CSE 512: PHASE 3 REPORT

THE FIRST ORDER

Ankit Nadig:1211213650, anadig@asu.edu
Anshuman Bora, 1211247437, abora3@asu.edu
Saumya Parikh: 1211191654, ssparik1@asu.edu
Vraj Delhivala: 1211213637, vdelhiva@asu.edu

Problem Definition:

Given a collection of NYC Yellow Cab taxi trip records for the month of January in 2015, identify the fifty most significant hot spot cells in time and space using the Getis-ord statistic. Due to the large volume of data, the operations are to be performed using Hadoop File System and Spark, to decrease the data processing time and get relatively quicker results. To help with the processing time and unnecessary calculations, we are only required to perform analysis on an envelope ranging from latitude: 40.5 N - 40.9 N and longitude: 73.7 W - 74.25 W.

Getis-ord Statistic:

Getis-ord statistic is a commonly used statistic used to identify statistically significant clusters in spatial data. It provides z-score and p-values which help determine where features with either low or high values are clustered spatially. The function is defined as [1]:

$$G_i^* = \frac{\sum_{j=1}^{n} w_{i,j}x_j - \bar{X}\sum_{j=1}^{n} w_{i,j}}{S\sqrt{\frac{\left[n\sum_{j=1}^{n} w_{i,j}^2 - \left(\sum_{j=1}^{n} w_{i,j}\right)^2\right]}{n-1}}}$$

where xj is the attribute value for the cell j, wi, j is the spatial weight between cell i and j, n is equal to the total number of cells, for simplicity the spatial weight is assumed to be equal 1, if the cells i and j are neighbors, else 0:

$$\bar{X} = \frac{\sum_{j=1}^{n} x_j}{n} \qquad S = \sqrt{\frac{\sum_{j=1}^{n} x_j^2}{n} - (\bar{X})^2}$$

A space time cube is created based on the latitude, longitude and time, which is used to compute the neighbourhood for each cell. Each cell will have 26 neighbours comprising of the preceding, following and current time periods. It is assumed that the weight of each neighbor cell is equal. The cube would be created as [1]:

Our output will also be in accordance to the coordinate system laid down by this space time cube system. Ex-

cell_x, cell_y, date, gscore

Where,
cell_x = latitude for that cell,
cell_y = longitude for that cell,
day = day step for that cell,
gscore = Getis-ord value for that cell.

Algorithm:

As an overview, our approach can be broken down into three main parts. The first part strips the data to use only the required attributes and then divides this data based on the cells, where each cell is uniquely identified by the lat-long values and the day. The next part involves calculating the attribute values for the each cells and finding the neighbors of the cell. In the final part, we calculate the statistic of each cell using the Getis-Ord formula. We use the spark flatMapToPair and reduceByKey functions for mapping and reducing the data. Our approach in detail is as follows:
Note- We will be using the phrase Cell Identifier to denote a particular cell, that is of the format- cell_x, cell_y, day.

Step 1- (Mapping the Input): First, we parse the input and extract the latitude, longitude and day values for each row. Then, we find the corresponding cell that this row belongs to and emit a key-value pair, with the cell identifier as key and “1” as the value. The “1” denotes the presence of a data-point in a cell. We map the entire input following this step.

Step 2 (Reducing the input): In the reducer, we sum the values for the entries with equal keys. Since every data-point is associated with a particular cell, this gives us the attribute value for each cell. The attribute value is nothing but the count of the data-points in a particular cell. Following this step, we get a reduced attributeRDD that has the attribute value for each cell. Using this, we create another RDD called squaredRDD, that gives us the sum of the squared attribute values over our collection. We do this by mapping the squared values of the attributeRDD to a common key, "SquaredSum". We then reduce this RDD by adding all the values with the common key. We get a single entry RDD that has the squared attribute sum as the value.

Step 3 (Mapping the neighbors of a cell): Now that we have sum of attribute values, the squared attribute sum, we can derive the mean of the attribute sum and the standard deviation using the formula that has been provided. Next we get the list of neighbors for each cell based on their latitude-longitude and day values. For every neighbor in the list of neighbors for a cell, we emit a key-value pair where the key is the cell identifier for that neighbor while the value consists of the attribute value and 1 (comma separated). The 1 indicates the presence of a neighbor. We stored this in a RDD called the neighborPairRDD.

Step 4 (Reducing the neighbors): We reduce the neighborPairRDD by adding up the attribute values and the 1s. This effectively gives us the sum of the attribute values of the all the neighbors of a cell and the count of neighbors. We have a neighborReducedRDD that has all the values associated needed to find the Getis-ord value. We find the Getis score for every cell and create a RDD called gscoresRDD that emits a key-value pair for every cell. Here, the key is a combination of the Getis-ord value and the cell Identifier (comma separated.). We are adding the Getis-ord to the key to leverage the sortByKey function available in Spark.

Step 5: The gscoresRDD is then sorted based on their getis score, in a descending order. The cell identifier is then converted to our corresponding space-time cube system and the top 50 entries are written to a file.

Command used to run the Spark Job:

./bin/spark-submit --master local[*] --class dds.phase3.App /home/vraj/phase3.jar hdfs://localhost:54310/input hdfs://localhost:54310/output

Here is a sample of the output as required by the submission guidelines. This sample contains the top 20 outputs.

40.75,-73.99,13,64.86948222817657
40.75,-73.99,14,64.43715615183363
40.75,-73.99,21,63.853017994932024

40.75,-73.99,12,63.19456980364423
40.75,-73.99,8,62.69533916025946
40.75,-73.99,15,62.34855899979065
40.75,-73.99,22,62.34128121291778
40.75,-73.99,7,62.18168062360039
40.75,-73.99,29,61.88482352747006
40.75,-73.99,20,61.62678129466172
40.75,-73.98,13,60.37615107677071
40.75,-73.99,6,60.23838384807195
40.75,-73.98,14,59.740046967987354
40.74,-73.99,29,59.521968722744084
40.75,-73.98,21,59.222685697656125
40.74,-73.99,14,59.18042346160489
40.75,-73.99,16,59.174933201332365
40.74,-73.99,8,59.151950716470665
40.74,-73.99,15,59.06857536861126

References:

[1] ACM SIGSPATIAL Cup 2016 Problem Definition:
http://sigspatial2016.sigspatial.org/giscup2016/problem