RECOMMENDING ACTIONS FROM A K-MEANS CLUSTER ANALYSIS IN PYSPARK

Prepared by

Ken Wood

December 12, 2020

Attribute Selection

features_used = ["sum(count_gameclicks)", "sum(count_hits)", "sum(price)"]

Attribute	**Rationale for Selection**
"sum(count_gameclicks)" by userId	"count_gameclicks" is provided in the 'combined_data.csv' file. Does the number of game_clicks generated by a userId correlate with money spent?
"sum(count_hits)" by userId	"count_hits" is provided in the 'combined_data.csv' file. Does the number of hits generated by a userId correlate with money spent?
"sum(price)" by userId	"price" paid is provided by userId in the 'buy-clicks.csv' file. Total amount of money spent by user on the game.

Training Data Set Creation

The training data set used for this analysis is shown below (first 5 lines):

userId	sum(count_gameclicks)	sum(count_hits)	sum(price)
231	262	28	63.0
2032	638	59	20.0
233	250	29	28.0
34	665	79	95.0
1234	590	73	53.0
1434	772	89	9.0
1634	2546	266	27.0
1235	367	39	40.0
1835	734	94	27.0
236	606	70	43.0
436	4392	494	43.0
1436	622	65	16.0
1636	317	29	25.0
2236	299	36	15.0
1837	714	90	67.0
38	1425	155	30.0
239	526	50	20.0
439	379	33	25.0
1639	1806	225	155.0
1640	559	73	72.0

only showing top 20 rows

Dimensions of the final data set:

userId	sum(count_gameclicks)	sum(count_hits)	sum(price)	features_unscaled	features
231	262	28	63.0	[262.0,28.0,63.0]	[-0.6344202852895...
2032	638	59	20.0	[638.0,59.0,20.0]	[-0.0018534459321...
233	250	29	28.0	[250.0,29.0,28.0]	[-0.6546085886733...
34	665	79	95.0	[665.0,79.0,95.0]	[0.04357023668131...
1234	590	73	53.0	[590.0,73.0,53.0]	[-0.0826066594671...
1434	772	89	9.0	[772.0,89.0,9.0]	[0.22358260851973...
1634	2546	266	27.0	[2546.0,266.0,27.0]	[3.20808679208386...
1235	367	39	40.0	[367.0,39.0,40.0]	[-0.4577726306817...
1835	734	94	27.0	[734.0,94.0,27.0]	[0.15965298113786...
236	606	70	43.0	[606.0,70.0,43.0]	[-0.0556889216221...
436	4392	494	43.0	[4392.0,494.0,43.0]	[6.31372079595049...
1436	622	65	16.0	[622.0,65.0,16.0]	[-0.0287711837771...
1636	317	29	25.0	[317.0,29.0,25.0]	[-0.5418905614473...
2236	299	36	15.0	[299.0,36.0,15.0]	[-0.5721730165230...
1837	714	90	67.0	[714.0,90.0,67.0]	[0.12600580883161...
38	1425	155	30.0	[1425.0,155.0,30.0]	[1.32216278431870...
239	526	50	20.0	[526.0,50.0,20.0]	[-0.1902776108471...
439	379	33	25.0	[379.0,33.0,25.0]	[-0.4375843272980...
1639	1806	225	155.0	[1806.0,225.0,155.0]	[1.96314141675271...
1640	559	73	72.0	[559.0,73.0,72.0]	[-0.1347597765418...

only showing top 20 rows

Cluster Centers

The code used in creating cluster centers is given below:

We can now perform K-Means clustering to generate 3 clusters:

In [18]:
```
kmeans = KMeans(k=3, seed=1)
model = kmeans.fit(scaledData)
transformed = model.transform(scaledData)
```

Print the center of these three clusters...

In [19]:
```
centers = model.clusterCenters()
centers
```

Out[19]:
```
[array([-0.31333318, -0.33179717, -0.37859269]),
 array([ 2.35752989,  2.3388565 , -0.03049294]),
 array([-0.04642581,  0.05337151,  1.8170137 ])]
```

Convert the cluster centers back to their original scale...

In [20]:
```
centers = [cluster * scalerModel.std + scalerModel.mean
    for cluster in model.clusterCenters()]
centers
```

Out[20]:
```
[array([ 452.85532995,   50.17766497,   24.24111675]),
 array([ 2040.42592593,   229.05555556,    38.66666667]),
 array([ 611.5060241 ,   75.97590361,  115.22891566])]
```

In [ ]:

Cluster centers formed are given in the table below

Cluster #	Center [sum(count_gameclicks), sum(count_hits), sum(price)]
1	452.85532995, 50.17766497, 24.24111675
2	2040.42592593, 229.05555556, 38.66666667
3	611.5060241, 75.97590361, 115.22891566

These clusters can be differentiated from each other as follows:

Cluster 1 is different from the others in that users with a smaller number of gameclicks and count_hits tend to have the lowest number of in-game purchases.

Cluster 2 is different from the others in that the users who generate large gameclicks and count_hits numbers spend slightly more in in-game purchases.

Cluster 3 is different from the others in that the users who generate an intermediate number of gameclicks and count_hits generate the most purchases

Below you can see the summary of the train data set:

Select the features column make the data persist:

```
In [17]: scaledData = scaledData.select("features")
         scaledData.persist()

Out[17]: DataFrame[features: vector]

In [18]: scaledData.show()

+--------------------+
|            features|
+--------------------+
|[-0.6344202852895...|
|[-0.0018534459321...|
|[-0.6546085886733...|
|[0.04357023668131...|
|[-0.0826066594671...|
|[0.22358260851973...|
|[3.20808679208386...|
|[-0.4577726306817...|
|[0.15965298113786...|
|[-0.0556889216221...|
|[6.31372079595049...|
|[-0.0287711837771...|
|[-0.5418905614473...|
|[-0.5721730165230...|
|[0.12600580883161...|
|[1.32216278431870...|
|[-0.1902776108471...|
|[-0.4375843272980...|
|[1.96314141675271...|
|[-0.1347597765418...|
+--------------------+
only showing top 20 rows
```

Recommended Actions

Action Recommended	Rationale for the action
Increase ads to users who play a lot	It was seen that users who play a lot are also the users who spend less. If we increase ads to users who play a lot, it will promote these users to spend more and therefore increase the revenue
Show higher price ads to users who spend more	If we show higher price ads to users who spend more, we can increase the revenue faster. The users who spend more also do not play too much, thus by showing them the more valuable ads first, we can increase the revenue faster.