Machine Learning - Citi Bank

April 30, 2021

1 Machine Learning - Citi Bank

I will be using this data set to illustrate different predictive analytic methods. The point of this analysis will be to showcase the use of pyspark pipelines and various classification modes and some of thier outputs

This will be important or helpful for any predictive analysis you will have to do. The pipe lets you work with big data easier and pyspark is significantly faster when it comes to processing speed

[1]:
```
#library used
from pyspark.ml.regression import GeneralizedLinearRegression
from pyspark.ml.classification import LogisticRegression
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import OneHotEncoder
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StandardScaler
from pyspark.ml import Pipeline
import pandas as pd
import numpy as np
import sklearn
import pandas as pd
from pyspark.ml.linalg import Vectors
from pyspark.ml.stat import Correlation
from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.feature import StringIndexer, VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.sql.functions import translate
from pyspark.sql.types import IntegerType
from pyspark.sql.functions import rand,when
from pyspark.ml.feature import VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.regression import GBTRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator
import pyspark.sql.functions as func
```

```python
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.classification import DecisionTreeClassifier
import pyspark.ml.evaluation as ev
import matplotlib.pyplot as plt
```

[2]:
```python
#connecting spark and calling the data
spark = SparkSession.builder.master('local[2]').config("spark.executor.
 ↪memory","1g").config("spark.drive.memory","1g").appName('spark_sh_data').
 ↪getOrCreate()
df = spark.read.options(header=None, nullValue='NULL', inferSchema=True).
 ↪option('nullValue','null').csv('bikeClean2.csv')
df.head()
```

[2]:
```
Row(_c0='tripduration', _c1='starttime', _c2='stoptime', _c3='start station id',
_c4='start station name', _c5='start station latitude', _c6='start station
longitude', _c7='end station id', _c8='end station name', _c9='end station
latitude', _c10='end station longitude', _c11='bikeid', _c12='usertype',
_c13='birth year', _c14='gender', _c15='subscriber', _c16='day_of_week',
_c17='weekend', _c18='duration_minutes')
```

[3]:
```python
#drop not needed columns
dfnew = df.drop(*['_c4', '_c8', '_c12'])
```

[4]:
```python
#change columns to int so can perform predicitons and easier for pipe
newDf1 = dfnew.withColumn('_c0', df["_c0"].cast(IntegerType())).
 ↪withColumn('_c1', df["_c1"].cast(IntegerType())).withColumn('_c2', df["_c2"].
 ↪cast(IntegerType()))\
        .withColumn('_c3', df["_c3"].cast(IntegerType())).withColumn('_c5',␣
 ↪df["_c5"].cast(IntegerType()))\
        .withColumn('_c6', df["_c6"].cast(IntegerType())).withColumn('_c7',␣
 ↪df["_c7"].cast(IntegerType()))\
        .withColumn('_c9', df["_c9"].cast(IntegerType())).withColumn('_c10',␣
 ↪df["_c10"].cast(IntegerType()))\
        .withColumn('_c11', df["_c11"].cast(IntegerType())).withColumn('_c13',␣
 ↪df["_c13"].cast(IntegerType()))\
        .withColumn('_c14', df["_c14"].cast(IntegerType())).withColumn('_c15',␣
 ↪df["_c15"].cast(IntegerType()))\
        .withColumn('_c17', df["_c17"].cast(IntegerType())).withColumn('_c18',␣
 ↪df["_c18"].cast(IntegerType()))
```

[5]:
```python
#check schema
newDf1.printSchema()
```

```
root
 |-- _c0: integer (nullable = true)
 |-- _c1: integer (nullable = true)
```

```
 |-- _c2: integer (nullable = true)
 |-- _c3: integer (nullable = true)
 |-- _c5: integer (nullable = true)
 |-- _c6: integer (nullable = true)
 |-- _c7: integer (nullable = true)
 |-- _c9: integer (nullable = true)
 |-- _c10: integer (nullable = true)
 |-- _c11: integer (nullable = true)
 |-- _c13: integer (nullable = true)
 |-- _c14: integer (nullable = true)
 |-- _c15: integer (nullable = true)
 |-- _c16: string (nullable = true)
 |-- _c17: integer (nullable = true)
 |-- _c18: integer (nullable = true)
```

[6]:
```
newDf1.show()
```

```
+----+----+----+----+----+----+----+----+-----+----+----+----+-----------+-
---+----+
| _c0| _c1| _c2| _c3| _c5| _c6| _c7| _c9|_c10| _c11|_c13|_c14|_c15|
_c16|_c17|_c18|
+----+----+----+----+----+----+----+----+-----+----+----+----+-----------+-
---+----+
|null|null|null|null|null|null|null|null|null|
null|null|null|null|day_of_week|null|null|
| 897|   1|   1| 493|  40| -73| 454|  40| -73|18340|1966|   1|   2|     Monday|
0|  14|
| 267|   1|   1|3143|  40| -73|3226|  40| -73|21458|1996|   1|   1|     Monday|
0|   4|
|2201|   1|   1| 317|  40| -73|3469|  40| -73|39874|1986|   1|   2|     Monday|
0|  36|
|1660|   1|   1| 249|  40| -74| 369|  40| -74|38865|1988|   1|   2|     Monday|
0|  27|
| 109|   1|   1|3552|  40| -73|3538|  40| -73|30256|1997|   1|   2|     Monday|
0|   1|
| 106|   1|   1|3593|  40| -73|3592|  40| -73|16875|1988|   1|   2|     Monday|
0|   1|
| 550|   1|   1|3507|  40| -73|3553|  40| -73|34139|1992|   1|   2|     Monday|
0|   9|
| 338|   1|   1| 478|  40| -73| 388|  40| -74|39703|1995|   1|   2|     Monday|
0|   5|
| 469|   1|   1| 514|  40| -74| 458|  40| -74|28266|1989|   1|   2|     Monday|
0|   7|
| 562|   1|   1| 116|  40| -74| 462|  40| -74|26757|1965|   1|   2|     Monday|
0|   9|
|2420|   1|   1|3054|  40| -73|3054|  40| -73|28840|1986|   1|   2|     Monday|
0|  40|
```

```
|1058|    1|    1|3434|  40| -73|3434|  40| -73|21038|1984|    1|    2|     Monday|
0|  17|
| 257|    1|    1|3119|  40| -73|3118|  40| -73|25280|1988|    1|    2|     Monday|
0|   4|
| 305|    1|    1| 402|  40| -73| 461|  40| -73|34925|1965|    1|    2|     Monday|
0|   5|
| 366|    1|    1|3578|  40| -73|3581|  40| -73|38092|1988|    2|    2|     Monday|
0|   6|
|1006|    1|    1| 382|  40| -73| 545|  40| -73|17972|1969|    0|    1|     Monday|
0|  16|
| 936|    1|    1|3674|  40| -73|3795|  40| -73|31672|1965|    1|    2|     Monday|
0|  15|
| 584|    1|    1| 455|  40| -73|3711|  40| -73|38455|1989|    1|    2|     Monday|
0|   9|
|1147|    1|    1|3619|  40| -73|3118|  40| -73|16115|1994|    1|    2|     Monday|
0|  19|
+----+----+----+----+----+----+----+----+----+-----+----+----+----+-----------+-
---+----+
only showing top 20 rows
```

[7]:
```
#convert your string to int
stringIndexer = StringIndexer().setInputCol("_c16").setOutputCol("_c16_index").
 ↪setHandleInvalid("skip")
_16_index_model=stringIndexer.fit(newDf1)
_16_index_df=_16_index_model.transform(newDf1)
```

[8]:
```
#use the one hot encoder on that stringindexer
encoder = OneHotEncoder().setInputCols(["_c16_index"]).
 ↪setOutputCols(["_16_encoded"])
encoder_model=encoder.fit(_16_index_df)
encoder_df=encoder_model.transform(_16_index_df)
```

[9]:
```
#assemble the rest of columns
assembler = VectorAssembler().
 ↪setInputCols(['_c0','_c1','_c2','_c3','_c5','_c6','_c7','_c9',
                                                              ␣
 ↪'_c10','_c11','_c13','_c14','_c17','_c18'])\
                    .setOutputCol("vectorized_features")\
                        .setHandleInvalid("skip")

assembler_df = assembler.transform(encoder_df)
```

[11]:
```
#for your response variable
label_indexer = StringIndexer().setInputCol("_c15").setOutputCol("label")

label_indexer_model=label_indexer.fit(assembler_df)
```

```
label_indexer_df = label_indexer_model.transform(assembler_df)
```

[13]:
```
#always scale your model
scaler = StandardScaler().setInputCol("vectorized_features").
 ↪setOutputCol("features")
scaler_model=scaler.fit(label_indexer_df)
scaler_df = scaler_model.transform(label_indexer_df)
pd.set_option('display.max_colwidth', 100)
```

[14]:
```
#put it all in a pipe
pipeline_stages=Pipeline().
 ↪setStages([stringIndexer,encoder,assembler,label_indexer,scaler])

pipeline_model=pipeline_stages.fit(newDf1)
pipeline_df=pipeline_model.transform(newDf1)
```

[ ]:
```

```

2 Analysis

After you fit you pipe now you can begin your analysis. The pipe is also interesting because you could send that piped model to anyone you want with a pickle allowing them to access that large data set

[15]:
```
#train model
train, test = pipeline_df.randomSplit([0.8,0.2], seed=56)
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))
```

```
Training Dataset Count: 1745133
Test Dataset Count: 435877
```

[16]:
```
#logisticRegression
lr = LogisticRegression(featuresCol = 'features', labelCol = 'label',␣
 ↪maxIter=10)
lrModel = lr.fit(train)
lr_summary=lrModel.summary
```

These are all methods of evaluating the model

[17]:
```
lr_summary.areaUnderROC
```

[17]: 0.7093301779037383

[18]:
```
lr_summary.accuracy
```

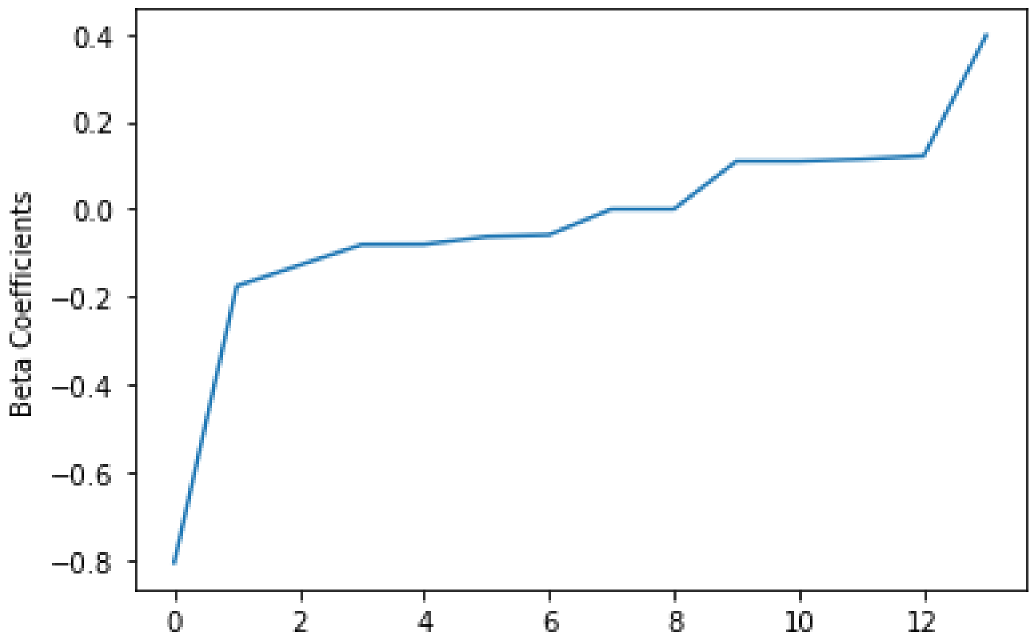
[18]: 0.8649357957244519

[19]:
```python
lr_summary.precisionByLabel
```

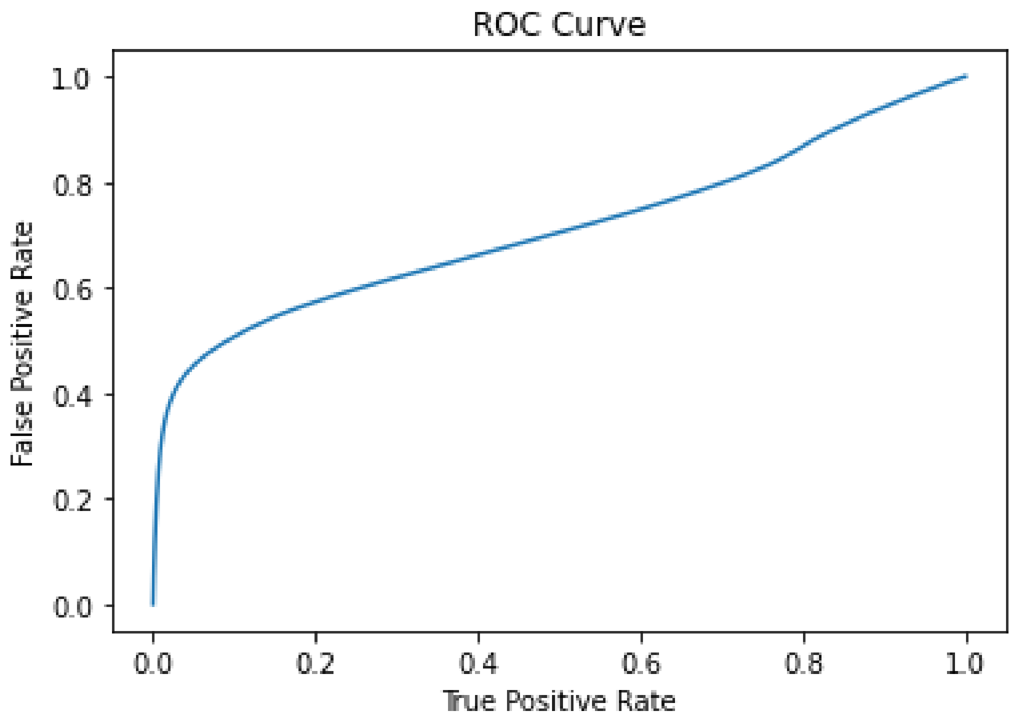
[19]: [0.8636634276746995, 0.8895521691986615]

These are visual representation to evaluate the model

[20]:
```python
beta = np.sort(lrModel.coefficients)
plt.plot(beta)
plt.ylabel('Beta Coefficients')
plt.show()
```



[21]:
```python
trainingSummary = lrModel.summary
roc = trainingSummary.roc.toPandas()
plt.plot(roc['FPR'],roc['TPR'])
plt.ylabel('False Positive Rate')
plt.xlabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
print('Training set areaUnderROC: ' + str(trainingSummary.areaUnderROC))
```




```
Training set areaUnderROC: 0.7093301779037383
```

Other ways to evalute accuracy of your model

[23]:
```python
#Evaluate our Logistic Regression model.
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()
predictions = lrModel.transform(test)
print('Test Area Under ROC', evaluator.evaluate(predictions))
```

```
Test Area Under ROC 0.70777726934561
```

[24]:
```python
#you can check your predictions
predictions.select('label','rawPrediction','probability','prediction').
 ↪toPandas().head(5)
```

[24]:
```
   label                                rawPrediction  \
0    0.0  [1.6449381313946918, -1.6449381313946918]
1    0.0  [1.8621994191595799, -1.8621994191595799]
2    0.0  [2.4427193225463792, -2.4427193225463792]
3    1.0  [-0.0151081397858579, 0.0151081397858579]
4    0.0  [1.6164169816927116, -1.6164169816927116]

                                 probability  prediction
```

```
0   [0.8382057503093875, 0.1617942496906125]        0.0
1   [0.8655531023455828, 0.1344468976544172]        0.0
2  [0.9200273960522601, 0.07997260394773986]        0.0
3   [0.4962230368961011, 0.5037769631038989]        1.0
4   [0.8343003948531139, 0.1656996051468861]        0.0
```

[25]:
```
from pyspark.ml.evaluation import RegressionEvaluator

lr_evaluator = RegressionEvaluator(predictionCol="prediction", \
                 labelCol='label',metricName="r2")
print("R Squared (R2) on test data = %g" % lr_evaluator.evaluate(predictions))
```

```
R Squared (R2) on test data = 0.0574956
```

[26]:
```
from sklearn.metrics import confusion_matrix

y_true = predictions.select("label")
y_true = y_true.toPandas()

y_pred = predictions.select("prediction")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred)
cnf_matrix
```

[26]:
```
array([[358059,   2351],
       [ 56462,  19005]])
```

3 TakeAway

By the end of this you should be able to

```
1) Follow the steps to pipe your model with pyspark


2) Be able to evaluate a model with logistic regression as well as create some models
```

[ ]:
```

```