databricks Memudu_Sadia_BigData_Project2

(https://databricks.com)

MEMUDU Alimatou Sadia

Big Data Project

Dataset Description

Dataset: https://www.kaggle.com/datasets/shelvigarg/credit-card-buyers

This dataset is a zip file composed of 2 csv files, test data credit card.csv and train data credit card.csv however only the train data credit card.csv will be used for this project.

Happy Customer Bank is a mid-sized private bank that deals in all kinds of banking products, like Savings accounts, Current accounts, investment products, credit products, among other offerings. The bank also cross-sells products to its existing customers and to do so they use different kinds of communication like tele-calling, e-mails, recommendations on net banking, mobile banking, etc. In this case, the Happy Customer Bank wants to cross sell its credit cards to its existing customers. The bank has identified a set of customers that are eligible for taking these credit cards. Now, the bank is looking for your help in identifying customers that could show higher intent towards a recommended credit card.

Data Variables

ID: Unique Identifier for a row

Gender: Gender of the Customer

Age: Age of the Customer (in Years)

Region_Code: Code of the Region for the customers

Occupation: Occupation Type for the customer

Channel_Code: Acquisition Channel Code for the Customer (Encoded)

Vintage: Vintage for the Customer (In Months)

Credit_Product: If the Customer has any active credit product (Home loan,Personal loan, Credit Card etc.)

AvgAccountBalance: Average Account Balance for the Customer in last 12 Months

Is_Active: If the Customer is Active in last 3 Months

Is_Lead(Target): If the Customer is interested for the Credit Card

0 : Customer is not interested

1 : Customer is interested

Goal:

Build a full machine learning pipeline to predict if the customers will be interested in getting a credit card using PySpark.

Data Collection

```
# Importing libraries
```

```
import pyspark
from pyspark.sql import SparkSession
from pyspark import pandas as pd
from pyspark.sql.functions import col,isnan, when, count
from pyspark.sql.functions import count_distinct
from pyspark.sql import SparkSession
from pyspark import pandas as pd
from pyspark.ml import Pipeline
from pyspark.ml.feature import
(OneHotEncoder,StandardScaler,StringIndexer,VectorAssembler)
from pyspark.ml.stat import Correlation
from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit
from pyspark.ml.classification import LogisticRegression,GBTClassifier,
RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, CrossValidatorModel,
ParamGridBuilder
```

```
#pip install kaggle
#import kaggle
```

```
# copying my kaggle.json file to kaggle local folder
# src_path = r"C:\Users\Alimat sadia\Downloads\kaggle.json"
# dst_path = r"C:\Users\Alimat sadia\.kaggle\kaggle.json"
# shutil.copy(src_path, dst_path)
# print('Copied')

# dbutils.fs.cp("dbfs:///FileStore/tables/kaggle_token/kaggle.json",
"file:/root/.kaggle/")
# !kaggle datasets download -d shelvigarg/credit-card-buyers
# # # unzipping data

# !unzip /databricks/driver/credit-card-buyers.zip
```

```
spark=SparkSession.builder.getOrCreate()
```

```
spark
```

SparkSession - hive

SparkContext

[Spark UI]

Version
```
v3.3.0
```
Master
```
local[8]
```
AppName
```
Databricks Shell
```

Data Exploration

```
# Read data file
df = spark.read.csv("/FileStore/tables/dataset/train_data_credit_card.csv",
header=True, inferSchema=True)
```

```
# to check all the columns and their datatype
df.printSchema()
```

```
root
 |-- ID: string (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Region_Code: string (nullable = true)
 |-- Occupation: string (nullable = true)
 |-- Channel_Code: string (nullable = true)
 |-- Vintage: integer (nullable = true)
 |-- Credit_Product: string (nullable = true)
 |-- Avg_Account_Balance: integer (nullable = true)
 |-- Is_Active: string (nullable = true)
 |-- Is_Lead: integer (nullable = true)
```

```
#Number of rows and columns
print("The shape of the data is ",(df.count(), len(df.columns)))
```

```
The shape of the data is  (245725, 11)
```

```
# statistical summary of the data columns
df.describe().toPandas().transpose()
```

	0	1	2	3	
summary	count	mean	stddev	min	
ID	245725	None	None	222A8XWS	ZZZZ
Gender	245725	None	None	Female	
Age	245725	43.85630684708516	14.828671804648	23	
Region_Code	245725	None	None	RG250	I
Occupation	245725	None	None	Entrepreneur	Self_Em
Channel_Code	245725	None	None	X1	
Vintage	245725	46.95914131651236	32.35313570875409	7	
Credit_Product	216400	None	None	No	
Avg_Account_Balance	245725	1128403.1010194323	852936.3560692756	20790	103
Is_Active	245725	None	None	No	
Is_Lead	245725	0.23720826126767727	0.4253718824871881	0	

```
# viewing dataset
df.show(15)
```

```
+--------+------+---+-----------+------------+------------+-------+--------
------+-------------------+---------+-------+
|      ID|Gender|Age|Region_Code|   Occupation|Channel_Code|Vintage|Credit_P
roduct|Avg_Account_Balance|Is_Active|Is_Lead|
+--------+------+---+-----------+------------+------------+-------+--------
------+-------------------+---------+-------+
|NNVBBKZB|Female| 73|      RG268|       Other|          X3|     43|
No|            1045696|       No|      0|
|IDD62UNG|Female| 30|      RG277|    Salaried|          X1|     32|
No|             581988|       No|      0|
|HD3DSEMC|Female| 56|      RG268|Self_Employed|          X3|     26|
No|            1484315|      Yes|      0|
|BF3NC7KV|  Male| 34|      RG270|    Salaried|          X1|     19|
No|             470454|       No|      0|
|TEASRWXV|Female| 30|      RG282|    Salaried|          X1|     33|
No|             886787|       No|      0|
|ACUTYTWS|  Male| 56|      RG261|Self_Employed|          X1|     32|
No|             544163|      Yes|      0|
|ETQCZFEJ|  Male| 62|      RG282|       Other|          X3|     20|
null|            1056750|      Yes|      1|
|JJNJUQMQ|Female| 48|      RG265|Self_Employed|          X3|     13|
```

```
# Displaying uniques values of each column

for col_name in df.columns:
    unique_val = df.select(col_name).distinct().collect()
    print(f" {col_name}")
    print(f"Unique values count: {len(unique_val)}")
```

```
 ID
Unique values count: 245725
 Gender
Unique values count: 2
 Age
Unique values count: 63
 Region_Code
Unique values count: 35
 Occupation
Unique values count: 4
 Channel_Code
Unique values count: 4
 Vintage
Unique values count: 66
 Credit_Product
Unique values count: 3
 Avg_Account_Balance
Unique values count: 135292
 Is_Active
Unique values count: 2
 Is_Lead
```

Number of unique value per columns

```
# checking for missing value
```

```
# Find Count of Null, None, NaN of All DataFrame Columns
df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in
df.columns]
   ).show()
```

```
+---+------+---+-----------+----------+------------+-------+--------------+-
------------------+---------+-------+
| ID|Gender|Age|Region_Code|Occupation|Channel_Code|Vintage|Credit_Product|A
vg_Account_Balance|Is_Active|Is_Lead|
+---+------+---+-----------+----------+------------+-------+--------------+-
------------------+---------+-------+
|  0|     0|  0|          0|         0|           0|      0|         29325|
0|        0|      0|
+---+------+---+-----------+----------+------------+-------+--------------+-
```

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

Credict Product has the highest number of missing value and is the only column with missing value

```
# Value counts of Credit_Products columns

df.groupBy('Credit_Product').count().orderBy('count',
ascending=False).show()
```

```
+--------------+------+
|Credit_Product| count|
+--------------+------+
|            No|144357|
|           Yes| 72043|
|          null| 29325|
+--------------+------+
```

Now we can see the most refrequent value of this columns is No, so we can replace the missing values by the most frequent one instead of deleting the rows

```
df = df.fillna({"Credit_Product":'No'})
df.show()
```

```
+--------+------+---+-----------+-------------+------------+-------+--------
------+-------------------+---------+-------+
|      ID|Gender|Age|Region_Code|   Occupation|Channel_Code|Vintage|Credit_P
roduct|Avg_Account_Balance|Is_Active|Is_Lead|
+--------+------+---+-----------+-------------+------------+-------+--------
------+-------------------+---------+-------+
|NNVBBKZB|Female| 73|      RG268|        Other|          X3|     43|
No|            1045696|       No|      0|
|IDD62UNG|Female| 30|      RG277|     Salaried|          X1|     32|
No|             581988|       No|      0|
|HD3DSEMC|Female| 56|      RG268|Self_Employed|          X3|     26|
No|            1484315|      Yes|      0|
|BF3NC7KV|  Male| 34|      RG270|     Salaried|          X1|     19|
No|             470454|       No|      0|
|TEASRWXV|Female| 30|      RG282|     Salaried|          X1|     33|
No|             886787|       No|      0|
|ACUTYTWS|  Male| 56|      RG261|Self_Employed|          X1|     32|
No|             544163|      Yes|      0|
|ETQCZFEJ|  Male| 62|      RG282|        Other|          X3|     20|
No|            1056750|      Yes|      1|
|JJNJUQMQ|Female| 48|      RG265|Self_Employed|          X3|     13|
```

```
df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in
df.columns]
   ).show()
```

```
+---+------+---+-----------+----------+------------+-------+--------------+-
------------------+--------+-------+
| ID|Gender|Age|Region_Code|Occupation|Channel_Code|Vintage|Credit_Product|A
vg_Account_Balance|Is_Active|Is_Lead|
+---+------+---+-----------+----------+------------+-------+--------------+-
------------------+--------+-------+
|  0|     0|  0|          0|         0|           0|      0|             0|
0|        0|      0|
+---+------+---+-----------+----------+------------+-------+--------------+-
------------------+--------+-------+
```

Great News ! our data is free of missing values

Information Extraction With Spark SQL

```
# we can easily regiter a DataFrame as a table

df.createOrReplaceTempView("data")
```

```
# Distinct region

df.select(
    count_distinct("Region_Code").alias("distinct region code")
).show()
```

```
+--------------------+
|distinct region code|
+--------------------+
|                  35|
+--------------------+
```

```
spark.sql("SELECT * FROM data").show()
```

```
+--------+------+---+-----------+------------+------------+-------+--------
------+-------------------+--------+-------+
|      ID|Gender|Age|Region_Code|   Occupation|Channel_Code|Vintage|Credit_P
roduct|Avg_Account_Balance|Is_Active|Is_Lead|
```

```
+--------+------+---+----------+-------------+------------+-------+--------
------+------------------+--------+-------+
|NNVBBKZB|Female| 73|     RG268|        Other|          X3|     43|
No|           1045696|      No|      0|
|IDD62UNG|Female| 30|     RG277|     Salaried|          X1|     32|
No|            581988|      No|      0|
|HD3DSEMC|Female| 56|     RG268|Self_Employed|          X3|     26|
No|           1484315|     Yes|      0|
|BF3NC7KV|  Male| 34|     RG270|     Salaried|          X1|     19|
No|            470454|      No|      0|
|TEASRWXV|Female| 30|     RG282|     Salaried|          X1|     33|
No|            886787|      No|      0|
|ACUTYTWS|  Male| 56|     RG261|Self_Employed|          X1|     32|
No|            544163|     Yes|      0|
|ETQCZFEJ|  Male| 62|     RG282|        Other|          X3|     20|
No|           1056750|     Yes|      1|
|JJNJUQMQ|Female| 48|     RG265|Self_Employed|          X3|     13|
```

```
# Number of male with active credict card

spark.sql("SELECT  count( DISTINCT ID)  as Number_of_male_active FROM data
where data.Credit_Product='Yes' and data.Gender='Male'").show()
```

```
+--------------------+
|Number_of_male_active|
+--------------------+
|               41738|
+--------------------+
```

```
# Percentage of male and female users who has been Active in the last 3
Months
spark.sql("SELECT  data.Gender,ROUND((SUM(CASE WHEN data.Is_Active='Yes'
THEN 1 ELSE 0 END) / COUNT(*) )* 100,2) as percentage FROM data GROUP BY
data.Gender").show()
```

```
+------+----------+
|Gender|percentage|
+------+----------+
|Female|     35.41|
|  Male|     41.69|
+------+----------+
```

```
# Target variable Is_Lead Distribution

spark.sql("SELECT data.Is_Lead, COUNT(data.Is_lead) as distinct ,
(COUNT(data.Is_lead) / (SELECT COUNT(data.Is_lead) FROM data))*100  as
percentage FROM data GROUP BY data.Is_Lead").show()
```

```
+-------+--------+------------------+
|Is_Lead|distinct|        percentage|
+-------+--------+------------------+
|      1|   58288|23.720826126767726|
|      0|  187437| 76.27917387323228|
+-------+--------+------------------+
```

Our data is imbalanced, the number of 1 is more larger than the number of 0.

```
# Average account balance per region
spark.sql("SELECT  data.Region_Code , (AVG(data.Avg_Account_Balance)) as
Average_account_balance from data Group By data.Region_Code order by
Average_account_balance desc").show()
```

```
+-----------+-----------------------+
|Region_Code|Average_account_balance|
+-----------+-----------------------+
|      RG283|     1475574.2942276313|
|      RG284|     1473165.5680641823|
|      RG268|     1463900.0817610063|
|      RG254|      1407392.135394933|
|      RG253|     1374248.0543595264|
|      RG262|     1200158.0458612975|
|      RG276|     1059816.6505065123|
|      RG269|     1027003.1523591505|
|      RG274|      996307.0546727204|
|      RG277|      983289.3474972711|
|      RG261|      981286.4003668282|
|      RG282|      953867.7062961056|
|      RG278|      913786.0779363337|
|      RG281|      889643.6980168859|
|      RG272|      885223.5024752475|
|      RG255|       868249.742814668|
|      RG257|      858105.4543517457|
|      RG273|      856303.2921947965|
```

RG283 is the region with highest average account balance

Data Preprocessing

The ID column is not necessary in this data let's remove it

```
df= df.drop("ID")
df.show()
```

```
+------+---+-----------+-------------+------------+-------+--------------+--
-----------------+---------+-------+
|Gender|Age|Region_Code|   Occupation|Channel_Code|Vintage|Credit_Product|Av
g_Account_Balance|Is_Active|Is_Lead|
+------+---+-----------+-------------+------------+-------+--------------+--
-----------------+---------+-------+
|Female| 73|      RG268|        Other|          X3|     43|            No|
1045696|       No|      0|
|Female| 30|      RG277|     Salaried|          X1|     32|            No|
581988|       No|      0|
|Female| 56|      RG268|Self_Employed|          X3|     26|            No|
1484315|      Yes|      0|
|  Male| 34|      RG270|     Salaried|          X1|     19|            No|
470454|       No|      0|
|Female| 30|      RG282|     Salaried|          X1|     33|            No|
886787|       No|      0|
|  Male| 56|      RG261|Self_Employed|          X1|     32|            No|
544163|      Yes|      0|
|  Male| 62|      RG282|        Other|          X3|     20|            No|
1056750|      Yes|      1|
|Female| 48|      RG265|Self_Employed|          X3|     13|            No|
```

```
#Float type columns of this data frame are:
target_col='Is_Lead'
numerical_cols = [df.dtypes[i][0] for i in range(len(df.dtypes)) if
df.dtypes[i][1]=='int']
numerical_cols.pop(-1)
numerical_cols
```

```
Out[23]: ['Age', 'Vintage', 'Avg_Account_Balance']
```

```
#Define string-type columns:
string_cols = [df.dtypes[i][0] for i in range(len(df.dtypes)) if
df.dtypes[i][1]=='string']
string_cols
```

```
Out[24]: ['Gender',
 'Region_Code',
 'Occupation',
 'Channel_Code',
 'Credit_Product',
 'Is_Active']
```

In order to feed this data to a model we have to transform the categorical features into number using OneHotEncoder

```
# string columns encoding


string_cols_indexer = [f"{string_col}_indexer" for string_col in
string_cols]
string_cols_vector  = [f"{string_col}_vector" for string_col in
string_cols_indexer]

string_cols_encode_stage= [
     StringIndexer(
 inputCols=string_cols,
 outputCols=string_cols_indexer,
 ),
 OneHotEncoder(
 inputCols=string_cols_indexer,
 outputCols=string_cols_vector)]





# numerical data scaling

numerical_scale_stage = [
 VectorAssembler(
 inputCols=numerical_cols,
 outputCol="assembled_numeric_cols",
 ),
 StandardScaler(
 inputCol="assembled_numeric_cols",
 outputCol="scaled_numeric_cols",
 ),
]



# group all the features into one columns


features_assembler = [
 VectorAssembler(
 inputCols=string_cols_vector + ["scaled_numeric_cols"],
 outputCol="features",
 )
]
```

```
target_stage = [StringIndexer(inputCol=target_col, outputCol="label")]


# Display effect on the whole data
Pipeline(stages=target_stage).fit(df).transform(df).show(5)
```

```
+------+---+-----------+------------+-----------+-------+--------------+--
-----------------+---------+-------+-----+
|Gender|Age|Region_Code|   Occupation|Channel_Code|Vintage|Credit_Product|Av
g_Account_Balance|Is_Active|Is_Lead|label|
+------+---+-----------+------------+-----------+-------+--------------+--
-----------------+---------+-------+-----+
|Female| 73|      RG268|        Other|          X3|     43|            No|
1045696|       No|      0|  0.0|
|Female| 30|      RG277|     Salaried|          X1|     32|            No|
581988|       No|      0|  0.0|
|Female| 56|      RG268|Self_Employed|          X3|     26|            No|
1484315|      Yes|      0|  0.0|
|  Male| 34|      RG270|     Salaried|          X1|     19|            No|
470454|       No|      0|  0.0|
|Female| 30|      RG282|     Salaried|          X1|     33|            No|
886787|       No|      0|  0.0|
+------+---+-----------+------------+-----------+-------+--------------+--
-----------------+---------+-------+-----+
only showing top 5 rows
```

```
# Splitting the data into train and test data

train, test = df.randomSplit(weights=[0.8, 0.2], seed=42)
print(f"Number of data in the train set: {train.count()}")
print(f"Number of data in the test set: {test.count()}")
```

```
Number of data in the train set: 196584
Number of data in the test set: 49141
```

```
# combining all the stages displayed above into a single pipeline
pipe = Pipeline(stages=string_cols_encode_stage + numerical_scale_stage +
features_assembler + target_stage)
data_preprocessing = pipe.fit(train)
```

```
data_preprocessing_train =
data_preprocessing.transform(train).select("features", "label")
data_preprocessing_train.show(5, truncate=False)
```

```
+---------------------------------------------------------------------------
--------------------------+-----+
|features
|label|
+---------------------------------------------------------------------------
--------------------------+-----+
|(46,[12,35,40,42,43,44,45],[1.0,1.0,1.0,1.0,1.5527608114781,0.4327398259957
207,1.1804160076654502])   |0.0  |
|(46,[26,36,38,41,43,44,45],[1.0,1.0,1.0,1.0,1.6202721511075826,0.4018298384
245978,0.4340912254307288])|0.0  |
|(46,[26,36,38,42,43,44,45],[1.0,1.0,1.0,1.0,1.6202721511075826,0.4018298384
245978,1.5836082643242448])|0.0  |
|(46,[26,36,38,41,43,44,45],[1.0,1.0,1.0,1.0,1.6202721511075826,0.4327398259
957207,0.6636636693035469])|0.0  |
|(46,[26,36,38,42,43,44,45],[1.0,1.0,1.0,1.0,1.6202721511075826,0.4327398259
957207,0.7558367647533992])|0.0  |
+---------------------------------------------------------------------------
--------------------------+-----+
only showing top 5 rows
```

```
data_preprocessing_test =
data_preprocessing.transform(test).select("features", "label")
data_preprocessing_test.show(5, truncate=False)
```

```
+---------------------------------------------------------------------------
----------------------------------+-----+
|features
|label|
+---------------------------------------------------------------------------
----------------------------------+-----+
|(46,[26,36,38,42,43,44,45],[1.0,1.0,1.0,1.0,1.6202721511075826,0.4018298384
245978,0.6337383310425381])       |0.0  |
|(46,[26,36,38,42,43,44,45],[1.0,1.0,1.0,1.0,1.6202721511075826,0.4636498135
668436,0.6812042865299271])       |0.0  |
|(46,[26,36,38,42,43,44,45],[1.0,1.0,1.0,1.0,1.6202721511075826,0.4636498135
668436,0.9455009278527056])       |0.0  |
|(46,[11,36,38,41,43,44,45],[1.0,1.0,1.0,1.0,1.6202721511075826,0.4327398259
957207,0.6210376792389842])       |0.0  |
|(46,[17,37,38,41,42,43,44,45],[1.0,1.0,1.0,1.0,1.0,1.6202721511075826,0.432
7398259957207,1.2018123896293065])|0.0  |
+---------------------------------------------------------------------------
----------------------------------+-----+
only showing top 5 rows
```

Model Building and Cross Validation

Logistic Regression

```
lr_model = LogisticRegression()

# hyperparameter tuning
paramGrid = (
 ParamGridBuilder()
 .addGrid(lr_model.maxIter, [50, 100,150])
 .addGrid(lr_model.regParam, [0.0,0.5, 1.0])
 .addGrid(lr_model.elasticNetParam, [0.0, 0.5,1.0])
 .build()
)

# Create a CrossValidator
evaluator = BinaryClassificationEvaluator()
```

```
cv = CrossValidator(
 estimator=lr_model,
 estimatorParamMaps=paramGrid,
 evaluator=evaluator,
 parallelism=8,
 numFolds=5,
)

cross_val_model = cv.fit(data_preprocessing_train)
```

```
# Extract the best model
best_model_lr = cross_val_model.bestModel
print(best_model_lr)

predictions_lr=best_model_lr.transform(data_preprocessing_test)
# evalutae best result result
print( "Logistic regression accuracy score on the test
set:",evaluator.evaluate(predictions_lr),"\n")

print("Logistic Regression Prediction Table")
predictions_lr.select('label', 'prediction', 'probability').show(20)
```

```
LogisticRegressionModel: uid=LogisticRegression_73cebbe1b8bc, numClasses=2,
numFeatures=46
Logistic regression accuracy score on the test set: 0.7341620461882972

Logistic Regression Prediction Table
+-----+----------+--------------------+
|label|prediction|         probability|
+-----+----------+--------------------+
|  0.0|       0.0|[0.89483941713461...|
|  0.0|       0.0|[0.89282481916175...|
|  0.0|       0.0|[0.89285797885386...|
|  0.0|       0.0|[0.90262157197535...|
|  0.0|       0.0|[0.96799815751839...|
|  0.0|       0.0|[0.91890817892784...|
|  0.0|       0.0|[0.91673685764783...|
|  0.0|       0.0|[0.96909688418369...|
|  0.0|       0.0|[0.89521304444288...|
|  0.0|       0.0|[0.89528125833428...|
|  0.0|       0.0|[0.92154765495802...|
|  0.0|       0.0|[0.91794971142531...|
|  0.0|       0.0|[0.91798555104292...|
```

Gradient-Boosted Tree

```
gbt_model = GBTClassifier()

# hyperparameter tuning
paramGrid = (ParamGridBuilder()
             .addGrid(gbt_model.maxDepth, [2, 4, 6])
             .addGrid(gbt_model.maxIter, [10, 15, 20])
             .build())

# Create a CrossValidator
evaluator = BinaryClassificationEvaluator()
```

```
cv = CrossValidator(
 estimator=gbt_model,
 estimatorParamMaps=paramGrid,
 evaluator=evaluator,
 parallelism=8,
 numFolds=5,
)

cross_val_model = cv.fit(data_preprocessing_train)
```

```
# Extract the best model
best_model_gbt = cross_val_model.bestModel
print(best_model_gbt)

predictions_gbt=best_model_gbt.transform(data_preprocessing_test)

# evalutae best result result
print( "GBTClassifier accuracy score on the test
set:",evaluator.evaluate(predictions_gbt),"\n")

print("GBTClassifier Prediction Table")
predictions_gbt.select('label', 'prediction', 'probability').show(20)
```

```
GBTClassificationModel: uid = GBTClassifier_df12f6811d27, numTrees=20, numCl
asses=2, numFeatures=46
GBTClassifier accuracy score on the test set: 0.7888099034858672

GBTClassifier Prediction Table
+-----+----------+--------------------+
|label|prediction|         probability|
+-----+----------+--------------------+
|  0.0|       0.0|[0.89088839340541...|
|  0.0|       0.0|[0.88507501011413...|
|  0.0|       0.0|[0.88698151346515...|
|  0.0|       0.0|[0.91464453799921...|
|  0.0|       0.0|[0.93863371810767...|
|  0.0|       0.0|[0.91464453799921...|
|  0.0|       0.0|[0.91464453799921...|
|  0.0|       0.0|[0.94204513281611...|
|  0.0|       0.0|[0.90268628663352...|
|  0.0|       0.0|[0.90268628663352...|
|  0.0|       0.0|[0.91813274435838...|
|  0.0|       0.0|[0.92035135496544...|
|  0.0|       0.0|[0.92035135496544...|
```

Random Forest

```
# randomForest

rf_model = RandomForestClassifier()

# hyperparameter tuning
paramGrid = (
 ParamGridBuilder()
 .addGrid(rf_model.maxDepth, [5, 7, 10])
 .addGrid(rf_model.numTrees, [30,50,100])
 .build()
)
```

```
# Create a CrossValidator
evaluator = BinaryClassificationEvaluator()
cv = CrossValidator(
 estimator=rf_model,
 estimatorParamMaps=paramGrid,
 evaluator=evaluator,
 parallelism=8,
 numFolds=5,
)

cross_val_model = cv.fit(data_preprocessing_train)


# Extract the best model
best_model_rf = cross_val_model.bestModel
print(best_model_rf)

predictions_rf=best_model_rf.transform(data_preprocessing_test)

# evalutae best result result
print( "RandomForestClassifier accuracy score on the test
set:",evaluator.evaluate(predictions_rf),"\n")

print("RandomForestClassifier Prediction Table")
predictions_rf.select('label', 'prediction', 'probability').show(20)
```

```
RandomForestClassificationModel: uid=RandomForestClassifier_9dd4dd3636fa, nu
mTrees=100, numClasses=2, numFeatures=46
RandomForestClassifier accuracy score on the test set: 0.7786447919491395

RandomForestClassifier Prediction Table
+-----+----------+--------------------+
|label|prediction|         probability|
```

```
+-----+----------+--------------------+
|  0.0|       0.0|[0.90150021148626...|
|  0.0|       0.0|[0.90208502185240...|
|  0.0|       0.0|[0.90152274208503...|
|  0.0|       0.0|[0.92797679267685...|
|  0.0|       0.0|[0.94355516985084...|
|  0.0|       0.0|[0.92950653515829...|
|  0.0|       0.0|[0.92475646105034...|
|  0.0|       0.0|[0.94692517148005...|
|  0.0|       0.0|[0.90534943231272...|
|  0.0|       0.0|[0.90528069876450...|
|  0.0|       0.0|[0.93199627291390...|
|  0.0|       0.0|[0.93122995403098...|
```

Summary

We got close prediction results between Linear Regression, GBTClassifier, and Random Forest model, however the GBTClassifier is the best model for this project with the highest accuracy of 0.78.