CIS5560 Term Project Tutorial

Authors: Lekha Ajit Kumar, Sushmitha Dandu, Dauren Omarov, Navyasree Sriramoju

Instructor: Jongwook Woo

Date: 05/11/2023

Lab Tutorial

Lekha Ajit Kumar (lajitku@calstatela.edu)

Sushmitha Dandu (sdandu3@calstatela.edu)

Dauren Omarov (domarov@calstatela.edu)

Navyasree Sriramoju (nsriram@calstatela.edu)

IBM Transactions for Anti Money Laundering (AML) Predictive Analysis using machine learning models in Spark ML

Objectives

List what your objectives are. The objective of the lab is to build a model that predicts the insights into the patterns and characteristics of both legitimate and laundering transactions using the following machine learning algorithms:

- Logistic Regression
- Gradient Boost Tree
- Decision Tree
- Random Forest
- Factorization Machine
- Support Vector Machine

Platform Specifications

HDFS ORACLE SPECIFICATION

- Hadoop Version: 3.1.2
- No. of CPUs: 4
- PySpark version: 3.0.2
- Nodes: 5
- Total Storage:390.7 GB
- CPU speed: 1995.3 MHz

DATABRICKS SPECIFICATION

- Databricks Community Version: 10.4 LTS (includes Apache Spark 3.1.1, Scala 2.12)
- File System: DBFS (Data Bricks File System)
- Nodes: 1
- Python Version: 3.10.4

Dataset Specifications

- Dataset Name: IBM Transactions for Anti Money Laundering (AML)
- Dataset Size: 2.98 GB
- Dataset URL: https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml
- Dataset Format: csv

Step 1: Get data manually from the Data source to Databricks.

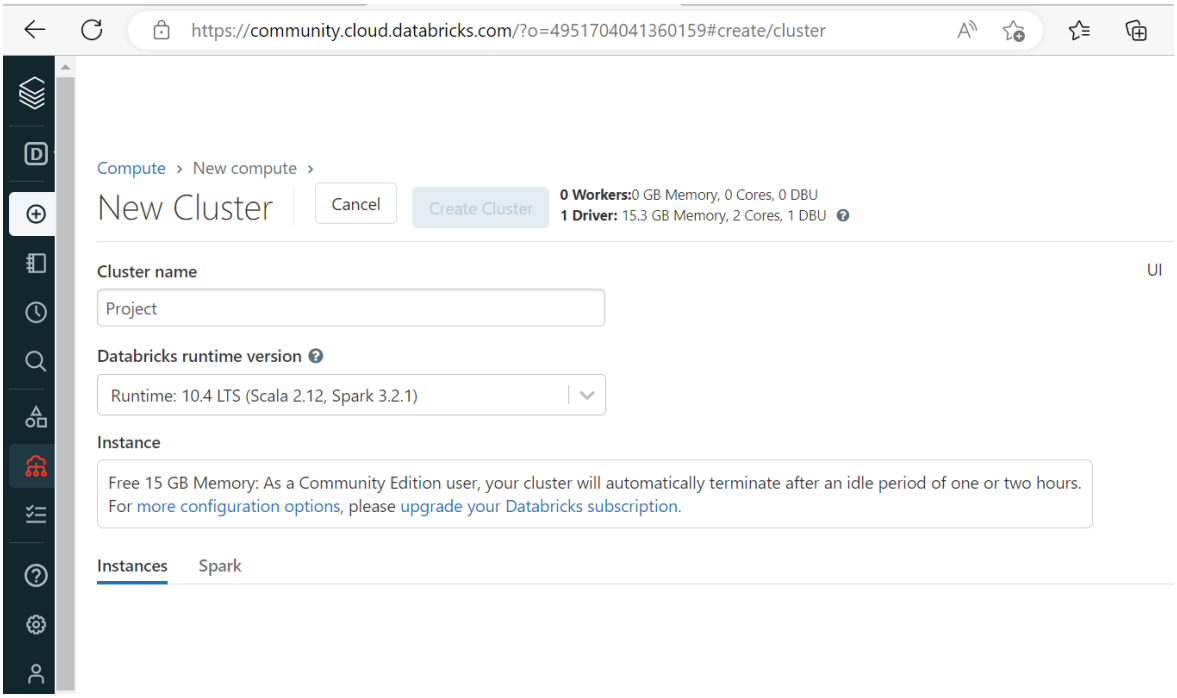
1. Login to Kaggle.
2. Download the files 'LI_Medium_Trans.csv' and 'LI-Medium_Patterns.txt' of IBM Transactions for Anti Money Laundering (AML) Dataset: https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml
3. Concatenate the above downloaded files with the following code. Or download the concatenated file from this GitHub link https://github.com/Lekha19202/CIS-5560-big-data-science-project/blob/main/money_%20Laundering.csv

```
import pandas as pd
import random
file1=pd.read_csv('LI-Medium_Trans.csv')
file2=pd.read_csv('LI-Medium_Patterns.csv')
concat_file = pd.concat([file1,file2])
random.shuffle(concat_file)
concat_file.to_csv('dataset.csv',index=False)
```

4. Sign into your Databricks account.

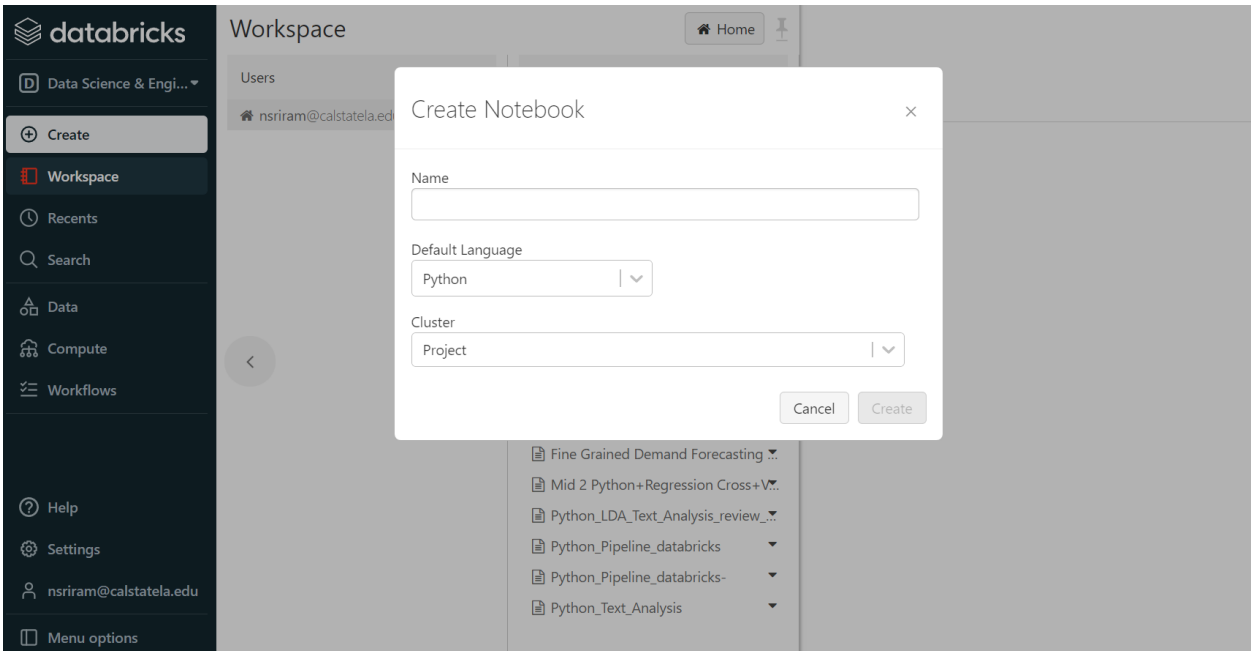
5. Go to Clusters option on the left and click on create cluster.

6. Give the cluster name and click create cluster.

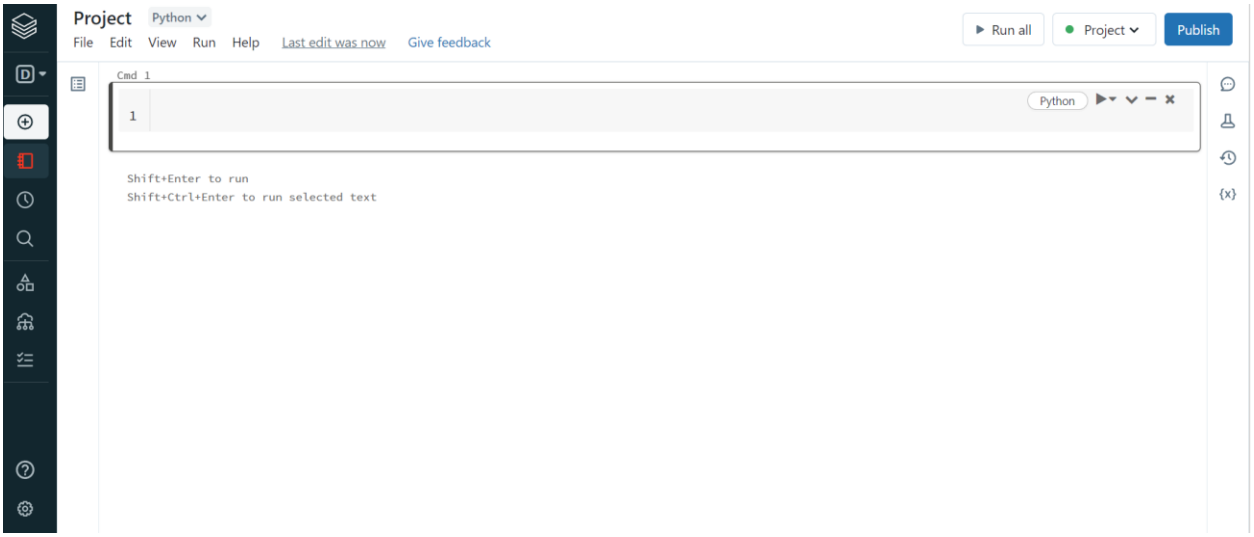


Step 2: Create a notebook and Load Data File

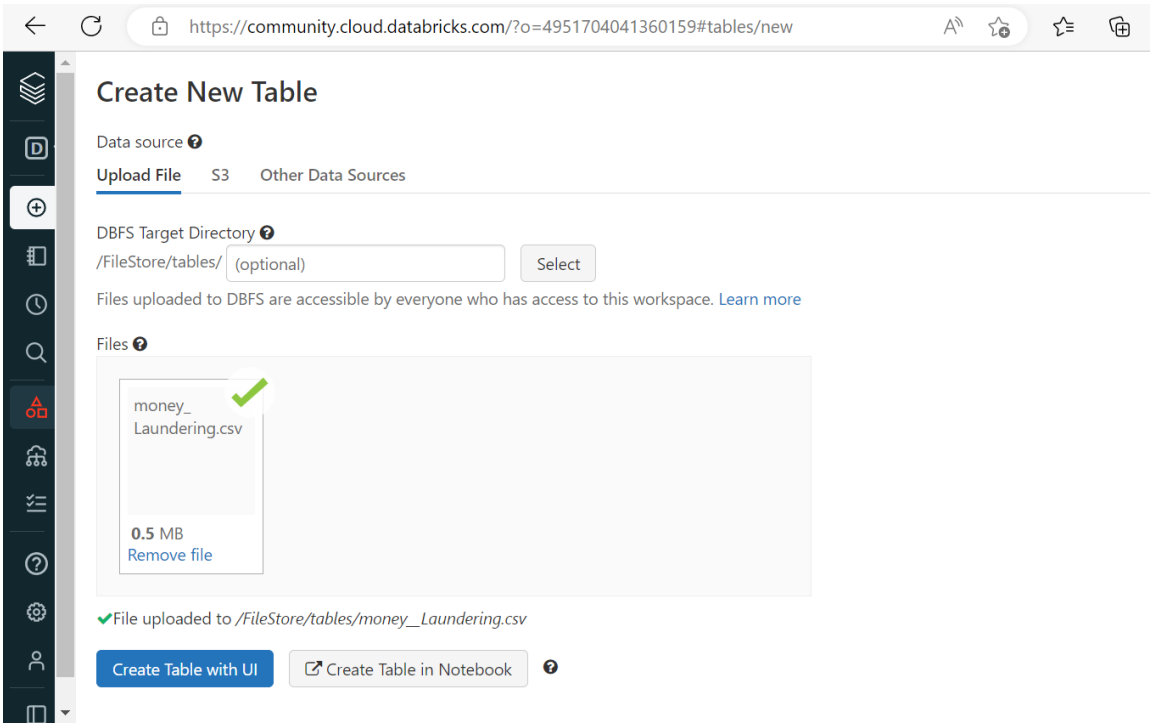
A notebook is a GUI to write a code as collection of cells that run computations on an Apache Spark cluster. To create a notebook in the workspace:
In the Workspace folder, select Create > Notebook.



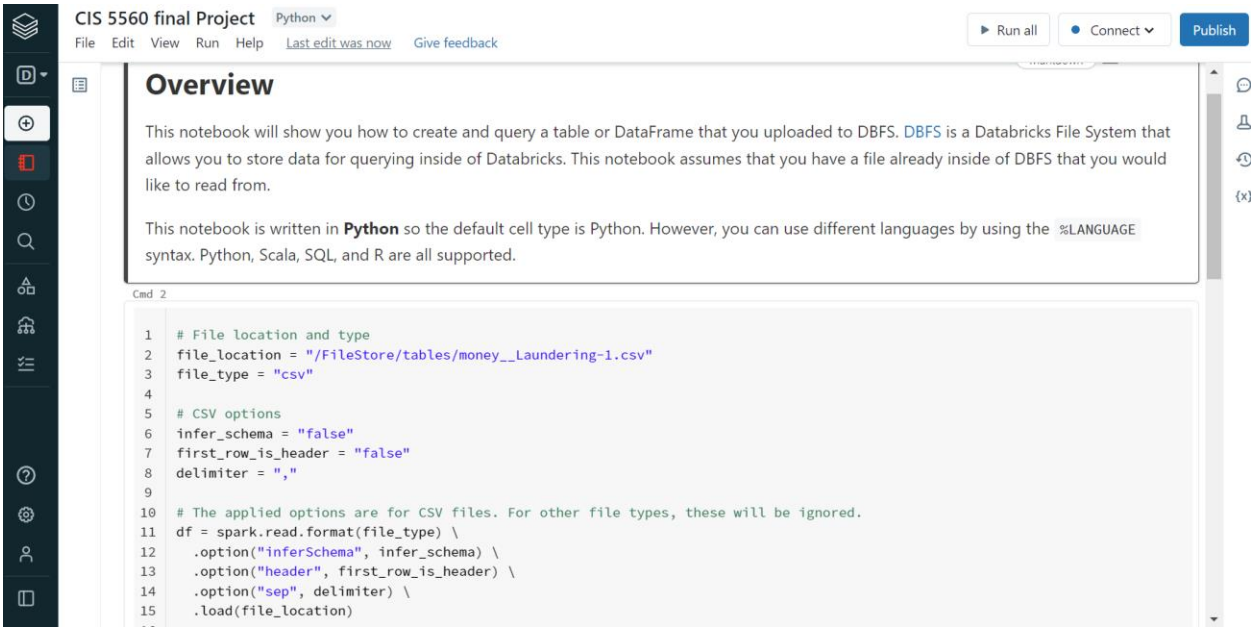
The notebook opens with an empty cell at the top, in which you can add your code.



Go to your Databricks page and select the 'Data' option from the left menu bar. Next, click on 'Create Table' and choose the file you wish to upload. Finally, upload the CSV file from the dataset.



It automatically generates a Spark and markdown codes at the notebook, which read the data file and display it.



Make sure if the cluster is attached – detached will generate an error, now you can make changes to the Code.

```
# Import Spark SQL and Spark ML libraries
from pyspark.sql.types import *
from pyspark.sql.functions import *

from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression, FMRegressor, RandomForestRegressor,
GBTRegressionModel, GBTRegressor
from pyspark.ml.classification import DecisionTreeClassifier, LogisticRegression, LinearSVC
from pyspark.ml.feature import VectorAssembler, MinMaxScaler, StringIndexer,VectorIndexer
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator, TrainValidationSplit
from pyspark.ml.evaluation import RegressionEvaluator, BinaryClassificationEvaluator,
MulticlassClassificationEvaluator

from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from time import time
```

```python
# Import Spark SQL and Spark ML libraries
from pyspark.sql.types import *
from pyspark.sql.functions import *

from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression, FMRegressor, RandomForestRegressor, GBTRegressionModel, GBTRegressor
from pyspark.ml.classification import DecisionTreeClassifier, LogisticRegression, LinearSVC
from pyspark.ml.feature import VectorAssembler, MinMaxScaler, StringIndexer,VectorIndexer
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator, TrainValidationSplit
from pyspark.ml.evaluation import RegressionEvaluator, BinaryClassificationEvaluator, MulticlassClassificationEvaluator

from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from time import time


```

Python

1

Command took 1.60 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

DataFrame Schema , that should be a Table schema.

```
# DataFrame Schema, that should be a Table schema
Schema = StructType([
  StructField("Timestamp", StringType(), False),
  StructField("From Bank", IntegerType(), False),
  StructField("Account 1", StringType(), False),
  StructField("To Bank", IntegerType(), False),
  StructField("Account 2", StringType(), False),
  StructField("Amount Received", FloatType(), False),
  StructField("Receiving Currency", StringType(), False),
  StructField("Amount Paid", FloatType(), False),
  StructField("Payment Currency", StringType(), False),
  StructField("Payment Format", StringType(), False),
  StructField("Is Laundering", IntegerType(), False),
])
```

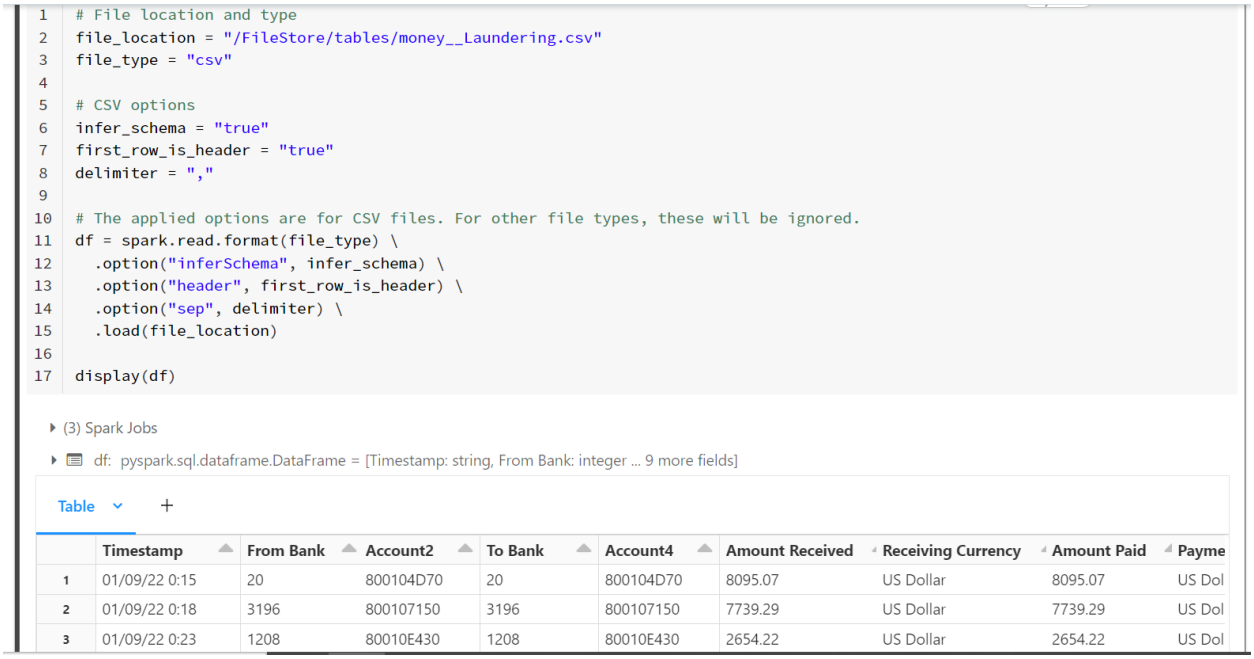
```python

# DataFrame Schema, that should be a Table schema
Schema = StructType([
  StructField("Timestamp", StringType(), False),
  StructField("From Bank", IntegerType(), False),
  StructField("Account 1", StringType(), False),
  StructField("To Bank", IntegerType(), False),
  StructField("Account 2", StringType(), False),
  StructField("Amount Received", FloatType(), False),
  StructField("Receiving Currency", StringType(), False),
  StructField("Amount Paid", FloatType(), False),
  StructField("Payment Currency", StringType(), False),
  StructField("Payment Format", StringType(), False),
  StructField("Is Laundering", IntegerType(), False),
])
```

Command took 0.07 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Load Source Data- We need to load the data.

```
# File location and type
file_location = ["/FileStore/tables/money__Laundering.csv"]
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","

df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(df)
```

Change the code to infer the schema – data types of each column and to set the first row as header. Then, select play menu, especially, “Run Cell” to execute the present cell only.
# CSV options
infer_schema = "true"
first_row_is_header = "true



```
# File location and type
file_location = "/FileStore/tables/money__Laundering.csv"
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(df)
```

(3) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [Timestamp: string, From Bank: integer ... 9 more fields]

Table +

	Timestamp	From Bank	Account2	To Bank	Account4	Amount Received	Receiving Currency	Amount Paid	Payme
1	01/09/22 0:15	20	800104D70	20	800104D70	8095.07	US Dollar	8095.07	US Dol
2	01/09/22 0:18	3196	800107150	3196	800107150	7739.29	US Dollar	7739.29	US Dol
3	01/09/22 0:23	1208	80010E430	1208	80010E430	2654.22	US Dollar	2654.22	US Dol

Step 3: Data Cleaning

Here we have converted hexadecimal values in the 'Account2' and 'Account4' columns to their decimal equivalents, adding new columns with the converted values to the Data Frame.

```
# Converting hexa decimal to integer
df = df.withColumn('Account2', conv(df['Account2'], 16, 10))
df = df.withColumn('Account4', conv(df['Account4'], 16, 10))
df.show()
```

```
1  # Converting hexa decimal to integer
2  df = df.withColumn('Account2', conv(df['Account2'], 16, 10))
3  df = df.withColumn('Account4', conv(df['Account4'], 16, 10))
4  df.show()
5
```

▸ (1) Spark Jobs

```
+-------------+---------+-----------+-------+-----------+---------------+-----------------+-----------+----------------+-------
-------+-------------+
|    Timestamp|From Bank|   Account2|To Bank|   Account4|Amount Received|Receiving Currency|Amount Paid|Payment Currency|Payment
Format|Is Laundering|
+-------------+---------+-----------+-------+-----------+---------------+-----------------+-----------+----------------+-------
-------+-------------+
|01/09/22 0:15|       20|34360806768|     20|34360806768|        8095.07|         US Dollar|    8095.07|       US Dollar|  Reinv
estment|            0|
|01/09/22 0:18|     3196|34360815952|   3196|34360815952|        7739.29|         US Dollar|    7739.29|       US Dollar|  Reinv
estment|            0|
|01/09/22 0:23|     1208|34360845360|   1208|34360845360|        2654.22|         US Dollar|    2654.22|       US Dollar|  Reinv
estment|            0|
|01/09/22 0:19|     3203|34360846976|   3203|34360846976|       13284.41|         US Dollar|   13284.41|       US Dollar|  Reinv
estment|            0|
|01/09/22 0:27|       20|34360806688|     20|34360806688|           9.72|         US Dollar|       9.72|       US Dollar|  Reinv
estment|            0|
|01/09/22 0:29|       20|34360806768|     20|34360806768|           5.38|         US Dollar|       5.38|       US Dollar|  Reinv
estment|            0|
|01/09/22 0:08|     1208|34360845360|   1208|34360845360|           7.66|         US Dollar|       7.66|       US Dollar|  Reinv
estment|            0|
```

Step 4: Creating Data Frame & Splitting the Data

```
data = df.select("Timestamp", "From Bank", "Account2", "To Bank", "Account4", "Amount Received",
"Receiving Currency", "Amount Paid", "Payment Currency", "Payment Format", ((col("Is
Laundering")).cast("Double").alias("label")))

data.show()
```

This step is to split the data into Train and Test data in the ratio of 70:30. Training dataset is used to build a model and Testing dataset is used to Test the model built.

```
# Split the data
splits = data.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1].withColumnRenamed("label", "trueLabel")
```

```python
#creating dataframe
data = df.select("Timestamp", "From Bank", "Account2", "To Bank", "Account4", "Amount Received", "Receiving Currency",
"Amount Paid", "Payment Currency", "Payment Format", ((col("Is Laundering")).cast("Double").alias("label")))
data.show()

# Split the data
splits = data.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1].withColumnRenamed("label", "trueLabel")
```

(1) Spark Jobs

```
+-------------+---------+-----------+-------+-----------+---------------+------------------+-----------+----------------+-------
-------+-----+
|    Timestamp|From Bank|   Account2|To Bank|   Account4|Amount Received|Receiving Currency|Amount Paid|Payment Currency|Payment
Format|label|
+-------------+---------+-----------+-------+-----------+---------------+------------------+-----------+----------------+-------
-------+-----+
|01/09/22 0:15|       20|34360806768|     20|34360806768|        8095.07|         US Dollar|    8095.07|       US Dollar|  Reinv
estment|  0.0|
|01/09/22 0:18|     3196|34360815952|   3196|34360815952|        7739.29|         US Dollar|    7739.29|       US Dollar|  Reinv
estment|  0.0|
|01/09/22 0:23|     1208|34360845360|   1208|34360845360|        2654.22|         US Dollar|    2654.22|       US Dollar|  Reinv
estment|  0.0|
|01/09/22 0:19|     3203|34360846976|   3203|34360846976|       13284.41|         US Dollar|   13284.41|       US Dollar|  Reinv
estment|  0.0|
|01/09/22 0:27|       20|34360806688|     20|34360806688|           9.72|         US Dollar|       9.72|       US Dollar|  Reinv
```

Finding the count of training and testing rows:

```
#Finding the count of training and testing rows
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, " Testing Rows:", test_rows)
```

Step 5: Random Forest Regression

Run Random Forest Regression algorithm using Train Split Validation and Cross Validation

```
# RandomForestRegressor

rf = RandomForestRegressor(labelCol="label", featuresCol="features")
```

```python
# RandomForestRegressor

rf = RandomForestRegressor(labelCol="label", featuresCol="features")

```

Command took 0.13 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Tune Parameters:

Tuning parameters to find the best model for your data. To do this we are using CrossValidator and trainValidationSplit class to evaluate each combination of parameters defined in a ParameterGrid against multiple folds of the data split into training and validation datasets, in order to find the best performing parameters.

```
paramGrid = ParamGridBuilder() \
  .addGrid(rf.maxDepth, [2, 3]) \
  .addGrid(rf.maxBins, [5, 10]) \
  .addGrid(rf.minInfoGain, [0.0]) \
  .build()
```

```
1  paramGrid = ParamGridBuilder() \
2    .addGrid(rf.maxDepth, [2, 3]) \
3    .addGrid(rf.maxBins, [5, 10]) \
4    .addGrid(rf.minInfoGain, [0.0]) \
5    .build()
```

```
Command took 0.03 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project
```

Define the Pipeline:

Defining a pipeline that creates a feature vector and trains the models

```
pipeline = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5, strIdx6, assembler, minMax, rf])


start = time()

#tvs = TrainValidationSplit(estimator=pipeline,
evaluator=BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="prediction",
metricName="areaUnderROC"), estimatorParamMaps=paramGrid, trainRatio=0.8)

model = pipeline.fit(train)
#model = tvs.fit(train)

end = time()
phrase = 'Random Forest tvs testing'
print('{} takes {} seconds'.format(phrase, (end - start))) #round(end - start, 2)))

time_rf_tvs = end - start
```

```
pipeline = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5, strIdx6, assembler, minMax, rf])


start = time()

#tvs = TrainValidationSplit(estimator=pipeline, evaluator=BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="prediction", metricName="areaUnderROC"), estimatorParamMaps=paramGrid, trainRatio=0.8)

model = pipeline.fit(train)
#model = tvs.fit(train)

end = time()
phrase = 'Random Forest tvs testing'
print('{} takes {} seconds'.format(phrase, (end - start))) #round(end - start, 2)))

time_rf_tvs = end - start
```

▸ (22) Spark Jobs

```
Random Forest tvs testing takes 19.382533073425293 seconds
```

Feature Importance:

Feature importance provides insights into which features are most relevant to the prediction task, helps in feature selection, and aids in interpreting and explaining the model's behavior.

#feature importance

import pandas as pd

featureImp = pd.DataFrame(list(zip(finalVect.getInputCols(),rfModel.featureImportances)),columns=["feature", "importance"])

featureImp.sort_values(by="importance", ascending=False)

Python

```
#feature importance
import pandas as pd
featureImp = pd.DataFrame(list(zip(finalVect.getInputCols(),rfModel.featureImportances)),columns=["feature", "importance"])
featureImp.sort_values(by="importance", ascending=False)
```

	feature	importance
0	TimestampIdx	0.357747
1	From Bank	0.254410
2	Account2	0.211565
3	To Bank	0.176279

Command took 0.87 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

```python
import pandas as pd
featureImp = pd.DataFrame(list(zip(assembler.getInputCols(),rfModel.featureImportances)),columns=["feature", "importance"])
featureImp.sort_values(by="importance", ascending=False)
```

	feature	importance
0	From Bank	0.357747
1	To Bank	0.254410
2	Amount Received	0.211565
3	Amount Paid	0.176279

```
import pandas as pd

featureImp = pd.DataFrame(list(zip(assembler.getInputCols(),rfModel.featureImportances)),columns=["feature", "importance"])

featureImp.sort_values(by="importance", ascending=False)
```

Tune Parameters:

```
# TODO: params refered to the reference above

paramGrid2 = (ParamGridBuilder() \

        .addGrid(rf.maxDepth, [3, 5]) \

        .addGrid(rf.maxBins, [10, 15]) \

        .addGrid(rf.minInfoGain, [0.0]) \

        .build())
```

```python
# TODO: params refered to the reference above
paramGrid2 = (ParamGridBuilder() \
              .addGrid(rf.maxDepth, [3, 5]) \
              .addGrid(rf.maxBins, [10, 15]) \
              .addGrid(rf.minInfoGain, [0.0]) \
              .build())
```

Define the Pipeline2:

```
#Randomforest in TrainValidationSplit

pipelinetvs = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5,
strIdx6,catVect,catIdx,assembler, minMax, rf])

start = time()

tvs2 = TrainValidationSplit(estimator=pipelinetvs,
evaluator=BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="prediction",
metricName="areaUnderROC"), estimatorParamMaps=paramGrid2, trainRatio=0.8)


# the second best model

modeltvs = tvs2.fit(train)
```

Python

```
#Randomforest in TrainValidationSplit
pipelinetvs = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5, strIdx6,catVect,catIdx,assembler, minMax, rf])

start = time()

tvs2 = TrainValidationSplit(estimator=pipelinetvs, evaluator=BinaryClassificationEvaluator(labelCol="label",
rawPredictionCol="prediction", metricName="areaUnderROC"), estimatorParamMaps=paramGrid2, trainRatio=0.8)

# the second best model
modeltvs = tvs2.fit(train)

end = time()
phrase = 'Random Forest tvs2 testing'
print('{} takes {} seconds'.format(phrase, (end - start))) #round(end - start, 2)))


time_rf_tvs2 = end - start
```

▸ (57) Spark Jobs

```
Random Forest tvs2 testing takes 47.87916326522827 seconds
```

Cross Validator with parameters

```
# TODO: params refered to the reference above
paramGridCV = ParamGridBuilder() \
  .addGrid(rf.maxDepth, [2, 3]) \
  .addGrid(rf.maxBins, [5, 10]) \
  .addGrid(rf.minInfoGain, [0.0]) \
  .build()
```

```
# TODO: params refered to the reference above
paramGridCV = ParamGridBuilder() \
  .addGrid(rf.maxDepth, [2, 3]) \
  .addGrid(rf.maxBins, [5, 10]) \
  .addGrid(rf.minInfoGain, [0.0]) \
  .build()
```

Command took 0.04 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

```
#randomforest with CrossValidator

pipelineCV = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5,
strIdx6,catVect,catIdx,assembler, minMax, rf])

start = time()

# TODO: K = 3

# K=3, 5

K = 3

cv =  CrossValidator(estimator=pipelineCV, estimatorParamMaps=paramGridCV,
evaluator=BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="prediction",
metricName="areaUnderROC"),numFolds=K)


# the third best model

modelCV = cv.fit(train)


end = time()

phrase = 'Random Forest testing'

print('{} takes {} seconds'.format(phrase, (end - start))) #round(end - start, 2)))


time_rf_cv = end - start
```

```python
#randomforest with CrossValidator
pipelineCV = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5, strIdx6,catVect,catIdx,assembler, minMax, rf])

start = time()

# TODO: K = 3
# K=3, 5
K = 3
cv =  CrossValidator(estimator=pipelineCV, estimatorParamMaps=paramGridCV,
evaluator=BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="prediction",
metricName="areaUnderROC"),numFolds=K)

# the third best model
modelCV = cv.fit(train)

end = time()
phrase = 'Random Forest testing'
print('{} takes {} seconds'.format(phrase, (end - start))) #round(end - start, 2)))

time_rf_cv = end - start
```

▸ (60) Spark Jobs

```
Random Forest testing takes 79.80046606063843 seconds
```

Test the Model:

```
# list prediction

predictiontvs = modeltvs.transform(test)

prediction = model.transform(test)

predictionCV = modelCV.transform(test)
```

```python
# list prediction
predictiontvs = modeltvs.transform(test)
prediction = model.transform(test)
predictionCV = modelCV.transform(test)
```

```
Command took 2.10 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project
```

Examine the Predicted and Actual Values:

```
predicted = prediction.select("features","prediction", "trueLabel")

predictedtvs = predictiontvs.select("features","prediction", "trueLabel")

predictedCV = predictionCV.select("features","prediction", "trueLabel")

predictedCV.show(20)

predicted.show(20)

predictedtvs.show(20)
```

```
predicted = prediction.select("features","prediction", "trueLabel")
predictedtvs = predictiontvs.select("features","prediction", "trueLabel")
predictedCV = predictionCV.select("features","prediction", "trueLabel")
predictedCV.show(20)
predicted.show(20)
predictedtvs.show(20)
```

▸ (3) Spark Jobs

```
+--------------------+--------------------+---------+
|            features|          prediction|trueLabel|
+--------------------+--------------------+---------+
|[0.0,0.0,20.69,20...| 0.00472830868366402|      0.0|
|[0.0,0.0,11.21,11...|0.005339254864739365|      0.0|
|[11.0,11.0,13.71,...|0.003289956931697...|      0.0|
|[11.0,11.0,11026....| 0.03843439571042922|      0.0|
|[20.0,0.0,85.29,8...| 0.00472830868366402|      0.0|
|[20.0,0.0,26384.6...|0.039872747462395525|      0.0|
| [20.0,20.0,7.8,7.8]|0.003900903112773051|      0.0|
|[20.0,20.0,360.43...|0.003289956931697...|      0.0|
|[70.0,0.0,36218.6...| 0.01399815808785205|      0.0|
|[214.0,214.0,24.2...|0.001823239105346004|      0.0|
|[214.0,1208.0,284...|  0.0219841751564542|      0.0|
|[544.0,544.0,841....|0.005224183210633434|      0.0|
|[544.0,544.0,16.6...|0.001823239105346004|      0.0|
|[718.0,718.0,9.38...|0.002434185286421349|      0.0|
|[741.0,741.0,2085...| 0.03843439571042922|      0.0|
```

Calculate the Precision and Recall for Random Forest Cross Validator:

```
#Precision and recal for random forest cv
tp = float(predictedCV.filter("prediction >= 0.01 AND truelabel == 1").count())
fp = float(predictedCV.filter("prediction >= 0.01 AND truelabel == 0").count())
tn = float(predictedCV.filter("prediction >= 0.0 AND truelabel == 0").count())
fn = float(predictedCV.filter("prediction >= 0.0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("TP", tp),
 ("FP", fp),
 ("TN", tn),
 ("FN", fn),
 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

```
#Precision and recal for random forest cv

tp = float(predictedCV.filter("prediction >= 0.01 AND truelabel == 1").count())
fp = float(predictedCV.filter("prediction >= 0.01 AND truelabel == 0").count())
tn = float(predictedCV.filter("prediction >= 0.0 AND truelabel == 0").count())
fn = float(predictedCV.filter("prediction >= 0.0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("TP", tp),
 ("FP", fp),
 ("TN", tn),
 ("FN", fn),
 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

▸ (11) Spark Jobs

```
+---------+------------------+
|   metric|             value|
+---------+------------------+
|       TP|              47.0|
|       FP|             875.0|
|       TN|            1664.0|
|       FN|              50.0|
|Precision|0.0509761388286334|
|   Recall|0.4845360824742268|
+---------+------------------+
```

Calculate the Precision and Recall for Random Forest Train Validation Split:

```
#Precision and recal for random forest tvs
tp = float(predictedtvs.filter("prediction >= 0.01 AND truelabel == 1").count())
fp = float(predictedtvs.filter("prediction >= 0.01 AND truelabel == 0").count())
tn = float(predictedtvs.filter("prediction >= 0.0 AND truelabel == 0").count())
fn = float(predictedtvs.filter("prediction >= 0.0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("TP", tp),
 ("FP", fp),
 ("TN", tn),
 ("FN", fn),
 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

```
#Precision and recal for random forest tvs
tp = float(predictedtvs.filter("prediction >= 0.01 AND truelabel == 1").count())
fp = float(predictedtvs.filter("prediction >= 0.01 AND truelabel == 0").count())
tn = float(predictedtvs.filter("prediction >= 0.0 AND truelabel == 0").count())
fn = float(predictedtvs.filter("prediction >= 0.0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
  ("TP", tp),
 ("FP", fp),
 ("TN", tn),
 ("FN", fn),
 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

▸ (11) Spark Jobs

```
+---------+------------------+
|   metric|             value|
+---------+------------------+
|       TP|              46.0|
|       FP|             594.0|
|       TN|            1664.0|
|       FN|              50.0|
|Precision|          0.071875|
|   Recall|0.4791666666666667|
+---------+------------------+
```

Finding the AUC Value of Random Forest:

```
#the auc of the three random forest models

evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")

auc_tvs1_rf = evaluator.evaluate(prediction)

print("AUC = ", auc_tvs1_rf)

evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")

auc_tvs_rf = evaluator.evaluate(predictiontvs)

print("AUC = ", auc_tvs_rf)

evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")

auc_cv_rf = evaluator.evaluate(predictionCV)

print("AUC = ", auc_cv_rf)
```

Python

```
1  #the auc of the three random forest models
2  evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")
3  auc_tvs1_rf = evaluator.evaluate(prediction)
4  print("AUC = ", auc_tvs1_rf)
5
6  evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")
7  auc_tvs_rf = evaluator.evaluate(predictiontvs)
8  print("AUC = ", auc_tvs_rf)
9
10 evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")
11 auc_cv_rf = evaluator.evaluate(predictionCV)
12 print("AUC = ", auc_cv_rf)
13
```

▸ (9) Spark Jobs

```
AUC =  0.8856730769230771
AUC =  0.8702524038461544
AUC =  0.8637439903846159
```

Step 6: Gradient Boost Tree

```
#GBT

gbt = GBTRegressor(labelCol="label", featuresCol="features")
```

```
#GBT

gbt = GBTRegressor(labelCol="label", featuresCol="features")

```

Command took 0.10 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Tune Parameters:

Tuning parameters to find the best model for your data. To do this we are using CrossValidator anf trainValidationSplit class to evaluate each combination of parameters defined in a ParameterGrid against multiple folds of the data split into training and validation datasets, in order to find the best performing parameters.

```
paramGrid2 = (ParamGridBuilder() \
        .addGrid(gbt.maxDepth, [3, 5]) \
        .addGrid(gbt.maxBins, [10, 15]) \
        .addGrid(gbt.minInfoGain, [0.0]) \
        .build())
```

```
paramGrid2 = (ParamGridBuilder() \
              .addGrid(gbt.maxDepth, [3, 5]) \
              .addGrid(gbt.maxBins, [10, 15]) \
              .addGrid(gbt.minInfoGain, [0.0]) \
              .build())

```

Command took 0.04 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Define the Pipeline:

```
#GBT using TrainValidationSplit

pipelinegbt = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5,
strIdx6,catVect,catIdx,assembler, minMax, gbt])


start2 = time()


gbt_tvs = TrainValidationSplit(estimator=pipelinegbt,
evaluator=BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="prediction",
metricName="areaUnderROC"), estimatorParamMaps=paramGrid2, trainRatio=0.8)


model = gbt_tvs.fit(train)


end2 = time()

phrase = 'GBT testing'

print('{} takes {} seconds'.format(phrase, (end2 - start2))) #round(end - start, 2)))


time_gbt_tvs= end - start
```

Python

```
#GBT using TrainValidationSplit
pipelinegbt = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5, strIdx6,catVect,catIdx,assembler, minMax, gbt])

start2 = time()

gbt_tvs = TrainValidationSplit(estimator=pipelinegbt, evaluator=BinaryClassificationEvaluator(labelCol="label",
rawPredictionCol="prediction", metricName="areaUnderROC"), estimatorParamMaps=paramGrid2, trainRatio=0.8)

model = gbt_tvs.fit(train)

end2 = time()
phrase = 'GBT testing'
print('{} takes {} seconds'.format(phrase, (end2 - start2))) #round(end - start, 2)))

time_gbt_tvs= end - start
```

▸ (50) Spark Jobs

```
GBT testing takes 97.19177389144897 seconds
```

Command took 1.62 minutes -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Prediction:

```
prediction_gbt_tvs = model.transform(test)

predicted_gbt_tvs = prediction_gbt_tvs.select("normFeatures", "prediction", "trueLabel")

predicted_gbt_tvs.show()


prediction_gbt_cv = model2.transform(test)

predicted_gbt_cv = prediction_gbt_cv.select("normFeatures", "prediction", "trueLabel")

predicted_gbt_cv.show()
```

Python

```
1  prediction_gbt_tvs = model.transform(test)
2  predicted_gbt_tvs = prediction_gbt_tvs.select("normFeatures", "prediction", "trueLabel")
3  predicted_gbt_tvs.show()
4
5  prediction_gbt_cv = model2.transform(test)
6  predicted_gbt_cv = prediction_gbt_cv.select("normFeatures", "prediction", "trueLabel")
7  predicted_gbt_cv.show()
```

▸ (2) Spark Jobs

```
+--------------------+--------------------+----------+
|        normFeatures|          prediction|trueLabel|
+--------------------+--------------------+----------+
|[0.0,0.0,8.593460...|0.002893057193973484|      0.0|
|[0.0,0.0,4.656002...|0.002893057193973484|      0.0|
|[3.50581966063665...|0.011470633515300048|      0.0|
|[3.50581966063665...|  0.0368942167174964|      0.0|
|[6.37421756479392...|-0.01028854104942413|      0.0|
|[6.37421756479392...|0.035697693437056914|      0.0|
|[6.37421756479392...|-1.86154722278293...|      0.0|
|[6.37421756479392...|-1.86154722278293...|      0.0|
|[2.23097614767787...| 0.00109443685482925|      0.0|
|[6.82041279432949...|-0.00180699010460...|      0.0|
|[6.82041279432949...|0.002479718108519391|      0.0|
|[0.00173378717762...|-3.68863112976126...|      0.0|
|[0.00173378717762...|-0.00180699010460...|      0.0|
```

Calculate the Precision and Recall for Random Forest Train Validation Split:

```
tp = float(prediction_gbt_tvs.filter("prediction >= 0 AND truelabel == 1").count())
fp = float(prediction_gbt_tvs.filter("prediction >= 0 AND truelabel == 0").count())
tn = float(prediction_gbt_tvs.filter("prediction <= 0 AND truelabel == 0").count())
fn = float(prediction_gbt_tvs.filter("prediction <= 0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("Precision", tp / (tp + tn)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

Python

```
1  tp = float(prediction_gbt_tvs.filter("prediction >= 0 AND truelabel == 1").count())
2  fp = float(prediction_gbt_tvs.filter("prediction >= 0 AND truelabel == 0").count())
3  tn = float(prediction_gbt_tvs.filter("prediction <= 0 AND truelabel == 0").count())
4  fn = float(prediction_gbt_tvs.filter("prediction <= 0 AND truelabel == 1").count())
5  metrics2 = spark.createDataFrame([
6   ("Precision", tp / (tp + tn)),
7   ("Recall", tp / (tp + fn))],["metric", "value"])
8  metrics2.show()
```

▸ (11) Spark Jobs

```
+---------+-------------------+
|   metric|              value|
+---------+-------------------+
|Precision|0.08431703204047218|
|   Recall|                1.0|
+---------+-------------------+
```

Command took 3.10 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Calculate the Precision and Recall for Random Forest Cross Validator:

```
tp = float(prediction_gbt_cv.filter("prediction >= 0 AND truelabel == 1").count())
fp = float(prediction_gbt_cv.filter("prediction >= 0 AND truelabel == 0").count())
tn = float(prediction_gbt_cv.filter("prediction <= 0 AND truelabel == 0").count())
fn = float(prediction_gbt_cv.filter("prediction <= 0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("Precision", tp / (tp + tn)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

```python
tp = float(prediction_gbt_cv.filter("prediction >= 0 AND truelabel == 1").count())
fp = float(prediction_gbt_cv.filter("prediction >= 0 AND truelabel == 0").count())
tn = float(prediction_gbt_cv.filter("prediction <= 0 AND truelabel == 0").count())
fn = float(prediction_gbt_cv.filter("prediction <= 0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("Precision", tp / (tp + tn)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

(11) Spark Jobs

```
+---------+-------------------+
|   metric|              value|
+---------+-------------------+
|Precision|0.07492354740061162|
|   Recall|               0.98|
+---------+-------------------+
```

Command took 1.84 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Finding the AUC Value of Gradient Boost Tree:

#AUC for the GBT

evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")

auc_tvs_gbt = evaluator.evaluate(prediction_gbt_tvs)

print("AUC = ", auc_tvs_gbt)

evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")

auc_cv_gbt = evaluator.evaluate(prediction_gbt_cv)

print("AUC = ", auc_cv_gbt)

```python
#AUC for the GBT
evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")
auc_tvs_gbt = evaluator.evaluate(prediction_gbt_tvs)
print("AUC = ", auc_tvs_gbt)

evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")
auc_cv_gbt = evaluator.evaluate(prediction_gbt_cv)
print("AUC = ", auc_cv_gbt)
```

(6) Spark Jobs

```
AUC =  0.913707932692308
AUC =  0.9016526442307693
```

Command took 0.96 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Step 7: Factorization Machine

```
#FM

fm = FMRegressor(labelCol="label", featuresCol="normFeatures")
```

```
#FM
fm = FMRegressor(labelCol="label", featuresCol="normFeatures")
```

Command took 0.09 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Tune Parameters:

```
paramGrid2 = (ParamGridBuilder() \

        .addGrid(fm.maxIter, [5, 10])\

        .build())
```

```
paramGrid2 = (ParamGridBuilder() \
              .addGrid(fm.maxIter, [5, 10])\
              .build())

```

Command took 0.04 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Define the Pipeline:

```
#FM using TrainValidationSplit

pipelinefm = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5,
strIdx6,catVect,catIdx,assembler, minMax, fm])


start3 = time()


fm_tvs = TrainValidationSplit(estimator=pipelinefm,
evaluator=BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="prediction",
metricName="areaUnderROC"), estimatorParamMaps=paramGrid2, trainRatio=0.8)




# the second best model

model = fm_tvs.fit(train)


end3 = time()

phrase = 'FM testing'

print('{} takes {} seconds'.format(phrase, (end3 - start3))) #round(end - start, 2)))


time_fm_tvs= end - start
```

```python
#FM using TrainValidationSplit
pipelinefm = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5, strIdx6,catVect,catIdx,assembler, minMax, fm])

start3 = time()

fm_tvs = TrainValidationSplit(estimator=pipelinefm, evaluator=BinaryClassificationEvaluator(labelCol="label",
rawPredictionCol="prediction", metricName="areaUnderROC"), estimatorParamMaps=paramGrid2, trainRatio=0.8)


# the second best model
model = fm_tvs.fit(train)

end3 = time()
phrase = 'FM testing'
print('{} takes {} seconds'.format(phrase, (end3 - start3))) #round(end - start, 2)))

time_fm_tvs= end - start
```

▸ (82) Spark Jobs

```
FM testing takes 19.095678329467773 seconds
```

Command took 19.16 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

```
#FM using CrossValidator


start3 = time()


fm_cv = CrossValidator(estimator=pipelinefm,
evaluator=BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="prediction",
metricName="areaUnderROC"), estimatorParamMaps=paramGrid2, numFolds=K)


model2 = fm_cv.fit(train)


end3 = time()

phrase = 'FM testing'

print('{} takes {} seconds'.format(phrase, (end3 - start3))) #round(end - start, 2)))


time_fm_cv= end - start
```

```python
#FM using CrossValidator

start3 = time()

fm_cv = CrossValidator(estimator=pipelinefm, evaluator=BinaryClassificationEvaluator(labelCol="label",
rawPredictionCol="prediction", metricName="areaUnderROC"), estimatorParamMaps=paramGrid2, numFolds=K)

model2 = fm_cv.fit(train)

end3 = time()
phrase = 'FM testing'
print('{} takes {} seconds'.format(phrase, (end3 - start3))) #round(end - start, 2)))

time_fm_cv= end - start
```

▸ (76) Spark Jobs

```
FM testing takes 35.307175636291504 seconds
```

Command took 35.36 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Prediction:

```
prediction_fm_tvs = model.transform(test)

predicted_fm_tvs = prediction_fm_tvs.select("normFeatures", "prediction", "trueLabel")

predicted_fm_tvs.show()


prediction_fm_cv = model2.transform(test)

predicted_fm_cv = prediction_fm_cv.select("normFeatures", "prediction", "trueLabel")

predicted_fm_cv.show()
```

```
prediction_fm_tvs = model.transform(test)
predicted_fm_tvs = prediction_fm_tvs.select("normFeatures", "prediction", "trueLabel")
predicted_fm_tvs.show()

prediction_fm_cv = model2.transform(test)
predicted_fm_cv = prediction_fm_cv.select("normFeatures", "prediction", "trueLabel")
predicted_fm_cv.show()
```

▸ (2) Spark Jobs

```
+--------------------+-------------------+---------+
|        normFeatures|         prediction|trueLabel|
+--------------------+-------------------+---------+
|[0.0,0.0,8.593460...| 0.4130625233665066|      0.0|
|[0.0,0.0,4.656002...| 0.4130625245148541|      0.0|
|[3.50581966063665...| 0.4130906812354846|      0.0|
|[3.50581966063665...|0.41308934827658067|      0.0|
|[6.37421756479392...|0.41310728000317326|      0.0|
|[6.37421756479392...|0.41310410075453646|      0.0|
|[6.37421756479392...| 0.4131137149206395|      0.0|
|[6.37421756479392...| 0.4131136722661596|      0.0|
|[2.23097614767787...| 0.4132148444738608|      0.0|
|[6.82041279432949...|0.41360919622105397|      0.0|
```

Calculate the Precision and Recall for Random Forest Train Validation Split:

```
#Precision and Recall for fm tvs

tp = float(prediction_fm_tvs.filter("prediction >= 0.48 AND truelabel == 1").count())

fp = float(prediction_fm_tvs.filter("prediction >= 0.48 AND truelabel == 0").count())

tn = float(prediction_fm_tvs.filter("prediction <= 0.49 AND truelabel == 0").count())

fn = float(prediction_fm_tvs.filter("prediction <= 0.49 AND truelabel == 1").count())

metrics2 = spark.createDataFrame([

 ("Precision", tp / (tp + tn)),

 ("Recall", tp / (tp + fn))],["metric", "value"])

metrics2.show()
```

```
#Precision and Recall for fm tvs
tp = float(prediction_fm_tvs.filter("prediction >= 0.48 AND truelabel == 1").count())
fp = float(prediction_fm_tvs.filter("prediction >= 0.48 AND truelabel == 0").count())
tn = float(prediction_fm_tvs.filter("prediction <= 0.49 AND truelabel == 0").count())
fn = float(prediction_fm_tvs.filter("prediction <= 0.49 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("Precision", tp / (tp + tn)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

▸ (11) Spark Jobs

```
+---------+-------------------+
|   metric|              value|
+---------+-------------------+
|Precision|0.011111111111111112|
|   Recall| 0.34615384615384615|
+---------+-------------------+
```

Command took 2.69 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Calculate the Precision and Recall for Random Forest Cross Validator:

```
#Precision and Recall for fm cv


tp = float(prediction_fm_cv.filter("prediction >= 0.48 AND truelabel == 1").count())
fp = float(prediction_fm_cv.filter("prediction >= 0.48 AND truelabel == 0").count())
tn = float(prediction_fm_cv.filter("prediction <= 0.49 AND truelabel == 0").count())
fn = float(prediction_fm_cv.filter("prediction <= 0.49 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("Precision", tp / (tp + tn)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

```
#Precision and Recall for fm cv

tp = float(prediction_fm_cv.filter("prediction >= 0.48 AND truelabel == 1").count())
fp = float(prediction_fm_cv.filter("prediction >= 0.48 AND truelabel == 0").count())
tn = float(prediction_fm_cv.filter("prediction <= 0.49 AND truelabel == 0").count())
fn = float(prediction_fm_cv.filter("prediction <= 0.49 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("Precision", tp / (tp + tn)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

▸ (11) Spark Jobs

```
+---------+--------------------+
|   metric|               value|
+---------+--------------------+
|Precision|0.011111111111111112|
|   Recall| 0.34615384615384615|
+---------+--------------------+
```

Command took 2.85 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Finding the AUC Value of Factorization Machine:

```
#auc of FM

evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")

auc_tvs_fm = evaluator.evaluate(predicted_fm_tvs)

print("AUC = ", auc_tvs_fm)


evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")

auc_cv_fm = evaluator.evaluate(predicted_fm_cv)

print("AUC = ", auc_cv_fm)
```

```python
#auc of FM
evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")
auc_tvs_fm = evaluator.evaluate(predicted_fm_tvs)
print("AUC = ", auc_tvs_fm)

evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")
auc_cv_fm = evaluator.evaluate(predicted_fm_cv)
print("AUC = ", auc_cv_fm)
```

(6) Spark Jobs

```
AUC =  0.7187259615384515
AUC =  0.7187259615384515
```

Command took 1.14 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Step 8: Support Vector Machines

```
lsvc = LinearSVC(labelCol="label", featuresCol="features", maxIter=50)


pipelinesvc = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5,
strIdx6,catVect,catIdx,assembler, minMax, lsvc])


#SVM with CrossValidator

start4 = time()

svc_cv = CrossValidator(estimator=pipelinesvc,
evaluator=BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="prediction",
metricName="areaUnderROC"), estimatorParamMaps=paramGrid2, numFolds=3)


modelsvc_cv = svc_cv.fit(train)


end4 = time()

phrase = 'SVM testing'

print('{} takes {} seconds'.format(phrase, (end4 - start4))) #round(end - start, 2)))


time_svm_cv= end - start
```

```python
lsvc = LinearSVC(labelCol="label", featuresCol="features", maxIter=50)

pipelinesvc = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5, strIdx6,catVect,catIdx,assembler, minMax, lsvc])

#SVM with CrossValidator

start4 = time()

svc_cv = CrossValidator(estimator=pipelinesvc, evaluator=BinaryClassificationEvaluator(labelCol="label",
rawPredictionCol="prediction", metricName="areaUnderROC"), estimatorParamMaps=paramGrid2, numFolds=3)

modelsvc_cv = svc_cv.fit(train)

end4 = time()
phrase = 'SVM testing'
print('{} takes {} seconds'.format(phrase, (end4 - start4))) #round(end - start, 2)))

time_svm_cv= end - start

# svc_tvs = TrainValidationSplit(estimator=pipelinesvc, evaluator=RegressionEvaluator(), estimatorParamMaps=paramGrid2,
trainRatio=0.8)
# svc_cv = CrossValidator(estimator=pipelinesvc, evaluator=RegressionEvaluator(), estimatorParamMaps=paramGrid2, numFolds=3)
```

▸ (100) Spark Jobs

```
SVM testing takes 139.36338591575623 seconds
```

Command took 2.33 minutes -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

```
#SVM with TrainValidationSplit

start4 = time()


svc_tvs = TrainValidationSplit(estimator=pipelinesvc,
evaluator=BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="prediction",
metricName="areaUnderROC"), estimatorParamMaps=paramGrid2, trainRatio=0.8)


modelsvc_tvs = svc_tvs.fit(train)


end4 = time()

phrase = 'SVM testing'

print('{} takes {} seconds'.format(phrase, (end4 - start4))) #round(end - start, 2)))


time_svm_tvs= end - start
```

```python
#SVM with TrainValidationSplit
start4 = time()

svc_tvs = TrainValidationSplit(estimator=pipelinesvc, evaluator=BinaryClassificationEvaluator(labelCol="label",
rawPredictionCol="prediction", metricName="areaUnderROC"), estimatorParamMaps=paramGrid2, trainRatio=0.8)

modelsvc_tvs = svc_tvs.fit(train)

end4 = time()
phrase = 'SVM testing'
print('{} takes {} seconds'.format(phrase, (end4 - start4))) #round(end - start, 2)))

time_svm_tvs= end - start
```

(100) Spark Jobs

```
SVM testing takes 47.452725410461426 seconds
```

Command took 47.52 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

```
predictionSVM_tvs = modelsvc_tvs.transform(test)

predictedSVM_tvs = predictionSVM_tvs.select("normFeatures", "prediction", "trueLabel")

predictedSVM_tvs.show()


predictionSVM_cv = modelsvc_cv.transform(test)

predictedSVM_cv = predictionSVM_cv.select("normFeatures", "prediction", "trueLabel")

predictedSVM_cv.show()
```

```
predictionSVM_tvs = modelsvc_tvs.transform(test)
predictedSVM_tvs = predictionSVM_tvs.select("normFeatures", "prediction", "trueLabel")
predictedSVM_tvs.show()

predictionSVM_cv = modelsvc_cv.transform(test)
predictedSVM_cv = predictionSVM_cv.select("normFeatures", "prediction", "trueLabel")
predictedSVM_cv.show()
```

▸ (2) Spark Jobs

```
+--------------------+----------+---------+
|        normFeatures|prediction|trueLabel|
+--------------------+----------+---------+
|[0.0,0.0,8.593460...|       0.0|      0.0|
|[0.0,0.0,4.656002...|       0.0|      0.0|
|[3.50581966063665...|       0.0|      0.0|
|[3.50581966063665...|       0.0|      0.0|
|[6.37421756479392...|       0.0|      0.0|
|[6.37421756479392...|       0.0|      0.0|
|[6.37421756479392...|       0.0|      0.0|
|[6.37421756479392...|       0.0|      0.0|
|[2.23097614767787...|       0.0|      0.0|
|[6.82041279432949...|       0.0|      0.0|
|[6.82041279432949...|       0.0|      0.0|
|[0.00173378717762...|       0.0|      0.0|
|[0.00173378717762...|       0.0|      0.0|
|[0.00228834410576...|       0.0|      0.0|
```

Calculate the Precision and Recall for SVM Cross Validator:

```
#Precision and Recall for svm cv


tp = float(predictionSVM_cv.filter("prediction <= 1.0 AND truelabel == 1").count())

fp = float(predictionSVM_cv.filter("prediction <= 1.0 AND truelabel == 0").count())

tn = float(predictionSVM_cv.filter("prediction == 0.0 AND truelabel == 0").count())

fn = float(predictionSVM_cv.filter("prediction == 0.0 AND truelabel == 1").count())

metrics2 = spark.createDataFrame([

 ("Precision", tp / (tp + fp)),

 ("Recall", tp / (tp + fn))],["metric", "value"])

metrics2.show()
```

```
#Precision and Recall for svm tvs

tp = float(predictionSVM_cv.filter("prediction <= 1.0 AND truelabel == 1").count())
fp = float(predictionSVM_cv.filter("prediction <= 1.0 AND truelabel == 0").count())
tn = float(predictionSVM_cv.filter("prediction == 0.0 AND truelabel == 0").count())
fn = float(predictionSVM_cv.filter("prediction == 0.0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

▸ (11) Spark Jobs

```
+---------+--------------------+
|   metric|               value|
+---------+--------------------+
|Precision|0.029171528588098017|
|   Recall|                 0.5|
+---------+--------------------+
```

Command took 3.10 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Calculate the Precision and Recall for SVM Train Validation Split:

```
#Precision and Recall for svm tvs


tp = float(predictionSVM_tvs.filter("prediction <= 1.0 AND truelabel == 1").count())
fp = float(predictionSVM_tvs.filter("prediction <= 1.0 AND truelabel == 0").count())
tn = float(predictionSVM_tvs.filter("prediction == 0.0 AND truelabel == 0").count())
fn = float(predictionSVM_tvs.filter("prediction == 0.0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

```
#Precision and Recall for svm tvs

tp = float(predictionSVM_tvs.filter("prediction <= 1.0 AND truelabel == 1").count())
fp = float(predictionSVM_tvs.filter("prediction <= 1.0 AND truelabel == 0").count())
tn = float(predictionSVM_tvs.filter("prediction == 0.0 AND truelabel == 0").count())
fn = float(predictionSVM_tvs.filter("prediction == 0.0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

▸ (11) Spark Jobs

```
+---------+--------------------+
|   metric|               value|
+---------+--------------------+
|Precision|0.029171528588098017|
|   Recall|                 0.5|
+---------+--------------------+
```

Command took 1.96 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

AUC Values:

evaluatorSVM_tvs = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")

auc_SVM_tvs = evaluatorSVM_tvs.evaluate(predictionSVM_tvs)

print("AUC = ", auc_SVM_tvs)

evaluatorSVM_cv = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")

auc_SVM_cv = evaluatorSVM_cv.evaluate(predictionSVM_cv)

print("AUC = ", auc_SVM_cv)

Python

```
evaluatorSVM_tvs = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction",
metricName="areaUnderROC")
auc_SVM_tvs = evaluatorSVM_tvs.evaluate(predictionSVM_tvs)
print("AUC = ", auc_SVM_tvs)

evaluatorSVM_cv = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction",
metricName="areaUnderROC")
auc_SVM_cv = evaluatorSVM_cv.evaluate(predictionSVM_cv)
print("AUC = ", auc_SVM_cv)
```

▸ (6) Spark Jobs

```
AUC =  0.5
AUC =  0.5
```

Command took 1.23 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Step 9: Logistic Regression

```
lr = LogisticRegression(labelCol="label",featuresCol="normFeatures",maxIter=10,regParam=0.3)
```

```
1  lr = LogisticRegression(labelCol="label",featuresCol="normFeatures",maxIter=10,regParam=0.3)
2
```

Command took 0.10 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Tuning Parameters:

```
#paramGrid2 = (ParamGridBuilder() \
#             .build())
paramGrid2=(ParamGridBuilder()
            .addGrid(lr.regParam, [0.01, 0.1, 0.5, 1.0, 2.0])
            .addGrid(lr.elasticNetParam, [0.0, 0.25, 0.5, 0.75, 1.0])
            .addGrid(lr.maxIter, [1, 5, 10, 20, 50])
            .build())
```

```
1  #paramGrid2 = (ParamGridBuilder() \
2  #             .build())
3
4  paramGrid2=(ParamGridBuilder()
5              .addGrid(lr.regParam, [0.01, 0.1, 0.5, 1.0, 2.0])
6              .addGrid(lr.elasticNetParam, [0.0, 0.25, 0.5, 0.75, 1.0])
7              .addGrid(lr.maxIter, [1, 5, 10, 20, 50])
8              .build())
```

Command took 0.02 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Define the Pipeline:

```
#LogisticRegression with TrainValidationSplit

pipelinelr = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5,
strIdx6,catVect,catIdx,assembler, minMax, lr])

start5 = time()


lr_tvs = TrainValidationSplit(estimator=pipelinelr, evaluator=RegressionEvaluator(),
estimatorParamMaps=paramGrid2, trainRatio=0.8)


# the second best model

model = lr_tvs.fit(train)


end5 = time()

phrase = 'Logistic Regression testing'

print('{} takes {} seconds'.format(phrase, (end5 - start5))) #round(end - start, 2)))


time_lr_tvs= end - start
```

Python

```
1  #LogisticRegression with TrainValidationSplit
2  pipelinelr = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5, strIdx6,catVect,catIdx,assembler, minMax, lr])
3
4  start5 = time()
5
6  lr_tvs = TrainValidationSplit(estimator=pipelinelr, evaluator=RegressionEvaluator(), estimatorParamMaps=paramGrid2,
   trainRatio=0.8)
7
8  # the second best model
9  model = lr_tvs.fit(train)
10
11 end5 = time()
12 phrase = 'Logistic Regression testing'
13 print('{} takes {} seconds'.format(phrase, (end5 - start5))) #round(end - start, 2)))
14
15 time_lr_tvs= end - start
16
```

▸ (79) Spark Jobs

```
Logistic Regression testing takes 516.7596139907837 seconds
```

Command took 8.61 minutes -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

##LogisticRegression with CrossValidator

start5 = time()

lr_cv = CrossValidator(estimator=pipelinelr, evaluator=RegressionEvaluator(), estimatorParamMaps=paramGrid2, numFolds=3)

model2 = lr_cv.fit(train)

end5 = time()

phrase = 'Logistic Regression testing'

print('{} takes {} seconds'.format(phrase, (end5 - start5))) #round(end - start, 2)))

time_lr_cv= end - start

Python

```
1  ##LogisticRegression with CrossValidator
2
3  start5 = time()
4
5  lr_cv = CrossValidator(estimator=pipelinelr, evaluator=RegressionEvaluator(), estimatorParamMaps=paramGrid2, numFolds=3)
6
7  model2 = lr_cv.fit(train)
8
9  end5 = time()
10 phrase = 'Logistic Regression testing'
11 print('{} takes {} seconds'.format(phrase, (end5 - start5))) #round(end - start, 2)))
12
13 time_lr_cv= end - start
14
```

▸ (93) Spark Jobs

```
Logistic Regression testing takes 1422.19579911232 seconds
```

Command took 23.70 minutes -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Prediction:

```
prediction_lr_tvs = model.transform(test)

predicted_lr_tvs = prediction_lr_tvs.select("normFeatures", "prediction", "trueLabel")

predicted_lr_tvs.show()


prediction_lr_cv = model2.transform(test)

predicted_lr_cv = prediction_lr_cv.select("normFeatures", "prediction", "trueLabel")

predicted_lr_cv.show()
```

Python

```
1  prediction_lr_tvs = model.transform(test)
2  predicted_lr_tvs = prediction_lr_tvs.select("normFeatures", "prediction", "trueLabel")
3  predicted_lr_tvs.show()
4
5  prediction_lr_cv = model2.transform(test)
6  predicted_lr_cv = prediction_lr_cv.select("normFeatures", "prediction", "trueLabel")
7  predicted_lr_cv.show()
8
```

▸ (2) Spark Jobs

```
+--------------------+----------+---------+
|        normFeatures|prediction|trueLabel|
+--------------------+----------+---------+
|[0.0,0.0,8.593460...|       0.0|      0.0|
|[0.0,0.0,4.656002...|       0.0|      0.0|
|[3.50581966063665...|       0.0|      0.0|
|[3.50581966063665...|       0.0|      0.0|
|[6.37421756479392...|       0.0|      0.0|
|[6.37421756479392...|       0.0|      0.0|
|[6.37421756479392...|       0.0|      0.0|
|[6.37421756479392...|       0.0|      0.0|
|[2.23097614767787...|       0.0|      0.0|
|[6.82041279432949...|       0.0|      0.0|
|[6.82041279432949...|       0.0|      0.0|
```

Calculating Precision and Recall Values for Logistic Regression Cross Validation:

```
# Precision and Recall for lr cv
tp = float(prediction_lr_cv.filter("prediction >= 0 AND truelabel == 1").count())
fp = float(prediction_lr_cv.filter("prediction >= 0 AND truelabel == 0").count())
tn = float(prediction_lr_cv.filter("prediction <= 0 AND truelabel == 0").count())
fn = float(prediction_lr_cv.filter("prediction <= 0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

```
1  # Precision and Recall for lr cv
2  tp = float(prediction_lr_cv.filter("prediction >= 0 AND truelabel == 1").count())
3  fp = float(prediction_lr_cv.filter("prediction >= 0 AND truelabel == 0").count())
4  tn = float(prediction_lr_cv.filter("prediction <= 0 AND truelabel == 0").count())
5  fn = float(prediction_lr_cv.filter("prediction <= 0 AND truelabel == 1").count())
6  metrics2 = spark.createDataFrame([
7   ("Precision", tp / (tp + fp)),
8   ("Recall", tp / (tp + fn))],["metric", "value"])
9  metrics2.show()
```

▸ (11) Spark Jobs

```
+---------+--------------------+
|   metric|               value|
+---------+--------------------+
|Precision|0.029171528588098017|
|   Recall|  0.5154639175257731|
+---------+--------------------+
```

Command took 2.53 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Calculating Precision and Recall Values for Logistic Regression Train Validation Split:

```
# Precision and Recall for lr tvs
tp = float(prediction_lr_tvs.filter("prediction >= 0 AND truelabel == 1").count())
fp = float(prediction_lr_tvs.filter("prediction >= 0 AND truelabel == 0").count())
tn = float(prediction_lr_tvs.filter("prediction <= 0 AND truelabel == 0").count())
fn = float(prediction_lr_tvs.filter("prediction <= 0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

```
1  # Precision and Recall for lr tvs
2  tp = float(prediction_lr_tvs.filter("prediction >= 0 AND truelabel == 1").count())
3  fp = float(prediction_lr_tvs.filter("prediction >= 0 AND truelabel == 0").count())
4  tn = float(prediction_lr_tvs.filter("prediction <= 0 AND truelabel == 0").count())
5  fn = float(prediction_lr_tvs.filter("prediction <= 0 AND truelabel == 1").count())
6  metrics2 = spark.createDataFrame([
7   ("Precision", tp / (tp + fp)),
8   ("Recall", tp / (tp + fn))],["metric", "value"])
9  metrics2.show()
```

▸ (11) Spark Jobs

```
+---------+-------------------+
|   metric|              value|
+---------+-------------------+
|Precision|0.029171528588098017|
|   Recall|  0.5154639175257731|
+---------+-------------------+
```

Command took 1.87 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Calculating AUC Values:

```
evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")

auc_tvs_lr = evaluator.evaluate(prediction_lr_tvs)

print("AUC = ", auc_tvs_lr)


evaluator1 = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")

auc_cv_lr = evaluator1.evaluate(prediction_lr_cv)

print("AUC = ", auc_cv_lr)
```

Python

```
1  evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")
2  auc_tvs_lr = evaluator.evaluate(prediction_lr_tvs)
3  print("AUC = ", auc_tvs_lr)
4
5  evaluator1 = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")
6  auc_cv_lr = evaluator1.evaluate(prediction_lr_cv)
7  print("AUC = ", auc_cv_lr)
8
```

▸ (6) Spark Jobs

```
AUC =  0.5296995192307693
AUC =  0.53
```

Command took 0.88 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Step 10: Decision Tree Classifier

```
dt = DecisionTreeClassifier(labelCol="label", featuresCol="features")
```

```
1  dt = DecisionTreeClassifier(labelCol="label", featuresCol="features")
2
```

Command took 0.07 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Tuning Parameters:

```
paramGrid2=(ParamGridBuilder()
        .addGrid(dt.maxDepth, [2, 5, 10, 20, 30])
        .addGrid(dt.maxBins, [10, 20, 40, 80, 100])
        .build())
```

```
1  paramGrid2=(ParamGridBuilder()
2              .addGrid(dt.maxDepth, [2, 5, 10, 20, 30])
3              .addGrid(dt.maxBins, [10, 20, 40, 80, 100])
4              .build())
```

Command took 0.04 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Define Pipeline for Decision Tree Classifier with Train Validation Split:

```
#DecisionTreeClassifier with TrainValidationSplit

pipelinedt = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5,
strIdx6,catVect,catIdx,assembler, minMax, dt])


start6 = time()


dt_tvs = TrainValidationSplit(estimator=pipelinedt, evaluator=RegressionEvaluator(),
estimatorParamMaps=paramGrid2, trainRatio=0.8)


# the second best model

model = dt_tvs.fit(train)


end6 = time()

phrase = 'Decision Tree testing'

print('{} takes {} seconds'.format(phrase, (end6 - start6))) #round(end - start, 2)))


time_dt_tvs= end - start
```

Python

```
1  #DecisionTreeClassifier with TrainValidationSplit
2  pipelinedt = Pipeline(stages=[strIdx1, strIdx2, strIdx3, strIdx4, strIdx5, strIdx6,catVect,catIdx,assembler, minMax, dt])
3
4  start6 = time()
5
6  dt_tvs = TrainValidationSplit(estimator=pipelinedt, evaluator=RegressionEvaluator(), estimatorParamMaps=paramGrid2,
   trainRatio=0.8)
7
8  # the second best model
9  model = dt_tvs.fit(train)
10
11 end6 = time()
12 phrase = 'Decision Tree testing'
13 print('{} takes {} seconds'.format(phrase, (end6 - start6))) #round(end - start, 2)))
14
15 time_dt_tvs= end - start
16
```

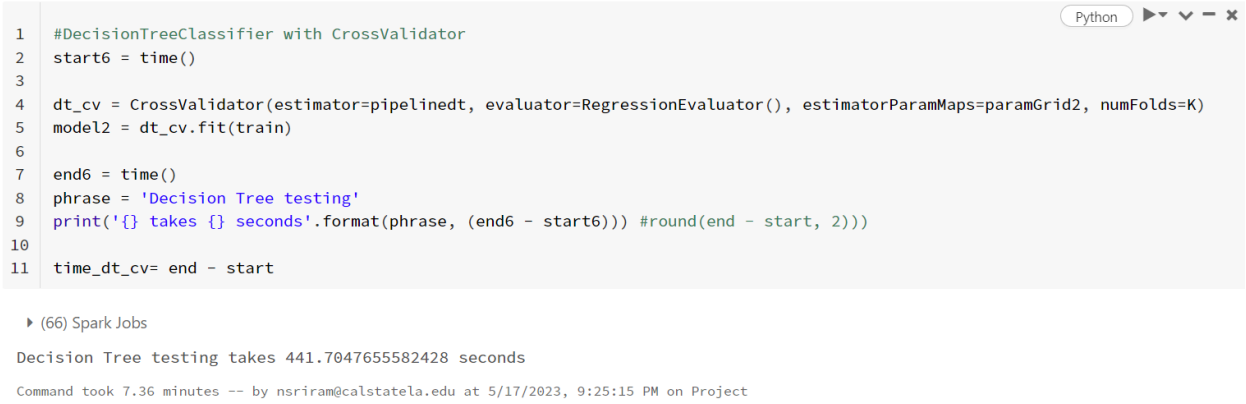
▸ (64) Spark Jobs

```
Decision Tree testing takes 144.8159306049347 seconds
```

Command took 2.42 minutes -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Define Pipeline for Decision Tree Classifier with Cross Validation:

```
#DecisionTreeClassifier with CrossValidator

start6 = time()


dt_cv = CrossValidator(estimator=pipelinedt, evaluator=RegressionEvaluator(),
estimatorParamMaps=paramGrid2, numFolds=K)

model2 = dt_cv.fit(train)


end6 = time()

phrase = 'Decision Tree testing'

print('{} takes {} seconds'.format(phrase, (end6 - start6))) #round(end - start, 2)))


time_dt_cv= end - start
```



```
#DecisionTreeClassifier with CrossValidator
start6 = time()

dt_cv = CrossValidator(estimator=pipelinedt, evaluator=RegressionEvaluator(), estimatorParamMaps=paramGrid2, numFolds=K)
model2 = dt_cv.fit(train)

end6 = time()
phrase = 'Decision Tree testing'
print('{} takes {} seconds'.format(phrase, (end6 - start6))) #round(end - start, 2)))

time_dt_cv= end - start
```

```
(66) Spark Jobs
Decision Tree testing takes 441.7047655582428 seconds
Command took 7.36 minutes -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project
```

Prediction:

```
prediction_dt_tvs = model.transform(test)

predicted_dt_tvs = prediction_dt_tvs.select("normFeatures", "prediction", "trueLabel")

predicted_dt_tvs.show()

prediction_dt_cv = model2.transform(test)

predicted_dt_cv = prediction_dt_cv.select("normFeatures", "prediction", "trueLabel")

predicted_dt_cv.show()
```

```
prediction_dt_tvs = model.transform(test)
predicted_dt_tvs = prediction_dt_tvs.select("normFeatures", "prediction", "trueLabel")
predicted_dt_tvs.show()

prediction_dt_cv = model2.transform(test)
predicted_dt_cv = prediction_dt_cv.select("normFeatures", "prediction", "trueLabel")
predicted_dt_cv.show()
```

▸ (2) Spark Jobs

```
+--------------------+----------+---------+
|        normFeatures|prediction|trueLabel|
+--------------------+----------+---------+
|[0.0,0.0,8.593460...|       0.0|      0.0|
|[0.0,0.0,4.656002...|       0.0|      0.0|
|[3.50581966063665...|       0.0|      0.0|
|[3.50581966063665...|       0.0|      0.0|
|[6.37421756479392...|       0.0|      0.0|
|[6.37421756479392...|       0.0|      0.0|
|[6.37421756479392...|       0.0|      0.0|
|[6.37421756479392...|       0.0|      0.0|
|[2.23097614767787...|       0.0|      0.0|
|[6.82041279432949...|       0.0|      0.0|
|[6.82041279432949...|       0.0|      0.0|
|[0.00173378717762...|       0.0|      0.0|
|[0.00173378717762...|       0.0|      0.0|
|[0.00228834410576...|       0.0|      0.0|
```

Calculating the Precision and Recall Values for Decision Tree Classifier with TVS:

```
tp = float(prediction_dt_tvs.filter("prediction <= 1.0 AND truelabel == 1").count())

fp = float(prediction_dt_tvs.filter("prediction <= 1.0 AND truelabel == 0").count())

tn = float(prediction_dt_tvs.filter("prediction == 0.0 AND truelabel == 0").count())

fn = float(prediction_dt_tvs.filter("prediction == 0.0 AND truelabel == 1").count())

metrics2 = spark.createDataFrame([


 ("Precision", tp / (tp + fp)),

 ("Recall", tp / (tp + fn))],["metric", "value"])

metrics2.show()
```

```
tp = float(prediction_dt_tvs.filter("prediction <= 1.0 AND truelabel == 1").count())
fp = float(prediction_dt_tvs.filter("prediction <= 1.0 AND truelabel == 0").count())
tn = float(prediction_dt_tvs.filter("prediction == 0.0 AND truelabel == 0").count())
fn = float(prediction_dt_tvs.filter("prediction == 0.0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([

 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

▸ (11) Spark Jobs

```
+---------+--------------------+
|   metric|               value|
+---------+--------------------+
|Precision|0.029171528588098017|
|   Recall|  0.5882352941176471|
+---------+--------------------+
```

Command took 3.80 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Calculating the Precision and Recall Values for Decision Tree Classifier with CV:

```
tp = float(prediction_dt_cv.filter("prediction <= 1.0 AND truelabel == 1").count())
fp = float(prediction_dt_cv.filter("prediction <= 1.0 AND truelabel == 0").count())
tn = float(prediction_dt_cv.filter("prediction == 0.0 AND truelabel == 0").count())
fn = float(prediction_dt_cv.filter("prediction == 0.0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

```
tp = float(prediction_dt_cv.filter("prediction <= 1.0 AND truelabel == 1").count())
fp = float(prediction_dt_cv.filter("prediction <= 1.0 AND truelabel == 0").count())
tn = float(prediction_dt_cv.filter("prediction == 0.0 AND truelabel == 0").count())
fn = float(prediction_dt_cv.filter("prediction == 0.0 AND truelabel == 1").count())
metrics2 = spark.createDataFrame([
 ("Precision", tp / (tp + fp)),
 ("Recall", tp / (tp + fn))],["metric", "value"])
metrics2.show()
```

▸ (11) Spark Jobs

```
+---------+--------------------+
|   metric|               value|
+---------+--------------------+
|Precision|0.029171528588098017|
|   Recall|                 0.5|
+---------+--------------------+
```

Command took 2.42 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project

Calculate the TVS & CV AUC Values of Decision Tree Classifier:

```
#AUC of Decision Tree

evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")

auc_tvs_dt = evaluator.evaluate(predicted_dt_tvs)

print("AUC = ", auc_tvs_dt)


evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")

auc_cv_dt = evaluator.evaluate(predicted_dt_cv)

print("AUC = ", auc_cv_dt)
```

```python
#AUC of Decision Tree
evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")
auc_tvs_dt = evaluator.evaluate(predicted_dt_tvs)
print("AUC = ", auc_tvs_dt)

evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")
auc_cv_dt = evaluator.evaluate(predicted_dt_cv)
print("AUC = ", auc_cv_dt)
```

```
▸ (6) Spark Jobs
AUC =  0.6415865384615385
AUC =  0.5
Command took 2.23 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:15 PM on Project
```

Step 11: Compare the Results of all Algorithms Used

```
#all metrics in a tabular format

metrics = spark.createDataFrame([

 ("auc_tvs1_rf", auc_tvs1_rf),

 ("auc_tvs_rf", auc_tvs_rf),

 ("auc_cv_rf", auc_cv_rf),

 ("auc_tvs_gbt", auc_tvs_gbt),

 ("auc_cv_gbt", auc_cv_gbt),

 ("auc_tvs_fm", auc_tvs_fm),

 ("auc_cv_fm",auc_cv_fm),

 ("auc_tvs_svm", auc_SVM_tvs),

 ("auc_cv_svm", auc_SVM_cv),

 ("auc_tvs_lr",auc_tvs_lr),

 ("auc_cv_lr",auc_cv_lr),

 ("auc_tvs_dt",auc_tvs_dt),

 ("auc_cv_dt",auc_cv_dt),


 ],["metric", "value"])


metrics.show()
```

```
#all metrics in a tabular format
metrics = spark.createDataFrame([
 ("auc_tvs1_rf", auc_tvs1_rf),
 ("auc_tvs_rf", auc_tvs_rf),
 ("auc_cv_rf", auc_cv_rf),
 ("auc_tvs_gbt", auc_tvs_gbt),
 ("auc_cv_gbt", auc_cv_gbt),
 ("auc_tvs_fm", auc_tvs_fm),
 ("auc_cv_fm",auc_cv_fm),
 ("auc_tvs_svm", auc_SVM_tvs),
 ("auc_cv_svm", auc_SVM_cv),
 ("auc_tvs_lr",auc_tvs_lr),
 ("auc_cv_lr",auc_cv_lr),
 ("auc_tvs_dt",auc_tvs_dt),
 ("auc_cv_dt",auc_cv_dt),

 ],["metric", "value"])

metrics.show()
```

▸ (3) Spark Jobs

▸ (3) Spark Jobs

metric	value
auc_tvs1_rf	0.8856730769230771
auc_tvs_rf	0.8702524038461544
auc_cv_rf	0.8637439903846159
auc_tvs_gbt	0.913707932692308
auc_cv_gbt	0.9016526442307693
auc_tvs_fm	0.7187259615384515
auc_cv_fm	0.7187259615384515
auc_tvs_svm	0.5
auc_cv_svm	0.5
auc_tvs_lr	0.5296995192307693
auc_cv_lr	0.53
auc_tvs_dt	0.6415865384615385
auc_cv_dt	0.5

Command took 0.58 seconds -- by nsriram@calstatela.edu at 5/17/2023, 9:25:16 PM on Project

References

1. URL of Data Source: https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml
2. URL of your GitHub: https://github.com/Lekha19202/CIS-5560-big-data-science-project
3. URL of References:
    i. https://towardsdatascience.com/multi-class-text-classification-with-pyspark-7d78d022ed35
    ii. https://spark.apache.org/docs/latest/ml-classification-regression.html#regression