1. Pre-Processing Code:

import sys

import os

#Replace it with file location to which the file has be uploaded

#rawData = sc.textFile(<File location>).cache()

rawData = sc.textFile("/FileStore/tables/xi8s0vex1500739277781/rec\_log\_train.txt").cache()

rawSampledRDD\_1 = rawData.map(lambda x: x.split('\t')).map(lambda x: (int(x[0]),int(x[1]),float(x[2]))).filter(lambda x: x[2] == 1.0)

rawSampledRDD\_0 = rawData.map(lambda x: x.split('\t')).map(lambda x: (int(x[0]),int(x[1]),float(x[2]))).filter(lambda x: x[2] == -1.0)

rawSampled = rawSampledRDD\_0.sample(False,0.2).cache()

#combine both data with 1 and -1 labeled

combinedRDD = rawSampledRDD\_1.union(rawSampled)

#save the generated file to a location.

#Replace the <file tmp location> to desired location

#combinedRDD.map(lambda x: str(x[0]) + '\t' + str(x[1]) + '\t' + str(x[2])).saveAsTextFile(<file tmp location>)

combinedRDD.map(lambda x: str(x[0]) + '\t' + str(x[1]) + '\t' + str(x[2])).saveAsTextFile("/FileStore/tables/xi8s0vex1500739277781/rec\_log\_train\_tmp.txt")

rawSampledRDD\_1.count()

rawSampled.count()

combinedRDD.collect()

1. GBT training:

import sys

import os

from pyspark.ml.linalg import Vectors

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.feature import Word2Vec

from pyspark.ml.classification import GBTClassifier

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from pyspark.ml import Pipeline

# get the tmp data and convert to a tuple

# replace the file location with the file location in which the tmp file is saved

# rawData = sc.textFile(<Tmp File Location>).cache()

rawData = sc.textFile("/FileStore/tables/xi8s0vex1500739277781/rec\_log\_train\_tmp.txt").cache()

rawSampled = rawData.sample(False,1).cache()

def get\_tuple(data):

item = data.split('\t')

if(float(item[2]) == -1):

return item[0],item[1], 0.0

else:

return item[0],item[1],1.0

testRDD = rawSampled.map(get\_tuple).cache()

testDF = spark.createDataFrame(testRDD, ["user","product","label"])

testDF.describe("label").show()

# get the data from user\_key\_word file and generate feature vectors

def convert\_data(data):

values = data.split(';')

vec = map(lambda x: x.split(':'),values)

buff = sorted(vec,key=lambda x: int(x[0]))

result = ''

for x in buff:

result += x[0]+':'

result = result[:-1]

return result

# upload the user\_key\_word.txt and replace the <file location> with the new location

# userFeatureRDD = sc.textFile(<file location>).cache()

userFeatureRDD = sc.textFile("/FileStore/tables/dv4ow79y1501801226776/user\_key\_word.txt").cache()

userLibSvmRDD = userFeatureRDD.map(lambda x: x.split('\t')).map(lambda x: x[0]+' '+convert\_data(x[1]))

inp = userLibSvmRDD.map(lambda x: x.split(' ')).map(lambda x: (x[0],x[1].split(':')))

inpDF = spark.createDataFrame(inp, ["user","input"])

word2vec = Word2Vec(vectorSize=5, minCount=0, inputCol="input", outputCol="result")

model = word2vec.fit(inpDF)

result = model.transform(inpDF)

# get the item keywords form item.txt and generate feature vectors

# upload the file item.txt and replace the <file location> with the new file location

# itemRawRDD = sc.textFile(<file location>)

itemRawRDD = sc.textFile("/FileStore/tables/7ahtri9r1501802021762/item.txt")

itemTmpRDD = itemRawRDD.map(lambda x: x.split('\t')).map(lambda x: (x[0],x[2].split(';')))

itemTmpDF = spark.createDataFrame(itemTmpRDD, ["product","input"])

word2vec = Word2Vec(vectorSize=5, minCount=0, inputCol="input", outputCol="categories")

model = word2vec.fit(itemTmpDF)

itemDF = model.transform(itemTmpDF)

# combine all features and generate a final feature vector comprising of user and item features

trainDF = testDF.join(itemDF, itemDF.product == testDF.product)

outDF = trainDF.join(result,result.user == trainDF.user)

assembler = VectorAssembler(inputCols=["categories", "result"],outputCol="features")

output = assembler.transform(outDF)

#train data

(trainingData, testData) = output.randomSplit([0.8, 0.2])

print 'Training: %s, test: %s\n' % (trainingData.count(),testData.count())

gbt = GBTClassifier(labelCol="label", featuresCol="features", maxIter=10)

pipeline = Pipeline(stages=[gbt])

model = pipeline.fit(trainingData)

#make prediction on test data

predictions = model.transform(testData)

predictions.describe("prediction","label").show()

graph = predictions[["prediction","label"]]

#convert the output into desired plot

display(graph)

evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")

accuracy = evaluator.evaluate(graph)

print("Test Accuracy = %g" % accuracy)

1. ALS algorithm:

import sys

import os

from pyspark.ml.feature import \*

from pyspark.ml.evaluation import \*

from pyspark.mllib.evaluation import \*

from pyspark.mllib.recommendation import ALS

# get the data from the tmp file generated

# replace the file location with the file location in which the tmp file is saved

# rawSampled = sc.textFile(<Tmp File Location>).cache()

rawSampled = sc.textFile("/FileStore/tables/xi8s0vex1500739277781/rec\_log\_train\_tmp.txt").cache()

#data of format (UserId)\t(ItemId)\t(Result)\t(Unix-timestamp)

#generate tuple

def get\_tuple(data):

item = data.split('\t')

return int(item[0]),int(item[1]),float(item[2])

testRDD = rawSampled.map(get\_tuple).cache()

#split data int0 training and test

trainingRDD, testRDD = testRDD.randomSplit([0.8,0.2])

print 'Training: %s, test: %s\n' % (trainingRDD.count(),testRDD.count())

# convert the outputs to 1 or -1

import math

def generalize(data):

val = 0.0

if(data[2] >= 0.0):

val = 1.0

else:

val = -1.0

return (data[0],data[1],val)

# trian the model

testForPredictRDD = testRDD.map(lambda x: (x[0],x[1]))

seed = 5L

iterations = 5

regularizationParameter = 0.1

rank = 3

tolerance = 0.03

model = ALS.train(trainingRDD, rank, seed=seed, iterations=iterations,

lambda\_=regularizationParameter)

predictedRatingsRDD = model.predictAll(testForPredictRDD).map(generalize)

# predict the values and test the model

predictedReformattedRDD = predictedRatingsRDD.map(lambda x: ((x[0],x[1]),x[2])).cache()

actualReformattedRDD = testRDD.map(lambda x: ((x[0],x[1]),x[2])).cache()

squaredErrorsRDD = predictedReformattedRDD.join(actualReformattedRDD).map(lambda x: [x[1][0],x[1][1]])

squaredDF = spark.createDataFrame(squaredErrorsRDD,["predicted","actual"])

evaluator = MulticlassClassificationEvaluator(labelCol="actual", predictionCol="predicted", metricName="accuracy")

accuracy = evaluator.evaluate(squaredDF)

squaredDF.describe("actual","predicted").show()

print accuracy

# convert the output into the desired plot

display(squaredDF)