MLlib分类实战

1. 逻辑回归详解
   1. 逻辑回归不是回归算法

逻辑回归不是回归算法，而是分类算法。特别是用在二分分类中。

* 1. 逻辑回归的数学基础

逻辑回归的具体公式如下：



与线性回归相同，这里的θ是逻辑回归的参数，即回归系数，如果再将其进一步变形，使其能够反映二元分类问题的公式，则公式为：



这里y值是有已有的数据集中数据和θ共同决定。实际上这个公式求计算是在满足一定条件下，最终取值的对数几率，即由数据集的可能性的比值的对数变换得到。通过公式表示为：



通过这个逻辑回归倒退公式可以看到，最终逻辑回归的计算可以转化成数据集的特征向量与系数θ共同完成，然后求得其加权和作为最终的判断结果。

* 1. 一元逻辑回归示例

数据集位置：D:\devs\data\spark\D07\u.txt

样本数据集如下：



代码位置：

D:\devs\studies\scala\HelloWorld\src\gtl\spark\java\example\C07\LogisticRegression.java

代码如下：

**package** gtl.spark.java.example.C07;  
  
**import** org.apache.spark.api.java.JavaRDD;  
**import** org.apache.spark.api.java.JavaSparkContext;  
**import** org.apache.spark.mllib.classification.LogisticRegressionModel;  
**import** org.apache.spark.mllib.linalg.Vector;  
**import** org.apache.spark.sql.SparkSession;  
**import** org.apache.spark.mllib.classification.LogisticRegressionWithSGD;  
**import** org.apache.spark.mllib.linalg.Vectors;  
**import** org.apache.spark.mllib.regression.LabeledPoint;  
  
*//一元逻辑回归示例***public class** LogisticRegression {  
 **public static** String *DATA\_FILE* = **"D:\\devs\\data\\spark\\D07\\u.txt"**;  
  
 **public static void** main(String[] args){  
 SparkSession spark = SparkSession  
 .*builder*()  
 .master(**"local"**)  
 .appName(**"LogisticRegression"**)  
 .getOrCreate();  
 JavaSparkContext sc = JavaSparkContext.*fromSparkContext*(spark.sparkContext());  
  
 JavaRDD<String> data = sc.textFile(*DATA\_FILE*); *//获取数据集* JavaRDD<LabeledPoint> parsedData = data.map(r -> {  
 String t[] = r.split(**"\\|"**); *//“|”是转义字符,必须得加"\\"* LabeledPoint s = **new** LabeledPoint(  
 Double.*valueOf*(t[0]),  
 Vectors.*dense*( Double.*valueOf*(t[1])));  
 **return** s;  
 }).cache(); *//转化数据格式* LogisticRegressionModel model = LogisticRegressionWithSGD.*train*(parsedData.rdd(),50); *//建立模型* Vector target = Vectors.*dense*(-1); *//创建测试值* **double** resulet = model.predict(target); *//根据模型计算结果* System.***out***.println(resulet); *//打印结果* }  
}

输出结果为：

1.0

* 1. 多元逻辑回归示例

本小节采用的例子是MLlib中自带的数据集sample\_libsvm\_data.txt。

libSVM的数据格式：

Label 1: value 2: value ... .

Label是类别的标识，比如0或者1，可根据需要自己随意定，比如100, 20, 13。

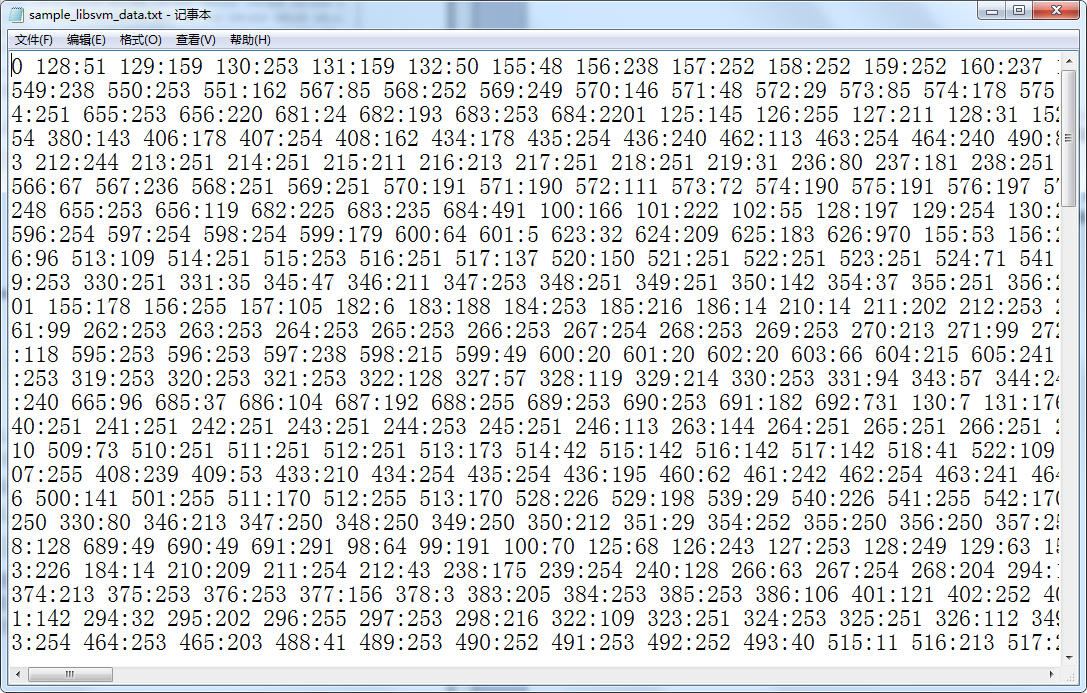
Value是要训练的数据，从分类的角度来说是特征值，数据之间使用空格隔开。而每个“:”用于标注向量的序号和向量值。列如数据：

1 1:12 3:7 4:1

指的是表示为1的那组数据集，第1个数据值为12，第3个数据值为7，第4个数据值为1，第二个数据缺失。特征冒号前面的（姑且称做序号）可以不连续。这样做的好处可以减少内存的使用，并提高矩阵内积时的运算速度。

数据集位置：D:\devs\data\spark\D07\sample\_libsvm\_data.txt

数据集如下：



代码位置：

D:\devs\studies\scala\HelloWorld\src\gtl\spark\java\example\C07\LogisticRegression2.java

代码如下：

**package** gtl.spark.java.example.C07;  
  
**import** org.apache.spark.api.java.JavaRDD;  
**import** org.apache.spark.api.java.JavaSparkContext;  
**import** org.apache.spark.mllib.classification.LogisticRegressionModel;  
**import** org.apache.spark.mllib.classification.LogisticRegressionWithSGD;  
**import** org.apache.spark.mllib.regression.LabeledPoint;  
**import** org.apache.spark.mllib.util.MLUtils;  
**import** org.apache.spark.sql.SparkSession;  
**import** java.util.Arrays;  
  
*// 多元逻辑回归示例***public class** LogisticRegression2 {  
 **public static** String *DATA\_FILE* = **"D:\\devs\\data\\spark\\D07\\sample\_libsvm\_data.txt"**;  
  
 **public static void** main(String[] args){  
 SparkSession spark = SparkSession  
 .*builder*()  
 .master(**"local"**)  
 .appName(**"LogisticRegression2"**)  
 .getOrCreate();  
 JavaSparkContext sc = JavaSparkContext.*fromSparkContext*(spark.sparkContext());  
  
 JavaRDD<LabeledPoint> data = MLUtils.*loadLibSVMFile*(sc.sc(),*DATA\_FILE*).toJavaRDD();*//读取数据文件* LogisticRegressionModel model = LogisticRegressionWithSGD.*train*(data.rdd(),50);*//训练数据模型* System.***out***.println(model.weights());*//打印θ值* System.***out***.println(model.weights().size());*//打印θ值个数* System.***out***.println(Arrays.*stream*(model.weights().toArray()).filter(x -> x != 0).toArray().**length**);*//打印θ值不为0的数* }  
}

输出结果为：

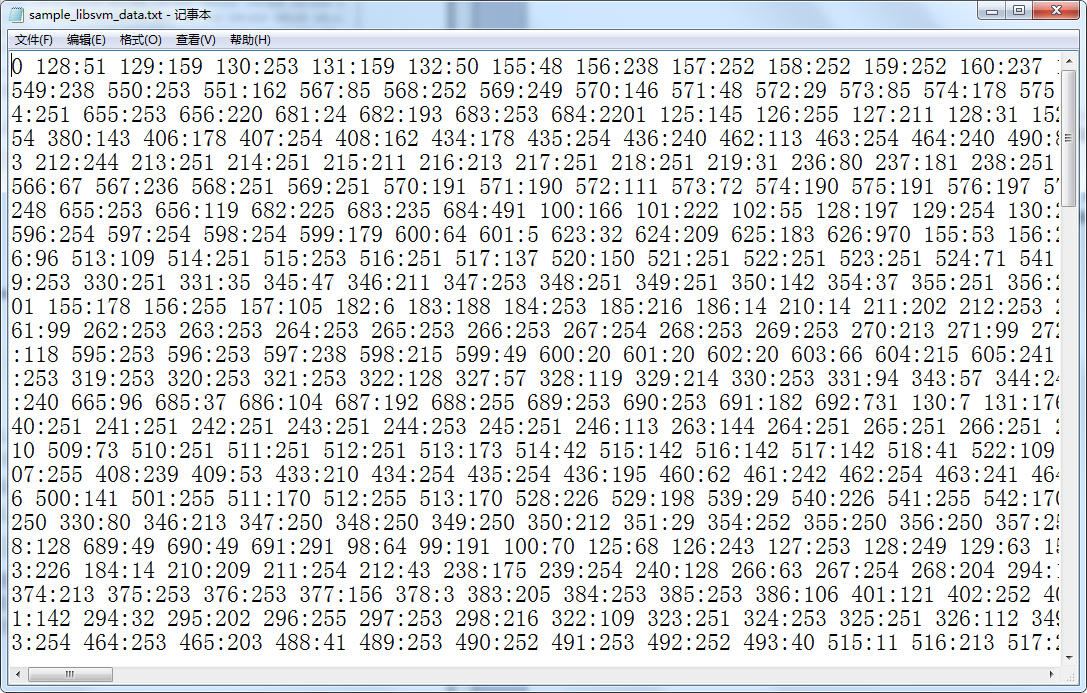


* 1. MLlib逻辑回归验证

MLlib中MulticlasMetrics类是对数据进行分类的类，其中包括各种方法，通过调用其中的accurate方法可以对验证数据进行验证。注意spark2.0后使用accurate方法，不使用precision方法。

数据集位置：D:\devs\data\spark\D07\sample\_libsvm\_data.txt

数据集如下：



代码位置：

D:\devs\studies\scala\HelloWorld\src\gtl\spark\java\example\C07\LogisticRegression3.java

代码如下：

**package** gtl.spark.java.example.C07;  
  
**import** org.apache.spark.api.java.JavaPairRDD;  
**import** org.apache.spark.api.java.JavaRDD;  
**import** org.apache.spark.api.java.JavaSparkContext;  
**import** org.apache.spark.mllib.classification.LogisticRegressionModel;  
**import** org.apache.spark.mllib.classification.LogisticRegressionWithSGD;  
**import** org.apache.spark.mllib.regression.LabeledPoint;  
**import** org.apache.spark.mllib.util.MLUtils;  
**import** org.apache.spark.mllib.evaluation.MulticlassMetrics;  
**import** org.apache.spark.sql.SparkSession;  
**import** scala.Tuple2;  
  
*//逻辑回归验证***public class** LogisticRegression3 {  
 **public static** String *DATA\_FILE* = **"D:\\devs\\data\\spark\\D07\\sample\_libsvm\_data.txt"**;  
  
 **public static void** main(String[] args) {  
 SparkSession spark = SparkSession  
 .*builder*()  
 .master(**"local"**)  
 .appName(**"LogisticRegression3"**)  
 .getOrCreate();  
 JavaSparkContext sc = JavaSparkContext.*fromSparkContext*(spark.sparkContext());  
  
 JavaRDD<LabeledPoint> data = MLUtils.*loadLibSVMFile*(sc.sc(),*DATA\_FILE*).toJavaRDD();*//读取数据集* JavaRDD<LabeledPoint> splits[] = data.randomSplit(**new double**[]{0.6, 0.4}, 11L); *//对数据集切分* JavaRDD<LabeledPoint> parsedData = splits[0];*//分割训练数据* JavaRDD<LabeledPoint> parseTtest = splits[1];*//分割测试数据* LogisticRegressionModel model = LogisticRegressionWithSGD.*train*(parsedData.rdd(),50);*//训练模型* System.***out***.println(model.weights());*//打印θ值* JavaPairRDD<Object, Object> predictionAndLabels = parseTtest.mapToPair(r -> {  
 **return new** Tuple2<Object, Object>(model.predict(r.features()), r.label());  
 }); *//存储测试和预测值* MulticlassMetrics metrics = **new** MulticlassMetrics(predictionAndLabels.rdd());*//创建验证类* **double** accuracy = metrics.accuracy();*//计算验证值* System.***out***.println(**"accuracy = "** + accuracy);*//打印验证值* }  
}

* 1. MLlib逻辑回归实例：胃癌的转移判断

某研究人员在探讨肾细胞癌转移的有关临床病理因素研究中，收集了一批行根治性肾切除术的患者的肾癌标本资料，从中抽取26例资料作为示例进行logistic回归分析（本例来自《卫生统计学》第四版第11章）。

数据说明：

y：肾细胞癌转移情况（有转移y=1；无转移y=0）；

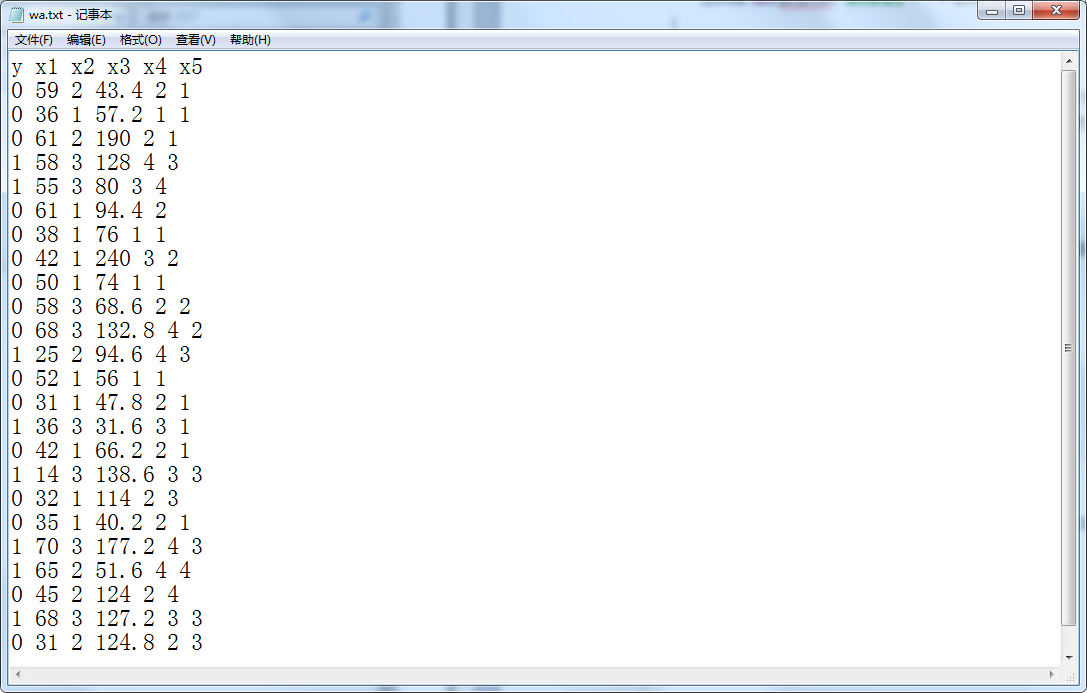
x1：确诊时患者的年龄（岁）；

x2：肾细胞癌血管内皮生长因子（VEGF），其阳性表述由低到高共3个等级；

x3：肾细胞癌组织内微血管数（MVC）；

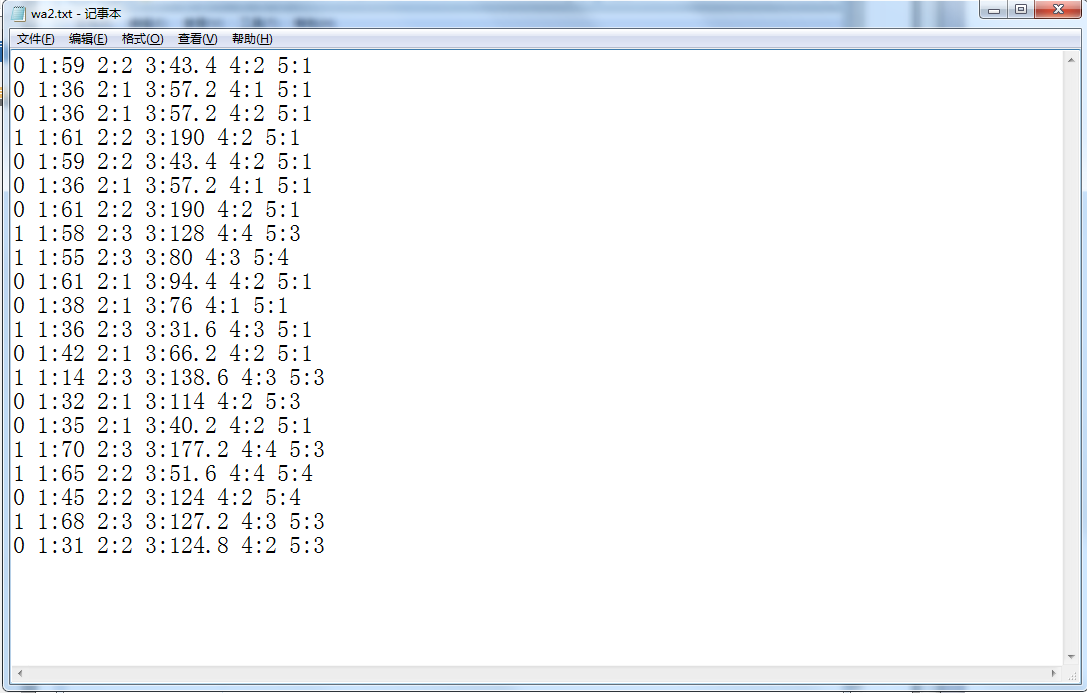
x4：肾癌细胞核组织学分级，由低到高共4级；

x5：肾细胞癌分期，由低到高共4期。



数据集位置：D:\devs\data\spark\D07\wa2.txt

数据集如下：



代码位置：

D:\devs\studies\scala\HelloWorld\src\gtl\spark\java\example\C07\LogisticRegression4.java

代码如下：

**package** gtl.spark.java.example.C07;  
  
**import** org.apache.spark.api.java.JavaPairRDD;  
**import** org.apache.spark.api.java.JavaRDD;  
**import** org.apache.spark.api.java.JavaSparkContext;  
**import** org.apache.spark.mllib.classification.LogisticRegressionModel;  
**import** org.apache.spark.mllib.classification.LogisticRegressionWithSGD;  
**import** org.apache.spark.mllib.evaluation.MulticlassMetrics;  
**import** org.apache.spark.mllib.linalg.Vectors;  
**import** org.apache.spark.mllib.regression.LabeledPoint;  
**import** org.apache.spark.mllib.util.MLUtils;  
**import** org.apache.spark.sql.SparkSession;  
**import** scala.Tuple2;  
  
*//逻辑回归***public class** LogisticRegression4 {  
 **public static** String *DATA\_FILE* = **"D:\\devs\\data\\spark\\D07\\wa2.txt"**;  
  
 **public static void** main(String[] args) {  
 SparkSession spark = SparkSession  
 .*builder*()  
 .master(**"local"**)  
 .appName(**"LogisticRegression4"**)  
 .getOrCreate();  
 JavaSparkContext sc = JavaSparkContext.*fromSparkContext*(spark.sparkContext());  
  
 JavaRDD<LabeledPoint> data = MLUtils.*loadLibSVMFile*(sc.sc(), *DATA\_FILE*).toJavaRDD();*//读取数据集* JavaRDD<LabeledPoint> splits[] = data.randomSplit(**new double**[]{0.7, 0.3}, 11L);*//对数据集切分* JavaRDD<LabeledPoint> parsedData = splits[0];*//分割训练数据* JavaRDD<LabeledPoint> parseTtest = splits[1];*//分割测试数据* LogisticRegressionModel model = LogisticRegressionWithSGD.*train*(parsedData.rdd(), 50);  
 JavaPairRDD<Object, Object> predictionAndLabels = parseTtest.mapToPair(r -> {  
 **return new** Tuple2<Object, Object>(model.predict(r.features()), r.label());  
 }); *//存储测试和预测值* MulticlassMetrics metrics = **new** MulticlassMetrics(predictionAndLabels.rdd());*//创建验证类* **double** accuracy = metrics.accuracy();*//计算验证值* **double** patient = model.predict(Vectors.dense(70, 3, 180.0, 4, 3));*//计算患者可能性* **if** (patient == 1)  
 System.***out***.println(**"患者的胃癌有几率转移。"**); *//做出判断* **else** System.***out***.println(**"患者的胃癌没有几率转移。"**);*//做出判断* }  
}

输出结果为：



1. 支持向量机详解
   1. 三角还是圆

通过找到支持向量从而获得分类平面的方法，称为支持向量机。因此支持向量机的目的就是，通过划分最优的平面从而使不同的类别分开。

* 1. 支持向量机的数学基础

线性回归模型可以使用如下公式来表示，即：



其中a和b分别是公式的系数。若将其推广到线性空间中，则公式如下：



这里人为将空间分为三部分，当f(x)=0时，可以讲x认为属于分割面上的点。而当f(x)>0时，可以将其近似地认为f(x)=1，从而将其确定为一个分类。以此类推，f(x)<0时，可以将其近似地认为f(x)=-1，从而将其确定为另一个分类。

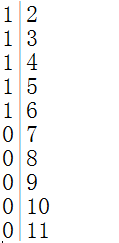
通过上述方法，支持向量机模型最终转化为一般的代数计算为题，将x的值带入公式计算f(x)的值从而判断x所属的位置。

下面的问题就转化为求方程系数的问题。即如何求得公式中w和b的大小从而确定公式。

* 1. 支持向量机的使用示例

数据集位置：D:\devs\data\spark\D07\u.txt

数据集如下：



代码位置：

D:\devs\studies\scala\HelloWorld\src\gtl\spark\java\example\C07\SVM.java

代码如下：

**package** gtl.spark.java.example.C07;  
  
**import** org.apache.spark.api.java.JavaRDD;  
**import** org.apache.spark.api.java.JavaSparkContext;  
**import** org.apache.spark.mllib.classification.SVMModel;  
**import** org.apache.spark.mllib.classification.SVMWithSGD;  
**import** org.apache.spark.mllib.linalg.Vectors;  
**import** org.apache.spark.mllib.regression.LabeledPoint;  
**import** org.apache.spark.sql.SparkSession;  
  
**public class** SVM {  
 **public static** String *DATA\_FILE* = **"D:\\devs\\data\\spark\\D07\\u.txt"**;  
  
 **public static void** main(String[] args){  
 SparkSession spark = SparkSession  
 .*builder*()  
 .master(**"local"**)  
 .appName(**"SVM"**)  
 .getOrCreate();  
 JavaSparkContext sc = JavaSparkContext.*fromSparkContext*(spark.sparkContext());  
  
 JavaRDD<String> data = sc.textFile(*DATA\_FILE*);*//获取数据集* JavaRDD<LabeledPoint> parsedData = data.map(r -> {  
 String t[] = r.split(**"\\|"**);*//“|”是转义字符,必须得加"\\"* LabeledPoint s = **new** LabeledPoint(  
 Double.*valueOf*(t[0]),  
 Vectors.*dense*( Double.*valueOf*(t[1])));  
 **return** s;  
 }).cache();*//转化数据格式* SVMModel model = SVMWithSGD.*train*(parsedData.rdd(),10);*//训练数据模型* System.***out***.println(model.weights());*//打印权重* System.***out***.println(model.intercept());*//打印截距* }  
}

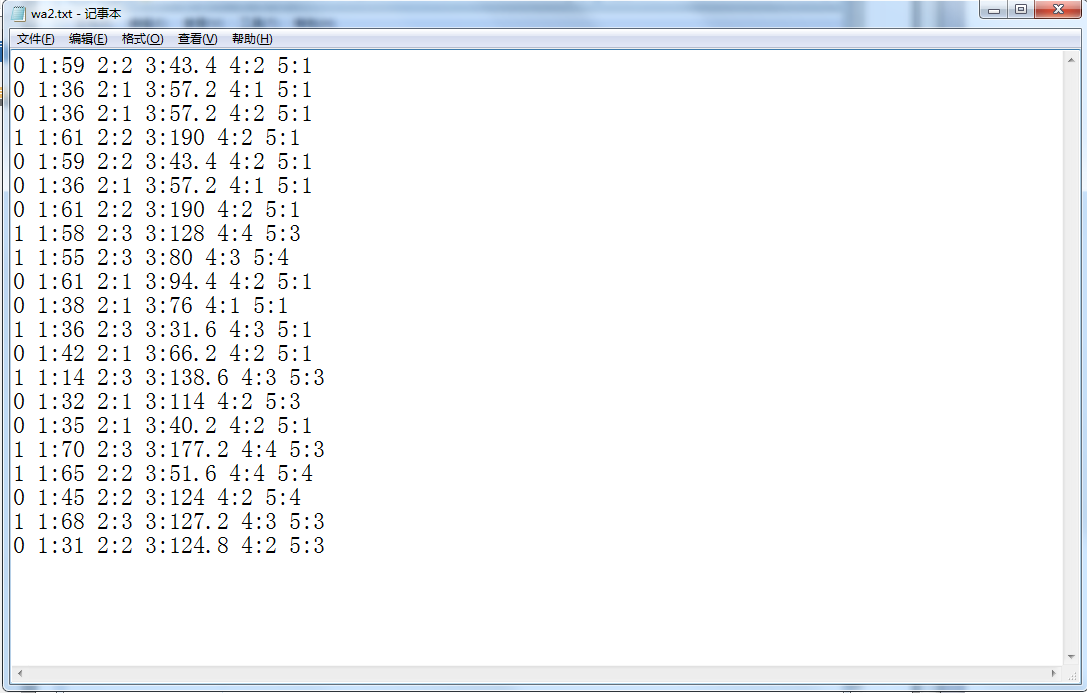
输出结果为：



* 1. 使用支持向量机

数据集位置：D:\devs\data\spark\D07\wa2.txt

数据集如下：



代码位置：

D:\devs\studies\scala\HelloWorld\src\gtl\spark\java\example\C07\SVMTest.java

代码如下：

**package** gtl.spark.java.example.C07;  
  
**import** org.apache.spark.api.java.JavaPairRDD;  
**import** org.apache.spark.api.java.JavaRDD;  
**import** org.apache.spark.api.java.JavaSparkContext;  
**import** org.apache.spark.mllib.classification.SVMModel;  
**import** org.apache.spark.mllib.classification.SVMWithSGD;  
**import** org.apache.spark.mllib.evaluation.MulticlassMetrics;  
**import** org.apache.spark.mllib.linalg.Vectors;  
**import** org.apache.spark.mllib.regression.LabeledPoint;  
**import** org.apache.spark.mllib.util.MLUtils;  
**import** org.apache.spark.sql.SparkSession;  
**import** scala.Tuple2;  
  
**public class** SVMTest {  
 **public static** String *DATA\_FILE* = **"D:\\devs\\data\\spark\\D07\\wa2.txt"**;  
  
 **public static void** main(String[] args) {  
 SparkSession spark = SparkSession  
 .*builder*()  
 .master(**"local"**)  
 .appName(**"SVMTest"**)  
 .getOrCreate();  
 JavaSparkContext sc = JavaSparkContext.*fromSparkContext*(spark.sparkContext());  
  
 JavaRDD<LabeledPoint> data = MLUtils.*loadLibSVMFile*(sc.sc(), *DATA\_FILE*).toJavaRDD();*//读取数据集* JavaRDD<LabeledPoint> splits[] = data.randomSplit(**new double**[]{0.7, 0.3}, 11L);*//对数据集切分* JavaRDD<LabeledPoint> parsedData = splits[0];*//分割训练数据* JavaRDD<LabeledPoint> parseTtest = splits[1];*//分割测试数据* SVMModel model = SVMWithSGD.*train*(parsedData.rdd(), 50);  
 JavaPairRDD<Object, Object> predictionAndLabels = parseTtest.mapToPair(r -> {  
 **return new** Tuple2<Object, Object>(model.predict(r.features()), r.label());  
 }); *//存储测试和预测值* MulticlassMetrics metrics = **new** MulticlassMetrics(predictionAndLabels.rdd());*//创建验证类* **double** accuracy = metrics.accuracy();*//计算验证值* System.***out***.println(**"accuracy = "** + accuracy);*//打印验证值* **double** patient = model.predict(Vectors.*dense*(70, 3, 180.0, 4, 3));*//计算患者可能性* **if** (patient == 1)  
 System.***out***.println(**"患者的胃癌有几率转移。"**); *//做出判断* **else** System.***out***.println(**"患者的胃癌没有几率转移。"**);*//做出判断* }  
}

输出结果为：



1. 朴素贝叶斯详解
   1. 朴素贝叶斯定律

朴素贝叶斯的数学表达可以如下定义：

V=(v1,v2,v3,...,vn)是一个待分项，而Vn为V的每个特征向量；

B=(b1,b2,b3,...,bn)是一个分类集合,bn为每个具体的分类；

如果需要测试某个Vn归属于B集合中的哪个具体分类，则需要计算P(bn|V)，即在V发生的条件下，归属于b1,b2,b3,...,bn中哪个可能性大。即：



因此，这个问题转变成每个待分项分配到集合中具体分类的概率是多少。而这个具体概率的求法可以使用贝叶斯定律：



而分子条件概率的求法可由如下公式求得：

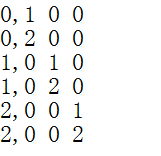


此为朴素贝叶斯计算公式。

* 1. MLlib朴素贝叶斯使用示例

数据集位置：D:\devs\data\spark\D07\bayes.txt

数据集如下：



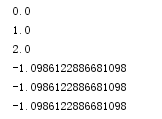
代码位置：

D:\devs\studies\scala\HelloWorld\src\gtl\spark\java\example\C07\Bayes.java

代码如下：

**package** gtl.spark.java.example.C07;  
  
**import** org.apache.spark.api.java.JavaRDD;  
**import** org.apache.spark.api.java.JavaSparkContext;  
**import** org.apache.spark.mllib.classification.NaiveBayes;  
**import** org.apache.spark.mllib.classification.NaiveBayesModel;  
**import** org.apache.spark.mllib.linalg.Vectors;  
**import** org.apache.spark.sql.SparkSession;  
**import** org.apache.spark.mllib.util.MLUtils;  
**import** org.apache.spark.mllib.regression.LabeledPoint;  
  
**import** java.util.Arrays;  
  
**public class** Bayes {  
 **public static** String *DATA\_FILE* = **"D:\\devs\\data\\spark\\D07\\Bayes.txt"**;  
  
 **public static void** main(String[] args) {  
 SparkSession spark = SparkSession  
 .*builder*()  
 .master(**"local"**)  
 .appName(**"Bayes"**)  
 .getOrCreate();  
 JavaSparkContext sc = JavaSparkContext.*fromSparkContext*(spark.sparkContext());  
 JavaRDD<String> datai = sc.textFile(*DATA\_FILE*);*//读取数据集* JavaRDD<LabeledPoint> data = datai.map(r -> {  
 String t[] = r.split(**","**);  
 String u[] = t[1].split(**" "**);  
 LabeledPoint s = **new** LabeledPoint(  
 Double.*valueOf*(t[0]),  
 Vectors.*dense*( Double.*valueOf*(u[0]), Double.*valueOf*(u[1]), Double.*valueOf*(u[2])));  
 **return** s;  
 });*//转化数据格式* NaiveBayesModel model = NaiveBayes.*train*(data.rdd(), 1.0);*//训练贝叶斯模型  
 //打印label值* **for**(**double** a: model.labels()){  
 System.***out***.println(a);  
 }  
 *//打印先验概率* **for**(**double** a: model.pi()){  
 System.***out***.println(a);  
 }  
  
*// double result = model.predict(Vectors.dense(0,0,10));//test* }  
}

输出结果为：



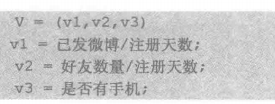
* 1. MLlib朴素贝叶斯实战：“僵尸粉”的鉴定

僵尸粉一般是指微博上的虚假分粉丝，花钱就可以买到“关注”，有实无名的微博粉丝。

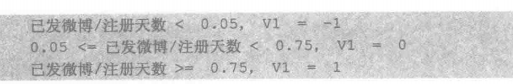
首先第一步需要对正常用户和虚假用户标记不同的标签，本例中，正常用户标记为1，而虚假的用户被标记为0.

其次是计算向量的设置。这里需要选着根据微博使用的特征所转化的向量。由于微博用户的使用有一定的规律，例如会经常发微博记录或者有大量感兴趣的其他关注，因此可以使用发帖数与注册时间的比值作为第一个参考向量，同时使用关注用户的数量与注册天数的比值作为第二个参考向量。同时对于正常的微博用户，还应注册手机作为接受信息使用，因此可以使用是否填写手机作为用户向量的一个判断标准。

即可得以下向量：



由于V1和V2都是一系列的连续数值构成，我们可以对其进行人为的划分，即：



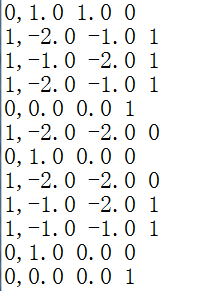
因此可以将v1的向量表示为(-1,0,1)，而v2的向量也可以如此表示。对于v3用-1表示为不使用手机而1表示使用手机。

对于数据集的获取，这里采用人工判定的2万个数据作为数据集，经过归并计算。

由于数据集出现负数，对于特征值进行了加2处理。

数据集位置：D:\devs\data\spark\D07\data.txt

数据集如下：



代码位置：

D:\devs\studies\scala\HelloWorld\src\gtl\spark\java\example\C07\LogisticRegression2.java

代码如下：

**package** gtl.spark.java.example.C07;  
  
**import** org.apache.spark.api.java.JavaPairRDD;  
**import** org.apache.spark.api.java.JavaRDD;  
**import** org.apache.spark.api.java.JavaSparkContext;  
**import** org.apache.spark.mllib.classification.NaiveBayes;  
**import** org.apache.spark.mllib.classification.NaiveBayesModel;  
**import** org.apache.spark.mllib.linalg.Vectors;  
**import** org.apache.spark.mllib.regression.LabeledPoint;  
**import** org.apache.spark.mllib.util.MLUtils;  
**import** org.apache.spark.sql.SparkSession;  
**import** scala.Tuple2;  
  
**public class** BayesTest {  
 **public static** String *DATA\_FILE* = **"D:\\devs\\data\\spark\\D07\\data.txt"**;  
  
 **public static void** main(String[] args) {  
 SparkSession spark = SparkSession  
 .*builder*()  
 .master(**"local"**)  
 .appName(**"BayesTest"**)  
 .getOrCreate();  
 JavaSparkContext sc = JavaSparkContext.*fromSparkContext*(spark.sparkContext());  
  
 JavaRDD<String> datai = sc.textFile(*DATA\_FILE*);*//读取数据集* JavaRDD<LabeledPoint> data = datai.map(r -> {  
 String t[] = r.split(**","**);  
 String u[] = t[1].split(**" "**);  
 LabeledPoint s = **new** LabeledPoint(  
 Double.*valueOf*(t[0]),  
 Vectors.dense(Double.*valueOf*(u[0]) + 2, Double.*valueOf*(u[1]) + 2, Double.*valueOf*(u[2]) + 2));  
 **return** s;  
 });*//转化数据格式* JavaRDD<LabeledPoint> splits[] = data.randomSplit(**new double**[]{0.7, 0.3}, 11L);*//对数据集切分* JavaRDD<LabeledPoint> trainingData = splits[0];*//分割训练数据* JavaRDD<LabeledPoint> testData = splits[1];*//分割测试数据* NaiveBayesModel model = NaiveBayes.*train*(trainingData.rdd(), 1.0);*//训练贝叶斯模型* JavaPairRDD<Object, Object> predictionAndLabels = testData.mapToPair(r -> {  
 **return new** Tuple2<Object, Object>(model.predict(r.features()), r.label());  
 }); *//存储测试和预测值* **double** accuracy = 1.0 \* predictionAndLabels.filter(r -> r.\_1().equals(r.\_2())).count() / testData.count();  
 System.***out***.println(**"Accuracy = "** + accuracy);*//打印准确度* }  
}

输出结果为：

