中 北 大 学

课程设计说明书

|  |  |  |  |
| --- | --- | --- | --- |
| **学 院、系：** | 软件学院 | | |
| **专 业：** | 软件工程 | | |
| **学 生 姓 名：** | 张丹 | **学 号：** | 1414010513 |
| **学 生 姓 名：** | 和泽 | **学 号：** | 1414010824 |
| **学 生 姓 名：** | 张佳丽 | **学 号：** | 1414010611 |
| **设 计 题 目：** | 汽车测评数据分类分析 | | |
|  |  | | |
| **起 迄 日 期:** | 2017年5月24日-2017年6月6日 | | |
| **指 导 教 师:** | 章永来 | | |

日期: 2017年5月 25日

**目录**

1、设计目的…………………………………………………………………………3

2、任务概述…………………………………………………………………………3

3、功能模块划分……………………………………………………………………4

4、功能描述…………………………………………………………………………4

4.1、数据存储方面………………………………………………………………4

4.2、数据处理与分析方面………………………………………………………4

4.3、数据可视化方面……………………………………………………………4

5、系统设计…………………………………………………………………………5

5.1、虚拟机中运行………………………………………………………………5

5.2、在IDE中运行………………………………………………………………5

6、程序运行数据及其结果…………………………………………………………6

6.1、数据集分类结果……………………………………………………………6

6.2 数据库设计………………………………………………………………… 7

7、课程设计心得……………………………………………………………………7

8、附录………………………………………………………………………………8

**1、设计目的**

据统计2017年其市场规模达到10600亿，2019年将突破1.2万亿。中国汽车产销量已位居世界第一，年上市新款乘用车（含改款车）达数百款。在中 国汽车保有量稳步增长的同时，汽车消费群体正在趋于年轻化，更多消费者通过看汽车测评完成初步选车。2015年以来，90后上牌量增幅高达 86.8%，85后购车最多达376万辆，多数80后进入置换购车阶段。在这样的背景下，对汽车测评成为汽车购置风向标。现今已是大数据时代，通过数据挖掘和云计算等技术，对大量数据进行分类分析，可以从大量信息中获取有价值的信息，利用可视化对分析好的数据进行图形化呈现，使消费者在购置汽车时可以快速获取自己想要了解的信息，与此同时，可以参考当下汽车盛行风向。总之，汽车测评数据分类分析是很有必要的。

**2、任务概述**

本次课设题目是汽车测评数据分类分析，具体的任务要求如下：

Car Evaluation是一个关于汽车测评的数据集，类别变量为汽车的测评（unacc，ACC，good，vgood）分别代表（不可接受，可接受，好，非常好），而6个属性变量分别为「买入价」，「维护费」，「车门数」，「可容纳人数」，「后备箱大小」，「安全性」。值得一提的是6个属性变量全部是有序类别变量，比如「可容纳人数」值可为「2，4，more」，「安全性」值可为「low, med, high」。

功能要求：

（1）数据存储可选择Hive，mysql，hdfs、Hase等；

（2）设计合适的分类算法对数据进行分类；

（3）计算分类方法的错误率；

（4）统计每分类的对象数量；

（5）进行javaweb编程，利用数据可视化方法将数据处理的结果在web页面上显示；

**3、功能模块划分**

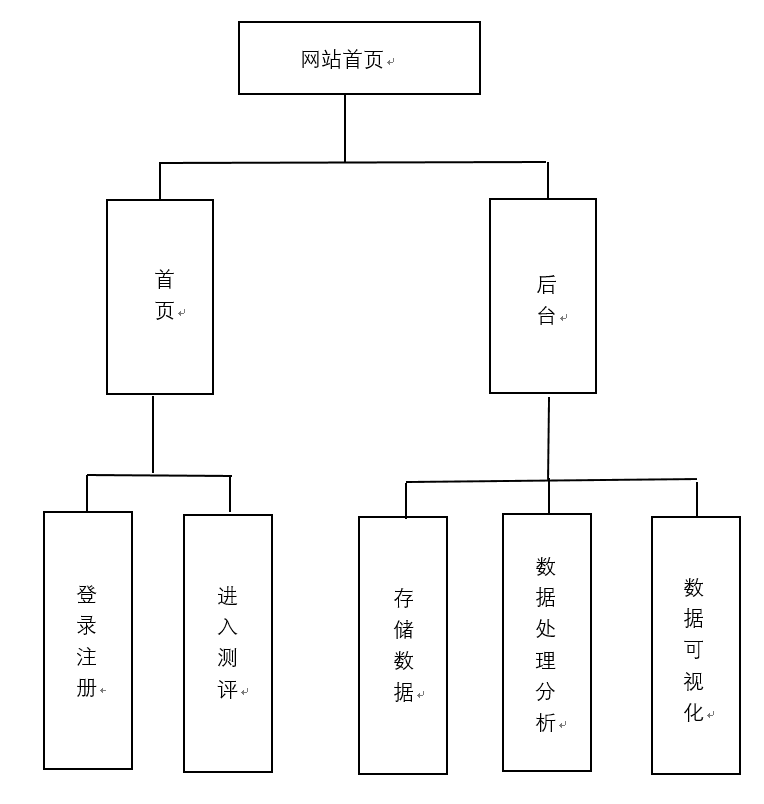


图1 总体功能模块图

**4、功能描述**

4.1、数据存储方面

将car.data.txt中的数据上传到hadoop的hdfs中，再将hdfs中存放的数据以“,”为分隔符存放到hive中创建相应的表中，并在hive中对数据进行简单的数据分析。

4.2、数据处理与分析方面

使用scala编程语言在spark框架中编写分类算法。分类算法有很多种，其中常用的有决策树（ID3、C4.5等）、朴素贝叶斯、支持向量机等。通过采用多种分类算法，计算出分类方法的错误率，从而选择最佳的分类方法。将分类出的结果存储到mysql中，由于mysql是hive的元数据库，因此与数据存储相关。

4.3、数据可视化方面

对数据集进行了分类以后，在eclipse中用java语言编写相应的代码。具体流程为：先编写ECherts中展示数据相关的jsp页面，然后编写所要展示的相关实体类，再编写相关的servlet，最后配置web.xml。运行jsp页面就可以得到所想要呈现的可视化图。

**5、系统设计**

5.1、虚拟机中运行(hadoop,hive,spark,scala,jar,IDE)

1. 启动集群：zkServer.sh
2. 启动hadoop：/apps/hadoop/sbin/start-all.sh
3. 在hdfs上创建目录、导入数据集及查看是否导入成功：

hdfs dfs -mkdir -p /data/cardata/car.data.txt

hdfs dfs –put car.data.txt /data/cardata

hdfs dfs –cat /data/cardata/car.data.txt | head -10

1. 启动hive之前先启动mysql数据库元

a）sudo service mysql start(输入密码)

b）启动hive：hive

c）hive创建表：create table car\_data(price string,fee string,doornum int,pernum int,large string,safety string,type string) row format delimited fields terminated by ',' stored as textfile;

d）hive中导入hdfs的文件：load data inpath ‘/data/cardata/car.data.txt' into table car\_data;

e）hive查询：select \* from car\_data limit 5;

1. spark启动后，输入scala，进入编程状态（输入相应的scala编程语句完成任务要求）。
2. 连接hive，分析hive表中的数据即可。
3. hive中数据通过sqoop导入mysql,eclipse中可视化即可。
4. 数据可视化

5.2： IDE中运行

1）Scala IDE中建项目，建文件（启动时发现assembly.jar包和scala不兼容：Build path中设置 fixed 2.10）。

2）分类算法参考http://spark.apache.org/docs/latest/ml-classification-regression.html：

找到下载spark-1.6.0-bin-hadoop2.6/examples/src/main/scala/org/apachespark

/examples/ml;mllib

3）算法中加载libsvm格式的文件。下载.xls，excel中frame的形式打开txt（替换为数值后），宏，转换成libsvm格式。（参考 <http://blog.csdn.net/smilehehe110/article/details/53326010>）

4）ml和mllib中算法区别：ml控制台输出，mllib输出到文件中。

5）统计每一属性的数量用 rdd.map()。

6）使用算法：

a）决策树（错误率）：

//ml决策树 结果输出到控制台。

//mllib决策树 结果输出到控制台和文件内。

b）多层感知器分类器（预测率）。

c）朴素贝叶斯（输出到文件）。

d）随机森林（错误率）：

//mllib 随机森林分类算法\_控制台输出\_文件保存。

//ml 随机森林分类算法\_控制台输出

e）分类算法的度量值

7）每个算法计算结果进行可视化：

a）每一分类属性的类别数目；

b）分类结果每一类别数目；

c）算法错误率的比较；

d）某一分类属性中每一类别的度量值。

**6、程序运行数据及其结果**

6.1、数据集分类结果

（1）每一汽车分类属性的分类结果：

买入价：(high,432) (med,432) (low,432) (vhigh,432)

维护费： (high,432) (med,432) (low,432) (vhigh,432)

车门数：(5more,432) (4,432) (2,432) (3,432)

可容纳人数：(4,576) (2,576) (more,576)

后备箱大小：(small,576) (med,576) (big,576)

安全性：(high,576) (med,576) (low,576)

（2）对汽车满意程度分类结果

(acc,384) (good,69) (unacc,1210) (vgood,65)

（3）算法错误率计算

a.决策树：mlTest Error = 0.1603603603603604

b.决策树：mllibTest Error = 0.10707803992740472

c.随机森林：ml Test Error = 0.0822429906542056

d.随机森林：mllib Test Error = 0.15789473684210525

e.朴素贝叶斯：结果已存在项目所在的文件中

f.多层感知分类器：Precision:0.725705329153605

g.Precision(0.0) = 0.8253012048192772

Precision(1.0) = 0.625

Precision(2.0) = 0.7894736842105263

Precision(3.0) = 0.3333333333333333

Recall(0.0) = 0.9362186788154897

Recall(1.0) = 0.5120481927710844

Recall(2.0) = 0.5555555555555556

Recall(3.0) = 0.07407407407407407

FPR(0.0) = 0.39545454545454545

FPR(1.0) = 0.10344827586206896

FPR(2.0) = 0.006329113924050633

FPR(3.0) = 0.006329113924050633

F1-Score(0.0) = 0.8772678762006404

F1-Score(1.0) = 0.5629139072847682

F1-Score(2.0) = 0.6521739130434783

F1-Score(3.0) = 0.12121212121212122

Weighted precision: 0.7532215757046234

Weighted recall: 0.77845220030349

Weighted F1 score: 0.7578842628015993

Weighted false positive rate: 0.2900132494682058

6.2 数据库设计

|  |  |  |
| --- | --- | --- |
| 属性 | 数据类型 | 是否为空/约束条件 |
| id | int(10) | 主键 |
| name | varchar（20） | 否 |
| num | varchar（10） | 否 |

建立的数据表都为上面表头形式，唯一不同的是表明。

**7、课程设计心得**

在这次课程设计中，从数据存储、数据统计、数据分类、数据分析到最后的可视化的过程中，首先是团队协作的力量，根据自己的所长进行合理分工，首先是对数据的分类，由于对这方面接触的很少，我们采用的方法是网上查找资料和实验室机房利用章鱼大数据平台进行学习，在学习中逐步建立起属于我们的大数据知识体系，然后通过给定的car.data数据集进行尝试，实践。尽管这样，在算法错误率计算中我们还是遇到麻烦，在尝试了多次以后还是一直报错，这次我们利用厦门大数据实验室的资料进行对代码修改，在解决了一个又一个错误之后，我们取得了最后的结果。最后一步就是数据可视化，在可视化中我们采用了百度的echarts技术，对数据进行饼图，漏斗图，柱图，折线图可视化操作，同时我们加入了前端技术，对首页和后台进行了美化。

总之，在这次课程设计中，我们基本了解了一系列大数据的流程，并从中学到了许多Hadoop生态圈中的技术，为未来的实训打下了不错的知识基础。

**附录：**

**1)car.scala**

import org.apache.spark.SparkConf;

import org.apache.spark.SparkContext;

import org.apache.spark.SparkContext.\_;

object car {

def main(args:Array[String]){

val conf=new SparkConf()

conf.setMaster("local").setAppName("wordcount")

val sc=new SparkContext(conf)

var rdd=sc.textFile("hdfs://localhost:9000/data/cardata/car.data.txt")

//每种属性统计一次

//第一个属性的数量

rdd.map(line=>(line.split(",")(0),1)).reduceByKey(\_+\_).collect().foreach(println)

rdd.map(line=>(line.split(",")(1),1)).reduceByKey(\_+\_).collect().foreach(println)

rdd.map(line=>(line.split(",")(2),1)).reduceByKey(\_+\_).collect().foreach(println)

rdd.map(line=>(line.split(",")(3),1)).reduceByKey(\_+\_).collect().foreach(println)

rdd.map(line=>(line.split(",")(4),1)).reduceByKey(\_+\_).collect().foreach(println)

rdd.map(line=>(line.split(",")(5),1)).reduceByKey(\_+\_).collect().foreach(println)

//最后一个属性的数量

rdd.map(line=>(line.split(",")(6),1)).reduceByKey(\_+\_).collect().foreach(println)

sc.stop()

}

}

**2)DecisionTreeClassificationExample.scala**  
// scalastyle:off println  
package org.apache.spark.examples.ml  
//ml决策树 结果输出到控制台  
import org.apache.spark.sql.SQLContext  
import org.apache.spark.{SparkContext, SparkConf}  
// $example on$  
import org.apache.spark.ml.Pipeline  
import org.apache.spark.ml.classification.DecisionTreeClassifier  
import org.apache.spark.ml.classification.DecisionTreeClassificationModel  
import org.apache.spark.ml.feature.{StringIndexer, IndexToString, VectorIndexer}  
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator  
// $example off$  
  
object DecisionTreeClassificationExample {  
  def main(args: Array[String]): Unit = {  
    val conf = new SparkConf().setMaster("local").setAppName("DecisionTreeClassificationExample")  
    val sc = new SparkContext(conf)  
    val sqlContext = new SQLContext(sc)  
    // $example on$  
    // Load the data stored in LIBSVM format as a DataFrame.  
    val data = sqlContext.read.format("libsvm").load("./data/mllib/car.data.libsvm.txt")  
  
    // Index labels, adding metadata to the label column.  
    // Fit on whole dataset to include all labels in index.  
    val labelIndexer = new StringIndexer()  
      .setInputCol("label")  
      .setOutputCol("indexedLabel")  
      .fit(data)  
    // Automatically identify categorical features, and index them.  
    val featureIndexer = new VectorIndexer()  
      .setInputCol("features")  
      .setOutputCol("indexedFeatures")  
      .setMaxCategories(4) // features with > 4 distinct values are treated as continuous  
      .fit(data)  
  
    // Split the data into training and test sets (30% held out for testing)  
    val Array(trainingData, testData) = data.randomSplit(Array(0.7, 0.3))  
  
    // Train a DecisionTree model.  
    val dt = new DecisionTreeClassifier()  
      .setLabelCol("indexedLabel")  
      .setFeaturesCol("indexedFeatures")  
  
    // Convert indexed labels back to original labels.  
    val labelConverter = new IndexToString()  
      .setInputCol("prediction")  
      .setOutputCol("predictedLabel")  
      .setLabels(labelIndexer.labels)  
  
    // Chain indexers and tree in a Pipeline  
    val pipeline = new Pipeline()  
      .setStages(Array(labelIndexer, featureIndexer, dt, labelConverter))  
  
    // Train model.  This also runs the indexers.  
    val model = pipeline.fit(trainingData)  
  
    // Make predictions.  
    val predictions = model.transform(testData)  
  
    // Select example rows to display.  
    predictions.select("predictedLabel", "label", "features").show(5)  
  
    // Select (prediction, true label) and compute test error  
    val evaluator = new MulticlassClassificationEvaluator()  
      .setLabelCol("indexedLabel")  
      .setPredictionCol("prediction")  
      .setMetricName("precision")  
    val accuracy = evaluator.evaluate(predictions)  
    println("Test Error = " + (1.0 - accuracy))  
  
    val treeModel = model.stages(2).asInstanceOf[DecisionTreeClassificationModel]  
    println("Learned classification tree model:\n" + treeModel.toDebugString)  
  
    // $example off$  
  }  
}

**3)DecisionTreeClassificationExample1.scala**// scalastyle:off println  
package org.apache.spark.examples.mllib  
//mllib决策树  结果输出到控制台和文件内  
// $example on$  
import org.apache.spark.mllib.tree.DecisionTree  
import org.apache.spark.mllib.tree.model.DecisionTreeModel  
import org.apache.spark.mllib.util.MLUtils  
// $example off$  
import org.apache.spark.{SparkConf, SparkContext}  
  
object DecisionTreeClassificationExample {  
  
  def main(args: Array[String]): Unit = {  
    val conf = new SparkConf().setMaster("local").setAppName("DecisionTreeClassificationExample")  
    val sc = new SparkContext(conf)  
  
    // $example on$  
    // Load and parse the data file.  
    val data = MLUtils.loadLibSVMFile(sc, "./data/mllib/car.data.libsvm.txt")  
    // Split the data into training and test sets (30% held out for testing)  
    val splits = data.randomSplit(Array(0.7, 0.3))  
    val (trainingData, testData) = (splits(0), splits(1))  
  
    // Train a DecisionTree model.  
    //  Empty categoricalFeaturesInfo indicates all features are continuous.  
    val numClasses = 4444  
    val categoricalFeaturesInfo = Map[Int, Int]()  
    val impurity = "gini"  
    val maxDepth = 5  
    val maxBins = 32  
  
    val model = DecisionTree.trainClassifier(trainingData, numClasses, categoricalFeaturesInfo,  
      impurity, maxDepth, maxBins)  
  
    // Evaluate model on test instances and compute test error  
    val labelAndPreds = testData.map { point =>  
      val prediction = model.predict(point.features)  
      (point.label, prediction)  
    }  
    val testErr = labelAndPreds.filter(r => r.\_1 != r.\_2).count().toDouble / testData.count()  
    println("Test Error = " + testErr)  
    println("Learned classification tree model:\n" + model.toDebugString)  
  
    // Save and load model  
    model.save(sc, "target/tmp/myDecisionTreeClassificationModel")  
    val sameModel = DecisionTreeModel.load(sc, "target/tmp/myDecisionTreeClassificationModel")  
    // $example off$  
  }  
}

**4)** **MulticlassMetricsExample.scala**

// scalastyle:off println

package org.apache.spark.examples.mllib

//度量分类算法的依据属性的计算

// $example on$

import org.apache.spark.mllib.classification.LogisticRegressionWithLBFGS

import org.apache.spark.mllib.evaluation.MulticlassMetrics

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.util.MLUtils

// $example off$

import org.apache.spark.{SparkContext, SparkConf}

object MulticlassMetricsExample {

def main(args: Array[String]): Unit = {

val conf = new SparkConf().setMaster("local").setAppName("MulticlassMetricsExample")

val sc = new SparkContext(conf)

// $example on$

// Load training data in LIBSVM format

val data = MLUtils.loadLibSVMFile(sc, "./data/mllib/car.data.mul.libsvm.txt")

// Split data into training (60%) and test (40%)

val Array(training, test) = data.randomSplit(Array(0.6, 0.4), seed = 11L)

training.cache()

// Run training algorithm to build the model

val model = new LogisticRegressionWithLBFGS()

.setNumClasses(4)

.run(training)

// Compute raw scores on the test set

val predictionAndLabels = test.map { case LabeledPoint(label, features) =>

val prediction = model.predict(features)

(prediction, label)

}

// Instantiate metrics object

val metrics = new MulticlassMetrics(predictionAndLabels)

// Confusion matrix

println("Confusion matrix:")

println(metrics.confusionMatrix)

// Overall Statistics

val precision = metrics.precision

val recall = metrics.recall // same as true positive rate

val f1Score = metrics.fMeasure

println("Summary Statistics")

println(s"Precision = $precision")

println(s"Recall = $recall")

println(s"F1 Score = $f1Score")

// Precision by label

val labels = metrics.labels

labels.foreach { l =>

println(s"Precision($l) = " + metrics.precision(l))

}

// Recall by label

labels.foreach { l =>

println(s"Recall($l) = " + metrics.recall(l))

}

// False positive rate by label

labels.foreach { l =>

println(s"FPR($l) = " + metrics.falsePositiveRate(l))

}

// F-measure by label

labels.foreach { l =>

println(s"F1-Score($l) = " + metrics.fMeasure(l))

}

// Weighted stats

println(s"Weighted precision: ${metrics.weightedPrecision}")

println(s"Weighted recall: ${metrics.weightedRecall}")

println(s"Weighted F1 score: ${metrics.weightedFMeasure}")

println(s"Weighted false positive rate: ${metrics.weightedFalsePositiveRate}")

// $example off$

}

}

**5) MultilayerPerceptronClassifierExample.scala**

// scalastyle:off println

package org.apache.spark.examples.ml

//多层感知器分类器

import org.apache.spark.{SparkContext, SparkConf}

import org.apache.spark.sql.SQLContext

// $example on$

import org.apache.spark.ml.classification.MultilayerPerceptronClassifier

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

// $example off$

/\*\*

\* An example for Multilayer Perceptron Classification.

\*/

object MultilayerPerceptronClassifierExample {

def main(args: Array[String]): Unit = {

val conf = new SparkConf().setMaster("local").setAppName("MultilayerPerceptronClassifierExample")

val sc = new SparkContext(conf)

val sqlContext = new SQLContext(sc)

// $example on$

// Load the data stored in LIBSVM format as a DataFrame.

val data = sqlContext.read.format("libsvm")

.load("./data/mllib/car.data.mul.libsvm.txt")

// Split the data into train and test

val splits = data.randomSplit(Array(0.6, 0.4), seed = 1234L)

val train = splits(0)

val test = splits(1)

// specify layers for the neural network:

// input layer of size 6 (features), two intermediate of size 5 and 4

// and output of size 4 (classes)

val layers = Array[Int](6, 5, 4, 4)

// create the trainer and set its parameters

val trainer = new MultilayerPerceptronClassifier()

.setLayers(layers)

.setBlockSize(128)

.setSeed(1234L)

.setMaxIter(100)

// train the model

val model = trainer.fit(train)

// compute precision on the test set

val result = model.transform(test)

val predictionAndLabels = result.select("prediction", "label")

val evaluator = new MulticlassClassificationEvaluator()

.setMetricName("precision")

println("Precision:" + evaluator.evaluate(predictionAndLabels))

// $example off$

sc.stop()

}

}

**6)** **NaiveBayesExample.scala**

// scalastyle:off println

package org.apache.spark.examples.mllib

//朴素贝叶斯

// $example on$

import org.apache.spark.mllib.classification.{NaiveBayes, NaiveBayesModel}

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.regression.LabeledPoint

// $example off$

import org.apache.spark.{SparkConf, SparkContext}

object NaiveBayesExample {

def main(args: Array[String]) : Unit = {

val conf = new SparkConf().setMaster("local").setAppName("NaiveBayesExample")

val sc = new SparkContext(conf)

// $example on$

val data = sc.textFile("./data/mllib/car.data.replace.txt")

val parsedData = data.map { line =>

val parts = line.split('\t')

LabeledPoint(parts(6).toDouble, Vectors.dense(parts(0).split('\t').map(\_.toDouble)))

}

// Split data into training (60%) and test (40%).

val splits = parsedData.randomSplit(Array(0.6, 0.4), seed = 11L)

val training = splits(0)

val test = splits(1)

val model = NaiveBayes.train(training, lambda = 1.0, modelType = "multinomial")

val predictionAndLabel = test.map(p => (model.predict(p.features), p.label))

val accuracy = 1.0 \* predictionAndLabel.filter(x => x.\_1 == x.\_2).count() / test.count()

// Save and load model

model.save(sc, "target/tmp/myNaiveBayesModel")

val sameModel = NaiveBayesModel.load(sc, "target/tmp/myNaiveBayesModel")

// $example off$

}

}

**7)** **RandomForestClassificationExample.scala**

// scalastyle:off println

package org.apache.spark.examples.mllib

//mllib 随机森林分类算法\_控制台输出\_文件保存

import org.apache.spark.{SparkContext, SparkConf}

// $example on$

import org.apache.spark.mllib.tree.RandomForest

import org.apache.spark.mllib.tree.model.RandomForestModel

import org.apache.spark.mllib.util.MLUtils

// $example off$

object RandomForestClassificationExample {

def main(args: Array[String]): Unit = {

val conf = new SparkConf().setMaster("local").setAppName("RandomForestClassificationExample")

val sc = new SparkContext(conf)

// $example on$

// Load and parse the data file.

val data = MLUtils.loadLibSVMFile(sc, "./data/mllib/car.data.libsvm.txt")

// Split the data into training and test sets (30% held out for testing)

val splits = data.randomSplit(Array(0.7, 0.3))

val (trainingData, testData) = (splits(0), splits(1))

// Train a RandomForest model.

// Empty categoricalFeaturesInfo indicates all features are continuous.

val numClasses = 4444

val categoricalFeaturesInfo = Map[Int, Int]()

val numTrees = 3 // Use more in practice.

val featureSubsetStrategy = "auto" // Let the algorithm choose.

val impurity = "gini"

val maxDepth = 4

val maxBins = 32

val model = RandomForest.trainClassifier(trainingData, numClasses, categoricalFeaturesInfo,

numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins)

// Evaluate model on test instances and compute test error

val labelAndPreds = testData.map { point =>

val prediction = model.predict(point.features)

(point.label, prediction)

}

val testErr = labelAndPreds.filter(r => r.\_1 != r.\_2).count.toDouble / testData.count()

println("Test Error = " + testErr)

println("Learned classification forest model:\n" + model.toDebugString)

// Save and load model

model.save(sc, "target/tmp/myRandomForestClassificationModel")

val sameModel = RandomForestModel.load(sc, "target/tmp/myRandomForestClassificationModel")

// $example off$

}

}

**8)** **RandomForestClassifierExample.scala**

// scalastyle:off println

package org.apache.spark.examples.ml

//ml 随机森林分类算法\_控制台输出

import org.apache.spark.sql.SQLContext

import org.apache.spark.{SparkConf, SparkContext}

// $example on$

import org.apache.spark.ml.Pipeline

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

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

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

// $example off$

object RandomForestClassifierExample {

def main(args: Array[String]): Unit = {

val conf = new SparkConf().setMaster("local").setAppName("RandomForestClassifierExample")

val sc = new SparkContext(conf)

val sqlContext = new SQLContext(sc)

// $example on$

// Load and parse the data file, converting it to a DataFrame.

val data = sqlContext.read.format("libsvm").load("./data/mllib/car.data.libsvm.txt")

// Index labels, adding metadata to the label column.

// Fit on whole dataset to include all labels in index.

val labelIndexer = new StringIndexer()

.setInputCol("label")

.setOutputCol("indexedLabel")

.fit(data)

// Automatically identify categorical features, and index them.

// Set maxCategories so features with > 4 distinct values are treated as continuous.

val featureIndexer = new VectorIndexer()

.setInputCol("features")

.setOutputCol("indexedFeatures")

.setMaxCategories(4)

.fit(data)

// Split the data into training and test sets (30% held out for testing)

val Array(trainingData, testData) = data.randomSplit(Array(0.7, 0.3))

// Train a RandomForest model.

val rf = new RandomForestClassifier()

.setLabelCol("indexedLabel")

.setFeaturesCol("indexedFeatures")

.setNumTrees(10)

// Convert indexed labels back to original labels.

val labelConverter = new IndexToString()

.setInputCol("prediction")

.setOutputCol("predictedLabel")

.setLabels(labelIndexer.labels)

// Chain indexers and forest in a Pipeline

val pipeline = new Pipeline()

.setStages(Array(labelIndexer, featureIndexer, rf, labelConverter))

// Train model. This also runs the indexers.

val model = pipeline.fit(trainingData)

// Make predictions.

val predictions = model.transform(testData)

// Select example rows to display.

predictions.select("predictedLabel", "label", "features").show(5)

// Select (prediction, true label) and compute test error

val evaluator = new MulticlassClassificationEvaluator()

.setLabelCol("indexedLabel")

.setPredictionCol("prediction")

.setMetricName("precision")

val accuracy = evaluator.evaluate(predictions)

println("Test Error = " + (1.0 - accuracy))

val rfModel = model.stages(2).asInstanceOf[RandomForestClassificationModel]

println("Learned classification forest model:\n" + rfModel.toDebugString)

// $example off$

sc.stop()

}

}