尚硅谷电商用户画像七

版本：V 1.0

张晨

# 第十章 挖掘类标签

## 10.1 挖掘类标签与机器学习

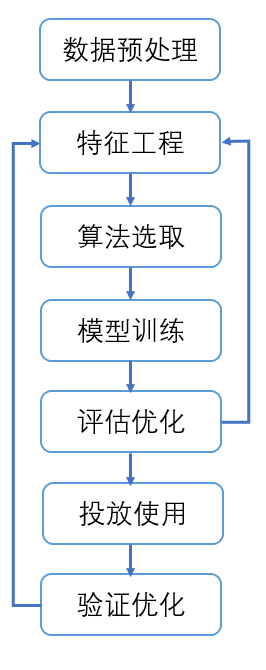
挖掘类标签需要应该用算法挖掘用户的相关特征，比如：性别预测、年龄预测、 用户流失预测、风险欺诈预测。

相比统计、规则类这些通过专业人员制定明确规则的标签，挖掘类的标签完全是另一套处理思路。

获得挖掘标签过程：



## 10.2 模型建立过程

整个挖掘的过程的核心就是建立、完善模型的过程。

一个模型完善的过程是个没有尽头的迭代。

1. **数据预处理**

主要是对数据的初步的清洗加工，这个过程一般可以在数仓中完成。

1. **特征工程**

主要是特征的选择和提取。比如想预测用户的流失，那就要**选择**哪些**指标字段**会和用户的流失有比较强的相关性。要从数仓中，把这些指标提取出来并进一步加工。

除了获得特征，还需要“**参考答案**”，比如抽选出来的这些用户特征，那这些用户到底是不是流失的，要标记出来，用于机器学习。

特征的选取往往不能一蹴而就，需要反复的迭代尝试。

1. **算法选取**

目前机器学习的算法种类繁多，比如**分类算法**领域中：决策树、随机森林、逻辑回归、GBDT、XGBoost。

**回归算法**领域中：线性回归、多项式回归、岭回归、Lasso回归、弹性回归。

在画像领域中，主要使用分类算法。但具体使用哪种分类算法，也是需要不断尝试验证的，没有一定的标准。

1. **模型训练**

通过代码实现“数据+算法=模型”。

1. **评估优化**

一般会把数据进行分组，训练组和验证组，然后对模型组进行准率的评估。

根据准确率，对模型进行优化：

优化一般主要是三个方面：

1. 特征选取和提炼
2. 算法的比较和选择
3. 算法的参数调整
4. **投放使用、验证优化**

把模型投放到实际的标签生产中去观察，比如预测流失的用户，一段时间是否真的会流失。

或者进行**A/B测试**，对预测的一部分用户采取某种措施，另一部分用户不作处理。观察两组人的变化效果。

通过实际生产中的预测效果，不断的反复调整模型、算法。

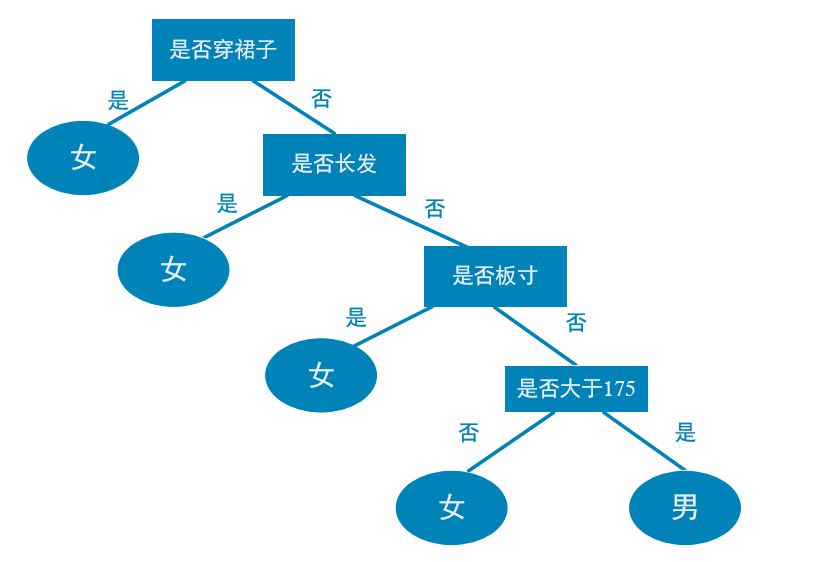
## 10.3 决策树算法介绍

### 10.3.1 什么是决策树

比如：判断学生男女

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | 特征(feature) | | | | 标签(label) |
| 学生ID | 发型 | 身高 | 是否穿裙子 | 年龄 | 性别 |
| 1 | 长发 | 159 | 否 | 80后 | 女 |
| 2 | 短发 | 168 | 否 | 00后 | 男 |
| 3 | 板寸 | 172 | 否 | 80后 | 男 |
| 4 | 短发 | 180 | 否 | 90后 | 男 |
| 5 | 短发 | 169 | 是 | 00后 | 女 |
| 6 | 短发 | 155 | 否 | 90后 | 女 |

### 10.3.2 构建决策树



### 10.3.3 特征的分类

1 **分类特征（离散特征）**

**比如：**发型、星座、地区、偏好品类、设备型号。

**2 二元特征**

也算是分类特征的一种。**比如**：性别、**是否**穿裙子，**是否**是新用户，**是否**即将流失。

**3连续值特征 （数值特征）**

**比如： 身高cm、收入、最近消费金额、浏览次数、最后一次登录时间。**

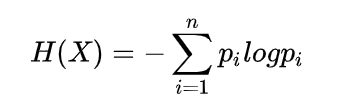
## 10.4 特征的度量

决策树的搭建的核心就是找到合适特征进行划分。那么什么样的特征更适合划分呢？

### 10.4.1 信息熵(entropy)

描述了信息不确定性的程度。

**香农公式：**

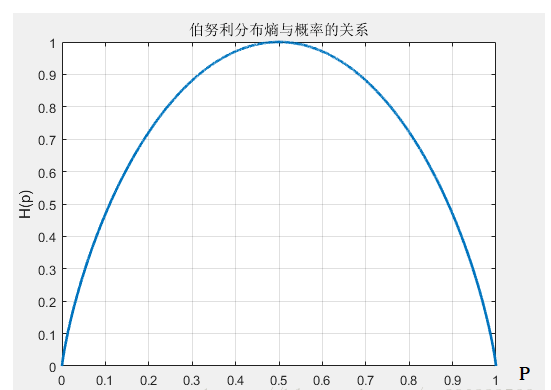


H(X)表示X的熵值。

而熵值其实和X本身没有关系，只和p有关系，p表示X在整个数列中分布的概率。

当p=0 或 1时，H(X)=0 , 这种情况下完全没有不确定。

当p=0.5时，H(X) 最大，即熵最大，也就是不确定性最大。



### 10.4.2 通过熵来度量特征的算法

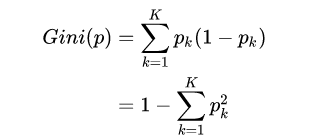
**信息增益**

通过特征A使得熵值减小的程度。

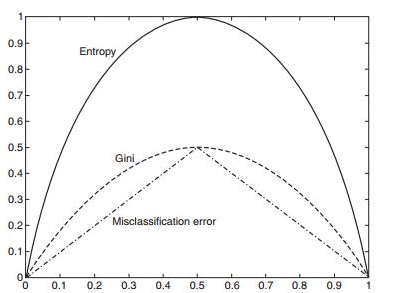
**信息增益率**

信息增益算法有些弊端，就是对取值数量较多的属性有所偏好，比如唯一性编号，虽然熵减效果特别好，但是其实不具备泛化能力，所以引入分裂信息来惩罚取值较多的特征。

### 10.4.3 基尼系数(Gini)

基尼系数也称为基尼不纯度，基尼系数与信息增益率内在含义非常接近。表示样本集合中一个随机选中的样本被分错的概率。基尼指数越小表示集合中被选中的样本被分错的概率越小，也就说集合的纯度越高，反之，集合越不纯。  
 

其实基尼指数,熵的曲线非常接近。



熵的最大值为1，基尼系数的最大值为0.5。

### 10.4.4 基尼系数计算：

以 是否穿裙子为例计算：

计算【是否穿裙子】的基尼系数，来判断该特征是否是好特征。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 是否穿裙子 | 个数 | label个数 | Gini计算 | Gini指数() |
| 是 | 1 | 男 0  女 1 | Gini= 1-[ (**0/1**)^2+ (**1/1**)^2]  =**0** | 加权平方  (1/6)\*0 +( 5/6)\* (12/25)  =**0.4** |
| 否 | 5 | 男 3  女 2 | Gini= 1-[ (**3/5**)^2+ (**2/5**)^2]  =**12/25** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 是否低于170 | 个数 | label个数 | Gini计算 | Gini指数() |
| 是 | 4 | 男 1  女 3 | Gini= 1-[ (**1/4**)^2+ (**3/4**)^2]  =**6/16** | 加权平方  (4/6)\*6/16 +( 2/6)\* 0  =**0.25** |
| 否 | 2 | 男 2  女 0 | Gini= 1-[ (**2/2**)^2+ (**0/2**)^2]  =**0** |

**橙色**部分为P值（出现概率），在0~1之间。

**紫色**部分为Gini系数，在0~0.5之间。

**因为0.4的Gini系数这个更接近0.5，其实不是一个好的特征。决策树会选择Gini系数小的特征放在更上面的位置。**

**连续值的Gini系数如何判断？**

决策树会使用**贪心算法**的策略，即会把连续值均分分成N个段，会假设在每个断点进行切分，逐个断点去计算Gini系数。取Gini系数小的断点，作为决策数据的分支判断节点。

## 10.5 用SparkMLlib实现决策树的使用

### 10.5.1 利用流水线完成模型训练

训练+预测的完成过程

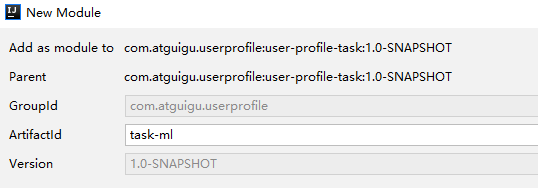


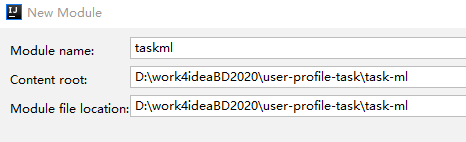
### 10.5.2 数据脚本

|  |
| --- |
| create table student  ( uid bigint ,  hair string,  height bigint ,  skirt string,  age string ,  gender string  ) |

模拟数据，参见画像物料。

### 10.5.3 新建模块task-ml





在pom.xml中加入

|  |
| --- |
| <**dependencies**>  <**dependency**>  <**groupId**>com.atguigu.userprofile</**groupId**>  <**artifactId**>task-common</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>  </**dependency**>   <**dependency**>  <**groupId**>org.apache.spark</**groupId**>  <**artifactId**>spark-mllib\_2.12</**artifactId**>  <**scope**>provided</**scope**>  <**exclusions**>  <**exclusion**>  <**groupId**>net.jpountz.lz4</**groupId**>  <**artifactId**>lz4</**artifactId**>  </**exclusion**>  </**exclusions**>   </**dependency**> </**dependencies**>   <**build**>  <**plugins**>  *<!-- 该插件用于将Scala代码编译成class文件 -->* <**plugin**>  <**groupId**>net.alchim31.maven</**groupId**>  <**artifactId**>scala-maven-plugin</**artifactId**>  <**version**>3.4.6</**version**>  <**executions**>  <**execution**>  *<!-- 声明绑定到maven的compile阶段 -->* <**goals**>  <**goal**>compile</**goal**>  <**goal**>testCompile</**goal**>  </**goals**>  </**execution**>  </**executions**>  </**plugin**>   <**plugin**>  <**groupId**>org.apache.maven.plugins</**groupId**>  <**artifactId**>maven-assembly-plugin</**artifactId**>  <**version**>3.0.0</**version**>  <**configuration**>  <**descriptorRefs**>  <**descriptorRef**>jar-with-dependencies</**descriptorRef**>  </**descriptorRefs**>  </**configuration**>  <**executions**>  <**execution**>  <**id**>make-assembly</**id**>  <**phase**>package</**phase**>  <**goals**>  <**goal**>single</**goal**>  </**goals**>  </**execution**>  </**executions**>  </**plugin**>  </**plugins**>  </**build**> |

### 10.5.4 创建流水线对象

|  |
| --- |
| class MyPipeline {   // 流水线对象  private var pipeline:Pipeline=null  // 流水线训练后模型  private var pipelineModel:PipelineModel=null    //// 以下为参数 ////////////////////  //最大分类树（用于识别连续值特征和分类特征）  private var maxCategories=5  // 最大分支数  private var maxBins=5  // 最大树深度  private var maxDepth=5  //最小分支包含数据条数  private var minInstancesPerNode=1  //最小分支信息增益  private var minInfoGain=0.0      var labelColumnName:String=null  var featureColumnNames:Array[String]=null  def setMaxCategories(maxCategories:Int): MyPipeline ={  this.maxCategories=maxCategories  this } def setMaxBins(maxBins:Int): MyPipeline ={  this.maxBins=maxBins  this } def setMaxDepth(maxDepth:Int): MyPipeline ={  this.maxDepth=maxDepth  this }  def setMinInstancesPerNode(minInstancesPerNode:Int): MyPipeline ={  this.minInstancesPerNode=minInstancesPerNode  this }  def setMinInfoGain(minInfoGain:Double): MyPipeline ={  this.minInfoGain=minInfoGain  this }  def setLabelColumnName(labelColumnName:String): MyPipeline ={  this.labelColumnName=labelColumnName  this } def setFeatureColumnNames(featureColumnNames:Array[String]): MyPipeline ={  this.featureColumnNames=featureColumnNames  this } |

### 10.5.5 增加4大流水线组件

#### 创建标签索引

|  |
| --- |
| ***//1创建标签索引***  **var** *labelColumnName*:String=**null**  **def** setLabelColumnName(labelColumnName:String): MyPipeline ={  **this**.*labelColumnName*=labelColumnName  **this** }    **def** createLabelIndexer(labelColumn:String): StringIndexer ={  **val** stringIndexer: StringIndexer = **new** StringIndexer()  .setInputCol(**labelColumn**).setOutputCol(**"label\_index"**)  stringIndexer  } |

**标签索引**的作用，把“参考答案”列做标识，同时把字符串值，按照出现概率转为矢量值。比如：男多女少的数据， 男转为 0 , 女 转为1

InputCols是输入列名，需要字符串。

OutputCol是输出列名，固定名称为**label\_index**。

#### 创建特征集合

|  |
| --- |
| ***// 2 创建特征集合***  **var** *featureColumnNames*:Array[String]=**null**  **def** setFeatureColumnNames(featureColumnNames:Array[String]): MyPipeline ={  **this**.*featureColumnNames*=featureColumnNames  **this** }  **def** createFeatureAssmble(featureColumn:Array[String]): VectorAssembler ={  **val** vectorAssembler = **new** VectorAssembler()  .setInputCols(**featureColumn**)  .setOutputCol(**"features\_assmble"**)  vectorAssembler  } |

**特征集合**的作用，标识哪些数据列是特征列，合并为一个“**features\_assmble**”

InputCols是输入列名，即上游数据源的特征列名，类型为字符串数组

OutputCol是输出列名。固定名称为**features\_assmble**。

#### 特征向量向量索引

|  |
| --- |
| ***//3 创建特征向量索引* def** createFeatureIndexer(): VectorIndexer ={  **val** vectorIndexer = **new** VectorIndexer()  .setInputCol(**"features\_assmble"**)  .setOutputCol(**"features\_index"**)  .setHandleInvalid(**"skip"**)  .setMaxCategories(*maxCategories*)  vectorIndexer  } |

**特征向量向量索引**的作用：

1. 把特征值区分为连续值特征和分类特征。区分依据是 *maxCategories* ，默认为5 ，特征出现不同值的个数超过这个值的视为连续值特征，否则为分类特征。
2. 会把所有分类特征，按照出现次数，转换为矢量，比如：短发数量>长发数量>板寸数量，则 短发数量→0，长发→1 ，板寸→2。
3. 参数说明

InputCol固定为**"features\_assmble"**   
 OutputCol 固定为**"features\_index"**

HandleInvalid ，设如果出现异常数据如何处理，一般选择skip即跳过该条问题数据。。

MaxCategories，默认值为5，小于等于该值的列为分类特征，其他为连续值特征。

#### 创建分类器

|  |
| --- |
| ***// 4 创建分类器* def** createClassifier(): DecisionTreeClassifier ={  **val** classifier = **new** DecisionTreeClassifier()  .setFeaturesCol(**"features\_index"**)  .setLabelCol(**"label\_index"**)  .setPredictionCol(**"prediction"**)  .setImpurity(**"gini"**)  .setMaxBins(*maxBins*)  .setMinInstancesPerNode(*minInstancesPerNode*)  .setMinInfoGain(*minInfoGain*)  .setMaxDepth(*maxDepth*)  classifier } |

作用： 设定核心的分类算法，和算法参数。

featuresCol:填入上游特征索引的名称**features\_index**。

labelCol: 填入上游标签索引的名称**label\_index**

PredictionCol： 填入未来预测后预测结果的列名：**prediction。**

Impurity： 可以选择**gini 或者 entropy**。

**剪枝参数：**

maxBins： **最大分支数** maxDepth：**最大树深度**   
 minInstancesPerNode：**最小分支包含数据条数** minInfoGain**：最小分支信息增益**

### 10.5.6 初始化对象

|  |
| --- |
| **def** init(): MyPipeline ={  **if**(*labelColumnName*==**null**) **throw new** RuntimeException(**"need set labelColumnName first! "**)  **if**(*featureColumnNames*==**null**) **throw new** RuntimeException(**"need set featureColumnNames first! "**)  *pipeline*= **new** Pipeline().setStages(*Array*(  createLabelIndexer(*labelColumnName*),  createFeatureAssmble(*featureColumnNames*),  createFeatureIndexer(),  createClassifier()  )  )   **this** } |

pipeline对象把搭建过程分为了4步，把***标签索引，特征集合，特征索引，分类器***，放在Stage中。

### 10.5.7 训练

|  |
| --- |
| ***// 训练* def** train(dataFrame:DataFrame): Unit ={  *pipelineModel* = *pipeline*.fit(dataFrame)  } |

把数据以DataFrame的结构传递进来，进行训练，可以获得流水线模型。

### 10.5.8 查看模型的决策树

|  |
| --- |
| ***//训练后 获得分类树* def** getModelTree():String={  **if**(*pipelineModel*==**null**) **throw new** RuntimeException(**"need training first! "**)  **val** classificationModel: DecisionTreeClassificationModel =   *pipelineModel*.stages(3).asInstanceOf[DecisionTreeClassificationModel]  classificationModel.toDebugString  } |

### 10.5.9 获得各个特征权重

|  |
| --- |
| ***//训练后获得特征权重* def** getFeatureImportance():String={  **if**(*pipelineModel*==**null**) **throw new** RuntimeException(**"need training first! "**)  **val** classificationModel: DecisionTreeClassificationModel  = *pipelineModel*.stages(3).asInstanceOf[DecisionTreeClassificationModel]  classificationModel.*featureImportances*.toString } |

从模型中的第三个阶段模型，获得决策树模型，提取转换后获得***featureImportances***

### 10.5.10 预测

|  |
| --- |
| ***//预测* def** predict(dataFrame: DataFrame): DataFrame ={  **if**(*pipelineModel*==**null**) **throw new** RuntimeException(**"need training first! "**)  *pipelineModel*.transform(dataFrame) } |

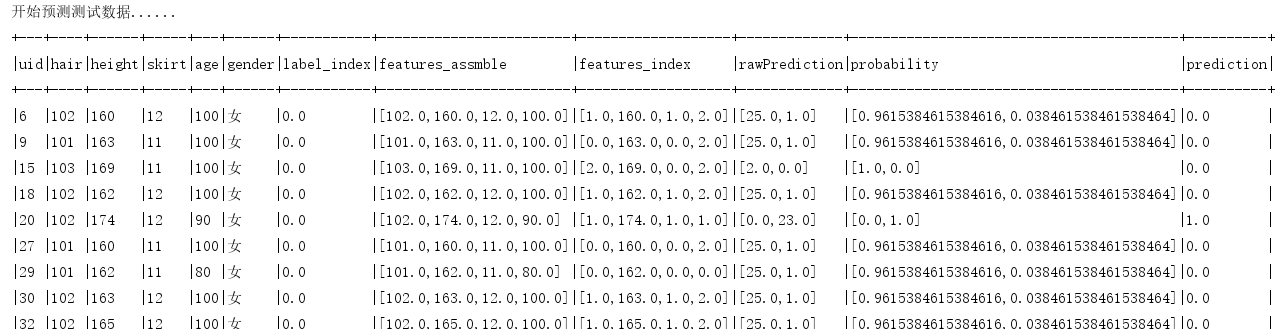
利用已有模型对新数据进行预测。

### 10.5.11 实现训练主程序

|  |
| --- |
| **def** main(args: Array[String]): Unit = {   **val** sparkConf: SparkConf = **new** SparkConf().setAppName(**"student\_gender\_app"**).setMaster(**"local[\*]"**)  **val** sparkSession: SparkSession = SparkSession.*builder*().config(sparkConf).enableHiveSupport().getOrCreate()   *println*(**"开始查询数据....."**)  **val** sql=  **"""  | select uid,  | case hair when '长发' then 101  | when '短发' then 102  | when '板寸' then 103 end as hair,  | height,  | case skirt when '是' then 11  | when '否' then 12 end as skirt,  | case age when '80后' then 80  | when '90后' then 90  | when '00后' then 100 end as age ,  | gender  | from user\_profile2077.student  """**.stripMargin  **val** dataFrame: DataFrame = sparkSession.sql(sql)  dataFrame.show(100)  *println*(**"切分数据....."**)  **val** *Array*(trainDF,testDF): Array[Dataset[Row]] = dataFrame.randomSplit(*Array*(0.7,0.3))   *println*(**"开始初始化流水线....."**)  **val** myPipeline: MyPipeline = **new** MyPipeline()  .setLabelColumnName(**"gender"**)  .setFeatureColumnNames(*Array*(**"hair"**,**"height"**,**"skirt"**,**"age"**))  .setMaxCategories(5)  .setMaxDepth(6)  .setMaxBins(3)  .init()   *println*(**"开始初始化训练......"**)  myPipeline.train(trainDF)  *println*(**"获得模型......"**)  *println*(myPipeline.getModelTree())  *println*(**"获得特征权重......"**)  *println*(myPipeline.getFeatureImportance())   *println*(**"开始预测测试数据......"**)  **val** predictDF: DataFrame = myPipeline.predict(testDF)  predictDF.show(100,**false**)  } |

### 10.5.12 解读预测数据

#### 预测



label\_index： 把原有的值，转换为0,1,2…的映射值，出现次数越小label值越小。

features\_assmble：成为特征的原值。

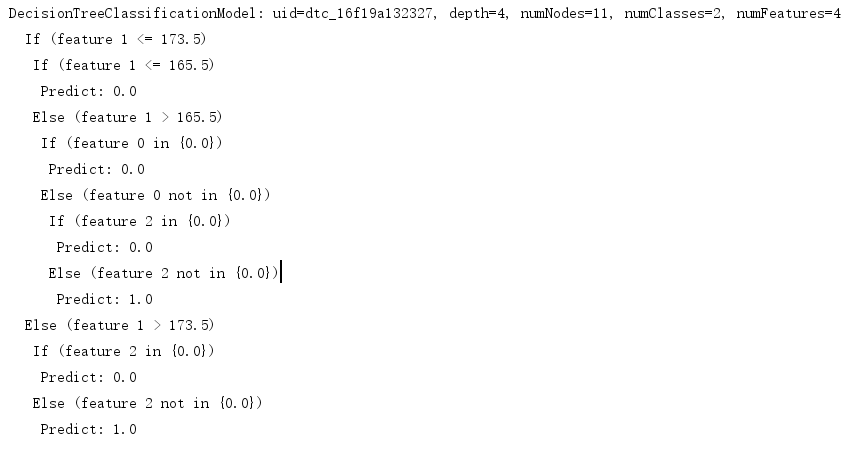
features\_index：转化为向量的特征值。

rawPrediction：根据决策树预测的可信权重，越大可信度越高。

probability： 把可信权重转换为概率。

prediction：选择概率大的一方作为权重值。

#### 决策数据



feature 0 /1/ 2/3 分别是原特征的1,2,3,4列

#### 查看特征权重



### 10.5.13转换预测列

由于预测列作为结果，分类器计算出来的还是矢量值。不方便应用。最好能够把矢量列还原成原值。

在流水线中加入转换方法。

|  |
| --- |
| ***//转换预测结果矢量* def** convertLabel(dataFrame:DataFrame): DataFrame ={  **if**(*pipelineModel*==**null**) **throw new** RuntimeException(**"need training first! "**)  **val** indexerModel: StringIndexerModel =  *pipelineModel*.stages(0).asInstanceOf[StringIndexerModel]  **val** converter = **new** IndexToString()  .setInputCol(**"prediction"**)  .setOutputCol(**"prediction\_origin"**)  .setLabels(indexerModel.labels)  **val** convertedDF: DataFrame = converter.transform(dataFrame)  convertedDF } |

训练程序中最后加入

|  |
| --- |
| *println*(**"把预测列转换为原始数据......"**) **val** convertedData: DataFrame = myPipeline.convertLabel(predictDF) convertedData.show(100) |

### 10.5.14 评估方法

目前的评估准确率主要是三个指标：

准确率（*accuracy*）、精确率(*precision*) 、召回率(*recall*)。

#### 准确率：

公式： 准确率= 所有判断正确/ 所有数据

准确率很容易理解就是就是判断对的除以总数。但是有些场合不适合使用准确率。

尤其是二元结果比例悬殊的情况： 比如判断用户是否是黄牛党，黄牛党的比例在总用户的比例也就是0.2%，如果用准确率评估模型的话，那还不如直接全猜用户不是黄牛党。这样就都有99.8%的准确率。

所以针对严重倾斜的二元结果可以采取精确率和召回率来衡量。 这两个的特点是着重于对二元结果中的某一方（一般是少数派）进行评估。

#### 精确率和召回率

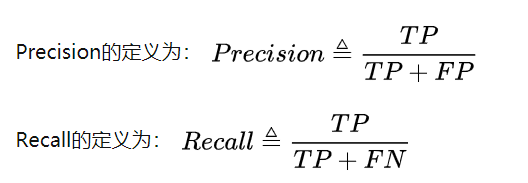
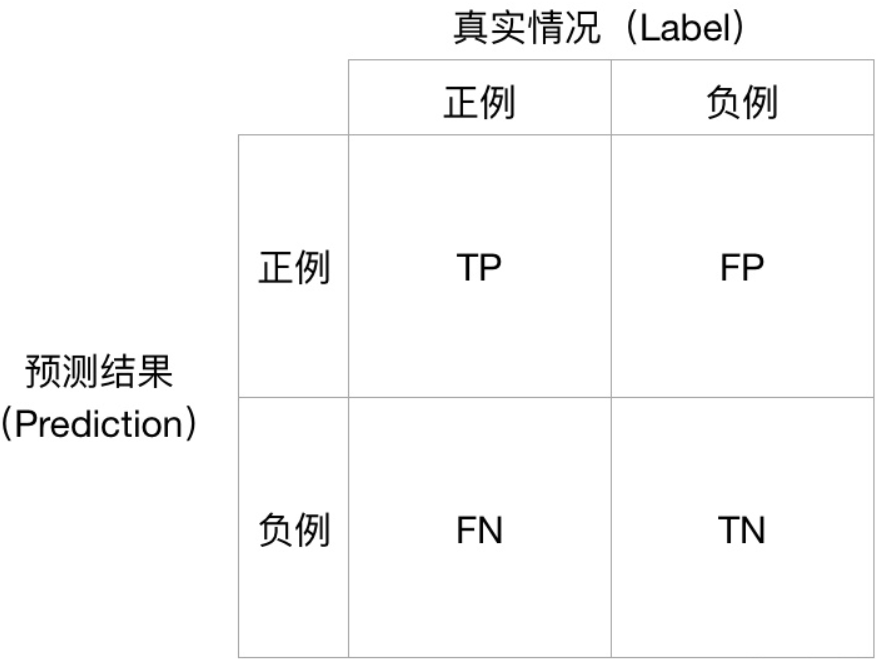
**四个概念：TP，FP，TN，FN**

TP（True Positive）：在判定为positive的样本中，判断正确的数目。

FP（False Positive）：在判定为positive的样本中，判断错误的数目。

TN（True Negative）：在判定为negative的样本中，判断正确的数目。

FN（False Negative）：在判定为negative的样本中，判断错误的数目。

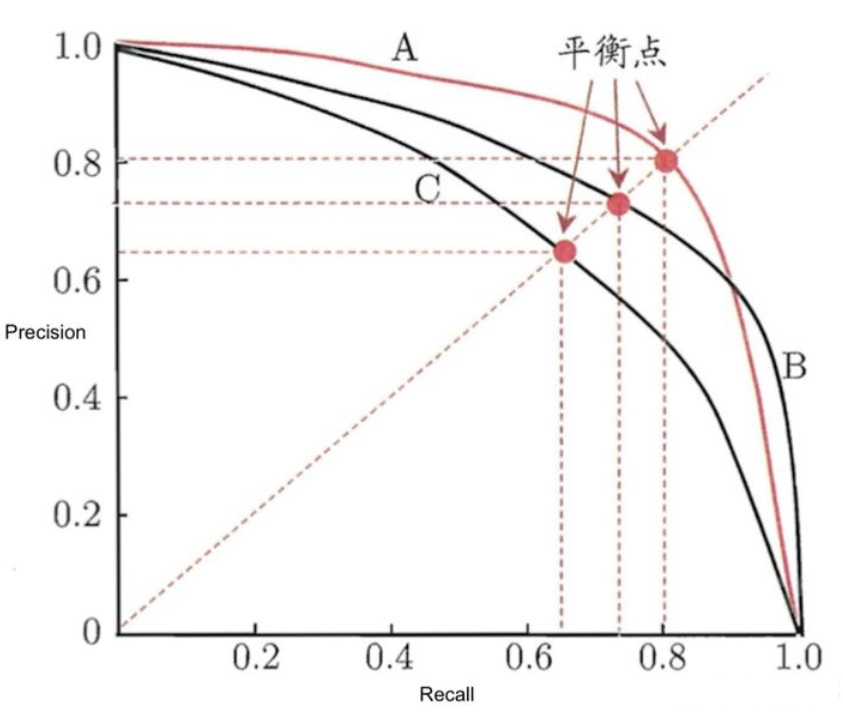


**Precision**从预测结果角度出发，描述了预测出来的正例结果中有多少是真实正例，即该预测的正例有多少是准确的

**Recall**从真实结果角度出发，描述了测试集中的真实正例有多少被挑选了出来，即真实的正例有多少被召回。

**技巧**：在实际工作中调节模型的过程，往往会造成这两个评估值形成**“跷跷板”**。就是一边大，另外一边就小。**Precision提高就要减少“错杀”，Recall提高就要减少“错放”。激进和保守本就矛盾。**实际情况要侧重于哪边要**看实际业务**的情况。

但如果只看数值上的综合收益，那二者相乘得到的最大值，是**理论**上的综合性能平衡点。



#### 代码实现

|  |
| --- |
| ***//评估* def** evaluate(dataFrame: DataFrame) ={  ***//把列调整为预测列和标签列的元组* val** predictionAndLabel: RDD[(Double, Double)] =  dataFrame.*rdd*.map { row =>  (row.getAs[Double](**"prediction"**)  , row.getAs[Double](**"label\_index"**))  }  ***//放入评估矩阵* val** metrics = **new** MulticlassMetrics(predictionAndLabel )  *println*(**"准确率："**+metrics.*accuracy*)  metrics.*labels*.foreach{label=>  *println*(**" 精确度（tp/(tp+fp)）："**+label+**" ->"**+metrics.precision(label) )  *println*(**" 召回率（tp/(tp+fn)）："**+label+**" ->"**+metrics.recall(label) )  }  ***//统计明细*** dataFrame.createTempView(**"predict\_table"**)  dataFrame.*sparkSession*.sql(  **""" select tp,fn, fp,tn,tt,ct,  | tp/(tp+fp) pRate,  | tp/(tp+fn) rRate,  | tt/ct aRate from (  | select sum(if(label\_index=1.0 and prediction=1.0 ,1,0)) tp ,  | sum(if(label\_index=1.0 and prediction=0.0 ,1,0)) fn ,  | sum(if(label\_index=0.0 and prediction=1.0 ,1,0)) fp ,  | sum(if(label\_index=0.0 and prediction=0.0 ,1,0)) tn ,  | sum(if(label\_index=prediction ,1,0)) tt ,  | count(\*) ct  | from predict\_table  | ) pt  """**.stripMargin ).show(100,**false**)  } |

主训练类中

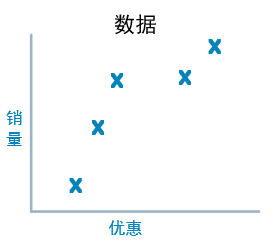
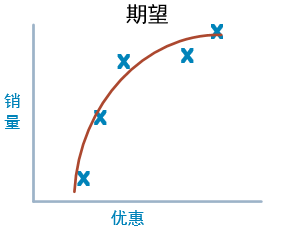
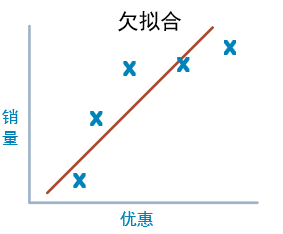
|  |
| --- |
| *println*(**"分析测试结果......"**)  myPipeline.evaluate(predictDF) |

### 10.5.15 拟合与剪枝

模型的生成是一个不断迭代的过程。一颗**差强人意**的决策树往往处于两个状态：**欠拟合**和过拟合。

#### 欠拟合

**欠拟合**就是模型没有很好地捕捉到数据特征，不能够很好地拟合数据。如下

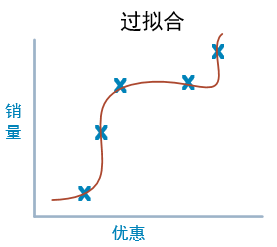
 

**解决办法**： 1 增加、调整特征项。

2 放宽用来剪枝的一些参数。比如提高树的深度。

#### 过拟合

**过拟合**就是模型把数据学习的太彻底，以至于把噪声数据的特征也学习到了。如下：



如果不对决策树设置任何限制，它可以生成一颗非常庞大的树，决策树的树叶节点所覆盖的训练样本都是“纯”的。这样决策树在训练样本上非常精准，但是在测试集上就没那么好了。  
 层数越多，叶结点越多，分的越细致，对训练数据分的也越深，越容易过拟合，导致对测试数据预测时反而效果差。要解决这个问题就需要对决策树进行**剪枝。**

#### ****剪枝****

在sparkMLlib中可以通过设置一些参数来达到剪枝的效果：

sparkMLlib目前只支持**预剪枝**，主要是通过设置参数完成。

剪枝参数（超参数）：

**maxDepth**：默认为5 ，最大树深度，树越深，记住的细节就越多。往往容易过拟合。

**maxBins**：默认32，针对某一个**连续值**特征最小可以产生的分支。用于避免因为一个连续值而产生的多个分支。同时还能降低计算消耗。该值最低不能低于2。

**minInstancesPerNode**：默认为1， 最小分支包含数据条数，过小容易产生很多细节分支。 **minInfoGain：**默认为0，最小分支信息增益，过小会增加无谓的分支。取值范围基尼系数(0-0.5), 熵(0-1.0)。

# 第十一章 任务六： 电商用户画像挖掘标签—性别预测

## 11．1 主要任务

任务目标根据用户相关数据，利用机器学习对用户性别进行预测。

## 11.2 实现步骤分析

1) 分析选取特征属性和标签(label)值

**训练阶段**

2) 通过SQL查询出特征和标签值。

3) 将特征和标签数据送入自定义流水线对象，进行模型训练。

4) 对模型进行评估

5) 将模型保存到HDFS

**预测阶段**

从HDFS中读取模型，

利用模型对数据进行预测

把矢量结果转换为标签的原值

**写入数据阶段**

在平台定义标签信息

读取标签信息创建标签表

将预测的结果写入标签表

## 11.3 选择反映用户性别的特征与真实标签(label)

### 11.3.1 特征选取

|  |  |  |
| --- | --- | --- |
| 特征 | 特征类型 | 来源表 |
| 显著男性类商品的访问时长 | 线性值特征 | dwd\_page\_log |
| 显著女性类商品的访问时长 | 线性值特征 | dwd\_page\_log |
| 最常访问的一级商品类目 | 分类特征 | dwd\_page\_log |
| 第二常访问的一级商品类目 | 分类特征 | dwd\_page\_log |
| 第三常访问的一级商品类目 | 分类特征 | dwd\_page\_log |

问题1： 为什么要选择三个排名的一级目录？

问题2： 显著商品访问为什么要取时长而不是次数？

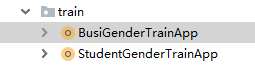
问题3: 显著男性类商品和女性商品类别由谁确定？

### 11.3.2 标签列(label)从何而来

标签列即认为是用户真实性别的“参考答案”。从dim\_user\_info表中，填写了用户信息的数据提取。

## 11.4 训练阶段开发

### 11.4.1 创建训练类



### 11.4.2 训练类代码

|  |
| --- |
| **def** main(args: Array[String]): Unit = {   **val** sparkConf: SparkConf =  **new** SparkConf().setAppName(**"busi\_gender\_train"**).setMaster(**"local[\*]"**)  **val** sparkSession: SparkSession =  SparkSession.*builder*().config(sparkConf).enableHiveSupport().getOrCreate()  ***//通过SQL 把特征及参考答案 一起取*** *println*(**"通过SQL 把特征及参考答案 一起取"**)  **val** querySql =  **s"""  | with user\_cate as  | (  | select user\_id ,category1\_id c1 ,during\_time from  | dwd\_page\_log pl join dim\_sku\_info ski  | on pl.page\_item=ski.id  | where ski.dt='2021-05-16' and pl.dt='2021-05-16'  | and page\_id='good\_detail' and page\_item\_type='sku\_id'  | ) ,  | user\_gender as  | (  | select id ,gender from dim\_user\_info  | where dt='9999-99-99' and gender<>''  | )  | select user\_id as uid , top1\_c1,top2\_c1,top3\_c1 ,  | male\_c1\_dur,female\_c1\_dur ,ug.gender  | from  | (  | select user\_id ,  | sum(if(ct\_rank=1,c1,0)) top1\_c1,  | sum(if(ct\_rank=2,c1,0)) top2\_c1,  | sum(if(ct\_rank=3,c1,0)) top3\_c1,  | sum(sum\_dur),  | sum( if(c1 in (3,4,16),sum\_dur,0 )) male\_c1\_dur ,  | sum( if(c1 in (8,12,15),sum\_dur,0 )) female\_c1\_dur  | from  | (  | select user\_id ,c1 ,sum(during\_time) sum\_dur,count(\*),  | row\_number()over(partition by user\_id order by count(\*) desc ) ct\_rank  | from user\_cate  | group by user\_id ,c1  | order by user\_id  | ) user\_c1  | group by user\_id  | )user\_ct join user\_gender ug on ug.id=user\_ct.user\_id  """**.stripMargin;  sparkSession.sql(**"use gmall2021"**)  **val** dataFrame: DataFrame = sparkSession.sql(querySql)  ***//把数据拆分训练集和测试集*** *println*(**"把数据拆分训练集和测试集"**)  **val** *Array*(trainDataFrame,testDataFrame) = dataFrame.randomSplit(*Array*(0.8,0.2))  ***//创建流水线*** *println*(**"创建流水线"**)  **val** myPipeline: MyPipeline = **new** MyPipeline().setLabelColumnName(**"gender"**)  .setFeatureColumnNames(*Array*(**"male\_c1\_dur"**,  **"female\_c1\_dur"**, **"top1\_c1"**,  **"top2\_c1"**, **"top3\_c1"**))  .setMaxDepth(7)  .init()  *println*(**"开始训练"**)  ***//把训练集投入流水线进行训练 //得到模型*** myPipeline.train(trainDataFrame)  ***//*** *println*(**"模型打印"**)  *println*(myPipeline.getDecisionTreeString())  *println*(myPipeline.getFeatureImportances())   ***//预测*** *println*(**"预测测试集"**) ***// val test2Datafram: DataFrame = testDataFrame.select("uid" , // "top1\_c1","top2\_c1","top3\_c1" ,"male\_c1\_dur","female\_c1\_dur")* val** predictedDataFrame: DataFrame = myPipeline.predict(testDataFrame)  ***//评估*** *println*(**"评估"**)  myPipeline.evaluate(predictedDataFrame)    **val** properties: Properties =  MyPropertiesUtil.*load*(**"config.properties"**)  **val** modelPath: String =  properties.getProperty(**"model.path"**)   myPipeline.save(modelPath)  } |

增加DataSQL类 ，用于把SQL单独提取出来提高复用，使代码清晰。

|  |
| --- |
|  |

MyPipeline的类中加入

|  |
| --- |
| ***//保存* def** save(path:String): Unit ={  **if**(*pipelineModel*==**null**) **throw new** RuntimeException(**"need training first! "**)  *pipelineModel*.write.overwrite().save(path) }   ***//加载* def** load(path:String): MyPipeline = {  *pipelineModel* = PipelineModel.*load*(path)  this }  ***//转换预测结果矢量* def** convertLabel(dataFrame:DataFrame): DataFrame ={  **if**(*pipelineModel*==**null**) **throw new** RuntimeException(**"need training first! "**)  **val** indexerModel: StringIndexerModel =  *pipelineModel*.stages(0).asInstanceOf[StringIndexerModel]  **val** converter = **new** IndexToString()  .setInputCol(**"prediction"**)  .setOutputCol(**"prediction\_origin"**)  .setLabels(indexerModel.labels)  **val** convertedDF: DataFrame = converter.transform(dataFrame)  convertedDF } |

在config.properties中加入模型在hdfs中的存储路径配置

|  |
| --- |
| model.path=hdfs://bigdata01:8020/train\_model/user\_profile2021/busi\_gender |

## 11.5 预测阶段的开发

### 11.5.1 主程序

|  |
| --- |
| **def** main(args: Array[String]): Unit = {  **val** sparkConf: SparkConf = **new** SparkConf()  .setAppName(**"busi\_gender\_train"**).setMaster(**"local[\*]"**)  **val** sparkSession: SparkSession = SparkSession.*builder*()  .config(sparkConf).enableHiveSupport().getOrCreate()   **var** taskId=**""  var** taskDate=**""  if**(args.length>0){  taskId=args(0)  taskDate=args(1)  }  ***//通过SQL 把特征数据集取出*** *println*(**"把特征数据集取出"**)  **val** querySql =  **s"""  | with user\_cate as  | (  | select user\_id ,category1\_id c1 ,during\_time from  | dwd\_page\_log pl join dim\_sku\_info ski  | on pl.page\_item=ski.id  | where ski.dt='$**taskDate**' and pl.dt='$**taskDate**'  | and page\_id='good\_detail' and page\_item\_type='sku\_id'  | ) ,  | user\_gender as  | (  | select id ,gender from dim\_user\_info  | where dt='9999-99-99' and gender is null  | )  | select user\_id as uid , top1\_c1,top2\_c1,top3\_c1 ,male\_c1\_dur,female\_c1\_dur  | from  | (  | select user\_id ,  | sum(if(ct\_rank=1,c1,0)) top1\_c1,  | sum(if(ct\_rank=2,c1,0)) top2\_c1,  | sum(if(ct\_rank=3,c1,0)) top3\_c1,  | sum(sum\_dur),  | sum( if(c1 in (3,4,16),sum\_dur,0 )) male\_c1\_dur ,  | sum( if(c1 in (8,12,15),sum\_dur,0 )) female\_c1\_dur  | from  | (  | select user\_id ,c1 ,sum(during\_time) sum\_dur,count(\*),  | row\_number()over(partition by user\_id order by count(\*) desc ) ct\_rank  | from user\_cate  | group by user\_id ,c1  | order by user\_id  | ) user\_c1  | group by user\_id  | )user\_ct join user\_gender ug on ug.id=user\_ct.user\_id  """**.stripMargin;  *println*(querySql)  sparkSession.sql(**"use gmall2021"**)  **val** dataFram: DataFrame = sparkSession.sql(querySql) ***// println("行数："+dataFram.count()) // dataFram.show(100,false)  //取出模型*** *println*(**"取出模型"**)  **val** properties: Properties = MyPropertiesUtil.*load*(**"config.properties"**)  **val** modelPath: String = properties.getProperty(**"model.path"**)  **val** myPipeline: MyPipeline = **new** MyPipeline().load(modelPath)  *println*(myPipeline.getDecisionTreeString())  *println*(myPipeline.getFeatureImportances())  ***// 预测*** *println*(**"预测"**)  **val** predictedDataFrame: DataFrame = myPipeline.predict(dataFram)  predictedDataFrame.show(100,**false**)  *println*(**"预测行数："**+predictedDataFrame.count()) ***// // 转换为原值*** *println*(**"转换为原值"**)  **val** predictedWithOrginDF: DataFrame = myPipeline.convertOriginLabel(predictedDataFrame)   predictedWithOrginDF.show(100,**false**)    } |

### 11.5.4 插入标签方法

|  |
| --- |
| **def** insertTag(taskId :String,taskDate :String, predictedDF: DataFrame,sparkSession: SparkSession): Unit ={   **val** properties: Properties = MyPropertiesUtil.*load*(**"config.properties"**)  **val** hdfsStorePath: String = properties.getProperty(**"hdfs-store.path"**)  **val** dwDBName: String = properties.getProperty(**"data-warehouse.dbname"**)  **val** upDBName: String = properties.getProperty(**"user-profile.dbname"**)   predictedDF.select(**"uid"**,**"prediction\_gender"**)  .cache().createTempView(**"predict\_result"**)   ***//创建表* val** tagInfo: TagInfo = TagInfoDAO.*getTagInfoByTaskId*(taskId)    ***//根据 标签名称建表 ，如果没有就建表 根据字段类型  //用tagcode作为表名* val** tableName: String = tagInfo.tagCode.toLowerCase   **val** createTableSql = **s""" create table if not exists $***userProfileDbName* **.$**tableName **( uid bigint,tag\_value string)   comment $**{tagInfo.tagName} **PARTITIONED BY (`dt` STRING)  ROW FORMAT DELIMITED FIELDS TERMINATED BY '\\t'   LOCATION '$**hdfsStorePath**/$**upDBName**/$**tableName**.toLowerCase/'"""   *// 如果预测没有计算的用户，用真实性别补充* val** insertSql= **s""" insert overwrite table $***userProfileDbName***.$**tableName **partition (dt='$**taskDate**' )  | select cast(uid as bigint) uid,   | if(prediction\_gender='M','男','女' )   | tag\_value from predict\_result  | union all  | select id, if(gender='M','男','女' )   | from $***wareHouseDbName***.dim\_user\_info ui  | where dt='9999-99-99'  | and (select count(1) as num from predict\_result pr where pr.uid=ui.id)=0  """**.stripMargin   *println*(**"-------开始创建表格-------------"**)  sparkSession.sql(createTableSql)  *println*(**"-------开始插入数据-------------"**)  *println*(insertSql)  sparkSession.sql(insertSql)   } |