Lab Program – VIII

Program on spark with ml using data set on regression

```
# make pyspark importable as a regular library.
#import findspark
#findspark.init('/opt/spark')
# create a SparkSession
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()

# load data
data = spark.read.csv("/FileStore/tables/BostonHousing.csv", header=True,
inferSchema=True)

# create features vector
feature_columns = data.columns[:-1] # here we omit the final column
from pyspark.ml.feature import VectorAssembler
assembler =
VectorAssembler(inputCols=feature_columns,outputCol="features")
data_2 = assembler.transform(data)

# train/test split
train, test = data_2.randomSplit([0.7, 0.3])

# define the model
from pyspark.ml.regression import LinearRegression
algo = LinearRegression(featuresCol="features", labelCol="medv")

# train the model
model = algo.fit(train)

# evaluation
evaluation_summary = model.evaluate(test)
evaluation_summary.meanAbsoluteError
evaluation_summary.rootMeanSquaredError
evaluation_summary.r2

# predicting values
predictions = model.transform(test)
predictions.select(predictions.columns[13:]).show()
# here I am filtering out some columns just for the figure to fit
```

Lab Program – IX
Program on spark with ml using data set on classification

```
#!pip install sparkmagic
#!pip install pyspark
```

#Part - I

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

from pyspark.ml import Pipeline
#from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import VectorAssembler, StringIndexer, VectorIndexer,
MinMaxScaler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import BinaryClassificationEvaluator

spark = SparkSession.builder.master("local[*]").getOrCreate()
```

#Part - II

```
#Load Source Data
csv = spark.read.csv("/FileStore/tables/flights.csv", inferSchema=True,
header=True)
csv.show(10)
```

#Part - III

```
#Prepare the Data for a Classification Model (Decision Tree Learning Model)
data = csv.select("DayofMonth", "DayOfWeek", "Carrier", "OriginAirportID",
"DestAirportID", "DepDelay", ((col("ArrDelay") >
15).cast("Int").alias("label")))
data.show(10)
```

#Part – IV

```
#Split the Data
splits = data.randomSplit([0.7, 0.3])
train = splits[0]
```

```
test = splits[1].withColumnRenamed("label", "trueLabel")
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, " Testing Rows:", test_rows)
```

#Part - V

```
#Define the Pipeline
strIdx = StringIndexer(inputCol = "Carrier", outputCol = "CarrierIdx")
catVect = VectorAssembler(inputCols = ["CarrierIdx", "DayofMonth",
"DayOfWeek", "OriginAirportID", "DestAirportID"], outputCol="catFeatures")
catIdx = VectorIndexer(inputCol = catVect.getOutputCol(), outputCol =
"idxCatFeatures")
numVect = VectorAssembler(inputCols = ["DepDelay"],
outputCol="numFeatures")
minMax = MinMaxScaler(inputCol = numVect.getOutputCol(),
outputCol="normFeatures")
featVect = VectorAssembler(inputCols=["idxCatFeatures", "normFeatures"],
outputCol="features")
lr =
LogisticRegression(labelCol="label",featuresCol="features",maxIter=10,regPar
am=0.3)
#dt = DecisionTreeClassifier(labelCol="label", featuresCol="features")
pipeline = Pipeline(stages=[strIdx, catVect, catIdx, numVect, minMax, featVect,
lr])
```

#Part - VI

```
#Run the Pipeline to train a model
piplineModel = pipeline.fit(train)
```

#Part – VII

```
#Generate label predictions
prediction = piplineModel.transform(test)
predicted = prediction.select("features", "prediction", "trueLabel")
predicted.show(100, truncate=False)
```

#Part – VIII

```
#Evaluating a Classification Model
tp = float(predicted.filter("prediction == 1.0 AND truelabel == 1").count())
fp = float(predicted.filter("prediction == 1.0 AND truelabel == 0").count())
tn = float(predicted.filter("prediction == 0.0 AND truelabel == 0").count())
fn = float(predicted.filter("prediction == 0.0 AND truelabel == 1").count())
```

```
pr = tp / (tp + fp)
re = tp / (tp + fn)
metrics = spark.createDataFrame([
 ("TP", tp),
 ("FP", fp),
 ("TN", tn),
 ("FN", fn),
 ("Precision", pr),
 ("Recall", re),
 ("F1", 2*pr*re/(re+pr))],["metric", "value"])
metrics.show()
```

#Part - IX

```
#Review the Area Under ROC (AUR)
evaluator = BinaryClassificationEvaluator(labelCol="trueLabel",
rawPredictionCol="rawPrediction", metricName="areaUnderROC")
aur = evaluator.evaluate(prediction)
print ("AUR = ", aur)
```

#Part - X

```
#View the Raw Prediction and Probability
prediction.select("rawPrediction", "probability", "prediction",
"trueLabel").show(100, truncate=False)
```

#Part - XI

```
#Tune Parameters
#Change the Discrimination Threshold
paramGrid = ParamGridBuilder().addGrid(lr.regParam, [0.3,
0.1]).addGrid(lr.maxIter, [10, 5]).addGrid(lr.threshold,[0.4, 0.3]).build()
cv = CrossValidator(estimator=pipeline,
evaluator=BinaryClassificationEvaluator(),
estimatorParamMaps=paramGrid,numFolds=2)
model = cv.fit(train)
```

#Part - XII

```
newPrediction = model.transform(test)
newPredicted = prediction.select("features", "prediction", "trueLabel")
newPredicted.show()
```

#Part – XIII

```
# Recalculate confusion matrix
```

```
tp2 = float(newPrediction.filter("prediction == 1.0 AND truelabel ==
1").count())
fp2 = float(newPrediction.filter("prediction == 1.0 AND truelabel ==
0").count())
tn2 = float(newPrediction.filter("prediction == 0.0 AND truelabel ==
0").count())
fn2 = float(newPrediction.filter("prediction == 0.0 AND truelabel ==
1").count())
pr2 = tp2 / (tp2 + fp2)
re2 = tp2 / (tp2 + fn2)
metrics2 = spark.createDataFrame([
 ("TP", tp2),
 ("FP", fp2),
 ("TN", tn2),
 ("FN", fn2),
 ("Precision", pr2),
 ("Recall", re2),
 ("F1", 2*pr2*re2/(re2+pr2))],["metric", "value"])
metrics2.show()
```

#Part – XIV

```
# Recalculate confusion matrix
tp2 = float(newPrediction.filter("prediction == 1.0 AND truelabel ==
1").count())
fp2 = float(newPrediction.filter("prediction == 1.0 AND truelabel ==
0").count())
tn2 = float(newPrediction.filter("prediction == 0.0 AND truelabel ==
0").count())
fn2 = float(newPrediction.filter("prediction == 0.0 AND truelabel ==
1").count())
pr2 = tp2 / (tp2 + fp2)
re2 = tp2 / (tp2 + fn2)
metrics2 = spark.createDataFrame([
 ("TP", tp2),
 ("FP", fp2),
 ("TN", tn2),
 ("FN", fn2),
 ("Precision", pr2),
 ("Recall", re2),
 ("F1", 2*pr2*re2/(re2+pr2))],["metric", "value"])
metrics2.show()
```

Lab Program – X

Program on Graph Analysis Use Case1 - Fraud Detection

// Do it in Kaggle

#Part - I

```
#Reading users data into dataframes
import pandas as pd

bank_accounts = pd.read_csv('/kaggle/input/scl2020-workshops/scl-competition/ptr-rd2-ungrd-rd2/bank_accounts.csv')
credit_cards = pd.read_csv('/kaggle/input/scl2020-workshops/scl-competition/ptr-rd2-ungrd-rd2/credit_cards.csv')
devices = pd.read_csv('/kaggle/input/scl2020-workshops/scl-competition/ptr-rd2-ungrd-rd2/devices.csv')
orders = pd.read_csv('/kaggle/input/scl2020-workshops/scl-competition/ptr-rd2-ungrd-rd2/orders.csv')

# Displaying dataframes
display(bank_accounts)
display(credit_cards)
display(devices)
display(orders)
```

#Part - II

```
# !python -m pip install networkx
import networkx as nx

# Create a graph with 'user_id' as nodes and edges to connect
# 'user_ids' that share the same 'bank_account', 'credit_card' or 'device'
users_graph = nx.Graph()

def add_user_id_link(G, user_id1, user_id2, attr1, attr2):
    """
    Function to add link in graph G for user_id1 and user_id2
    if they share the same 'attr' ('device', 'credit_card' or 'bank_account')
    """
    if attr1 == attr2:
        G.add_edge(user_id1, user_id2)

for df in [bank_accounts, credit_cards, devices]:
    # Ensure that 'bank_account', 'credit_card' and 'device' are str
```

```
    df.iloc[:,1] = df.iloc[:,1].astype('str')

    # Sorting by 'bank_account', 'credit_card' and 'device'
    df.sort_values(df.columns[1], inplace=True)

    # Compare if consecutive rows have the same 'user_id'
    # by adding shifted rows as new columns to easily compare consecutive rows
    df_next_row = df.shift(1)
    df_next_row.columns = df.columns + '2'
    df_combined = pd.concat([df, df_next_row], axis='columns')

    # If consecutive rows (columns 1 & 2 vs 3 & 4) share the same
    # 'bank_account', 'credit_card' or 'device', add link on users_graph
    df_combined.apply(lambda row: add_user_id_link(users_graph,
row['userid'], row['userid2'], row.values[1], row.values[3]), axis=1)
```

#Part - III

```
#Detecting fraudulent activities using users_graph
def is_fraud(G, buyer_id, seller_id):
    """
    Function to detect frauds by checking if there exists a path
    from buyer_id node to seller_id node in users_graph.
    """
    # Check if both buyer_id and seller_id are in users_graph
    if G.has_node(buyer_id) and G.has_node(seller_id):
        # Check if there is a path from buyer_id to seller_id
        if buyer_id in nx.algorithms.descendants(G, seller_id):
            return 1
    return 0

orders['is_fraud'] = orders.apply(lambda row: is_fraud(users_graph,
row.buyer_userid, row.seller_userid), axis=1)
display(orders)
```

#Part – IV

```
#Saving orderids and fraudulent flags as csv
df_frauds = orders[['orderid', 'is_fraud']]
df_frauds.to_csv('/kaggle/working/ans.csv', index=False)
```

Lab Program – XI

Program on Graph Analysis Use Case2 - Anti-Money Laundering (AML)

 # Do it in Kaggle

#Part - I

```
import os
import pandas as pd

pd.options.display.float_format = "{:,.2f}".format
```

#Part - II

```
df = pd.read_csv("/kaggle/input/ibm-transactions-for-anti-money-laundering-aml/HI-Large_Trans.csv", nrows=5_000)
print(df.columns); display(df)
del df
```

#Part - III

```
source = "/kaggle/input/ibm-transactions-for-anti-money-laundering-aml/HI-Large_Trans.csv"

df_calculated = pd.DataFrame()
df_len = 0

cols = ["Amount Received", "Receiving Currency", "Amount Paid"]
for i, chunk in enumerate(pd.read_csv(source, chunksize=10_000)):
    chunk_selection = chunk[cols]
    chunk_group = chunk_selection.groupby(cols[1], as_index=False).sum()
    df_calculated = pd.concat([df_calculated, chunk_group])
    df_len += len(chunk)
    # if i > 5:
    #     break

df_calculated = df_calculated.groupby(cols[1], as_index=False).sum()
```

#Part - IV

```
print(f">> total number of database rows: {df_len:,}")
display(df_calculated)
```