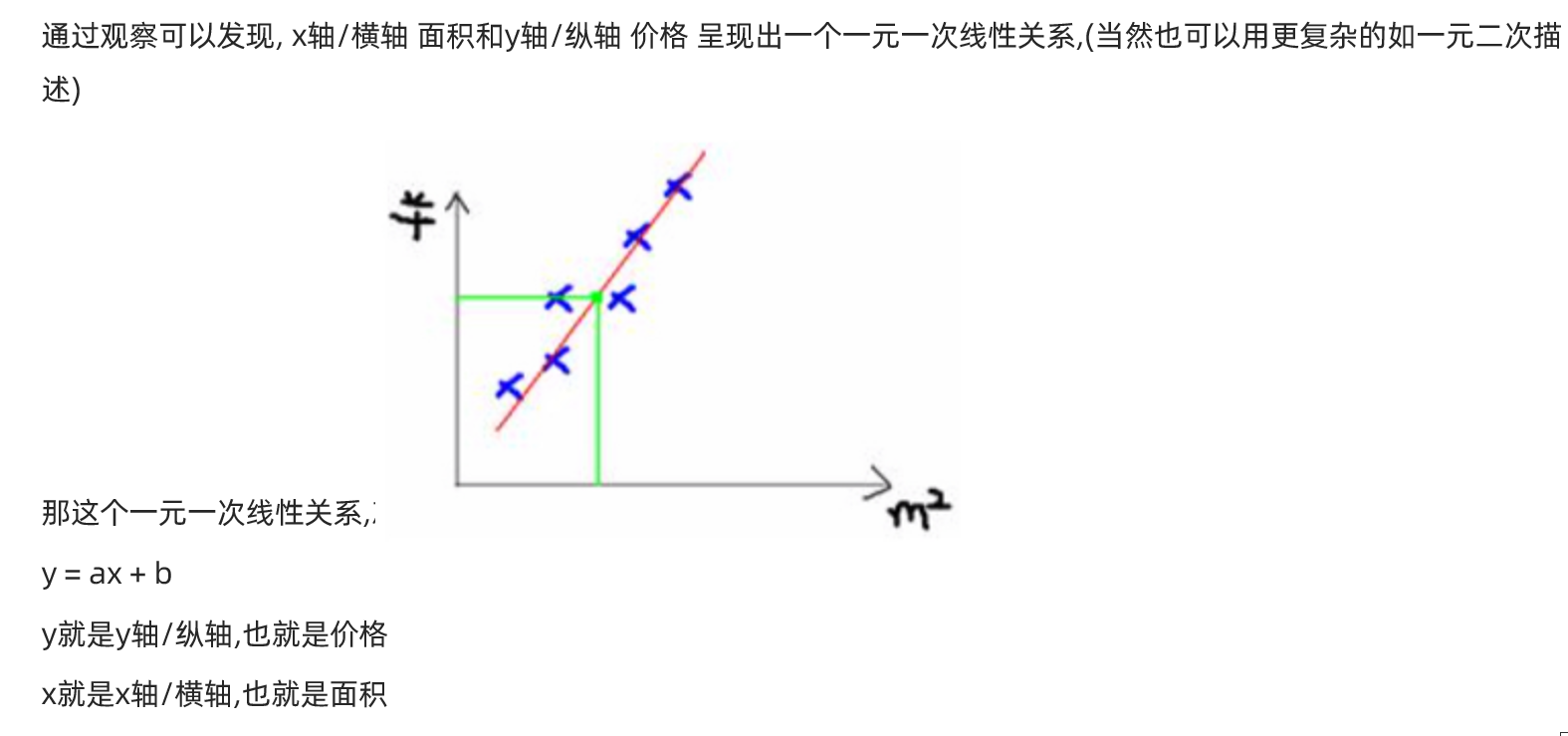
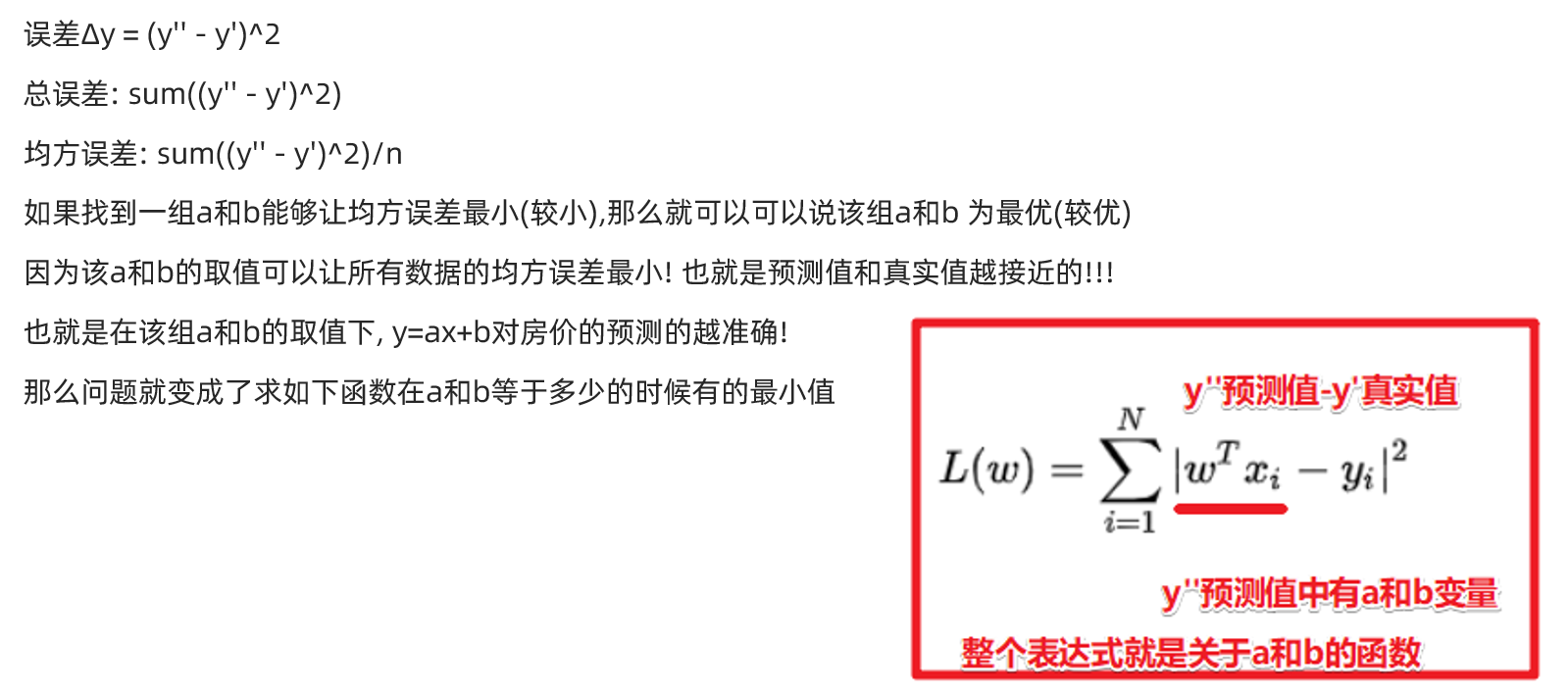
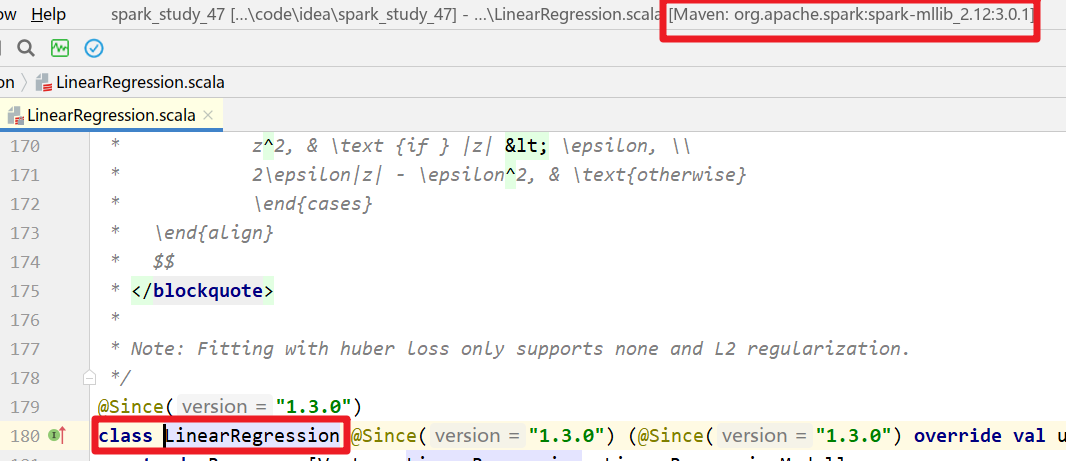
## 线性回归算法-房价预测案例

### 需求







特征列:  
|房屋编号mlsNum|城市city|平方英尺|卧室数bedrooms|卫生间数bathrooms|车库garage|年龄age|房屋占地面积acres|  
标签列:  
房屋价格price

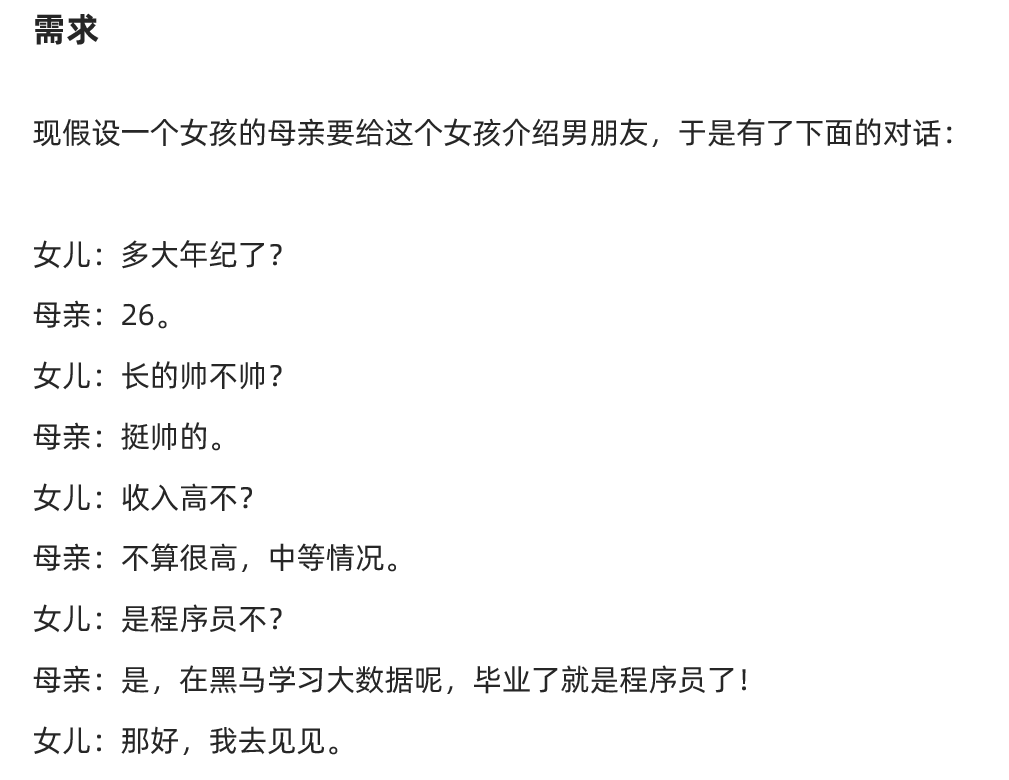
### 代码实现

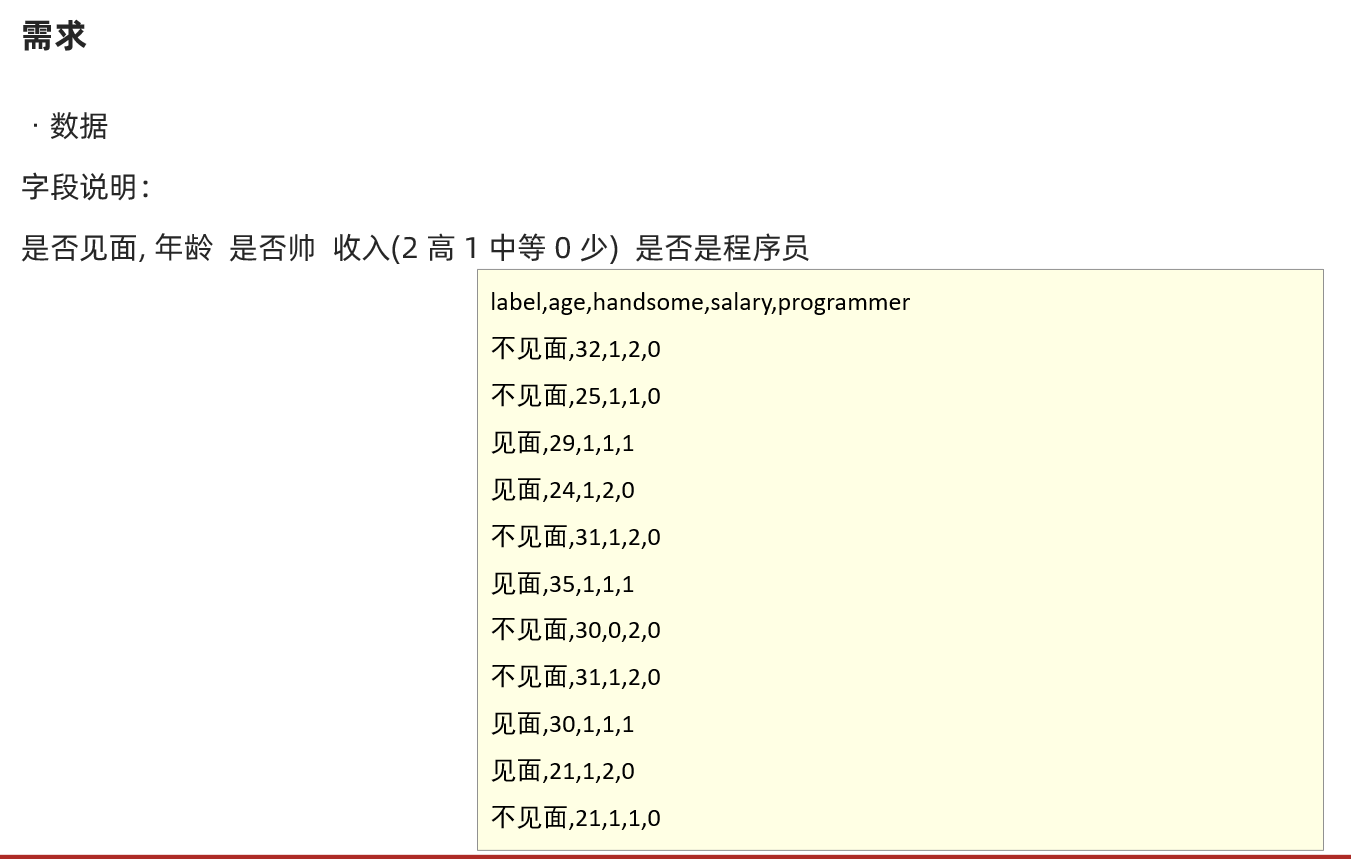
0.准备环境  
1.加载数据  
2.特征处理   
3.数据集划分0.8训练集/0.2测试集  
4.使用训练集训练线性回归模型  
5.使用测试集对模型进行测试  
6.计算误差rmse均方误差  
7.模型保存(save)方便后续使用(load)  
8.关闭资源

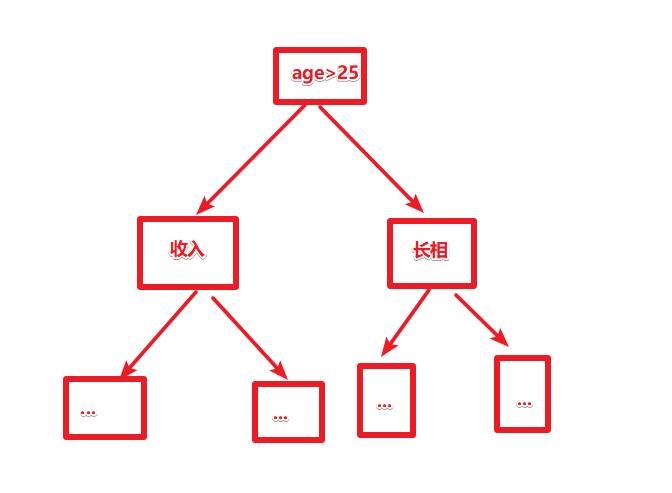
package cn.itcast.hello;  
  
import org.apache.spark.ml.evaluation.RegressionEvaluator;  
import org.apache.spark.ml.feature.VectorAssembler;  
import org.apache.spark.ml.regression.LinearRegression;  
import org.apache.spark.ml.regression.LinearRegressionModel;  
import org.apache.spark.sql.Dataset;  
import org.apache.spark.sql.Row;  
import org.apache.spark.sql.SparkSession;  
import org.apache.spark.sql.streaming.StreamingQueryException;  
  
import java.io.IOException;  
import java.util.concurrent.TimeoutException;  
  
/\*\*  
 \* Author itcast  
 \* Desc 演示使用Java语言开发SparkMlLib-线性回归算法-房价预测案例  
 \*/  
public class JavaSparkDemo05 {  
 public static void main(String[] args) throws TimeoutException, StreamingQueryException, IOException {  
 //0.TODO 准备环境  
 SparkSession spark = SparkSession.builder().appName("JavaSparkDemo").master("local[\*]")  
 .config("spark.sql.shuffle.partitions", "4")  
 .getOrCreate();  
 spark.sparkContext().setLogLevel("WARN");  
  
 //TODO 1.加载数据  
 Dataset<Row> homeDataDF = spark.read()  
 .format("csv")  
 .option("sep", "|")//指定分隔符  
 .option("header", "true")//是否有表头  
 .option("inferSchema", "true")//是否自动推断约束  
 .load("data/input/homeprice.data");  
 homeDataDF.printSchema();  
 homeDataDF.show();  
 /\*  
 root  
 |-- mlsNum: integer (nullable = true)  
 |-- city: string (nullable = true)  
 |-- sqFt: double (nullable = true)  
 |-- bedrooms: integer (nullable = true)  
 |-- bathrooms: integer (nullable = true)  
 |-- garage: integer (nullable = true)  
 |-- age: integer (nullable = true)  
 |-- acres: double (nullable = true)  
 |-- price: double (nullable = true)  
//|房屋编号|城市|平方英尺|卧室数|卫生间数|车库|年龄|房屋占地面积|房屋价格  
+-------+------------+-------+--------+---------+------+---+-----+---------+  
| mlsNum| city| sqFt|bedrooms|bathrooms|garage|age|acres| price|  
+-------+------------+-------+--------+---------+------+---+-----+---------+  
|4424109|Apple Valley| 1634.0| 2| 2| 2| 33| 0.04| 119900.0|  
|4404211| Rosemount|13837.0| 4| 6| 4| 17|14.46|3500000.0|  
|4339082| Burnsville| 9040.0| 4| 6| 8| 12| 0.74|2690000.0|  
 \*/  
  
 //TODO 2.特征处理  
 //特征选择  
 Dataset<Row> featuredDF = homeDataDF.select("sqFt", "age", "acres", "price");  
 //特征向量化  
 Dataset<Row> vectorDF = new VectorAssembler()  
 .setInputCols(new String[]{"sqFt", "age", "acres"})//指定要对哪些特征做向量化  
 .setOutputCol("features")//向量化之后的特征列列名  
 .transform(featuredDF);  
 vectorDF.printSchema();  
 vectorDF.show();  
 /\*  
 root  
 |-- sqFt: double (nullable = true)  
 |-- age: integer (nullable = true)  
 |-- acres: double (nullable = true)  
 |-- price: double (nullable = true)  
 |-- features: vector (nullable = true)  
  
 +-------+---+-----+---------+--------------------+  
 | sqFt|age|acres| price| features|  
 +-------+---+-----+---------+--------------------+  
 | 1634.0| 33| 0.04| 119900.0| [1634.0,33.0,0.04]|  
 |13837.0| 17|14.46|3500000.0|[13837.0,17.0,14.46]|  
 | 9040.0| 12| 0.74|2690000.0| [9040.0,12.0,0.74]|  
 \*/  
  
  
 //TODO 3.数据集划分0.8训练集/0.2测试集  
 Dataset<Row>[] arr = vectorDF.randomSplit(new double[]{0.8, 0.2}, 100);  
 Dataset<Row> trainSet = arr[0];  
 Dataset<Row> testSet = arr[1];  
  
 //TODO 4.构建线性回归模型并使用训练集训练  
 LinearRegressionModel model = new LinearRegression()  
 .setFeaturesCol("features")//设置特征列(应该设置向量化之后的)  
 .setLabelCol("price")//设置标签列(数据中已经标记好的原本的价格)  
 .setPredictionCol("predict\_price")//设置预测列(后续做预测时预测的价格)  
 .setMaxIter(10)//最大迭代次数  
 .fit(trainSet);//使用训练集进行训练  
  
 //TODO 5.使用测试集对模型进行测试/预测  
 Dataset<Row> testResult = model.transform(testSet);  
 testResult.show(false);  
  
 //TODO 6.计算误差rmse均方误差  
 double rmse = new RegressionEvaluator()//创建误差评估器  
 .setMetricName("rmse") //设置要计算的误差名称,均方根误差 (sum((y-y')^2)/n)^0.5  
 .setLabelCol("price")//设置真实值是哪一列  
 .setPredictionCol("predict\_price")//设置预测值是哪一列  
 .evaluate(testResult);//对数据中的真实值和预测值进行误差计算  
 System.out.println("rmse为:" + rmse);  
  
 //TODO 7.模型保存(save)方便后续使用(load)  
 //model.save("path");  
 //LinearRegressionModel lmodel = LinearRegressionModel.load("path");  
  
 //TODO 8.关闭资源  
 spark.stop();  
 }  
}

## 决策树算法-相亲决策案例

### 需求







### 思路

从历史数据中找出一些"关键因素"来作为树的节点/决策依据

在信息论中, 信息熵可以衡量系统的不确定性,如果让信息熵变小就可以让系统的不确定性变小/也就是变得更加确定!

而我们的目标也是让系统的不确定性变小, 让我们的程序能够更加特征尽快的做出确定性的决策/判断

所以现在的目标就变成了选择一些能够让系统的信息熵变小的特征作为树的节点/决策依据,也就是要选择信息增益较大的特征(信息增益指的是信息熵减少的程度!)

而且信息熵/信息增益能够量化!也就是能够计算!

总结: 决策树算法中需要选择当前信息增益最大的特征作为树的节点/决策依据,依次选择相应的特征作为树的节点/决策依据,直到形成一颗决策树!

### 代码实现

from ml import Pipeline  
from ml.classification import DecisionTreeClassifier, DecisionTreeClassificationModel  
from ml.evaluation import MulticlassClassificationEvaluator  
from ml.feature import StringIndexer, VectorAssembler, IndexToString  
from sql import SparkSession  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 """  
 演示pyspark-决策树分类案例-相亲决策案例  
 """  
 # 0.创建环境  
 spark = SparkSession.builder.appName("pyspark").master("local[\*]") \  
 .config("spark.sql.shuffle.partitions", "2") \  
 .getOrCreate()  
  
 # 0.加载数据  
 df = spark.read.format("csv")\  
 .option("sep", ",") \  
 .option("header", "true") \  
 .option("inferSchema", "true") \  
 .load("data/appointment.data")  
 df.printSchema()  
 df.show()  
  
 # 1.标签数值化  
 # 设置哪一列是字符串格式的标签列  
 # 设置标签数值化之后的列名  
 stringIndexer = StringIndexer()\  
 .setInputCol("label")\  
 .setOutputCol("label\_index").fit(df)  
  
 # 2.特征向量化  
 # 设置特征列有哪些列  
 # 设置特征向量化之后的列名  
 vectorAssembler = VectorAssembler() \  
 .setInputCols(["age", "handsome", "salary", "programmer"]) \  
 .setOutputCol("features")  
 # 后续可以使用pipeline做统一的训练和转换,所以这里不需要调用transform  
  
  
 # 3.构建决策树模型  
 # 设置向量化之后的特征列  
 # 设置数值化之后的标签列  
 # 设置预测列名称(预测出来的是数字)  
 decisionTreeClassifier = DecisionTreeClassifier()\  
 .setFeaturesCol("features")\  
 .setLabelCol("label\_index")\  
 .setPredictionCol("predict\_index")\  
 .setMaxDepth(5)  
 # 后续可以使用pipeline做统一的训练和转换,所以这里不需要调用fit  
  
 # 4.标签还原  
 # 预测出来的标签数值还原为字符串  
 # 设置数值化的预测列  
 # 设置转为字符串后的预测列的列名  
 # 设置数值和字符串的对应关系在哪里  
 indexToString = IndexToString().setInputCol("predict\_index")\  
 .setOutputCol("predict\_String")\  
 .setLabels(stringIndexer.labels)  
  
 # 5.数据集划分  
 trainSet,testSet = df.randomSplit(weights=[0.8,0.2])  
  
 # 6.构建pipeline  
 pipeline = Pipeline().setStages([stringIndexer,vectorAssembler,decisionTreeClassifier,indexToString])  
  
 # 7.使用训练集训练  
 model = pipeline.fit(trainSet)  
  
 # 8.使用测试集测试模型  
 testResult = model.transform(testSet)  
 testResult.show()  
  
 # 9.输出决策过程  
 print("模型决策过程为: \n %s" % model.stages[2].toDebugString)  
  
 # 10评估模型误差/好坏  
 accuracy = MulticlassClassificationEvaluator(labelCol="label\_index", predictionCol="predict\_index", metricName="accuracy")\  
 .evaluate(testResult)  
 print("测试集错误率为: %f" % (1.0 - accuracy))  
  
 # model.save("path")  
 # model2 =DecisionTreeClassificationModel.load("path")  
  
  
  
 spark.stop()