**گزارش تمرين شماره دو BIG DATA**

تاريخ تحويل : 18/8/98 نمره : 1.5

1. پايگاه داده KDDCUP99 که توضيحات آن در فايل پيوست آورده شده است داراي دو فايل آموزش kddcup.data.gz و تست corrected.gz ميباشد. ميخواهيم با استفاده از RANDOM FOREST در SPARK دقت کلاسبندي و زمان اجرا بر روي 4 هسته را براي پنج کلاس اصلي DOSو R2L و U2R و probing و NORMAL در حالات زير بدست آوريم.

ابتدا داده ها را با فرمت txt دخیره کرده و آنها را پارس میکنیم سپس با بکارگیری pipeline ، مسیری برای آماده سازی اطلاعات و مدل پیش میگیریم ، که در ابتدا ویژگی های کتگوریکال را به عدد هایی که متناسب با تعداد آنها فضای کمتری در را ذخیره میکند ، تبدیل میکنیم و سایر ویژگی های عددی را به صورت بردار در آورده و با اضافه کردن یک بخش دیگر به پایپ لاین ویژگی ها را در یک بردار به کمک assembler آماده کرده و سپس به مدل میدهیم ، سپس اطلاعات به پایپ لاین فیت میکنیم

برای بررسی روند اصلی ، دادگان اموزش را به دو دسته train و validation تقسیم میکنیم. تا از میزان یادگیری اطلاعات مطلع شویم. برای نمونه بر روی یک جنگل با 10 درخت ، 70 شاخه ای و عمق 10 نتایج زیر حاصل شد :

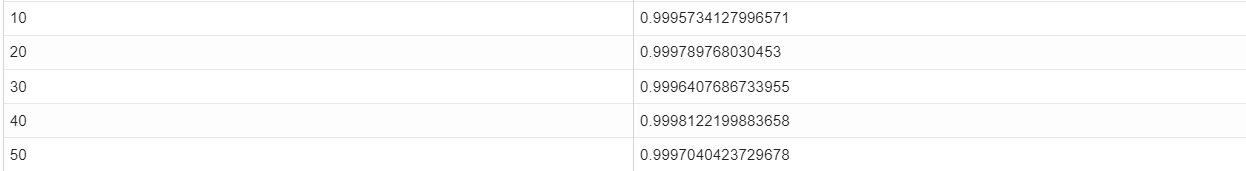
accuracy is on validation is 0.9995019747517527

accuracy is on test is 0.5277867980156191

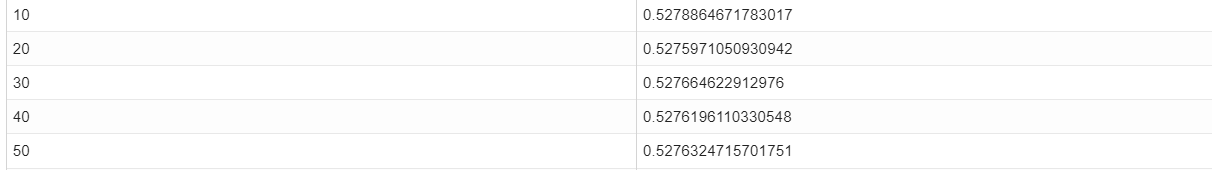
که به این معنا است که درخت داده های آموزش را به طور کامل فراگرفته و برای داده های تست نمیتواند عملکرد مناسبی داشته باشد. این حالت به این معنا است که دیتایی که برای تست مشخص شده ، با اطاعاتی که برای آموزش در نظر گرفته شده ، متفاوت است و از به مدل خوبی استفاده نشده یا اطلاعات برای مدل استفاده شده مناسب نیست ، با حال با تغییر در پارامتر های مدل به بررسی بیشتر مدل پیشنهادی میپردازیم:

الف: دو نمودار دقت و زمان اجرا را براي تعداد درخت 10-20-30-40-50 و حداکثر عمق درخت 10 رسم کنيد.

میزان دقت برای validation

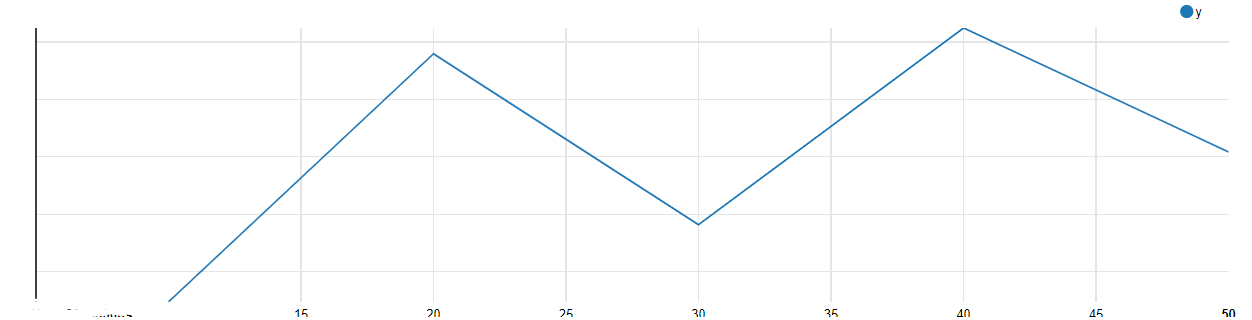


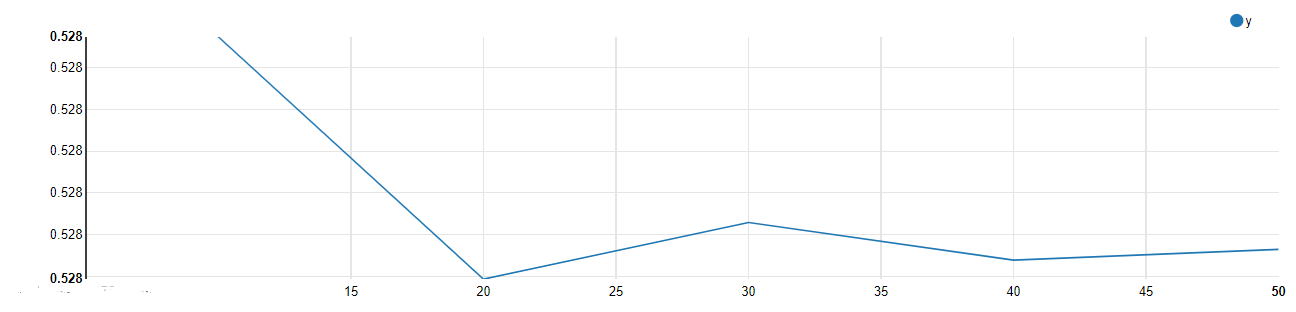
میزان دقت برای test



زمان اجرا







ب:بهترين جواب بخش الف را با حداکثر عمق درخت 20 و با معيارهاي GINI يا ENTROPY تکرار کنيد

gini in validation is 0.9999142743425148

gini in test is 0.5294715283783827

entropy in validation is 0.9999224386908467

entropy in test is 0.5277675072099386

ج : مقادير متريک هاي زير را براي بهترين جواب ارايه دهيد

f1 is0.5302292139352637

weightedPrecision is0.5329607544319456

weightedRecall is0.5294715283783827

accuracy is0.5294715283783827

خروجي گزارش شامل کد، نتايج، نمودار و تحليل ان مي باشد.

کد:

import org.apache.spark.\_

import org.apache.spark.SparkConf

import org.apache.spark.SparkContext.\_

import org.apache.spark.rdd.RDD.\_

import org.apache.spark.ml.linalg.{Vector, Vectors}

def time[R](block: => R): R = {

val t0 = System.nanoTime()

val result = block // call-by-name

val t1 = System.nanoTime()

println("Elapsed time: " + (t1 - t0) + "ns")

result

}

def parser(line:String)={

val num =line.split(",")

val label=num(41)

val x0=num(0).toDouble

val x1=num(1)

val x2=num(2)

val x3=num(3)

val x4=num.dropRight(1).drop(4).map(t=>t.toDouble)

(label,x0,x1,x2,x3 ,Vectors.dense(x4))

}

val Data = sc.textFile("C:/BigData/HW2/kddcup.txt").map(parser).toDF("label","f0","f1","f2","f3","f4")

val Array(trainData,valData) = Data.randomSplit(Array(0.9, 0.1), seed = 123)

val test = sc.textFile("C:/BigData/HW2/kddcup-test.txt").map(parser).toDF("label","f0","f1","f2","f3","f4")

//make string categorical thing to vector

import org.apache.spark.ml.feature.{IndexToString, StringIndexer, VectorIndexer}

//prepare label

val labelIndexer = new StringIndexer()

.setInputCol("label")

.setOutputCol("indexedLabel")

//prepare f1

val F1Indexer = new StringIndexer()

.setInputCol("f1")

.setOutputCol("Ft1")

//prepare f2

val F2Indexer = new StringIndexer()

.setInputCol("f2")

.setOutputCol("Ft2")

//prepare f3

val F3Indexer = new StringIndexer()

.setInputCol("f3")

.setOutputCol("Ft3")

import org.apache.spark.ml.feature.VectorAssembler

val assembler = new VectorAssembler()

.setInputCols(Array("f0","Ft1","Ft2","Ft3","f4"))

.setOutputCol("Features")

import org.apache.spark.ml.classification.{RandomForestClassificationModel, RandomForestClassifier}

val rf = new RandomForestClassifier()

.setLabelCol("indexedLabel")

.setFeaturesCol("Features")

.setImpurity("entropy")

.setMaxDepth(10)

.setNumTrees(10)

.setFeatureSubsetStrategy("auto")

.setSeed(123)

.setMaxBins(70)

import org.apache.spark.ml.Pipeline

val testData = new Pipeline().setStages(Array(labelIndexer, F1Indexer, F2Indexer, F3Indexer))

.fit(test).transform(test).select("f0","Ft1","Ft2","Ft3","f4","indexedLabel")

val pipeline = new Pipeline()

.setStages(Array(labelIndexer, F1Indexer, F2Indexer, F3Indexer, assembler ,rf ))

val model = pipeline.fit(trainData)

val predictionsV=model.transform(valData)

val predictionsT=model.transform(testData)

import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

val evaluator = new MulticlassClassificationEvaluator()

.setLabelCol("indexedLabel")

.setPredictionCol("prediction")

.setMetricName("accuracy")

println("accuracy is on validation is "+ evaluator.evaluate(predictionsV) )

println("accuracy is on test is "+ evaluator.evaluate(predictionsT) )

//part a

//Tree-10,20,30,40,50 -time -accuracy

import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

val seed = 100

val evaluator = new MulticlassClassificationEvaluator()

.setLabelCol("indexedLabel")

.setPredictionCol("prediction")

.setMetricName("accuracy")

println("%table\nx \ty")

(10 to 50 by 10).map(numtree =>(

numtree,time{evaluator.evaluate{new Pipeline().setStages(Array(labelIndexer, F1Indexer, F2Indexer, F3Indexer, assembler ,

new RandomForestClassifier()

.setLabelCol("indexedLabel")

.setFeaturesCol("Features")

.setImpurity("entropy")

.setMaxDepth(10)

.setNumTrees(numtree)

.setFeatureSubsetStrategy("auto")

.setSeed(123)

.setMaxBins(100)

))

.fit(trainData)

.transform(valData)

}

}

)

).foreach{case (x,y) => println(x + "\t" + y)}

//part a

//Tree-10,20,30,40,50 -time -accuracy in testdata

import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

val seed = 100

val evaluator = new MulticlassClassificationEvaluator()

.setLabelCol("indexedLabel")

.setPredictionCol("prediction")

.setMetricName("accuracy")

println("%table\nx \ty")

(10 to 50 by 10).map(numtree =>(

numtree,time{evaluator.evaluate{new Pipeline().setStages(Array(labelIndexer, F1Indexer, F2Indexer, F3Indexer, assembler ,

new RandomForestClassifier()

.setLabelCol("indexedLabel")

.setFeaturesCol("Features")

.setImpurity("entropy")

.setMaxDepth(10)

.setNumTrees(numtree)

.setFeatureSubsetStrategy("auto")

.setSeed(123)

.setMaxBins(100)

))

.fit(trainData)

.transform(testData)

}

}

)

).foreach{case (x,y) => println(x + "\t" + y)}

//gini in validation

println(

evaluator.evaluate{new Pipeline().setStages(Array(labelIndexer, F1Indexer, F2Indexer, F3Indexer, assembler ,

new RandomForestClassifier()

.setLabelCol("indexedLabel")

.setFeaturesCol("Features")

.setImpurity("gini")

.setMaxDepth(20)

.setNumTrees(10)

.setFeatureSubsetStrategy("auto")

.setSeed(123)

.setMaxBins(100)

))

.fit(trainData)

.transform(valData)}

)

//gini in test

println(

evaluator.evaluate{new Pipeline().setStages(Array(labelIndexer, F1Indexer, F2Indexer, F3Indexer, assembler ,

new RandomForestClassifier()

.setLabelCol("indexedLabel")

.setFeaturesCol("Features")

.setImpurity("gini")

.setMaxDepth(20)

.setNumTrees(10)

.setFeatureSubsetStrategy("auto")

.setSeed(123)

.setMaxBins(100)

))

.fit(trainData)

.transform(testData)}

)

//entropy in val

println(

evaluator.evaluate{new Pipeline().setStages(Array(labelIndexer, F1Indexer, F2Indexer, F3Indexer, assembler ,

new RandomForestClassifier()

.setLabelCol("indexedLabel")

.setFeaturesCol("Features")

.setImpurity("entropy")

.setMaxDepth(20)

.setNumTrees(10)

.setFeatureSubsetStrategy("auto")

.setSeed(123)

.setMaxBins(100)

))

.fit(trainData)

.transform(valData)}

)

//entropy in test

println(

evaluator.evaluate{new Pipeline().setStages(Array(labelIndexer, F1Indexer, F2Indexer, F3Indexer, assembler ,

new RandomForestClassifier()

.setLabelCol("indexedLabel")

.setFeaturesCol("Features")

.setImpurity("entropy")

.setMaxDepth(20)

.setNumTrees(10)

.setFeatureSubsetStrategy("auto")

.setSeed(123)

.setMaxBins(100)

))

.fit(trainData)

.transform(testData)}

)

val Model=new Pipeline().setStages(Array(labelIndexer, F1Indexer, F2Indexer, F3Indexer, assembler ,

new RandomForestClassifier()

.setLabelCol("indexedLabel")

.setFeaturesCol("Features")

.setImpurity("gini")

.setMaxDepth(20)

.setNumTrees(10)

.setFeatureSubsetStrategy("auto")

.setSeed(123)

.setMaxBins(100)

) )

.fit(trainData)

.transform(testData)

val p1 =evaluator.setMetricName("f1").evaluate(Model)

val p2 =evaluator.setMetricName("weightedPrecision").evaluate(Model)

val p3 =evaluator.setMetricName("weightedRecall").evaluate(Model)

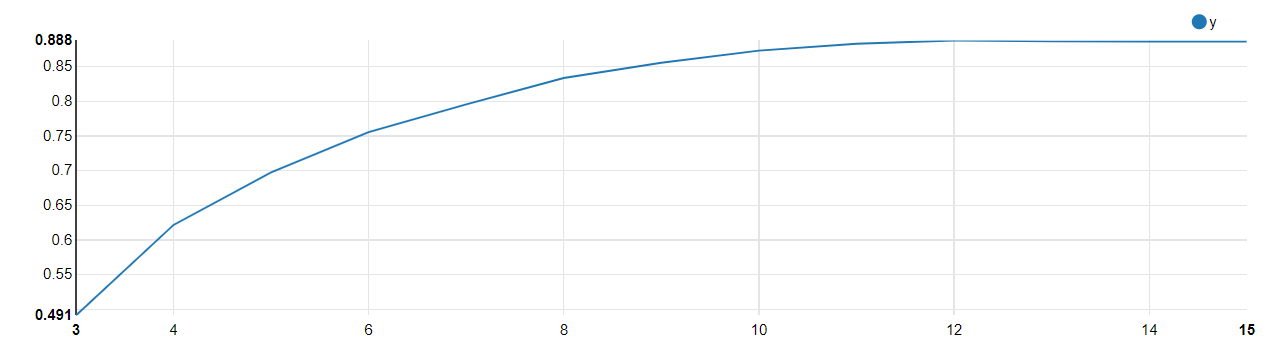
val p4 =evaluator.setMetricName("accuracy").evaluate(Model)

println("f1 is"+p1+"weightedPrecision is"+p2+"weightedRecall is"+p3+"accuracy is"+p4)

1. پايگاه داده MNIST که توضيحات آن در پيوست آورده شده براي آموزش و تست يک درخت تصميم گيري در SPARK استفاده مي شود

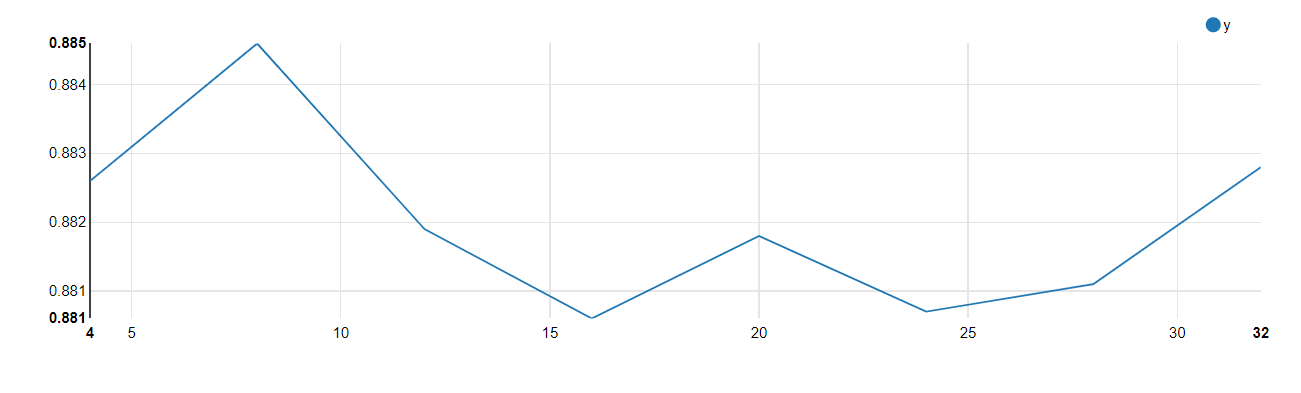
برای استفاده از اطلاعات ابتدا خارج از zeppelin داده ها به شکل csv (جدا شده با کاما) تبدیل شد ، سپس الگوریتم روی آن اجرا شد.

الف : نمودار دقت را با تغيير عمق درخت از k=3 to8 رسم نماييد.



میزان بهینه ماکزیمم عمق درخت 11 است.

ب: با در نظر گرفتن عمق بهينه درخت پارامتر MaxBins را بين مقادير 4 و 8 و 16 و 32 تغيير دهيد. نمودار دقت خروجي را بر حسب اين پارامتر رسم نماييد و تغييرات موجود در دقت نتايج را توضيح دهيد.



دامنه تغییرات به طور کلی در حدود همان 88 درصد دقت است ولی برای توجیح جزئیات آن MaxBins تعداد شاخه های خروجی هر نود است ، که بر اساس داکیومنت اسپارک باید بیشتر یا مساوی تعداد نوع ها در بردار ویژگی باشد ، از آن جایی که بردار ویژگی ما تصویر است و عمل رگرسیون انجام نمی دهیم پس عدد هر پیکسل به واقع یک نوع از آن دسته محسوب میشود ولی با بررسی این عدد دقت همچنان 88 درصد است، چرا که هر چند دامنه ها بیانگر دسته های مختلف هستند ولی دامنه تغییرات آن ها ثابت است (تعدادی در حدود 250 تعدادی در حدود 100 تعدادی در حدود 0 ) پس بجای انتخاب عدد 255 میتوان از عددهای کوچکتر استفاده نمود ، با توجه به انتخاب میزان بهینه عمق درخت 11 ویژگی در لایه های پایین تر به طور کامل از یکدیگر جدا شده و نتیجه را در حدود 88 درصد نگه میدارند با این حال برای توجیه نوسانات ، میتوان گفت که ما 10 دسته برای خروجی داریم و ویژگی ها را باید در 10 دسته تقسیم کنیم که در نواحی مضارب 10 نمودار دقت ، بهبود میابد

گزارش شامل کد و نتايج و نمودارهاي هر مرحله و تحليل جواب ها مي باشد.

کد:

import org.apache.spark.\_

import org.apache.spark.SparkConf

import org.apache.spark.SparkContext.\_

import org.apache.spark.rdd.RDD.\_

import org.apache.spark.ml.linalg.{Vector, Vectors}

//read

def parser(line:String)={

val num =line.split(",")

val label=num(0).toInt

val feature=num.drop(1).map(t=>t.toDouble)

(label ,Vectors.dense(feature) )

}

val training = sc.textFile("C:/BigData/HW2/mnist\_train.txt").map(parser).toDF("label", "features")

val testing = sc.textFile("C:/BigData/HW2/mnist\_test.txt").map(parser).toDF("label", "features")

import org.apache.spark.ml.classification.{DecisionTreeClassifier, DecisionTreeClassificationModel}

val dtc = new DecisionTreeClassifier()

.setImpurity("entropy")

.setMaxBins(255)

.setMaxDepth(11)

.setSeed(123)

.fit(training)

val perdiction = dtc.transform(testing)

perdiction.select("label","prediction").show(10)

import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

val evaluator = new MulticlassClassificationEvaluator()

.setLabelCol("label")

.setPredictionCol("prediction")

.setMetricName("accuracy")

val accuracy = evaluator.evaluate(perdiction)

println ("accuracy is :"+ accuracy\*100 +" percent \n" )

//2-1

//MaxDepth => 3 to 10

val evaluator = new MulticlassClassificationEvaluator()

.setLabelCol("label")

.setPredictionCol("prediction")

.setMetricName("accuracy")

println("%table\nx \ty")

(3 to 15).map(MaxDepth=>(MaxDepth,evaluator.evaluate{

new DecisionTreeClassifier()

.setImpurity("entropy")

.setMaxBins(32)

.setMaxDepth(MaxDepth)

.setSeed(123)

.fit(training)

.transform(testing)

}

)

).foreach{case (x,y) => println(x + "\t" + y)}

//2-1

//MaxBins => 3 to 10

val evaluator = new MulticlassClassificationEvaluator()

.setLabelCol("label")

.setPredictionCol("prediction")

.setMetricName("accuracy")

println("%table\nx \ty")

(4 to 32 by 4).map(MaxBins=>(MaxBins,evaluator.evaluate{

new DecisionTreeClassifier()

.setImpurity("entropy")

.setMaxBins(MaxBins)

.setMaxDepth(11)

.setSeed(123)

.fit(training)

.transform(testing)

}

)

).foreach{case (x,y) => println(x + "\t" + y)}