Big Data Project LoudAcre Mobile Kmeans Clustering

import org.apache.spark.mllib.linalg.Vectors import org.apache.spark.mllib.clustering.KMeans import org.apache.spark.sql.functions._

1. Loading the Dataset

val filename = "C:/spark/loudacre_BigData/loudacre_Dataset" val loudacre_data = sc.textFile(filename)

Create a new dataframe having only latitude and longitude

val loudacre_data_split = loudacre_data.map(x => x.split(",")).map(s => (s(3),s(4))).toDF
loudacre_data_split.show(10)

2. Cleaning

 Many of the values are zero in both the columns, customer might have not provided their information about their location. we will remove these rows.

loudacre_data_split.filter(\$"_1"==="0").show(10)

Convert dataframe back to RDD

val la_rdd = la_df.rdd.map(row => List(row.getString(0),row.getString(1)))

Create a vector RDD

val vectors = la_rdd.map(s => Vectors.dense(s(0).toDouble,s(1).toDouble)).cache()

3. Train the model

val numClusters = 3 val numIterations = 20 val kmeansmodel =

KMeans.train(vectors,numClusters,numIterations)

Display the center points of each cluster

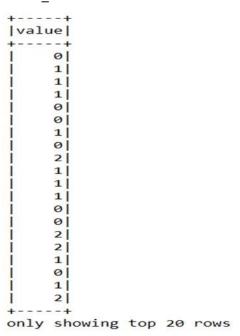
kmeansmodel.clusterCenters.foreach(println)

```
[34.54436372366799,-118.05513586535038]
[39.92266331983432,-121.38464957351843]
[35.08592000544937,-112.57643826547802]
```

Create a broadcast variable for the kmeans model

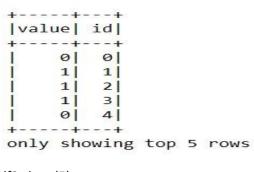
kmeansmodel.computeCost(vectors)

```
val kmeansModel_BC = sc.broadcast(kmeansmodel)
val cluster_df = kmeansModel_BC.value.predict(vectors).toDF
cluster_df.show
```



• Create a new column called "id", which will be used for joining the two dataframes

```
val df1 = cluster_df.withColumn("id",monotonically_increasing_id())
val df2 = la_df.withColumn("id",monotonically_increasing_id())
df1.show(5)
```



df2.show(5)

• Join the two dataframes based on "id" column

val df3 = df2.join(df1,"id")

df3.show

value	_2	_1	id
1	-119.659926097	39.4708861702	26
0	-117.787423338	33.8514465953	29
1	-121.020788448	39.5548840745	474
1	-122.490145742	42.8588445306	964
1	-123.772498927	42.0521070411	1677
0	-118.248366666	34.206121509	1697
1	-121.0370419	36.5039511635	1806
0	-118.13214751	34.1840832076	1950
1	-119.337989148	39.4458400243	2040
1	-121.511788497	37.9311384736	2214
0	-117.67549234	34.1648878746	2250
2	-115.053517304	36.1947181592	2453
1	-120.975396248	38.3183374986	2509
1	-121.845985179	38.1666641293	2529
0	-120.722388463	36.3195030742	2927
1	-122.29490043	37.801310103	3091
1	-122.403638455	38.3391887988	3506
2	-109.124901699	33.5908087287	3764
1	-121.72570178	37.1075594103	4590
2	-111.98383465	34.0175895081	4823

• The number of point for each cluster in the dataset

df3.groupBy("value").count().show()

```
| value | count |
| value | count |
| 1 | 177812 |
| 2 | 65686 |
| 0 | 188359 |
```

• We remove the "id" column and name the other three columns

val newNames = Seq("latitude","longitude","cluster") val finaldf = df3.drop("id").toDF(newNames:_*) finaldf.show

uster	longitude cl	latitude
	-	
1	-119.659926097	-
e	-117.787423338	33.8514465953
1	-121.020788448	39.5548840745
1	-122.490145742	42.8588445306
1	-123.772498927	42.0521070411
e	-118.248366666	34.206121509
1	-121.0370419	36.5039511635
e	-118.13214751	34.1840832076
1	-119.337989148	39.4458400243
1	-121.511788497	37.9311384736
e	-117.67549234	34.1648878746
2	-115.053517304	36.1947181592
1	-120.975396248	38.3183374986
1	-121.845985179	38.1666641293
e	-120.722388463	36.3195030742
1	-122.29490043	37.801310103
1	-122.403638455	
2	-109.124901699	
1	-121.72570178	: [
2	-111.98383465	: [[[[[[[[[[[[[[[[[[[

• Display the points for each cluster

finaldf.filter(\$"cluster" === "0").show

latitude	longitude	cluster
t	L	+
	-117.787423338	
	-118.248366666	0
34.1840832076	-118.13214751	0
34.1648878746	-117.67549234	0
36.3195030742	-120.722388463	0
34.2298811925	-117.877102556	0
34.0669923073	-118.192176947	0
33.0262077564	-116.812861708	0
34.1521333194	-116.638723297	0
32.9596188021	-116.805661125	0
33.9249343611	-117.881397679	0
Te	-116.674524717	
33.0339156527	-116.837511787	0
37.038360427	-119.588745345	0
32.9965920016	-116.685453769	ø
E0	-120.285702621	
	-118.2386197	
	-116.724525124	
	-116.307005506	3500,000
	-117.770690361	0

only showing top 20 rows

finaldf.filter(\$"cluster" === "1").show

latitude	longitude	cluster
39.4708861702	-119.659926097	1
39.5548840745	-121.020788448	1
42.8588445306	-122.490145742	1
42.0521070411	-123.772498927	1
36.5039511635	-121.0370419	1
39.4458400243	-119.337989148	1
37.9311384736	-121.511788497	1
38.3183374986	-120.975396248	1
38.1666641293	-121.845985179	1
37.801310103	-122.29490043	1
38.3391887988	-122.403638455	1
37.1075594103	-121.72570178	1
38.4745414188	-122.134143962	1
37.6543322066	-121.588940213	1
45.3409177163	-117.542333377	1
45.1781132987	-117.661882259	1
42.6990076523	-122.637552833	1
38.7301629267	-121.408276173	1
45.4371672468	-117.700782699	1
37.5033221854	-121.525720878	1

only showing top 20 rows

finaldf.filter(\$"cluster" === "2").show

uster	longitude cl	latitude
2	-115.053517304	36.1947181592 -
2	-109.124901699	33.5908087287 -
2	-111.98383465	34.0175895081
2	-114.56289137	39.6628949081
2	-111.542615553	33.6230375795 -
2	-111.408292531	35.0789202973 -
2	-114.731425185	36.0838304774 -
2	-113.613971565	36.2981261276 -
2	-111.641629449	32.2734814339 -
2	-112.003312523	33.5206745813 -
2	-115.006223963	36.661607919
2	-111.484732167	33.3561047647
2	-111.861572714	33.3954391971 -
2	-111.791915607	32.2866036613 -
2	-111.506443379	32.3938041755
2	-111.513103129	33.6603116416
2	-111.316399941	35.0455801515
2	-111.825495401	33.7285608333
2	-110.614014808	32.5350707596 -
2	-111.304704398	

only showing top 20 rows

4. Conclusion

• Successfully divided the user's location data into three clusters with centers [34.544,-118.055],[39.922,-121.384],[35.085,-112.576]. This will further help the company improve their service by maximising the coverage for its users.