FACULTÉ DES SCIENCES DHAR EL MAHRAZ
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH

Mini-Project: Analysis on Uber Event Data

Big Data Analytics and Smart Systems

Realized By:
BERRAG AYOUB

Supervised By:
Pr. FRIKH BOUCHRA

2020/2021

Part 1 : Cluster Analysis on Uber Event Data to Detect and Visualize Popular Uber Locations

We will discover the clusters of Uber data based on the longitude and latitude, then we will analyze the cluster centers by date/time, usings Spark SQL.

- ## Load the Data from a File into a DataFrame

First, we import the packages needed for Spark ML clustering and SQL, than we specify the schema with a Spark **StructType** and a Scala case class.

```
import org.apache.spark._
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._
import org.apache.spark.sql._
import org.apache.spark.sql.Dataset
import org.apache.spark.sql.types.TimestampType
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.clustering.{KMeans, KMeansModel}
import org.apache.spark.sql.types.TimestampType
```

Next we load the data from a CSV file into a Spark DataFrame, specifying the data source and schema to load into the DataFrame, as shown below

```
+----------------+-------+--------+------+
|              dt|    lat|     lon|  base|
+----------------+-------+--------+------+
|9/1/2014 0:01:00|40.2201|-74.0021|B02512|
|9/1/2014 0:01:00|  40.75|-74.0027|B02512|
|9/1/2014 0:03:00|40.7559|-73.9864|B02512|
|9/1/2014 0:06:00| 40.745|-73.9889|B02512|
|9/1/2014 0:11:00|40.8145|-73.9444|B02512|
|9/1/2014 0:12:00|40.6735|-73.9918|B02512|
|9/1/2014 0:15:00|40.7471|-73.6472|B02512|
|9/1/2014 0:16:00|40.6613|-74.2691|B02512|
|9/1/2014 0:32:00|40.3745|-73.9999|B02512|
|9/1/2014 0:33:00|40.7633|-73.9773|B02512|
|9/1/2014 0:33:00|40.7467|-73.6131|B02512|
|9/1/2014 0:37:00|40.8105|  -73.96|B02512|
|9/1/2014 0:38:00| 40.679|-74.0111|B02512|
|9/1/2014 0:39:00|40.4023|-73.9839|B02512|
|9/1/2014 0:48:00|40.7378|-74.0395|B02512|
|9/1/2014 0:48:00|40.7214|-73.9884|B02512|
|9/1/2014 0:49:00|40.8646|-73.9081|B02512|
|9/1/2014 1:08:00|40.7398|-74.0061|B02512|
|9/1/2014 1:17:00|40.6793|-74.0116|B02512|
|9/1/2014 1:19:00|40.7328|-73.9875|B02512|
+----------------+-------+--------+------+
```

- ## Define Features Array

In order for the features to be used by a machine learning algorithm, they are transformed and put into feature vectors, which are vectors of numbers representing the value for each feature. Below, a VectorAssembler transformer is used to return a new DataFrame with the input columns lat, lon in a vector features column. The df2 DataFrame with the features column is cached, since it will be used iteratively by the k-means estimator to create a model.

```
+-----------------+-------+--------+------+------------------+
|               dt|    lat|     lon|  base|          features|
+-----------------+-------+--------+------+------------------+
|9/1/2014 0:01:00|40.2201|-74.0021|B02512|[40.2201,-74.0021]|
|9/1/2014 0:01:00|  40.75|-74.0027|B02512|  [40.75,-74.0027]|
|9/1/2014 0:03:00|40.7559|-73.9864|B02512|[40.7559,-73.9864]|
|9/1/2014 0:06:00| 40.745|-73.9889|B02512| [40.745,-73.9889]|
|9/1/2014 0:11:00|40.8145|-73.9444|B02512|[40.8145,-73.9444]|
|9/1/2014 0:12:00|40.6735|-73.9918|B02512|[40.6735,-73.9918]|
|9/1/2014 0:15:00|40.7471|-73.6472|B02512|[40.7471,-73.6472]|
|9/1/2014 0:16:00|40.6613|-74.2691|B02512|[40.6613,-74.2691]|
|9/1/2014 0:32:00|40.3745|-73.9999|B02512|[40.3745,-73.9999]|
|9/1/2014 0:33:00|40.7633|-73.9773|B02512|[40.7633,-73.9773]|
|9/1/2014 0:33:00|40.7467|-73.6131|B02512|[40.7467,-73.6131]|
|9/1/2014 0:37:00|40.8105|  -73.96|B02512|  [40.8105,-73.96]|
|9/1/2014 0:38:00| 40.679|-74.0111|B02512| [40.679,-74.0111]|
|9/1/2014 0:39:00|40.4023|-73.9839|B02512|[40.4023,-73.9839]|
|9/1/2014 0:48:00|40.7378|-74.0395|B02512|[40.7378,-74.0395]|
|9/1/2014 0:48:00|40.7214|-73.9884|B02512|[40.7214,-73.9884]|
|9/1/2014 0:49:00|40.8646|-73.9081|B02512|[40.8646,-73.9081]|
|9/1/2014 1:08:00|40.7398|-74.0061|B02512|[40.7398,-74.0061]|
|9/1/2014 1:17:00|40.6793|-74.0116|B02512|[40.6793,-74.0116]|
|9/1/2014 1:19:00|40.7328|-73.9875|B02512|[40.7328,-73.9875]|
+-----------------+-------+--------+------+------------------+
```

- K-means Model

Next, we create a k-means estimator; we set the parameters to define the number of clusters and the column name for the cluster IDs. Then we use the k-means estimator fit method, on the VectorAssembler transformed DataFrame, to train and return a k-means model.

```
Final Centers:
[40.73207616599548,-73.99775800997581]
[40.768763598033644,-73.97217024526178]
[40.62582412010329,-73.97703811891023]
[40.76951584967321,-73.50817510893253]
[40.78363000141837,-73.87949407008126]
[40.65904219070612,-73.78295348012725]
[40.71237815343643,-73.94402168814342]
[40.989510394110034,-73.79590134257224]
[40.69783126827318,-74.20463852150976]
[40.68240359480648,-73.98053296895468]
```

We use the k-means model summary and k-means model summary predictions methods, which return the clusterIDs added as a column in a new DataFrame, in order to further analyze the clustering. Then we register the DataFrame as a temporary table in order to run SQL statements on the table.

lat	lon	base	features	dt	cid
40.2201	-74.0021	B02512	[40.2201,-74.0021]	2014-09-01 00:01:00	2
40.75	-74.0027	B02512	[40.75,-74.0027]	2014-09-01 00:01:00	0
40.7559	-73.9864	B02512	[40.7559,-73.9864]	2014-09-01 00:03:00	1
40.745	-73.9889	B02512	[40.745,-73.9889]	2014-09-01 00:06:00	0
40.8145	-73.9444	B02512	[40.8145,-73.9444]	2014-09-01 00:11:00	1
40.6735	-73.9918	B02512	[40.6735,-73.9918]	2014-09-01 00:12:00	9
40.7471	-73.6472	B02512	[40.7471,-73.6472]	2014-09-01 00:15:00	3
40.6613	-74.2691	B02512	[40.6613,-74.2691]	2014-09-01 00:16:00	8
40.3745	-73.9999	B02512	[40.3745,-73.9999]	2014-09-01 00:32:00	2
40.7633	-73.9773	B02512	[40.7633,-73.9773]	2014-09-01 00:33:00	1
40.7467	-73.6131	B02512	[40.7467,-73.6131]	2014-09-01 00:33:00	3
40.8105	-73.96	B02512	[40.8105,-73.96]	2014-09-01 00:37:00	1
40.679	-74.0111	B02512	[40.679,-74.0111]	2014-09-01 00:38:00	9
40.4023	-73.9839	B02512	[40.4023,-73.9839]	2014-09-01 00:39:00	2
40.7378	-74.0395	B02512	[40.7378,-74.0395]	2014-09-01 00:48:00	0
40.7214	-73.9884	B02512	[40.7214,-73.9884]	2014-09-01 00:48:00	0
40.8646	-73.9081	B02512	[40.8646,-73.9081]	2014-09-01 00:49:00	4
40.7398	-74.0061	B02512	[40.7398,-74.0061]	2014-09-01 01:08:00	0
40.6793	-74.0116	B02512	[40.6793,-74.0116]	2014-09-01 01:17:00	9
40.7328	-73.9875	B02512	[40.7328,-73.9875]	2014-09-01 01:19:00	0

- ## Questions answers

Which clusters had the highest number of pickups?

$$clusters.groupBy("cid").count().orderBy(desc("count")).show$$

In Spark SQL

$$spark.sql("select\ cid, count(cid)\ as\ count\ from\ uber\ group\ by\ cid").show$$

cid	count
1	344901
6	74179
3	2580
5	30890
9	66158
4	50460
8	10261
7	3281
2	14269
0	431157

Which hours of the day had the highest number of pickups?

$$clusters.select(hour(\$"dt").alias("hour"), \$"cid").groupBy("hour", "cid").agg(count("cid")$$
$$.alias("count")).orderBy(desc("count")).show$$

In Spark SQL

$$spark.sql("SELECT\ hour(uber.dt)\ as\ hr, count(cid)\ as\ ct\ FROM\ uber\ group\ By\ hour(uber.dt)").show$$

hour	cid	count
18	0	32181
17	0	31637
19	0	29872
16	0	28642
18	1	27035
17	1	27000
20	0	26874
21	0	26278
16	1	25567
15	0	25493
22	0	24025
19	1	23999
15	1	22679
14	0	21298
20	1	20780
14	1	18472
21	1	18354
23	0	18210
13	0	18125
8	0	17630

Which cluster/base combination had the highest number of pickups?

$$clusters.groupBy("cid","base").count().orderBy(desc("count")).show$$

cid	base	count
0	B02617	158974
1	B02617	128025
0	B02598	101745
0	B02682	84276
1	B02598	80324
0	B02764	71030
1	B02682	66147
1	B02764	57558
6	B02617	26078
9	B02617	23959
4	B02617	18291
6	B02764	16681
6	B02598	16405
9	B02598	15551
0	B02512	15132
6	B02682	13769
9	B02764	13390
1	B02512	12847
9	B02682	12078
4	B02598	11733

Part 2 : Cluster Analysis on Uber Event Data to Detect and Visualize Popular Uber Locations

In this part we will try to use spark streaming to analyze the Uber use case

• Loading the K-Means Model

The Spark KMeansModel class is used to load a k-means model, which was fitted on the historical Uber trip data and then saved Next, a Dataset of Cluster Center IDs and location is created to join later with the Uber trip locations.

```
val model = KMeansModel.load("model")

model: org.apache.spark.ml.clustering.KMeansModel = KMeans_31d3ba4bba02
```

```
case class Center(cid: Integer, clat: Double, clon: Double)   extends Serializable
var ac = new Array[Center](8)
var index: Int = 0
 model.clusterCenters.foreach(x => {
    ac(index) = Center(index, x(0), x(1));
    index += 1;
    })
 val ccdf = spark.createDataset(ac)
 ccdf.show

defined class Center
ac: Array[Center] = Array(null, null, null, null, null, null, null, null)
index: Int = 0
ccdf: org.apache.spark.sql.Dataset[Center] = [cid: int, clat: double ... 1 more field]
+---+------------------+------------------+
|cid|              clat|              clon|
+---+------------------+------------------+
|  0|40.731004063086196|  -73.997821843761|
|  1| 40.76671785028902|-73.97185792562814|
|  2| 40.65715454643839| -73.7784502515282|
|  3|40.759877422009396|-73.87319539537648|
|  4| 40.70030127111548| -74.2001973574177|
|  5| 40.77089895150719|-73.46007673656621|
|  6| 40.68624863582041|-73.96377484246057|
|  7| 40.88602293520361|-73.89210208311515|
+---+------------------+------------------+
```

• Reading Data from Kafka Topics

In order to read from Kafka, we must first specify the stream format, topic, and offset options.

```
val df1 = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "ec2-3-88-10-219.compute-1.amazonaws.com:9092").option("subscribe", "uber").option("startingOffsets","e
df1: org.apache.spark.sql.DataFrame = [key: binary, value: binary ... 5 more fields]
```

```
df1.printSchema()

root
 |-- key: binary (nullable = true)
 |-- value: binary (nullable = true)
 |-- topic: string (nullable = true)
 |-- partition: integer (nullable = true)
 |-- offset: long (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- timestampType: integer (nullable = true)
```

The next step is to parse and transform the binary values column into a Dataset of Uber objects.

- ## Parsing the Message Values into a Dataset of Uber Objects

A Scala Uber case class defines the schema corresponding to the CSV records. The parseUber function parses a comma separated value string into an Uber object. Than we register a user-defined function (UDF) to deserialize the message value strings using the parseUber function. Then we use the UDF in a select expression with a String Cast of the df1 column value, which returns a DataFrame of Uber objects.

```
case class Uber(Date: String, Lat: Double, Lon: Double, Base: String)   extends Serializable
 // Parse string into Uber case class
 def parseUber(str: String): Uber = {
    val p = str.split(",")
    Uber(p(0), p(1).toDouble, p(2).toDouble, p(3))
 }

defined class Uber
parseUber: (str: String)Uber
```

```
import spark.implicits._

spark.udf.register("deserialize",
    (message: String) => parseUber(message))

val df2 = df1.selectExpr("""deserialize(CAST(value as STRING)) AS message""").select($"message".as[Uber])
```

- ## Enriching the Dataset of Uber Objects with Cluster Center IDs and Location

A VectorAssembler is used to transform and return a new DataFrame with the latitude and longitude feature columns in a vector column.

```
val featureCols = Array("Lat", "Lon")
val assembler = new VectorAssembler().setInputCols(featureCols).setOutputCol("features")
val df3 = assembler.transform(df2)

featureCols: Array[String] = Array(lat, lon)
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_cbbfc7a94e64
df3: org.apache.spark.sql.DataFrame = [dt: string, lat: double ... 3 more fields]
```

```
val streamingquery = df3.writeStream.queryName("df3").format("memory").outputMode("append").start()

streamingquery: org.apache.spark.sql.streaming.StreamingQuery = org.apache.spark.sql.execution.streaming.StreamingQueryWrapper@78fa8da6
```

As result

```
%pyspark
df = sqlContext.sql("SELECT * FROM df3")
```

```
%pyspark
df.show()

+-------------------+-------+--------+------+------------------+
|          Date/Time|    Lat|     Lon|  Base|          features|
+-------------------+-------+--------+------+------------------+
|2014-08-01 00:00:00|40.7476|-73.9871|B02598|[40.7476,-73.9871]|
|2014-08-01 00:00:00|40.7424|-74.0044|B02598|[40.7424,-74.0044]|
|2014-08-01 00:00:00| 40.751|-73.9869|B02598| [40.751,-73.9869]|
|2014-08-01 00:00:00|40.7406|-73.9902|B02598|[40.7406,-73.9902]|
|2014-08-01 00:00:00|40.6994|-73.9591|B02617|[40.6994,-73.9591]|
|2014-08-01 00:00:00|40.6917|-73.9398|B02617|[40.6917,-73.9398]|
|2014-08-01 00:00:00|40.7063|-73.9223|B02617|[40.7063,-73.9223]|
|2014-08-01 00:00:00|40.6759|-74.0168|B02617|[40.6759,-74.0168]|
|2014-08-01 00:00:00|40.7617|-73.9847|B02617|[40.7617,-73.9847]|
|2014-08-01 00:00:00|40.6969|-73.9064|B02617|[40.6969,-73.9064]|
|2014-08-01 00:00:00|40.7623|-73.9751|B02617|[40.7623,-73.9751]|
|2014-08-01 00:00:00|40.6982|-73.9669|B02617|[40.6982,-73.9669]|
|2014-08-01 00:00:00|40.7553|-73.9253|B02617|[40.7553,-73.9253]|
|2014-08-01 00:00:00|40.7325|-73.9876|B02682|[40.7325,-73.9876]|
```

Than we writing to a Memory Sink

```
%pyspark
df_clusters = df_clusters.withColumnRenamed("Date/Time", "dt").withColumnRenamed("Lat", "lat").withColumnRenamed("Lon", "lon")
```

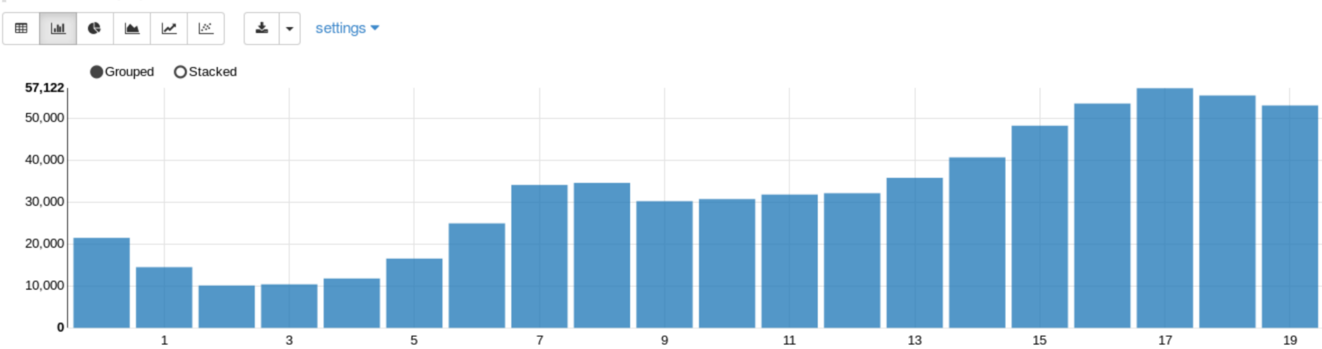
```
%pyspark
ctr = sc.parallelize([['40.73100406', '-73.99782184', '0'],
                      ['40.76671785', '-73.97185793', '1'],
                      ['40.65715455', '-73.77845025', '2'],
                      ['40.75987742', '-73.8731954', '3'],
                      ['40.70030127', '-74.20019736', '4'],
                      ['40.77089895', '-73.46007674', '5'],
                      ['40.68624864', '-73.96377484', '6'],
                      ['40.88602294', '-73.89210208', '7']]).\
    toDF(["lat", "lon", "centroid"])
```

```
%pyspark
ctr.registerTempTable("centroids")
df_clusters.registerTempTable("uber")
```

- Questions answers

Which hours have the highest number of pickups for cluster 0?

```
%sql
SELECT hour(dt) ,count(prediction) as count
FROM uber
GROUP BY hour(dt)
```



Which hours of the day and which cluster had the highest number of pickups?

```
%sql
SELECT hour(dt) AS hr, prediction, count(prediction) AS count
FROM uber
GROUP BY hour(dt), prediction
```

settings