databricks ds_checkpoint_4

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
import csv
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler

# Loads data.
dataset = spark.read.format("libsvm").load("/FileStore/tables/k_means_5.txt")

# Trains a k-means model.
kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(dataset)

# Make predictions
predictions = model.transform(dataset)

# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(predictions)
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 = 0.811722808257
Cluster Centers:
[ 2.8217591   1.73470971]
[ 1.36851089  0.85568353]
```

```
predictions.show()
```

```
+-------+--------------------+----------+
|  label|            features|prediction|
+-------+--------------------+----------+
|13788.0|(2,[0,1],[11.56,6...|         0|
| 7015.0|(2,[0,1],[8.29,5.2])|         0|
| 9758.0|(2,[0,1],[8.2,5.15])|         0|
| 8658.0|(2,[0,1],[8.64,4....|         0|
| 5193.0|(2,[0,1],[6.92,4....|         0|
|13095.0|(2,[0,1],[7.54,4....|         0|
```

```
|20941.0|(2,[0,1],[6.38,4....|         0|
|30799.0|(2,[0,1],[7.0,4.19])|         0|
|29033.0|(2,[0,1],[6.0,3.97])|         0|
|13082.0|(2,[0,1],[6.13,3....|         0|
|22613.0|(2,[0,1],[6.25,3....|         0|
|27646.0|(2,[0,1],[6.0,3.84])|         0|
| 3950.0|(2,[0,1],[5.75,3.8])|         0|
|17275.0|(2,[0,1],[5.75,3....|         0|
|  422.0|(2,[0,1],[5.57,3....|         0|
|22436.0|(2,[0,1],[6.0,3.63])|         0|
|  858.0|(2,[0,1],[5.33,3....|         0|
| 4807.0|(2,[0,1],[6.06,3....|         0|
|27778.0|(2,[0,1],[6.0,3.59])|         0|
|24403.0|(2,[0,1],[5.5,3.59])|         0|
+-------+--------------------+----------+
only showing top 20 rows


predictions = predictions.drop("features")


display(predictions)
```

label	prediction
13788	0
7015	0
9758	0
8658	0
5193	0
13095	0
20941	0
30799	0
29033	0

Showing the first 1000 rows.

```
officers_df = spark.read.format('csv').options(header='true',
inferSchema='true').load('/FileStore/tables/Officer_attributes.csv')
#display(officers_df)
officers_df.printSchema()
```

```
root
 |-- id: integer (nullable = true)
 |-- gender: string (nullable = true)
 |-- rank: string (nullable = true)
 |-- race: string (nullable = true)
 |-- tenure: integer (nullable = true)
 |-- age: integer (nullable = true)
 |-- number_of_awards: integer (nullable = true)
 |-- District_Number: integer (nullable = true)
```

```
type(predictions)

Out[25]: pyspark.sql.dataframe.DataFrame

joined_df = predictions.join(officers_df, predictions.label == officers_df.id,
'inner')
```

```
#joined_df = joined_df.join(dataset, dataset.)
display(joined_df)
#joined_df.printSchema()
```

label	prediction	id	gender	rank	race	tenure	age
1	1	1	M	Sergeant	White	13	47
2	1	2	F	Police Officer	Hispanic	13	38
5	1	5	M	Sergeant	White	64	88
6	1	6	M	Police Officer	White	24	50
7	1	7	M	Police Officer	White	23	46
15	1	15	M	Police Officer	White	6	35
16	0	16	M	Police Officer	Asian/Pacific	10	34
17	0	17	M	Police Officer	Black	29	60
18	1	18	M	Police Officer	Asian/Pacific	19	42

Showing the first 1000 rows.

```
officer_norm_df = spark.read.format('csv').options(header='true',
inferSchema='true').load('/FileStore/tables/officer_allegations_both.csv')
officer_norm_df = officer_norm_df.withColumnRenamed("CPDB Officer ID",
"officer_id")
officer_norm_df = officer_norm_df.withColumnRenamed("Normalized Self", "norm")
display(officer_norm_df)
```

officer_id	avg_allegation_count
13788	11.56
7015	8.29
9758	8.2
8658	8.64
5193	6.92
13095	7.54
20941	6.38
30799	7
29033	6

Showing the first 1000 rows.

```
joined_df = joined_df.join(officer_norm_df, joined_df.label ==
officer_norm_df.officer_id, 'inner')
```

```
joined_df = joined_df.withColumnRenamed("prediction", "cnt")
joined_df.drop('ID').collect()
display(joined_df)
#joined_df.printSchema()
```

label	cnt	id	gender	rank	race	tenure	age	number_of_a
1	1	1	M	Sergeant	White	13	47	88
2	1	2	F	Police Officer	Hispanic	13	38	100
5	1	5	M	Sergeant	White	64	88	5

6	1	6	M	Police Officer	White	24	50	2
7	1	7	M	Police	White	23	46	25

Showing the first 1000 rows.

```
joined_df.printSchema()
```

```
root
 |-- label: double (nullable = true)
 |-- cnt: integer (nullable = false)
 |-- id: integer (nullable = true)
 |-- gender: string (nullable = true)
 |-- rank: string (nullable = true)
 |-- race: string (nullable = true)
 |-- tenure: integer (nullable = true)
 |-- age: integer (nullable = true)
 |-- number_of_awards: integer (nullable = true)
 |-- District_Number: integer (nullable = true)
 |-- officer_id: integer (nullable = true)
 |-- avg_allegation_count: double (nullable = true)
 |-- norm: double (nullable = true)
```

```
data = spark.read.format('csv').options(header='true',
inferSchema='true').load('/FileStore/tables/d4_dataframe_values_1.csv')
display(data)
```

id	tenure	age	number_of_awards	District_Number	officer_id	avg_allegatic
1	13	47	88	14	1	1
2	13	38	100	15	2	2
5	64	88	5	25	5	2
6	24	50	2	20	6	1.2
7	23	46	25	16	7	1.75
15	6	35	32	6	15	1
16	10	34	48	1	16	3.8
17	29	60	6	19	17	3.57
18	19	42	43	19	18	1.67

Showing the first 1000 rows.

```
from pyspark.ml.feature import StringIndexer, VectorIndexer
from pyspark.sql.functions import col


labelIndexer = StringIndexer(inputCol="cnt",
outputCol="indexedLabel").fit(data)

data = data.select([col(c).cast("double").alias(c) for c in data.columns])
train, test = data.randomSplit([0.7, 0.3])

#featurecols = data.columns
#featurecols.remove('cnt')
#featurecols.remove('id')

#assembler = VectorAssembler(inputCols=featurecols, outputCol="features")
#output = assembler.transform(data)
#output.drop("cnt", "id", "gender", "rank", "race", "tenure", "age",
"number_of_awards", "District_Number")
data.printSchema()
```

```
root
 |-- id: double (nullable = true)
 |-- tenure: double (nullable = true)
 |-- age: double (nullable = true)
 |-- number_of_awards: double (nullable = true)
 |-- District_Number: double (nullable = true)
 |-- officer_id: double (nullable = true)
 |-- avg_allegation_count: double (nullable = true)
 |-- norm: double (nullable = true)
 |-- gender: double (nullable = true)
 |-- rank: double (nullable = true)
 |-- race: double (nullable = true)
 |-- cnt: double (nullable = true)
```

```
display(data)
```

id	tenure	age	number_of_awards	District_Number	officer_id	avg_allegatic
1	13	47	88	14	1	1
2	13	38	100	15	2	2
5	64	88	5	25	5	2

6	24	50	2	20	6	1.2
7	23	46	25	16	7	1.75
15	6	35	32	6	15	1
16	10	34	48	1	16	3.8

Showing the first 1000 rows.

```
from pyspark.ml.feature import VectorAssembler, VectorIndexer
featuresCols = data.columns
featuresCols.remove('cnt')

# This concatenates all feature columns into a single feature vector in a new
column "rawFeatures".
vectorAssembler = VectorAssembler(inputCols=featuresCols,
outputCol="rawFeatures")
# This identifies categorical features and indexes them.
vectorIndexer = VectorIndexer(inputCol="rawFeatures", outputCol="features",
maxCategories=10)
```

```
from pyspark.ml.regression import GBTRegressor
# Takes the "features" column and learns to predict "cnt"
gbt = GBTRegressor(labelCol="cnt")
```

```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import RegressionEvaluator
# Define a grid of hyperparameters to test:
#  - maxDepth: max depth of each decision tree in the GBT ensemble
#  - maxIter: iterations, i.e., number of trees in each GBT ensemble
# In this example notebook, we keep these values small.  In practice, to get the highest accuracy, you would likely want to try deeper trees (10 or higher) and more trees in the ensemble (>100).
paramGrid = ParamGridBuilder()\
  .addGrid(gbt.maxDepth, [2, 5])\
  .addGrid(gbt.maxIter, [10, 100])\
  .build()
# We define an evaluation metric.  This tells CrossValidator how well we are doing by comparing the true labels with predictions.
evaluator = RegressionEvaluator(metricName="rmse", labelCol=gbt.getLabelCol(), predictionCol=gbt.getPredictionCol())
# Declare the CrossValidator, which runs model tuning for us.
cv = CrossValidator(estimator=gbt, evaluator=evaluator, estimatorParamMaps=paramGrid)
```

```
from pyspark.ml import Pipeline
pipeline = Pipeline(stages=[vectorAssembler, vectorIndexer, cv])

data.printSchema()
```

```
root
 |-- id: double (nullable = true)
 |-- tenure: double (nullable = true)
 |-- age: double (nullable = true)
 |-- number_of_awards: double (nullable = true)
 |-- District_Number: double (nullable = true)
 |-- officer_id: double (nullable = true)
 |-- avg_allegation_count: double (nullable = true)
 |-- norm: double (nullable = true)
 |-- gender: double (nullable = true)
 |-- rank: double (nullable = true)
 |-- race: double (nullable = true)
 |-- cnt: double (nullable = true)
```

```
display(train)
```

id	tenure	age	number_of_awards	District_Number	officer_id	avg_allegatic
2	13	38	100	15	2	2

6	24	50	2	20	6	1.2
7	23	46	25	16	7	1.75
15	6	35	32	6	15	1
17	29	60	6	19	17	3.57
25	4	32	5	11	25	1
34	20	45	19	11	34	1.63

Showing the first 1000 rows.

```
train.cache()
#pipeline.cache()

pipelineModel = pipeline.fit(train)
predictions2 = pipelineModel.transform(test)
```

```
display(predictions2.select("cnt", "prediction", *featuresCols))
```

cnt	prediction	id	tenure	age	number_of_awards	District_Numl
1	0.9997861346294848	1	13	47	88	14
1	1.0400010254663474	5	64	88	5	25
0	-0.004269823179941717	16	10	34	48	1
1	0.9979148327238196	18	19	42	43	19
1	0.9975451639902313	19	19	69	14	17
1	0.9972960007874377	41	14	39	25	24
1	1.0131868029975208	42	17	51	177	24
1	0.9978426084872489	47	17	44	32	19
1	0.9737406947159603	50	18	55	13	17

Showing the first 1000 rows.

```
sum = pipelineModel.stages[-1]
```

```
sum.bestModel.featureImportances
```

```
Out[44]: SparseVector(11, {0: 0.0425, 1: 0.0582, 2: 0.0393, 3: 0.0988, 4: 0.007
3, 6: 0.4296, 7: 0.2971, 8: 0.0008, 9: 0.0263})
```

```
Out[46]: 'GBTRegressionModel (uid=GBTRegressor_4bf3b2a3656aff5d0b10) with 100
 trees\n  Tree 0 (weight 1.0):\n    If (feature 6 <= 2.025)\n     Predict: 1.0
\n    Else (feature 6 > 2.025)\n     If (feature 7 <= 1.2149999999999999)\n
   Predict: 0.8571428571428571\n     Else (feature 7 > 1.2149999999999999)\n
    Predict: 0.018495684340320593\n  Tree 1 (weight 0.1):\n    If (feature 6 <
= 2.145)\n     If (feature 6 <= 2.025)\n      Predict: 0.0\n     Else (feature
 6 > 2.025)\n      Predict: 0.4216632575172056\n    Else (feature 6 > 2.145)\n
     If (feature 7 <= 1.2149999999999999)\n      Predict: -1.3142857142857143
\n     Else (feature 7 > 1.2149999999999999)\n      Predict: -0.03699136868064
133\n  Tree 2 (weight 0.1):\n    If (feature 6 <= 2.145)\n     If (feature 6 <
= 2.025)\n      Predict: 0.0\n     Else (feature 6 > 2.025)\n      Predict: 0.
33733060601376275\n    Else (feature 6 > 2.145)\n     If (feature 7 <= 1.21499
99999999999)\n      Predict: -1.0514285714285712\n     Else (feature 7 > 1.214
9999999999999)\n      Predict: -0.029593094944512902\n  Tree 3 (weight 0.1):\n
    If (feature 7 <= 1.335)\n     If (feature 6 <= 2.025)\n      Predict: 0.0
\n     Else (feature 6 > 2.025)\n      Predict: 0.19765408673836177\n    Else
 (feature 7 > 1.335)\n     If (feature 6 <= 2.145)\n      Predict: -0.07242892
216727632\n     Else (feature 6 > 2.145)\n      Predict: -0.02367447595561042
\n  Tree 4 (weight 0.1):\n    If (feature 6 <= 2.145)\n     If (feature 6 <=
 2.025)\n      Predict: 1.7656801921375696E-4\n     Else (feature 6 > 2.025)\n
      Predict: 0.25274871990312675\n    Else (feature 6 > 2.145)\n     If (fea
```