%pyspark

from dateutil.parser import parse

from dateutil.parser import parse

from pyspark.sql.types import \*

from pyspark.mllib.classification import LogisticRegressionWithLBFGS, LogisticRegressionModel

from pyspark.mllib.classification import SVMWithSGD, SVMModel

from pyspark.mllib.regression import LabeledPoint

from pyspark.mllib.evaluation import BinaryClassificationMetrics

#loading trainHistory table with generated Features

trainHistory = sc.textFile("/Users/annu/Desktop/IDS/TRAIN\_HISTORY\_FEATURES.csv")

header = trainHistory.filter(lambda line: "CUSTOMER\_ID" in line)

trainHistory= trainHistory.subtract(header)

schemaString = "customer\_id offer\_id chain\_id market\_id repeat\_trips offer\_date category\_id min\_qty company\_id offer\_value brand\_id purchase\_times\_company purchase\_value\_company purchase\_quantity\_company repeat\_customer times\_company\_180 times\_company\_60 times\_company\_30 purchase\_times\_ccb purchase\_times\_category amount\_spent\_category bought\_category\_30 chain\_visit\_freq ratio\_returned\_bought\_cc"

fields = [StructField(field\_name, StringType(), True) for field\_name in schemaString.split()]

fields[4].dataType = FloatType()

fields[5].dataType = TimestampType()

fields[7].dataType = FloatType()

fields[9].dataType = FloatType()

fields[11].dataType = FloatType()

fields[12].dataType = FloatType()

fields[13].dataType = FloatType()

fields[14].dataType = IntegerType()

fields[15].dataType = FloatType()

fields[16].dataType = FloatType()

fields[17].dataType = FloatType()

fields[18].dataType = FloatType()

fields[19].dataType = FloatType()

fields[20].dataType = FloatType()

fields[21].dataType = FloatType()

fields[22].dataType = FloatType()

fields[23].dataType = FloatType()

schema = StructType(fields)

trainHistory = trainHistory.map(lambda l: l.split(",")).map(lambda p: (p[0],p[1],p[2],p[3],float(p[4]),parse(p[5]),p[6],float(p[7]),p[8],float(p[9]),p[10],float(p[11]),float(p[12]),float(p[13]),int(p[14]),float(p[15]),float(p[16]),float(p[17]),float(p[18]),float(p[19]),float(p[20]),float(p[21]),float(p[22]),float(p[23])))

trainHistoryDF = sqlContext.createDataFrame(trainHistory, schema)

#loading testHistory table

testHistory = sc.textFile("/Users/annu/Desktop/IDS/TEST\_HISTORY\_FEATURES.csv")

header = testHistory.filter(lambda line: "CUSTOMER\_ID" in line)

testHistory= testHistory.subtract(header)

schemaString = "customer\_id offer\_id chain\_id market\_id offer\_date category\_id min\_qty company\_id offer\_value brand\_id purchase\_times\_company purchase\_value\_company purchase\_quantity\_company times\_company\_180 times\_company\_60 times\_company\_30 purchase\_times\_ccb purchase\_times\_category amount\_spent\_category bought\_category\_30 chain\_visit\_freq ratio\_returned\_bought\_cc"

fields = [StructField(field\_name, StringType(), True) for field\_name in schemaString.split()]

fields[4].dataType = TimestampType()

fields[6].dataType = FloatType()

fields[8].dataType = FloatType()

fields[10].dataType = FloatType()

fields[11].dataType = FloatType()

fields[12].dataType = FloatType()

fields[13].dataType = FloatType()

fields[14].dataType = FloatType()

fields[15].dataType = FloatType()

fields[16].dataType = FloatType()

fields[17].dataType = FloatType()

fields[18].dataType = FloatType()

fields[19].dataType = FloatType()

fields[20].dataType = FloatType()

fields[21].dataType = FloatType()

schema = StructType(fields)

testHistory = testHistory.map(lambda l: l.split(",")).map(lambda p: (p[0],p[1],p[2],p[3],parse(p[4]),p[5],float(p[6]),p[7],float(p[8]),p[9],float(p[10]),float(p[11]),float(p[12]),float(p[13]),float(p[14]),float(p[15]),float(p[16]),float(p[17]),float(p[18]),float(p[19]),float(p[20]),float(p[21])))

testHistoryDF = sqlContext.createDataFrame(testHistory, schema)

#parsing data into LabeledPoint object

parsedTrainData= trainHistoryDF.map(lambda line:LabeledPoint(line[14],[line[7],line[9],line[11],line[12],line[13],line[15],line[16],line[17],line[18],line[19],line[20],line[21],line[22],line[23]]))

#partitioning the training data into training and testing sets

#parsedTrainData, parsedTestData = parsedTrainData.randomSplit([0.7, 0.3], seed = 1245)

parsedTestData= testHistoryDF.map(lambda line:LabeledPoint(line[0],[line[6],line[8],line[10],line[11],line[12],line[13],line[14],line[15],line[16],line[17],line[18],line[19],line[20],line[21]]))

#training the model with Logistic Regression with LBFGS - L1

logisticRegressionModel = LogisticRegressionWithLBFGS.train(parsedTrainData)

# evaluating the model on testing data

labelsAndPreds = parsedTestData.map(lambda p: (logisticRegressionModel.predict(p.features)))

labelsAndPreds.saveAsTextFile("/Users/annu/Desktop/predsFile")

#testingErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedTestData.count())

#print("Testing Error = " + str(testingErr))

# saving the model

#logisticRegressionModel.save(sc, "/Users/annu/Desktop/IDS/LRModel")

#logisticRegressionModel = LogisticRegressionModel.load(sc, "/Users/annu/Desktop/IDS/LRModel")

# Instantiate metrics object

#metrics = BinaryClassificationMetrics(labelsAndPreds)

# Area under precision-recall curve

#print("Area under PR = %s" % metrics.areaUnderPR)

# Area under ROC curve

#print("Area under ROC = %s" % metrics.areaUnderROC)