Big Data Computing

CISC 5957-Lab 2

Jonathan Ramos & Arna Sadia

Spark Programming

In this lab, we installed the Spark cluster, and based on the code in Project 1, we implemented the K-Means algorithm using PySpark. The lab has 3 parts.

Part 1

For each player, we define the comfortable zone of shooting as a matrix of,

{SHOT DIST, CLOSE DEF DIST, SHOT CLOCK}

In part 1 of the lab, we implemented a K-Means algorithm to classify each player's records into 4 comfortable zones. Considering the hit rate, we also figured out which zone is the best for James Harden, Chris Paul, Stephen Curry, and Lebron James.

1. Initializing spark session

```
spark = SparkSession.builder.appName("part1NBA").getOrCreate()
```

2. Loading the csv file into the spark context and filter based on player names and the given column names. Converting the DataFrame into RDD through map to prepare for the iteration.

```
dataset = spark.read.format("csv").load(sys.argv[1])

dataPoints = dataset.filter(dataset.playerName == 'Chris
Paul').select('SHOT_DIST', 'CLOSE_DEF_DIST', 'SHOT_CLOCK').na.drop()
dRDD = dataPoints.rdd.map(lambda r: (r[0], r[1], r[2]))
```

3. After the previous step, we implemented the K-Means algorithm by randomly selecting 4 rows as preliminary centroids.

```
ct = 4
centroidN = dRDD.takeSample(False, ct)
```

Then iteratively run the K-Means algorithm. We determine the separations between each data point and the four original centroids, then place each one in the cluster to which its closest centroid belongs. The new centroid is then calculated for each cluster by taking the mean of each column. We then compute the distance between each data point and the new centroids, giving us 4 new centroids. Until the new centroids are equal to the old centroids or until the maximum number of iterations has been achieved, this process is iterated.

First, we calculate the distance of each data to the centroid and then assign each data to its closest centroid. Then we find the mean iteration for each cluster to determine the new centroid: the distance between each data point and the NEW centroid. Finally, we stop the iteration when the original and new centroids are equal, or when the maximum number of iterations has been reached.

```
ctr = 0
centroid0 = centroidN

for m in range(40):
    map1 = dRDD.map(lambda r: closestCenter(r, centroid0))
    reduce1 = map1.groupByKey()
    map2 = reduce1.map(lambda x: calculateCentroid(x)).collect()
    newCentroid = map2
    converge = 0
    for i in range(k):
        if newCentroid[i] == centroid0[i]:
            converge += 1
        else:
            d = 0.0009
            cd = [round((f - l) ** 2, 6) for f, l in zip(newCentroid[i],
centroid0[i])]
            if all(v <= d for v in cd):
                converge += 1
    if converge >= 4:
        print(ctr)
        print(newCentroid)
        break
    else:
        ctr += 1
        print(ctr)
        centroid0 = newCentroid
        print(centroid0)
```

Part 2

In Part 2 of the lab, we implemented a K-Means algorithm using PySpark to find the probability of a black vehicle getting a ticket that has parked illegally at 34510, 10030, and 34050 (street codes). (Very rough prediction)

1. Initializing spark session

```
spark = SparkSession.builder.appName("part2Bonus").getOrCreate()
```

2. We loaded the CSV file as DataFrame and filtered it according to the color black. We selected the columns of Street code and then transformed the filtered dataframe into RDD.

```
dataset = spark.read.format("csv").load(sys.argv[1])
```

3. After the previous step, we implemented the K-Means algorithm by randomly selecting 4 rows as preliminary centroids.

```
ct = 4
centroidN = dRDD.takeSample(False, ct)
```

Then iteratively run the K-Means algorithm. We determine the separations between each data point and the four original centroids, then place each one in the cluster to which its closest centroid belongs. The new centroid is then calculated for each cluster by taking the mean of each column. We then compute the distance between each data point and the new centroids, giving us 4 new centroids. Until the new centroids are equal to the old centroids or until the maximum number of iterations has been achieved, this process is iterated.

First, we calculate the distance of each data to the centroid and then assign each data to its closest centroid. Then we find the mean iteration for each cluster to determine the new centroid: the distance between each data point and the NEW centroid. Finally, we stop the iteration when the original and new centroids are equal, or when the maximum number of iterations has been reached.

```
ctr = 0
centroidO = centroidN
```

```
for m in range(40):
    map1 = dRDD.map(lambda r: closestCenter(r, centroid0))
    reduce1 = map1.groupByKey()
    map2 = reduce1.map(lambda x: calculateCentroid(x)).collect()
    newCentroid = map2
    converge = 0
    for i in range(k):
        if newCentroid[i] == centroid0[i]:
            converge += 1
        else:
            diff = 0.0009
            closeDiff = [round((a - b) ** 2, 6) for a, b in
zip(newCentroid[i], centroid0[i])]
            if all(v <= diff for v in closeDiff):
                converge += 1
    if converge >= 4:
        print(ctr)
        print(newCentroid)
        break
    else:
        ctr += 1
        print(ctr)
        centroid0 = newCentroid
        print(centroid0)
```

4. Finally, we calculate the probability of a black vehicle getting a ticket.

```
streetCode = [34510, 10030, 34050]
clos = closestCenter(streetCode, ())

map3 = dRDD.filter(lambda x: closestCenter(x, newCentroid)[0] ==
clos[0]).collect()
count = len(map3)
token = dict(cntr(map3))
cntr = len(token)
maxValue = max(token.items(), key=itemgetter(1))[1]
probability = round(count / (maxValue * cntr), 6)
print(probability)
```

Part 3

In part 3 of the lab, we figured out when tickets are most likely to be issued by trying out different levels of parallelism, 2, 3, 4, 5. In order to do so, we set the configuration of spark-submit to the parallelism levels by adding the following line of code after spark-submit in the test.sh file:

–conf spark.default.parallelism=2

```
/usr/local/spark/bin/spark-submit --conf spark.default.parallelism=2 --
master=spark://$SPARK_MASTER:7077 ./part3.py
hdfs://$SPARK_MASTER:9000/CSIC5950-Lab2/Part 3/input/
```

The Readme.md files contain the screenshot of the code and the output.