some donor won't provide their addresses, or provide invalid address which cannot be processed by Google. Hence, exception should be handled when using the geocoding as a map function: **When Geocoding fails, it will return (0, 0) as the (Lat, Lng) pair**.

Filtering the invalid record, at last, we can get **53424** records as our enhanced gift list, which is stored in a binary **pickleFile** [**?**] format.

Remember the fact that one donor may submit multiple gifts from the same address, that means we should distinct the address / coordinate before trying to visualize the temporary result. After doing that, we can find **7217** distinct coordinates.

Using those processed coordinates, we can visualize the donor's location by using Map tool [**?**].

DonorID	Name	Amt	Date	Street	City	State	Others
001	John	$50.00	1/2/1999	12 Foo Ave	Bar	MO	...

.map() / transformation:

DonorID	Amt (Cent)	Date	Full Addr
001	5000	1/2/1999	12 Foo Ave, Bar, MO

.map()
Using GoogleGeocode API

DonorID	Amt (Cent)	Date	Full Addr	Latitude	Longitude
001	5000	1/2/1999	12 Foo Ave, Bar, MO	30.0001	-48.98

Figure 6: Data Preprocessing: Getting Coordinates

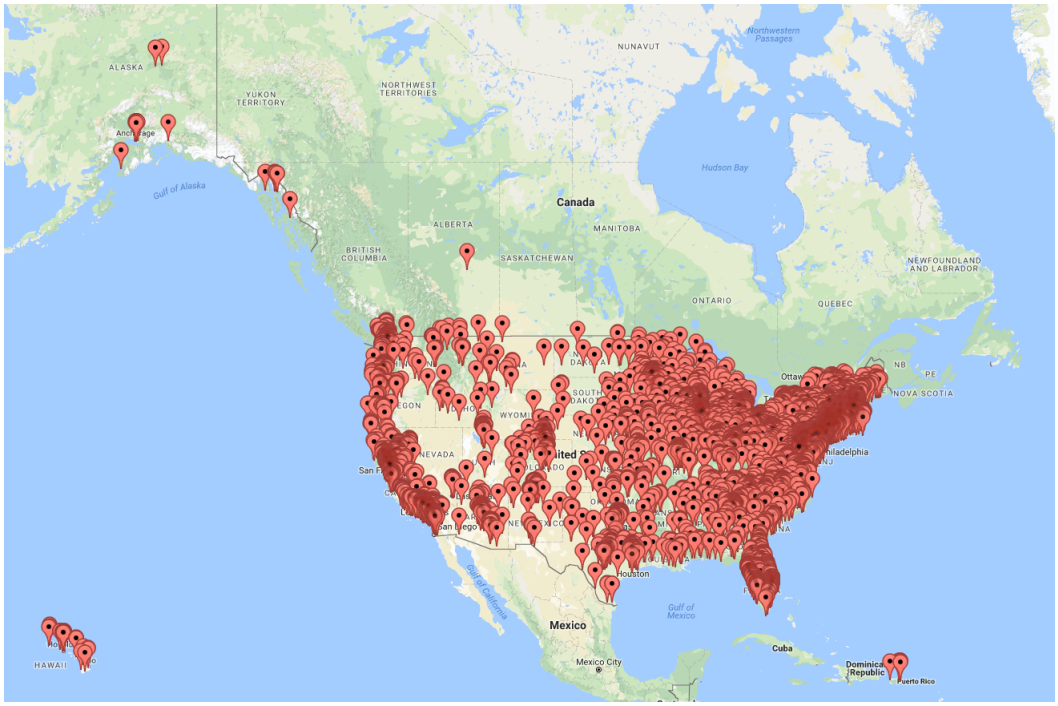

Figure 7: Visualization: Donors location

Getting Demographic Data for Statistics.

Given the fact that most donors are domestic donors, we will use the census data from FFIEC to find the potential relationship between donors' gift amount and their tract demographic information. We need to further exploit the coordinates information we get in previous stage to join more useful information:

- GEOID : String
- Tract Population : Integer
- Tract Below Poverty % : Float
- Tract Income Level: {Upper, Middle, Moderate, Low}

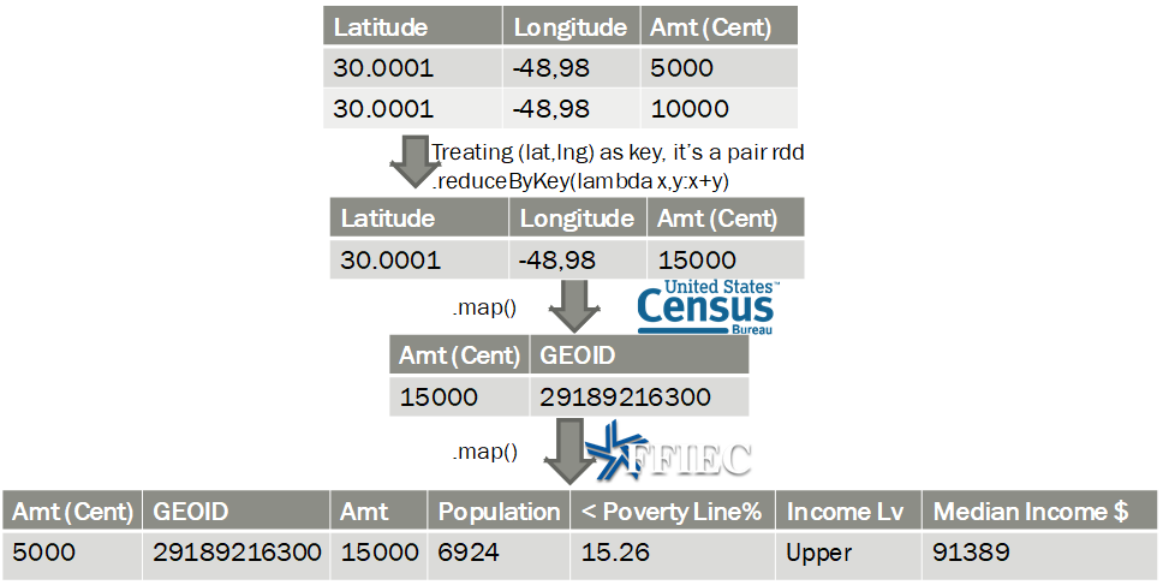

Figure 8: Data Preprocessing: Getting Census Data

As Fig. **??** shown, we first aggregate the gift amount by the same coordinates. Then, we fetch the GEOID by fabricating HTML GET request to US Census Geocoder. The map function will submit the coordinates, parse the result and get GEOID by using regular expression. Finally, we can submit the GEOID to FFIEC and get the related demographic data.

4.2 *k*-means Algorithm on Spark

Clustering approach

First, read the coordinates from input files and save them into a RDD, and persist the RDD.

Implementation: use a simple lambda function to transform the coordinates into the form of (latitude, longitude) and call persist().

Then select k points that are likely to be in different clusters. First, pick a random point; while there are fewer than k points, add the point whose minimum distance from the selected points is as large as possible. When the while loop ends, the initial k points are successfully selected.

Implementation: initKCenters is the function that returns the list of coordinates of the initial centers; it uses the **minimumDistance** function to calculate the minimum distance from the selected points.

Next, based on the k centroids, add each point in one cluster whose centroid is closest to the point. After all points are labeled with cluster index, recalculate the k centroids based on the new clusters. Calculate the sum of distance difference between each pair of old and new centroid. Iterate until the sum of distance difference between each pair of old and new centroid is below 0.1.

Implementation: In the iteration loop, for each iteration, initialize the **convergeDist** variable to 0; transform the initial RDD of coordinates, **latlongs**, into a RDD of coordinates in form of (x, y, z) with cluster index using **closestPoint** function and **toXYZ** function, and persist the new RDD, **latlongs_clustered**; use **reduceByKey** to calculate the sum of all points' coordinates and produce a new RDD, **sum_clustered**; use **countByKey** to calculate the number of points in each cluster and print it out; transform the **sum_clustered** into the form of (latitude, longitude) to obtain the new k centroids coordinates (here the **atan2()** function in the **toLatLong** function makes dividing the sum of coordinates to calculate mean coordinates unnecessary); calculate the sum of distance differences between the old and new centroids and update the **kCentroids**. Iterate until **convergeDist** < 0.1.

Save the clustered coordinates into a text file and save the k centroids into another csv file

Implementation: Transform the coordinates in the latest **latlongs_clustered** RDD into form of (latitude, longitude); sort the RDD by key to collect points of same clusters; transform the coordinates into CSV format; save the RDD into text file; add a header into a newly created csv file for containing the k centroids' coordinates; save the k centroids' coordinates into that file.

Once the computation is done and output files are ready, combine all the points' coordinates from the output files with the k centroids' coordinates into a single csv file, which is imported in Microsoft Excel for map visualization.

Implementation: Run "python aggregateClusters.py" to extract coordinates with cluster labels from non-empty output files in "output_csv" directory, and append them to the csv file containing the k centroids' coordinates. Then use Microsoft Excel to import the csv file and plot the locations on a flat map.

4.3 Cloud Execution

Cluster Setup

In order to execute our k-means clustering program on cloud, we create a cluster on **Amazon EMR** service with the following configuration:

- Software Configuration: Release: **emr-5.10.0**; Application: **Spark 2.2.0** on **Hadoop 2.7.3 YARN** with **Ganglia 3.7.2** and **Zeppelin 0.7.3**.
- Hardware Configuration: Instance type: **m4.large** with 3 instances (**1 master** and **2 core nodes**), each has **4 vCPU**, **8 GB memory**, and **32 GB EBS Storage**.

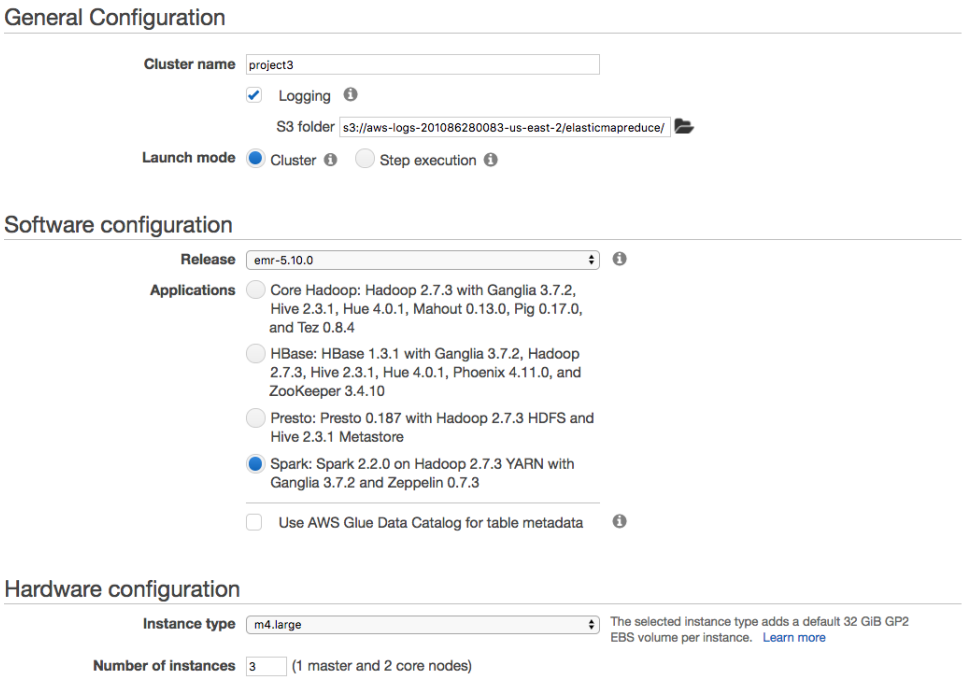

Figure 9: Configuring the Cluster

Data Storage

To store our data as well as our Spark program, we create a new bucket on **Amazon S3** storage and upload all necessary files on it. We also modify our Spark program so that it will read and write data to the bucket on S3 that we created.

Program Execution

Now we have everything set up. To execute our Spark k-means program, we *connect* to the Amazon EMR **master node** using **SSH** (Note that the *security configuration* of the master node is set so that it will only accept **SSH** from our specify IP address). Once accessed the master node, we can run our program with the *familiar* command:

```
$ spark-submit --master yarn-cluster kmeans.py distance_metric num_centers fileURL
```

where distance_metric is either **Euclidean** or **GreatCircle**, num_centers is the number of **centroids**, and fileURL is the **directory** of the data folder on Amazon S3.

5 Evaluation

5.1 k-means Algorithm on Spark: Compute and Visualize Clusters

Two datasets are used for computation: one includes locations from all US territories and UK **(Global)**; the other only includes the locations on the US mainland **(Mainland)**.

For the **Global** dataset, cases using $k = 10$ with **Euclidean** and **GreatCircle** distance are computed; for the **Mainland** dataset, cases using $k = 6$,10 with **Euclidean** and **GreatCircle** distance are computed.

The following figures show the effects of k values and datasets. When using **Global** dataset and k of 10 (Fig. 10), there are 4 clusters out of US mainland, which are Alaska, Hawaii, Puerto Rico and UK and 6 clusters in the US mainland. This observation indicates that the program handles correctly when there are clusters that are much distant from each other. When using Mainland dataset and k of 6 (Fig. 12), there are 6 clusters successfully produced in the US mainland, which are geographically and approximately

referring to Pacific, Mountain, West South Central, Great Lakes, South Atlantic, Northeast, respectively [**?**]. This clustering is practically reasonable in terms of demographical characteristics. Interestingly, the figure of **Global** dataset when $k = 10$ also shows different clustering pattern with 6 clusters in the US mainland. This pattern makes less sense than that of the Mainland dataset probably due to the 4 other clusters' influence in the clustering process. Fig. 11 shows that the US mainland is divided into 10 clusters ($k = 10$), which might be a little excessive because it further splits the Great Lakes region, Mountain region and Pacific region, respectively. Therefore, k of 10 and 6 should be the preferable values for Global and Mainland datasets, respectively.

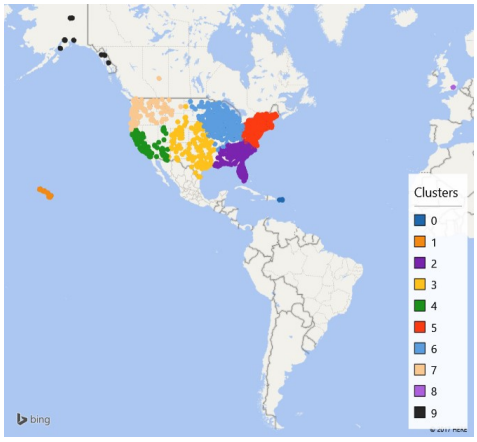

Figure 10: Great circle distance, k=10, Global dataset

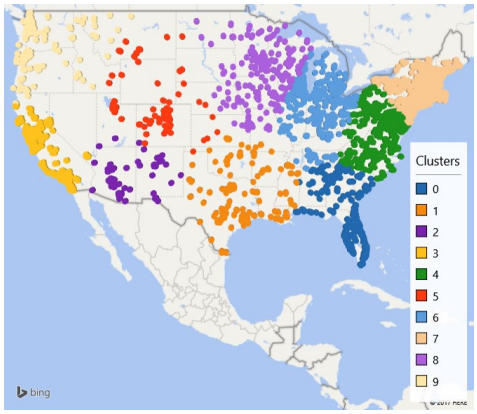

Figure 11: Great circle distance, k=10, Mainland dataset

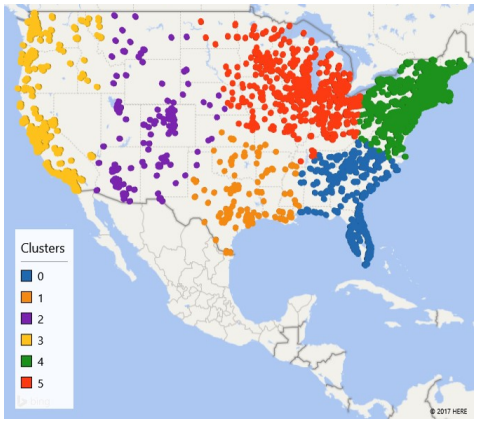

Figure 12: Great circle distance, k=6, Mainland dataset

The following two figures show the comparison between outcomes using **Euclidean** and **GreatCircle** distance. The clusters' shapes and centers' locations are all different for **Euclidean** and **GreatCircle** cases. The clusters using **Euclidean** distance are more skewed than those using **GreatCircle** distance, which is reasonable. Apparently, **GreatCircle** distance is favorable for geo-location clustering based on the test results.

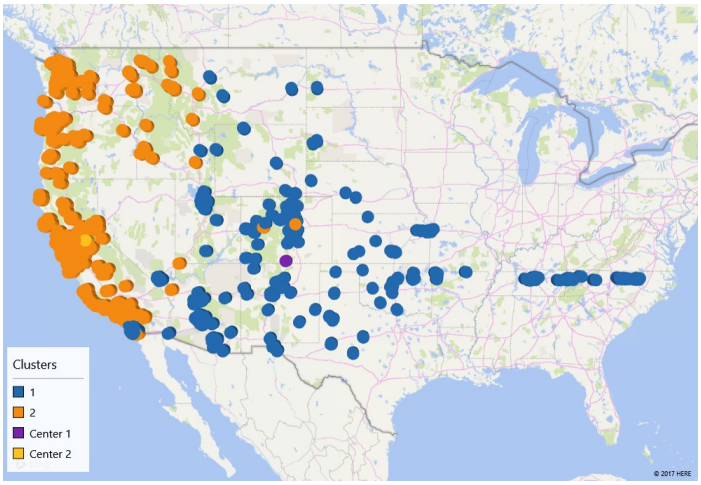

Figure 13: Euclidean distance, k=6, Mainland dataset (two clusters and their centers)

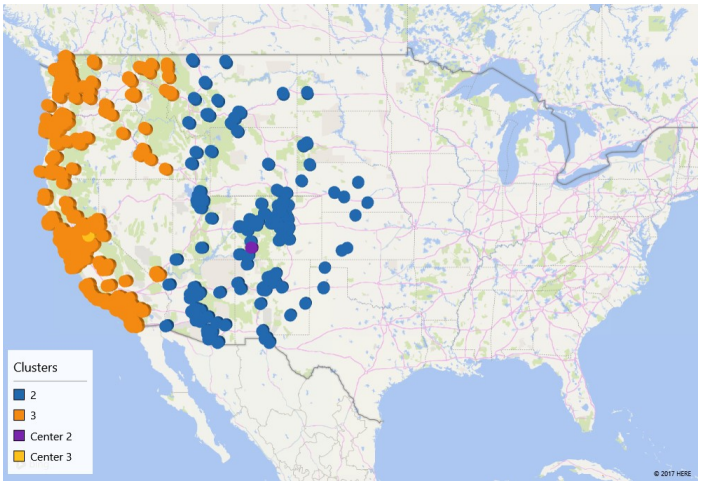

Figure 14: Great circle distance, k=6, Mainland dataset (two clusters and their centers)

5.2 *k*-means Algorithm on Spark: Runtime Analysis

To analyze the runtime of our k-means program, we run the program on the pseudo-cluster as well as on the real EMR cluster with different combinations of input arguments (Mainland data vs. Global, persist vs. non-persist, Euclidean vs. GreatCircle, number of centroids k = 6, 10). To get stable results, for each combination, we repeat three times and average the results. The final results are shown in the Table 1.

Table 1: Average runtime on Pseudo-distributed mode vs. Real Cluster for different input arguments

Persistence	Distance Metric	k	Data Set	Runtime (local) (sec)	Runtime (cloud) (sec)
Yes	Euclidean	10	Global	99	50
Yes	Euclidean	6	Mainland	136	68
Yes	Euclidean	10	Mainland	112	55
Yes	Great Circle	10	Global	91	45
Yes	Great Circle	6	Mainland	79	39
Yes	Great Circle	10	Mainland	90	44
No	Euclidean	10	Global	100	54
No	Euclidean	6	Mainland	151	69
No	Euclidean	10	Mainland	122	65
No	Great Circle	10	Global	96	50
No	Great Circle	6	Mainland	93	48
No	Great Circle	10	Mainland	103	50

As we observed, the implementation that uses **persistent RDDs** run faster (**lower runtime**) than the implementation **without using persistent RDDs**. In addition, we also observe that using Euclidean distance takes more iterations to converge than using Great Circle distance, hence, it has longer runtime.

5.3 Demographic Statistics

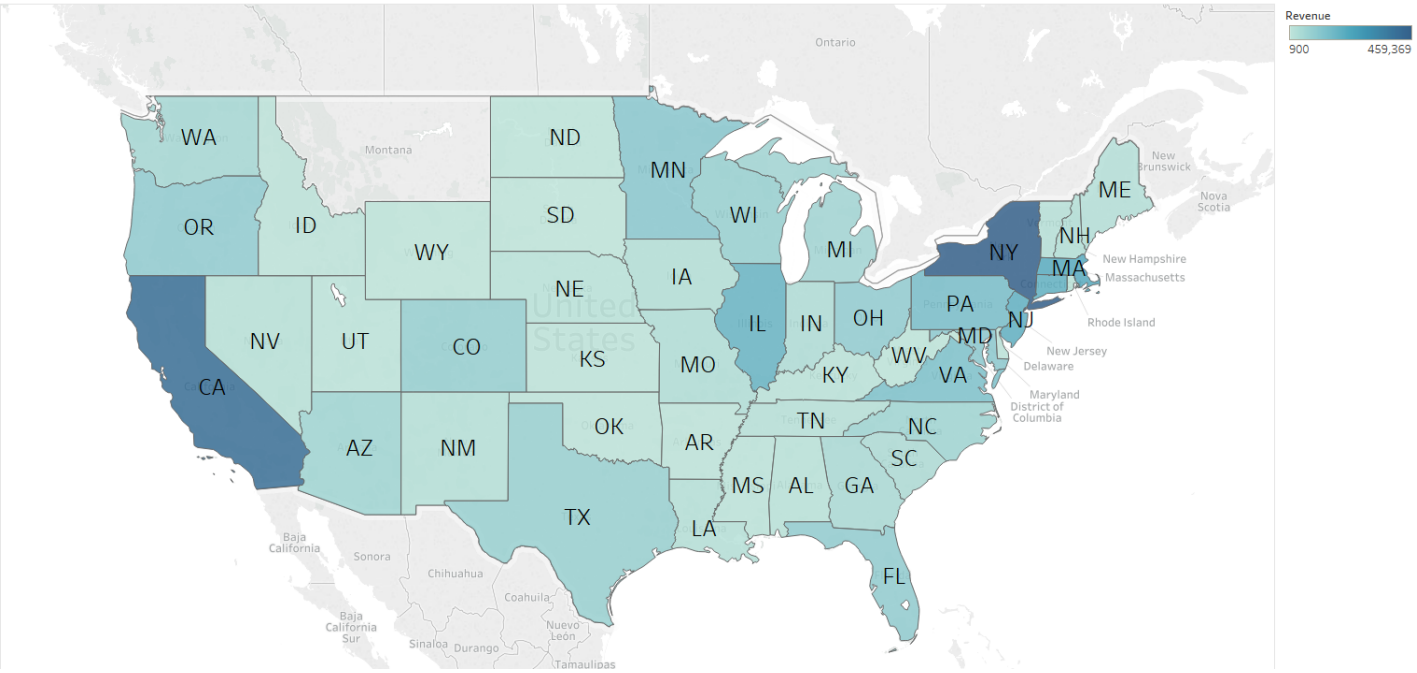

Figure 15: State Total Revenue (AK, HI, and PR excluded)

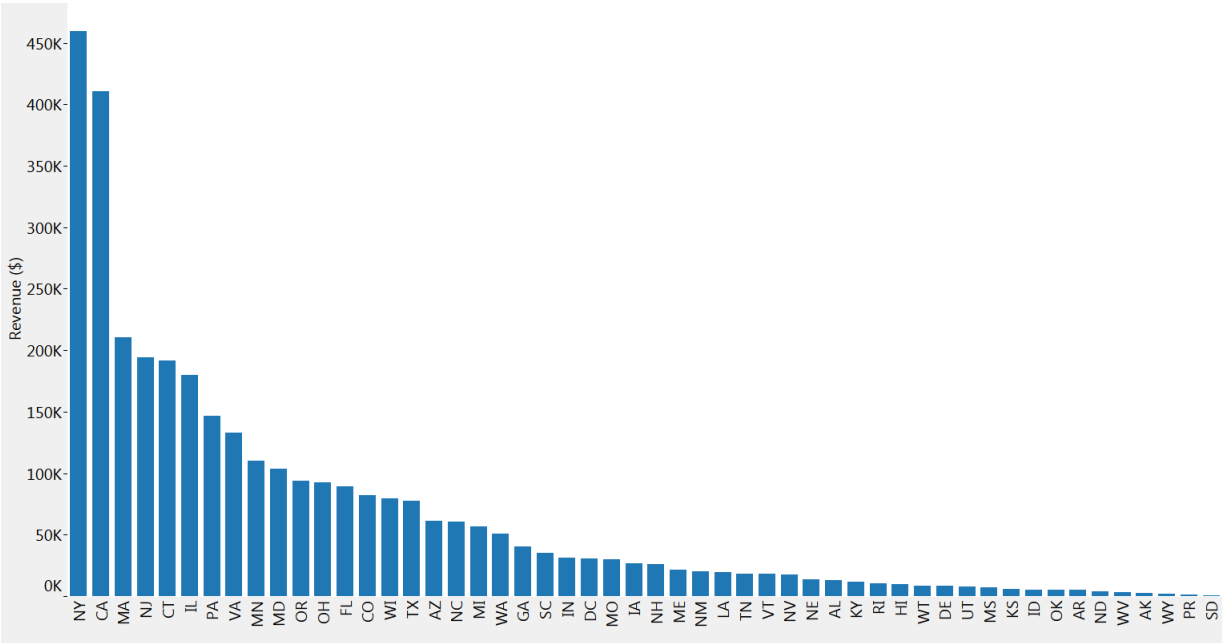

Figure 16: State Revenue, Descending Ordered

According to the GeoID, we can easily extract the stateID and group the gift records by state. As shown in Fig. **??** and Fig. **??**, donors from California and New York contribute the majority of revenue.

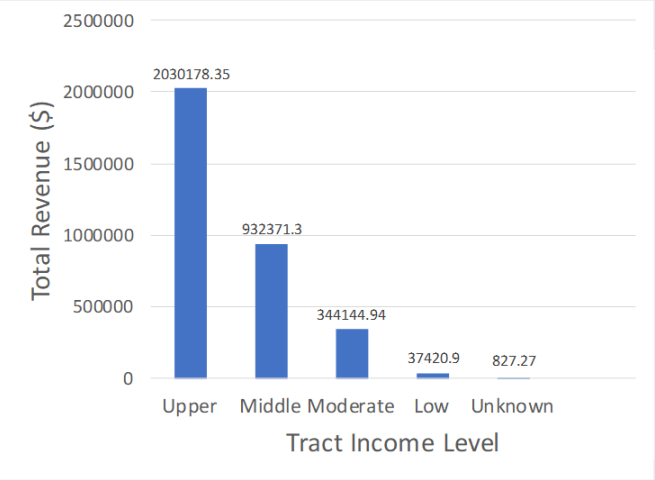

Figure 17: Total Revenue, grouped by Tract Income Level

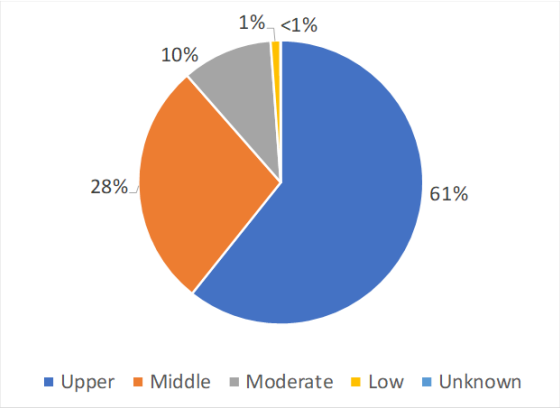

Figure 18: Total Revenue: Proportion

By using the "Tract Income Level" from FFIEC, we grouped and then aggregated the total revenue:

Clearly, the higher income level of a tract is, the institution get more gifts (Fig. **??**). Notice that the "Upper" bracket contributed the most (61%) revenue to this institution (Fig. **??**).

6 Lesson Learned

6.1 Haoran Li

I am in charge of the data preprocessing and some additional statistics. Specifically, I wrote spark programs to: (1) convert the address to GeoCode, (2) get the GeoID through GeoCode via US Census, and (3) get the Census data from GeoID via FFIEC; (4) provide demographic statistics in evaluation.

Converting address to coordinates is not such a painful process: thanks to Google Maps API, by applying an API Key, you can use that service. The runtime bottleneck is HTTP request to Google. Although the Google will rate limit our requests, we can still do 60,000 times geocoding in around 10 hours.

Some addresses in raw data are not correct, we are able to use zip code to do a coarse geocoding. However, there are some donors who refuse to disclose their addresses, even zip code. We treat those data as exceptions and returns coordinate (0,0) when using customized mapping function it in Spark. The Google Map API seems very stable and we successfully get all the data we want by only **ONE** run. But that service is not free (Fig. **??**).

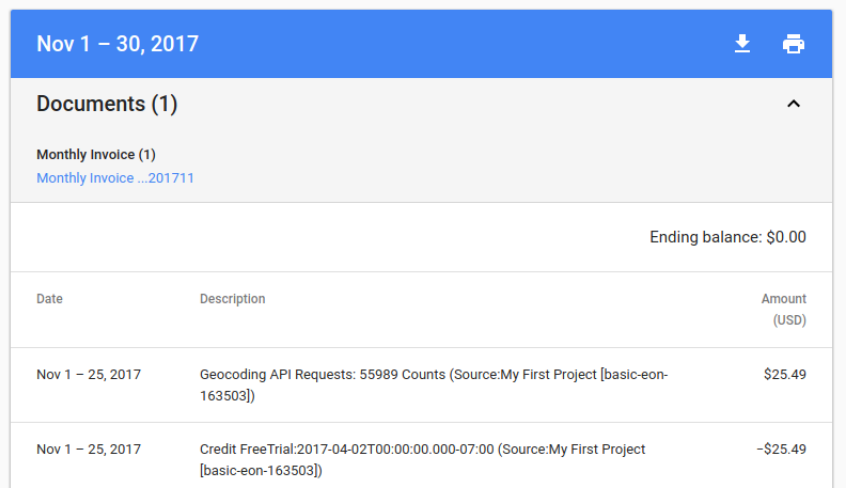

Figure 19: No Free Lunch: You pay for the GeoCoding Service

Scarping data form US Census and FFIEC is not very straightforward. First, they don't open a public API as Google Maps does. I reverse engineered their website code and find a walk-around to fabricate a "fake" form and then get the data by parsing the response. Second, those "service" are not so stable as Google's. I used a hybrid method by combining several exceptional handling functions with "micro batch submission" to finally get around 7,000 records transformed.

6.2 Zhichao Li

I am the major local developer and took charge of implementing the k-means algorithm using pyspark. Local development encompassed pseudo code writeup, API lookup, function implementation, incremental debugging, small input test, output post-processing using python script, map visualization (using 3D map in Microsoft Excel), large dataset test and runtime analysis. I also helped with deployment of the spark program on the AWS. Going through the runtime analysis both locally and on the cloud is quite rewarding in terms of getting to know the efficiency edge of cloud.

Through the process, I've become more familiar with RDD concepts, transformation and actions and learned how to leverage persistence to create more efficient spark program. I managed to write efficient scripts to processing the output data for easy visualization. Furthermore, by summarizing the results in this report, I gained more insights on how to iteratively produce reasonable results by interpreting the outcome and re-run the program with more reasonable parameters. Although k-means algorithm is a fundamental machine learning method, this project definitely empowers me to practice more complicated and valuable methods using cloud computing platform.

6.3 Hai Le

I am taking care of the cloud execution (on Amazon EMR) as well as assisting Zhichao with the Spark k-means implementation. I wrote the helper functions: *closestPoint*, *addPoints*, *EuclideanDistance*, and *GreatCircleDistance*. I created and managed the Amazon AWS Educate Account, uploaded all data and program to the S3 distributed storage, executed the program and collected the results.

So far in this course, every program I wrote was run locally. Therefore, working on a real cluster is a completely new experience for me. I had some trouble in the beginning such as creating the key-pair to SSH to the master node, configuring the security, accessing Zeppelin Notebook, Resource Manager, etc. However, after watching/reading the tutorials as well as doing some Google search, I can finally get everything done fast and correctly. Now I am more comfortable with the Amazon AWS services and ready to apply them for future Big data and cloud execution jobs.

References

[1] List of regions of the united states. https://en.wikipedia.org/wiki/List_of_regions_of_the_United_States.

[2] About the ffeic. https://www.ffiec.gov/about.htm, 2017. Accessed: 2017-12-04.

[3] Ffeic geocoding. https://geomap.ffiec.gov/FFIECGeocMap/GeocodeMap1.aspx, 2017. Accessed: 2017-12-04.

[4] Map tool. https://www.darrinward.com/lat-long/, 2017. Accessed: 2017-12-04.

[5] pyspark package. http://spark.apache.org/docs/2.1.0/api/python/pyspark.html, 2017. Accessed: 2017-12-04.

[6] United state census bureau. https://geocoding.geo.census.gov/, 2017. Accessed: 2017-12-04.

[7] What are geoids. https://www.census.gov/geo/reference/geoidentifiers.html, 2017. Accessed: 2017-12-04.