pyspark_Clustering

April 20, 2022

BIG DATA ANALYTICS

ASSIGNMENT 2 pySpark Clustering

Name: SAAD ATHER Roll No. 21L-7289

---

[1]:
```
import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession as ss
spark = ss.builder.appName('session').getOrCreate()
sc= spark.sparkContext
```

0.0.1 Question 1: Explore and preprocess the "Leaves" data given

[2]:
```
df = spark.read.csv('Leaves.csv',header=True,inferSchema=True)
```

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

```
root
 |-- ID: integer (nullable = true)
 |-- Month of absence: integer (nullable = true)
 |-- Day of the week: integer (nullable = true)
 |-- Distance from Residence to Work: integer (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Body mass index: integer (nullable = true)
 |-- Absenteeism time in hours: integer (nullable = true)
```

[ ]:
```
df.show(10)
```

a) Handle missing values

[5]:
```
df1 = df.dropna(how='any').dropDuplicates()
```

[6]:
```
df1.count()
```

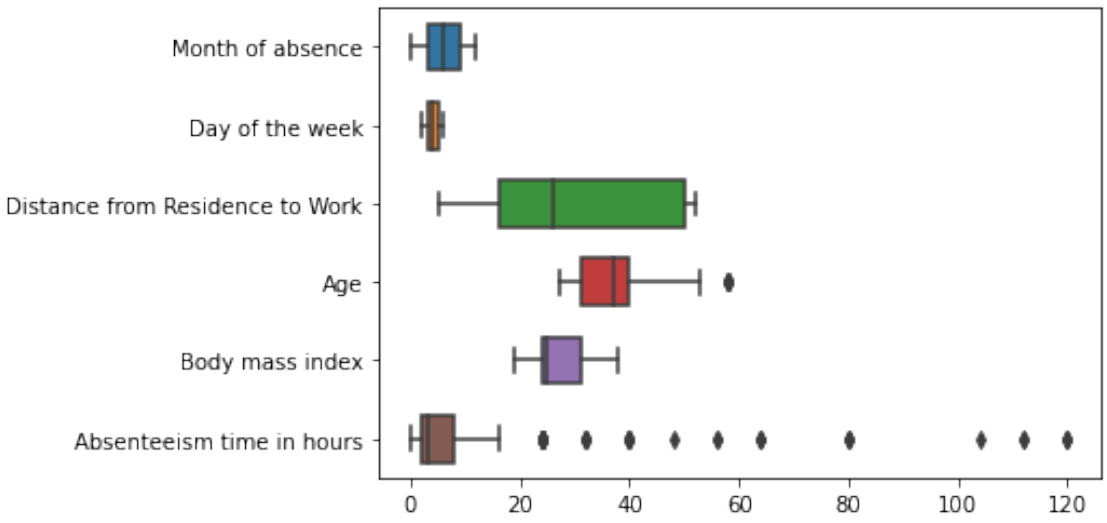
[6]: 648

b) Identify if an attribute has outliers or noise

[7]:
```
col1 = df.drop('ID').select('*').toPandas()
```

[8]:
```
import seaborn as sns
sns.boxplot(data=col1,width=0.6, orient='h')
```

[8]: <AxesSubplot:>



[9]:
```
print(' \nOutliers found in "Absenteeism time" & "Age" \n\n')

print('{} Outliers in Absenteeism time in hours '.format(df.select('Absenteeism␣
 ↪time in hours').where(
    df['Absenteeism time in hours'] > "20").count()))
print('{} Outliers in Age '.format(df.select('Age').where(
    df['Age'] > "57").count()),'\n\n')
```

```
Outliers found in "Absenteeism time" & "Age"


44 Outliers in Absenteeism time in hours
8 Outliers in Age
```

c) Apply measures of the central tendency and dispersion to analyze numeric attributes. That is, compute the mean, median, mode, range, variance, correlation for the attribute. Don't just give values explain analyze them.

[75]:
```
from pyspark.sql.functions import *
```

[144]:
```
print('Corelation between Age & Month of absence is
 ↪ {} '
      .format( df.corr('Age','Month of absence')))

print('Corelation between Day of the week & Distance from Residence to Work is
 ↪  {} '
      .format( df.corr('Day of the week','Distance from Residence to Work')))
```

```
Corelation between Age & Month of absence is
-0.001519544498427041
Corelation between Day of the week & Distance from Residence to Work is
0.11802558057950518
```

[145]:
```
## coreation between each attribute in matrix form


from pyspark.ml.feature import VectorAssembler
from pyspark.ml.stat import Correlation

newdb = VectorAssembler(inputCols=['Age','Month of absence','Day of the
 ↪week','Absenteeism time in hours',
          'Distance from Residence to Work'], outputCol="feat").transform(df)
df_for_corelmatrix = newdb.select('feat')

Correlmatrix = Correlation.corr(df_for_corelmatrix, 'feat').collect()[0]
for cor in Correlmatrix:
    print(cor)
```

```
DenseMatrix([[ 1.        , -0.00151954,  0.00445883,  0.0657597 , -0.14588637],
             [-0.00151954,  1.        , -0.00652801,  0.02434536, -0.00388726],
             [ 0.00445883, -0.00652801,  1.        , -0.12436061,  0.11802558],
             [ 0.0657597 ,  0.02434536, -0.12436061,  1.        , -0.08836282],
             [-0.14588637, -0.00388726,  0.11802558, -0.08836282,  1.        ]])
```

[ ]:
```
df.select('Age','Month of absence','Day of the week','Absenteeism time in
 ↪hours',
          'Distance from Residence to Work').summary().show()
```

Results deduce: By looking at the above data we analyzed that most of the employe lie in the age ranging between 27 to 58 and on average most of employee has an age of 35 years, similarly most of them prefeerd to take halfday/off on the 6th day of week that is on saturday. Employees took leaves for at max of 120 hours (5days OFF) and statistically speaking mean value is of 7hours that is a complete day Off if duty is 8hours per day. On average/mean people took Off/Halfday on 3rd day of week that is Wednesday. On average, employee residence distance from workplace is around 30Km and at max it is 52Km.

d) Would you apply preprocessing techniques like discretization or normalization on any attribute? Explain your answer. If yes, then apply the technique and share the results. Yes, i would apply preprocessing on 4 attributes which i will be considering the main features for my clustering. Since in clustering all the features should be in standerdize form so that the behaviour of the learning algorithum improves

[19]:
```
from pyspark.ml.feature import VectorAssembler
```

[20]:
```
feat_col = ['Month of absence'  ,'Day of the week','Absenteeism time in␣
↪hours','Distance from Residence to Work']
```

[21]:
```
feat_vector = VectorAssembler(inputCols=feat_col, outputCol='features')
```

[22]:
```
data = feat_vector.transform(df)
```

[23]:
```
final_data = data.select('ID','features')
```

[61]:
```
final_data
```

[61]:
```
DataFrame[ID: int, features: vector]
```

[24]:
```
final_data.show(4,truncate=False)
print('\n\n')
```

```
+---+------------------+
|ID |features          |
+---+------------------+
|11 |[7.0,3.0,4.0,36.0]|
|36 |[7.0,3.0,0.0,13.0]|
|3  |[7.0,4.0,2.0,51.0]|
|7  |[7.0,5.0,4.0,5.0] |
+---+------------------+
only showing top 4 rows
```

NOTE: I used three types of scaling method one is using normalize and the other is Standard-Scaler lastly is MinMax scaling, to see the difference between these two scaling techniques

Normalizer Scaling technique

[148]:
```
from pyspark.mllib.util import MLUtils
from pyspark.ml.feature import Normalizer
```

[149]:
```
normalizer = Normalizer(inputCol="features", outputCol="normFeatures", p=1.0)
normData = normalizer.transform(final_data)
```

[ ]:
```
normData.show(4,truncate=False)
```

StanderScaler Scaling technique

[151]:
```
from pyspark.ml.feature import StandardScaler
```

[152]:
```
scaler = StandardScaler(inputCol='features',outputCol=␣
↪'scalefeatures',withMean=False,withStd=True)
```

[153]:
```
scaled_data = scaler.fit(final_data).transform(final_data)
```

[ ]:
```
scaled_data.show(4,truncate=False)
```

MinMaxScaler Scaling Technique

[155]:
```
from pyspark.ml.feature import MinMaxScaler
```

[156]:
```
scaler = MinMaxScaler(inputCol="features",␣
↪outputCol="scaleFeatures_minmax",min=0,max=1)
```

[157]:
```
# Compute summary statistics and generate MinMaxScalerModel
minmaxscaler = scaler.fit(final_data).transform(final_data)
```

[ ]:
```
print("Features scaled to range: [%f, %f]" % (scaler.getMin(), scaler.getMax()))
minmaxscaler.select("features", "scaleFeatures_minmax").show(4,truncate=False)
```

Differnce between the above techniques:

- So what basically Normalizer does is normalizing each Vector to have unit norm.
- while StanderdScaler does is normalizing each feature to have unit standard deviation and zero mean.
- MinMaxScaler, rescales each feature to a specific range (often [0, 1]).

0.0.2 Question 2: CLUSTER THE DATA using the PySpark built-in K-means and bisecting K-means clustering algorithm (this is provided in the Spark ML Library).

a) Cluster the given dataSet using atmost 3-4 attributes to avoid curse of dimensionality. You can select the attribute based on the preprocessing

[159]:
```
from pyspark.ml.clustering import KMeans, BisectingKMeans
from pyspark.ml.evaluation import ClusteringEvaluator
```

For k=3:

[160]:
```
kmeans = KMeans(featuresCol='scalefeatures', k=3)
```

[161]:
```
model = kmeans.fit(scaled_data)
```

[162]:
```
pred = model.transform(scaled_data)
```

Sum of square error for Analyzing results

[163]:
```
# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Cluster Centers:
[1.77716172 2.92780981 0.40808429 3.29453656]
[2.09528486 2.34465262 6.40112111 1.3075606 ]
[1.86283105 2.68321855 0.39278773 1.39988903]
```

[ ]:
```

```

Silhouette co-efficient for Analyzing results

[164]:
```
evaluator = ClusteringEvaluator()
```

[165]:
```
silhouette = evaluator.evaluate(pred)
print("Silhouette with squared euclidean distance = " + str(silhouette))

# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Silhouette with squared euclidean distance = 0.7510103639752713
Cluster Centers:
[1.77716172 2.92780981 0.40808429 3.29453656]
[2.09528486 2.34465262 6.40112111 1.3075606 ]
[1.86283105 2.68321855 0.39278773 1.39988903]
```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

For k=5:

[166]:
```
kmeans = KMeans(featuresCol='scalefeatures', k=5)
```

[167]:
```
model = kmeans.fit(scaled_data)
```

[168]:
```
pred = model.transform(scaled_data)
```

[169]:
```
evaluator = ClusteringEvaluator()
```

[170]:
```
silhouette = evaluator.evaluate(pred)
print("Silhouette with squared euclidean distance = " + str(silhouette))

# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Silhouette with squared euclidean distance = -0.07439061799442737
Cluster Centers:
[2.75441729 1.85033414 0.41950022 2.241373  ]
[1.25135068 1.82882904 1.12919777 1.20825789]
[1.04720151 3.05337716 0.4142771  3.32149395]
[1.21550655 3.49982294 0.33938871 1.37594697]
[2.95047717 3.61966445 0.28307878 2.04856736]
```

For k=4:

[171]:
```
kmeans = KMeans(featuresCol='scaleFeatures_minmax', k=4)
```

[172]:
```
model = kmeans.fit(minmaxscaler)
```

[173]:
```
model.clusterCenters()
```

[173]:
```
[array([0.66222222, 0.16166667, 0.05405556, 0.80893617]),
 array([0.42457181, 0.23715415, 0.08023715, 0.26784963]),
 array([0.75931099, 0.78910615, 0.04115456, 0.43753714]),
 array([0.29957806, 0.81487342, 0.04382911, 0.76191759])]
```

[174]:
```
pred = model.transform(minmaxscaler).show()
```

```
+---+------------------+--------------------+----------+
| ID|          features|scaleFeatures_minmax|prediction|
+---+------------------+--------------------+----------+
| 11| [7.0,3.0,4.0,36.0]|[0.58333333333333…|         0|
| 36| [7.0,3.0,0.0,13.0]|[0.58333333333333…|         1|
|  3| [7.0,4.0,2.0,51.0]|[0.58333333333333…|         0|
|  7|  [7.0,5.0,4.0,5.0]|[0.58333333333333…|         2|
| 11| [7.0,5.0,2.0,36.0]|[0.58333333333333…|         2|
|  3| [7.0,6.0,2.0,51.0]|[0.58333333333333…|         3|
| 10| [7.0,6.0,8.0,52.0]|[0.58333333333333…|         3|
| 20| [7.0,6.0,4.0,50.0]|[0.58333333333333…|         3|
| 14|[7.0,2.0,40.0,12.0]|[0.58333333333333…|         1|
```

```
|  1| [7.0,2.0,8.0,11.0]|[0.58333333333333…|         1|
| 20| [7.0,2.0,8.0,50.0]|[0.58333333333333…|         0|
| 20| [7.0,3.0,8.0,50.0]|[0.58333333333333…|         0|
| 20| [7.0,4.0,8.0,50.0]|[0.58333333333333…|         0|
|  3| [7.0,4.0,1.0,51.0]|[0.58333333333333…|         0|
|  3| [7.0,4.0,4.0,51.0]|[0.58333333333333…|         0|
| 24| [7.0,6.0,8.0,25.0]|[0.58333333333333…|         2|
|  3| [7.0,6.0,2.0,51.0]|[0.58333333333333…|         3|
|  3| [7.0,2.0,8.0,51.0]|[0.58333333333333…|         0|
|  6| [7.0,5.0,8.0,29.0]|[0.58333333333333…|         2|
| 33| [8.0,4.0,2.0,25.0]|[0.66666666666666…|         2|
+---+-------------------+--------------------+----------+
only showing top 20 rows
```

For k=4:

[175]:
```
kmeans = KMeans(featuresCol='scaleFeatures_minmax', k=4)
```

[176]:
```
model = kmeans.fit(minmaxscaler)
```

[177]:
```
model.clusterCenters()
```

[177]:
```
[array([0.66222222, 0.16166667, 0.05405556, 0.80893617]),
 array([0.42457181, 0.23715415, 0.08023715, 0.26784963]),
 array([0.75931099, 0.78910615, 0.04115456, 0.43753714]),
 array([0.29957806, 0.81487342, 0.04382911, 0.76191759])]
```

[178]:
```
pred = model.transform(minmaxscaler).show()
```

```
+---+-------------------+--------------------+----------+
| ID|           features|scaleFeatures_minmax|prediction|
+---+-------------------+--------------------+----------+
| 11| [7.0,3.0,4.0,36.0]|[0.58333333333333…|         0|
| 36| [7.0,3.0,0.0,13.0]|[0.58333333333333…|         1|
|  3| [7.0,4.0,2.0,51.0]|[0.58333333333333…|         0|
|  7|  [7.0,5.0,4.0,5.0]|[0.58333333333333…|         2|
| 11| [7.0,5.0,2.0,36.0]|[0.58333333333333…|         2|
|  3| [7.0,6.0,2.0,51.0]|[0.58333333333333…|         3|
| 10| [7.0,6.0,8.0,52.0]|[0.58333333333333…|         3|
| 20| [7.0,6.0,4.0,50.0]|[0.58333333333333…|         3|
| 14|[7.0,2.0,40.0,12.0]|[0.58333333333333…|         1|
|  1| [7.0,2.0,8.0,11.0]|[0.58333333333333…|         1|
| 20| [7.0,2.0,8.0,50.0]|[0.58333333333333…|         0|
| 20| [7.0,3.0,8.0,50.0]|[0.58333333333333…|         0|
| 20| [7.0,4.0,8.0,50.0]|[0.58333333333333…|         0|
|  3| [7.0,4.0,1.0,51.0]|[0.58333333333333…|         0|
|  3| [7.0,4.0,4.0,51.0]|[0.58333333333333…|         0|
```

```
| 24| [7.0,6.0,8.0,25.0]|[0.58333333333333…|         2|
|  3| [7.0,6.0,2.0,51.0]|[0.58333333333333…|         3|
|  3| [7.0,2.0,8.0,51.0]|[0.58333333333333…|         0|
|  6| [7.0,5.0,8.0,29.0]|[0.58333333333333…|         2|
| 33| [8.0,4.0,2.0,25.0]|[0.66666666666666…|         2|
+---+-------------------+--------------------+----------+
only showing top 20 rows
```

For k=6:

[179]:
```
kmeans = KMeans(featuresCol='scalefeatures', k=6)
```

[180]:
```
model = kmeans.fit(scaled_data)
```

[181]:
```
model.clusterCenters()
```

[181]:
```
[array([2.82466203, 1.85897458, 0.39173528, 1.75026112]),
 array([1.20622277, 1.80946017, 1.19749234, 1.22150346]),
 array([2.04908299, 3.85054807, 0.37815902, 3.37278093]),
 array([1.02519839, 3.47560271, 0.33976539, 1.51583991]),
 array([1.38299887, 2.11688637, 0.41864475, 3.35652154]),
 array([2.79948529, 3.52279212, 0.30687193, 1.46497588])]
```

[182]:
```
pred = model.transform(scaled_data).show(10)
```

```
+---+-------------------+--------------------+----------+
| ID|           features|       scalefeatures|prediction|
+---+-------------------+--------------------+----------+
| 11| [7.0,3.0,4.0,36.0]|[2.03708250757275…|         0|
| 36| [7.0,3.0,0.0,13.0]|[2.03708250757275…|         0|
|  3| [7.0,4.0,2.0,51.0]|[2.03708250757275…|         4|
|  7|  [7.0,5.0,4.0,5.0]|[2.03708250757275…|         5|
| 11| [7.0,5.0,2.0,36.0]|[2.03708250757275…|         2|
|  3| [7.0,6.0,2.0,51.0]|[2.03708250757275…|         2|
| 10| [7.0,6.0,8.0,52.0]|[2.03708250757275…|         2|
| 20| [7.0,6.0,4.0,50.0]|[2.03708250757275…|         2|
| 14|[7.0,2.0,40.0,12.0]|[2.03708250757275…|         1|
|  1| [7.0,2.0,8.0,11.0]|[2.03708250757275…|         1|
+---+-------------------+--------------------+----------+
only showing top 10 rows
```

Using Bisecting Kmeans

For K = 6

[183]:
```
bisectkmean = BisectingKMeans(featuresCol='scaleFeatures_minmax',k=6)
```

[184]:
```
model = bisectkmean.fit(minmaxscaler)
```

[190]:
```
pred = model.transform(minmaxscaler)
```

[195]:
```
silhouette = evaluator.evaluate(pred)
print("\nSilhouette with squared euclidean distance = " + str(silhouette))

# Shows the result.
centers = model.clusterCenters()
print("\nCluster Centers: ")
for center in centers:
    print(center,'\n')

pred.show(20)
```

```
Silhouette with squared euclidean distance = -0.15614721296620132

Cluster Centers:
[0.31929348 0.23913043 0.07318841 0.28029602]

[0.22777778 0.11666667 0.07916667 0.92765957]

[0.82512626 0.24431818 0.07335859 0.33687943]

[0.70512821 0.10384615 0.0524359  0.901473  ]

[0.27666667 0.80428571 0.04466667 0.6312462 ]

[0.78733766 0.82467532 0.03863636 0.61605416]

+---+------------------+--------------------+----------+
| ID|          features|scaleFeatures_minmax|prediction|
+---+------------------+--------------------+----------+
| 11| [7.0,3.0,4.0,36.0]|[0.58333333333333…|         3|
| 36| [7.0,3.0,0.0,13.0]|[0.58333333333333…|         0|
|  3| [7.0,4.0,2.0,51.0]|[0.58333333333333…|         5|
|  7|  [7.0,5.0,4.0,5.0]|[0.58333333333333…|         5|
| 11| [7.0,5.0,2.0,36.0]|[0.58333333333333…|         5|
|  3| [7.0,6.0,2.0,51.0]|[0.58333333333333…|         5|
| 10| [7.0,6.0,8.0,52.0]|[0.58333333333333…|         5|
| 20| [7.0,6.0,4.0,50.0]|[0.58333333333333…|         5|
| 14|[7.0,2.0,40.0,12.0]|[0.58333333333333…|         0|
|  1| [7.0,2.0,8.0,11.0]|[0.58333333333333…|         0|
| 20| [7.0,2.0,8.0,50.0]|[0.58333333333333…|         3|
| 20| [7.0,3.0,8.0,50.0]|[0.58333333333333…|         3|
| 20| [7.0,4.0,8.0,50.0]|[0.58333333333333…|         5|
```

```
|  3| [7.0,4.0,1.0,51.0]|[0.58333333333333…|         5|
|  3| [7.0,4.0,4.0,51.0]|[0.58333333333333…|         5|
| 24| [7.0,6.0,8.0,25.0]|[0.58333333333333…|         5|
|  3| [7.0,6.0,2.0,51.0]|[0.58333333333333…|         5|
|  3| [7.0,2.0,8.0,51.0]|[0.58333333333333…|         3|
|  6| [7.0,5.0,8.0,29.0]|[0.58333333333333…|         5|
| 33| [8.0,4.0,2.0,25.0]|[0.66666666666666…|         2|
+---+-------------------+--------------------+----------+
only showing top 20 rows
```

[ ]: