Week 3 pySpark ML Clustering

May 1, 2024

1 Welcome to Week 3

```
[1]: from pyspark.sql import SparkSession
     from pyspark.ml.clustering import KMeans
     from pyspark.ml.feature import VectorAssembler
     from pyspark.ml.feature import StandardScaler
```

1.1 Build the model

Import Data

```
[2]: spark = SparkSession.builder.appName("clustering-w3").getOrCreate()
     df = spark.read.csv('data/w3_clustering.csv', header=True, inferSchema=True)
```

Let us display the number of lines in the data:

```
[3]: df.count()
```

```
[3]: 709
```

What does the data contain?

```
[4]: df
```

```
[4]: DataFrame[totalAdClicks: int, revenue: double]
```

Statistics about the data:

```
[5]: df.describe().toPandas().transpose()
```

```
[5]:                    0                  1                  2    3      4
     summary        count               mean             stddev  min    max
     totalAdClicks    709  32.208744710860366  16.384120817042554    1     73
     revenue          709   44.64880112834979  44.944528765285284  1.0  278.0
```

Drop all rows with NULL or NaN values

```
[6]: df = df.na.drop()
```

Let us look at the column names:

[7]:
```
df.columns
```

[7]:
```
['totalAdClicks', 'revenue']
```

Use VectorAssembler to gather all the features:

[8]:
```
featuresUsed = ['totalAdClicks', 'revenue']
assembler = VectorAssembler(inputCols=featuresUsed,␣
 ↪outputCol="features_unscaled")
assembled = assembler.transform(df)
```

Scale the features using StandardScaler:

[9]:
```
scaler = StandardScaler(inputCol="features_unscaled", outputCol="features",␣
 ↪withStd=True, withMean=True)
scalerModel = scaler.fit(assembled)
scaledData = scalerModel.transform(assembled)
```

Select the features column make the data persist:

[10]:
```
scaledData = scaledData.select("features")
scaledData.persist()
```

[10]:
```
DataFrame[features: vector]
```

We can now perform KMeans clustering to generate 2 clusters:

[11]:
```
kmeans = KMeans(k=2, seed=1)
model = kmeans.fit(scaledData)
transformed = model.transform(scaledData)
```

Print the center of these two clusters:

[12]:
```
centers = model.clusterCenters()
centers
```

[12]:
```
[array([0.84174521, 0.51884657]), array([-0.79780796, -0.49176392])]
```

1.2 Analyze center of these two clusters

Each array denotes the center for a cluster: - One Cluster is centered at *array([0.84174521, 0.51884657])* - Other Cluster is centered at *array([-0.79780796, -0.49176392])*

First number (field1) in each array refers to scaled verson of the number of ad-clicks and the second number (field2) is the scaled version of the revenue per user.

Compare the 1st number of each cluster to see how differently users in each cluster behave when it comes to clicking ads.

Compare the 2nd number of each cluster to see how differently users in each cluster behave when it comes to buying stuff.

In one cluster, in general, players click on ads much more often and spend more money on in-app purchases. Assuming that Eglence Inc. gets paid for showing ads and for hosting in-app purchase items, we can use this information to increase game's revenue by increasing the prices for ads we show to the frequent-clickers, and charge higher fees for hosting the in-app purchase items shown to the higher revenue generating buyers.

Note: This analysis requires you to compare the cluster centers and find any 'significant' differences in the corresponding feature values of the centers. The answer to this question will depend on the features you have chosen. Some features help distinguish the clusters remarkably while others may not tell you much. At this point, if you don't find clear distinguishing patterns, perhaps re-running the clustering model with different numbers of clusters and revising the features you picked would be a good idea.