Data Problem Report

Andre Foote

27th May 2018

Initial Cleaning

**DataSample.csv**

Changed “ TimeSt” to “TimeSt”, getting rid of the space that could cause errors in processing.

**POIList.csv**

Removed POI2 for being identical for POI1 under the assumption that this was a typo. The consequence of this is that records that would have fallen under POI2’s area were absorbed by the other POIs thus skewing the results. In further discussion POI2 will be referred to POIX.

Difficulties and Workarounds

**Use of Docker**

Docker installation and setup according to the provided instructions was straightforward. My understanding at that point was that the DataSample.csv and POIList.csv files in the datamr/data folder would be located and accessible somewhere within the container shell. I was not able to these files however and eventually I decided to continue with the assignment locally.

**Apache Spark Logic**

A lot of time was spent trying to determine the correct approach to solving the problems provided. I understand that the main advantage of Spark is its capacity to perform operations on large sets of distributed data, however I could not figure out a way to retrieve the kind of values I wanted using the appropriate Spark conventions.

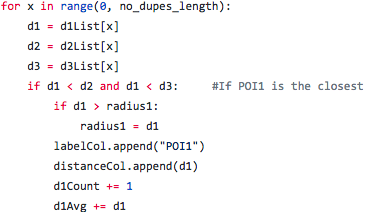
After a point, I decided to simply tackle the problems as best I could. In most instances RDDs and DataFrames were converted into lists on which I understand better how to operate.

Problems

**0. Cleanup:** Remove duplicate records



**1. Label:** assign each *request* to one of the *POI locations* that has minimum distance to the request location.

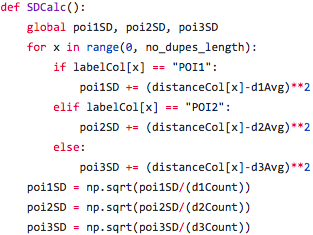


This code block is found in the distanceStats function. d1List, d2List and d3List are lists containing the distance between each request record and corresponding POIs. For example, d1List contains the distance between each record and POI1.

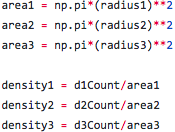
This code block contains 3 main conditional statements (only 1 is shown here) which append either “POI1”, “POI2”, or “POI3” to the list “labelCol” which acts as a record’s label.

This function also keeps track of other variables for additional calculations.

**2. Analysis I:** calculate the average and standard deviation of distance between the *POI location* to each of its assigned *requests*.

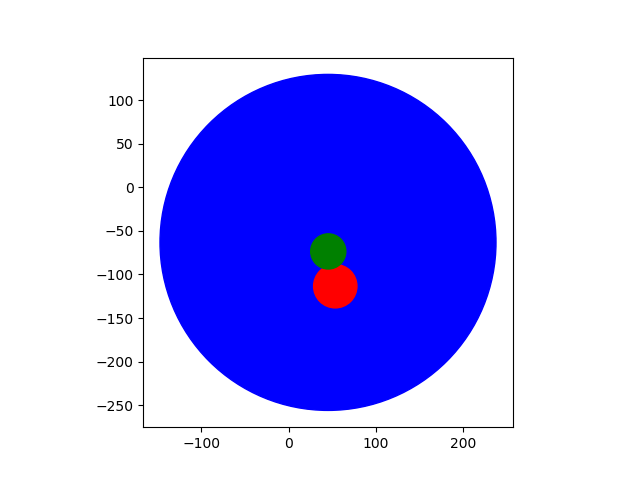


**2. Analysis II:** Draw circles that are centered at each of the POIs, each includes all *requests* assigned to the *POI location*. Find out the radius and density (i.e. request count/circle area) for each *POI location*.



|  |  |  |  |
| --- | --- | --- | --- |
|  | POI1 | POI2 | POI3 |
| Count | 9697 | 9817 | 484 |
| Average Distance | 3.348 | 35.978 | 57.205 |
| Standard Deviation | 3.858 | 30.574 | 56.237 |
| Radius | 24.852 | 20.155 | 192.705 |
| Area | 1940.307 | 1276.238 | 116663.714 |
| Density | 4.998 | 7.692 | 0.00415 |

POI1 = Red POI2 = Green POI3 = Blue



Conclusion

The radii of each POI was assigned as the distance of the farthest record under its label.

I initially suspected that the area of POI3 was so large because it inherited the majority of the deleted POIX. However, POI3 has much lower Count and Density values than the other POIs. This leads me to guess that POI3 represents the entirety of some kind of network while POI1 and POI2 are a subset that handle areas of high traffic.

I understand this was not the efficient way to solve these problems. Spark has a mean() function for RDDs that would probably have been more efficient than my solution. However, I hope that my logic proved sound, the code well-commented formatted, and that I kept the order of the algorithms reasonable.