Predicting Customer Churn

Yu chien (Calvin) Ma

Installing Dependencies and Initializing Spark

```
import findspark
findspark.init()
findspark.find()
```

```
'C:\\Users\\calvi\\anaconda3\\lib\\site-packages\\pyspark'
```

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
.appName('spark') \
.master('local[*]') \
.config('spark.sql.execution.arrow.pyspark.enabled', True) \
.config('spark.sql.session.timeZone', 'UTC') \
.config('spark.driver.memory','16G') \
.config('spark.ui.showConsoleProgress', True) \
.config('spark.sql.repl.eagerEval.enabled', True) \
.getOrCreate()

# Cloning repository
# !git clone
https://github.com/calvinma888/PySparkML_CustomerChurn.git

#path of dataset
path = r"C:\Users\calvi\Documents\Portfolio Projects\Predicting
Customer Churn\Churn_Modelling.csv"

#create a spark dataframe
df = spark.read.csv(path, header=True, inferSchema= True)

#display dataframe
df.show(5)
```

```
+---------+----------+--------+-----------+---------+------+---+------
+---------+-------------+---------+--------------+---------------
+------+
|RowNumber|CustomerId|  Surname|CreditScore|Geography|Gender|Age|
Tenure|  Balance|NumOfProducts|HasCrCard|IsActiveMember|
EstimatedSalary|Exited|
+---------+----------+--------+-----------+---------+------+---+------
+---------+-------------+---------+--------------+---------------
+------+
|        1|  15634602|Hargrave|        619|   France|Female| 42|
```

```
2|      0.0|            1|        1|             1|      101348.88|
1|
|        2|  15647311|    Hill|        608|    Spain|Female| 41|
1| 83807.86|            1|        0|             1|      112542.58|
0|
|        3|  15619304|    Onio|        502|   France|Female| 42|
8| 159660.8|            3|        1|             0|      113931.57|
1|
|        4|  15701354|    Boni|        699|   France|Female| 39|
1|      0.0|            2|        0|             0|       93826.63|
0|
|        5|  15737888|Mitchell|        850|    Spain|Female| 43|
2|125510.82|            1|        1|             1|        79084.1|
0|
+---------+----------+--------+-----------+---------+------+---+------
+---------+-------------+---------+--------------+---------------
+------+
only showing top 5 rows
```

```
#get the no.of rows & columns
print((df.count(), len(df.columns)))
```

```
(10000, 14)
```

```
#print schema
df.printSchema()
```

```
root
 |-- RowNumber: integer (nullable = true)
 |-- CustomerId: integer (nullable = true)
 |-- Surname: string (nullable = true)
 |-- CreditScore: integer (nullable = true)
 |-- Geography: string (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Tenure: integer (nullable = true)
 |-- Balance: double (nullable = true)
 |-- NumOfProducts: integer (nullable = true)
 |-- HasCrCard: integer (nullable = true)
 |-- IsActiveMember: integer (nullable = true)
 |-- EstimatedSalary: double (nullable = true)
 |-- Exited: integer (nullable = true)
```

```
#get the summary statistics
df.describe()
```

```
+-------+-----------------+-----------------+-------
+-----------------+---------+------+-----------------
+------------------+-----------------+------------------
```

```
+-------------------+-------------------+-----------------
+-------------------+
|summary|         RowNumber|       CustomerId|Surname|
CreditScore|Geography|Gender|               Age|           Tenure|
Balance|     NumOfProducts|          HasCrCard|     IsActiveMember|
EstimatedSalary|             Exited|
+-------+------------------+-----------------+-------
+-----------------+---------+------+-----------------
+------------------+-----------------+------------------
+-------------------+-------------------+-----------------
+-------------------+
|  count|              10000|            10000|  10000|
10000|    10000| 10000|             10000|             10000|
10000|              10000|               10000|              10000|
10000|              10000|
|   mean|             5000.5|  1.56909405694E7|   null|
650.5288|     null|  null|           38.9218|            5.0128|
76485.88928799961|             1.5302|             0.7055|
0.5151|100090.2398809998|             0.2037|
| stddev|2886.8956799071675|71936.18612274907|   null|
96.65329873613035|     null|  null|10.487806451704587|
2.8921743770496837|62397.40520238599|0.5816543579989917|
0.45584046447513327|0.49979692845891815|57510.49281769821|
0.40276858399486065|
|    min|                  1|         15565701|  Abazu|
350|   France|Female|                18|                 0|
0.0|                 1|                  0|                  0|
11.58|                  0|
|    max|              10000|         15815690| Zuyeva|
850|    Spain|  Male|                92|                10|
250898.09|                  4|                  1|                   1|
199992.48|                  1|
+-------+------------------+-----------------+-------
+-----------------+---------+------+-----------------
+------------------+-----------------+------------------
+-------------------+-------------------+-----------------
+-------------------+
```

Data Cleaning

```
# Checking the datatype of each column
df.dtypes
```

```
[('RowNumber', 'int'),
 ('CustomerId', 'int'),
 ('Surname', 'string'),
 ('CreditScore', 'int'),
 ('Geography', 'string'),
 ('Gender', 'string'),
```

```
 ('Age', 'int'),
 ('Tenure', 'int'),
 ('Balance', 'double'),
 ('NumOfProducts', 'int'),
 ('HasCrCard', 'int'),
 ('IsActiveMember', 'int'),
 ('EstimatedSalary', 'double'),
 ('Exited', 'int')]
```

```
# Separating categorical and numerical columns
categoricalColumns = [item[0] for item in df.dtypes if
item[1].startswith('string') ]
numericalColumns = list(set(df.columns)-set(categoricalColumns))

# We don't want the last item ("Exited") because that is the target
numericalColumns.remove("Exited")
```

```
categoricalColumns
```

```
['Surname', 'Geography', 'Gender']
```

```
numericalColumns
```

```
['CreditScore',
 'NumOfProducts',
 'Balance',
 'HasCrCard',
 'CustomerId',
 'Age',
 'IsActiveMember',
 'Tenure',
 'RowNumber',
 'EstimatedSalary']
```

```
# Converting categorical columns into indices
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder, StringIndexer,
VectorAssembler
from pyspark.ml.feature import MinMaxScaler

stages = []

# convert sting column to index column
indexer = StringIndexer(inputCols=categoricalColumns, outputCols=[x +
"Index" for x in categoricalColumns])

# one hot encode categorical columns to numerical vector columns
encoder = OneHotEncoder(inputCols=indexer.getOutputCols(), \
                        outputCols=[x + "_OHE" for x in
categoricalColumns])
```

```
# transform numerical columns to a single vector column

num_assembler = VectorAssembler(inputCols=numericalColumns,
outputCol='num_features',\
                                handleInvalid='keep')

# apply scaler Rescale each feature individually to a common range
[min, max] linearly using column summary statistics,
# which is also known as min-max normalization or rescaling.
scaler = MinMaxScaler(inputCol='num_features',
outputCol='scaled_num_features')

numericalScaled = ["scaled_num_features"]

assemblerInputs = [c + "_OHE" for c in categoricalColumns] +
numericalScaled
print(assemblerInputs)

# transform all create vector columns into one vector column
assembler = VectorAssembler(inputCols= assemblerInputs, \
                            outputCol='features')
```

```
['Surname_OHE', 'Geography_OHE', 'Gender_OHE', 'scaled_num_features']
```

```
# create a pipeline with above steps
data_pipeline = Pipeline(stages=[indexer,encoder, num_assembler,
scaler, assembler])

# fit pipeline and transform dataframe
dataset = data_pipeline.fit(df).transform(df)

# Now this dataset has the one-hot encoded categorical features and
the scaled numerical features
dataset.show(10)
```

```
+---------+----------+--------+-----------+---------+------+---+------
+---------+-------------+---------+--------------+---------------
+------+------------+--------------+-----------+------------------
+-------------+-------------+--------------------+--------------------
+--------------------+
|RowNumber|CustomerId| Surname|CreditScore|Geography|Gender|Age|
Tenure|  Balance|NumOfProducts|HasCrCard|IsActiveMember|
EstimatedSalary|Exited|SurnameIndex|GeographyIndex|GenderIndex|
Surname_OHE|Geography_OHE|   Gender_OHE|        num_features|
scaled_num_features|            features|
+---------+----------+--------+-----------+---------+------+---+------
+---------+-------------+---------+--------------+---------------
+------+------------+--------------+-----------+------------------
+-------------+-------------+--------------------+--------------------
+--------------------+
|        1|  15634602|Hargrave|        619|   France|Female| 42|
```

```
2|      0.0|            1|        1|             1|      101348.88|
1|      1958.0|           0.0|        1.0|(2931,[1958],[1.0])|(2,[0],
[1.0])|    (1,[],[])|[619.0,1.0,0.0,1....|[0.538,0.0,0.0,1....|(2944,
[1958,2931,...|
|        2|  15647311|    Hill|        608|    Spain|Female| 41|
1| 83807.86|            1|        0|             1|      112542.58|
0|        79.0|           2.0|        1.0|  (2931,[79],[1.0])|    (2,
[],[])|    (1,[],[])|[608.0,1.0,83807....|[0.516,0.0,0.3340...|(2944,
[79,2934,29...|
|        3|  15619304|    Onio|        502|   France|Female| 42|
8| 159660.8|            3|        1|             0|      113931.57|
1|       336.0|           0.0|        1.0| (2931,[336],[1.0])|(2,[0],
[1.0])|    (1,[],[])|[502.0,3.0,159660...|[0.304,0.66666666...|(2944,
[336,2931,2...|
|        4|  15701354|    Boni|        699|   France|Female| 39|
1|      0.0|            2|        0|             0|       93826.63|
0|       128.0|           0.0|        1.0| (2931,[128],[1.0])|(2,[0],
[1.0])|    (1,[],[])|[699.0,2.0,0.0,0....|[0.69800000000000...|(2944,
[128,2931,2...|
|        5|  15737888|Mitchell|        850|    Spain|Female| 43|
2|125510.82|            1|        1|             1|        79084.1|
0|        32.0|           2.0|        1.0|  (2931,[32],[1.0])|    (2,
[],[])|    (1,[],[])|[850.0,1.0,125510...|[1.0,0.0,0.500246...|(2944,
[32,2934,29...|
|        6|  15574012|     Chu|        645|    Spain|  Male| 44|
8|113755.78|            2|        1|             0|      149756.71|
1|        14.0|           2.0|        0.0|  (2931,[14],[1.0])|    (2,
[],[])|(1,[0],[1.0])|[645.0,2.0,113755...|[0.59,0.333333333...|(2944,
[14,2933,29...|
|        7|  15592531|Bartlett|        822|   France|  Male| 50|
7|      0.0|            2|        1|             1|        10062.8|
0|       631.0|           0.0|        0.0| (2931,[631],[1.0])|(2,[0],
[1.0])|(1,[0],[1.0])|[822.0,2.0,0.0,1....|[0.94400000000000...|(2944,
[631,2931,2...|
|        8|  15656148|  Obinna|        376|  Germany|Female| 29|
4|115046.74|            4|        1|             0|      119346.88|
1|      1269.0|           1.0|        1.0|(2931,[1269],[1.0])|(2,[1],
[1.0])|    (1,[],[])|[376.0,4.0,115046...|[0.05200000000000...|(2944,
[1269,2932,...|
|        9|  15792365|      He|        501|   France|  Male| 44|
4|142051.07|            2|        0|             1|        74940.5|
0|        57.0|           0.0|        0.0|  (2931,[57],[1.0])|(2,[0],
[1.0])|(1,[0],[1.0])|[501.0,2.0,142051...|[0.302,0.33333333...|(2944,
[57,2931,29...|
|       10|  15592389|      H?|        684|   France|  Male| 27|
2|134603.88|            1|        1|             1|       71725.73|
0|        44.0|           0.0|        0.0|  (2931,[44],[1.0])|(2,[0],
[1.0])|(1,[0],[1.0])|[684.0,1.0,134603...|[0.668,0.0,0.5364...|(2944,
[44,2931,29...|
```

```
+---------+----------+--------+-----------+---------+------+---+------
+---------+-------------+---------+--------------+---------------
+------+------------+--------------+-----------+-------------------
+-------------+-------------+--------------------+--------------------
+--------------------+
only showing top 10 rows


dataset.dtypes

[('RowNumber', 'int'),
 ('CustomerId', 'int'),
 ('Surname', 'string'),
 ('CreditScore', 'int'),
 ('Geography', 'string'),
 ('Gender', 'string'),
 ('Age', 'int'),
 ('Tenure', 'int'),
 ('Balance', 'double'),
 ('NumOfProducts', 'int'),
 ('HasCrCard', 'int'),
 ('IsActiveMember', 'int'),
 ('EstimatedSalary', 'double'),
 ('Exited', 'int'),
 ('SurnameIndex', 'double'),
 ('GeographyIndex', 'double'),
 ('GenderIndex', 'double'),
 ('Surname_OHE', 'vector'),
 ('Geography_OHE', 'vector'),
 ('Gender_OHE', 'vector'),
 ('num_features', 'vector'),
 ('scaled_num_features', 'vector'),
 ('features', 'vector')]
```

Creating the models

```
# split data into train, validation and test sets
train, validation_test = dataset.randomSplit([0.7, 0.3], seed = 100)
validation, test = validation_test.randomSplit([0.5, 0.5], seed = 100)

print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))

Training Dataset Count: 7033
Test Dataset Count: 1466

from pyspark.ml.classification import *
from time import *

# instantiate and train naive bayes
```

```
start = time()

nb = NaiveBayes(featuresCol='features', labelCol='Exited')
nb_model = nb.fit(train)

lr = LogisticRegression(featuresCol='features', labelCol='Exited')
lr_model = lr.fit(train)

svc = LinearSVC(featuresCol='features', labelCol='Exited')
svc_model = svc.fit(train)

rf = RandomForestClassifier(featuresCol='features', labelCol='Exited')
rf_model = rf.fit(train)

dt = DecisionTreeClassifier(featuresCol='features', labelCol='Exited')
dt_model = dt.fit(train)

gbt = GBTClassifier(featuresCol='features', labelCol='Exited')
gbt_model = gbt.fit(train)

end = time()

print ("Training took:",end-start, "seconds")
```

```
Training took: 28.099477767944336 seconds
```

```
from pyspark.ml.evaluation import *
# make baseline model predictions
# create lists with the models and their respective names
models = [nb_model,
          lr_model,
          svc_model,
          rf_model,
          dt_model,
          gbt_model]

model_names = ['naive bayes',
               'logistic regression',
               'linear svc',
               'random forest',
               'decision tree',
                'Gradient Boosting']

# for all models, make prediction, calculate f1 and area under the PR
scores and display results
for i in range(len(models)):
    model = models[i]
    model_name = model_names[i]

    # predict on validation set
    validation_prediction = model.transform(validation)
```

```
    # use MulticlassClassificationEvaluator to get f1 scores
    evaluator1 = MulticlassClassificationEvaluator(labelCol='Exited')

    # use BinaryClassificationEvaluator to get area under PR
    evaluator2 = BinaryClassificationEvaluator(
                 rawPredictionCol='prediction', labelCol='Exited')

    # make evaluation and print f1 and area under PR score per model
    print('')
    print('F1 score for {} on validation set: {}'.format(\
          (model_name), \
          (evaluator1.evaluate(validation_prediction,
{evaluator1.metricName:'f1'}))))

    print('Area under PR for {} on validation set:
{}'.format((model_name), \
          (evaluator2.evaluate(validation_prediction,
{evaluator2.metricName:'areaUnderPR'}))))
```

```
F1 score for naive bayes on validation set: 0.714324079804785
Area under PR for naive bayes on validation set: 0.09860093271152565

F1 score for logistic regression on validation set: 0.7429544816982064
Area under PR for logistic regression on validation set:
0.2882107955382399

F1 score for linear svc on validation set: 0.7322565865202919
Area under PR for linear svc on validation set: 0.27596826415365194

F1 score for random forest on validation set: 0.714982817564836
Area under PR for random forest on validation set: 0.1972018654230513

F1 score for decision tree on validation set: 0.8346652237983825
Area under PR for decision tree on validation set: 0.5304287785966667

F1 score for Gradient Boosting on validation set: 0.8466321651036282
Area under PR for Gradient Boosting on validation set:
0.5795530316500711
```

```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

# fit cv model and train it
start = time()

paramGrid_gbt = ParamGridBuilder() \
    .addGrid(gbt.maxDepth,[5, 10]) \
    .addGrid(gbt.maxIter,[5, 10]) \
    .build()
```

```
crossval = CrossValidator(estimator=gbt,
                          estimatorParamMaps=paramGrid_gbt,

evaluator=MulticlassClassificationEvaluator(labelCol='Exited',\
                          metricName='f1'),
                          numFolds=3)

# fit cv model and train it
start = time()
cvModel_gbt  = crossval.fit(train)
end = time()
print('Total training time for hyperparameter tuning on GBT model: {}
seconds'\
      .format(end-start))
```

```
Total training time for hyperparameter tuning on GBT model:
87.02806162834167 seconds
```

```
# predict on test set
results_final = cvModel_gbt.transform(test)
evaluator =
MulticlassClassificationEvaluator(predictionCol="prediction",labelCol=
'Exited')
print('Test Set Prediction Metrics:')
print('F-1 Score:{}'.format(evaluator.evaluate(results_final,
{evaluator.metricName: "f1"})))
print('Accuracy: {}'.format(evaluator.evaluate(results_final,
{evaluator.metricName: "accuracy"})))
print('')

# predict on validation set
results_final = cvModel_gbt.transform(validation)
evaluator =
MulticlassClassificationEvaluator(predictionCol="prediction",labelCol=
'Exited')
print('Validation Set Prediction Metrics:')
print('F-1 Score:{}'.format(evaluator.evaluate(results_final,
{evaluator.metricName: "f1"})))
print('Accuracy: {}'.format(evaluator.evaluate(results_final,
{evaluator.metricName: "accuracy"})))
```

```
Test Set Prediction Metrics:
F-1 Score:0.8371176844934723
Accuracy: 0.8485675306957708

Validation Set Prediction Metrics:
F-1 Score:0.8419843752867925
Accuracy: 0.854763491005996
```