**大数据电商用户画像项目**

**第八部分◎教案**

|  |  |  |
| --- | --- | --- |
| **课程教案版本** | **日期** | **备注** |
| **V1.0** | **20190515** |  |
|  |  |  |

# 第8章：挖掘类标签之聚类业务基础实战

# 项目架构再梳理

## 项目描述

企业级360°全方位用户画像是基于电商平台进行设计和开发，是面向注册会员的偏好、行为习惯和 人口属性的画像还原，同时也包括对商品信息的画像还原。提供用户喜好和商品特征帮助营销平台提升营销的精准度，也方便个性化推荐系统快速准确的为每个用户推荐相关的商品。



1.标签体系

标签体系模块基于本体论建立，包括基础标签和组合标签两个子模块。基础标签维护了人口属性、商业属性、行为属性和用户价值4个方面。组合标签是在基础标签的基础上按TGI创建的，用于反映目标群体在特定研究范围的强势和弱势。

2.标签引擎

标签引擎模块是用户画像运转的核心实现，维护标签的具体实现逻辑，包括标签依赖的数据源、规则和模型的信息。同时提供每一个标签引擎运行状态的可视化监控。当Super User登录时会出现新增标签及其引擎的审核。

3.画像模块

画像模块是对实体（用户/物品）信息全貌可视化的精准还原，包括对个体的全方位还原和群体的全方位还原。

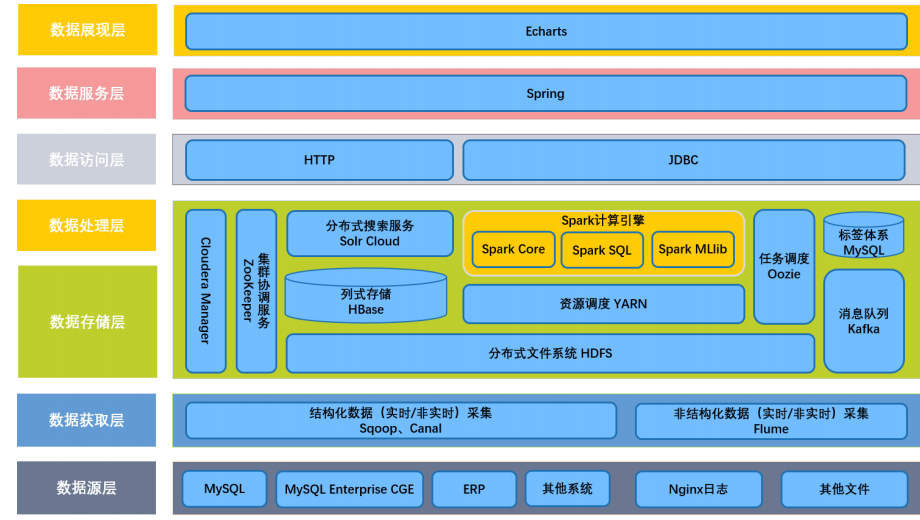
4.标签查询

标签查询模块是查看标签所覆盖的实体，用于查看标签体系中所有标签包含的商品详细信息和经过 脱敏后的用户信息。

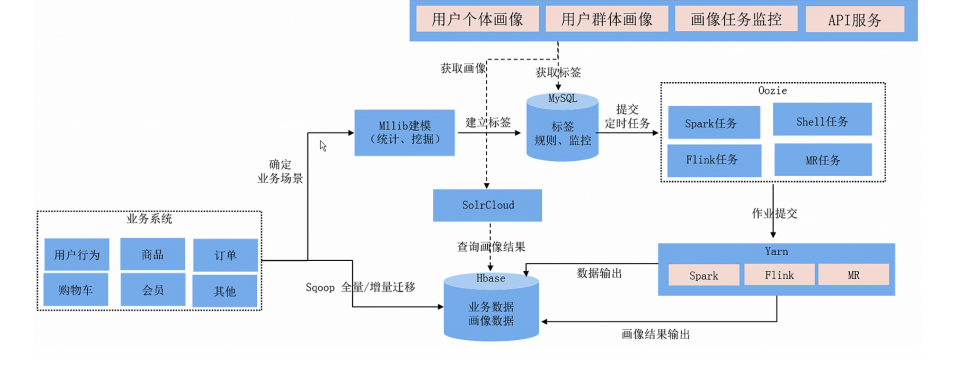
5.系统设置

系统设置模块主要包括用户管理、权限管理。

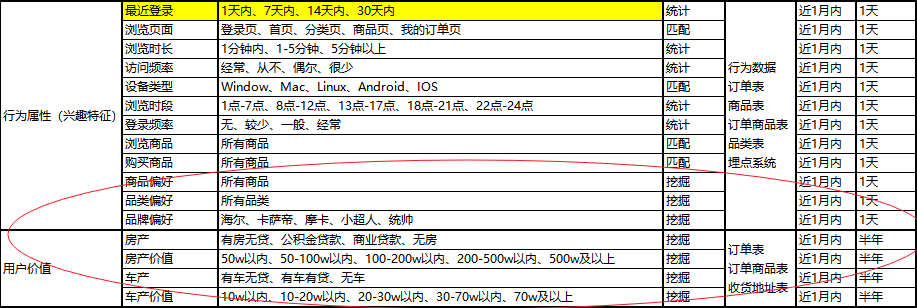
## 技术架构



## 项目功能架构



## 挖掘类标签的举例



# 2.用户活跃度模型-RFE

## 2.1RFE模型引入

RFE模型可以说是RFM模型的变体。 RFE模型基于用户的普通行为（非转化或交易行为）产生，它跟RFM类似都是使用三个维度做价值评估。

### 2.1.1RFE详解

RFE 模型是根据会员最近一次访问时间R（ Recency）、访问频率 F（Frequency）和页面互动度 E（Engagements）计算得出的RFE得分。 其中：

a.最近一次访问时间 R（ Recency）： 会员最近一次访问或到达网站的时间。

b.访问频率 F（ Frequency）：用户在特定时间周期内访问或到达的频率。

c.页面互动度 E（ Engagements）：互动度的定义可以根据不同企业或行业的交互情况而定，例如可以定义为页面 浏览时间、浏览商品数量、视频播放数量、点赞数量、转发数量等。

在RFE模型中，由于不要求用户发生交易，因此可以做未发生登录、 注册等匿名用户的行为价值分析， 也可以做实名用户分析。该模型常用来做用户活跃分群或价值区分， 也可用于内容型（例如论坛、新闻、资讯等）企业的会员分析。

RFM和 RFE模型的实现思路相同， 仅仅是计算指标发生变化。 对于RFE的数据来源， 可以从企业自己监控的用户行为日志获取，也可以从第三方网站分析工具获得。

### 2.1.2基于RFE模型的实践应用

在得到用户的RFE得分之后， 跟 RFM 类似也可以有两种应用思路：

1：基于三个维度值做用户群体划分和解读，对用户的活跃度做分析。 RFE得分为 313 的会员说明其访问频率低， 但是每次访问时的交互都非常不错， 此时重点要做用户回访频率的提升，例如通过活动邀请、 精准广告投放、会员活动推荐等提升回访频率。

2：基于RFE的汇总得分评估所有会员的活跃度价值，并可以做活跃度排名； 同时，该得分还可以作为输入维 度跟其他维度一起作为其他数据分析和挖掘模型的输入变量，为分析建模提供基础。

比如：

- 6忠诚 （1天内访问2次及以上，每次访问页面不重复）

- 5活跃 （2天内访问至少1次）

- 4回流 （3天内访问至少1次）

- 3新增 （注册并访问）

- 2不活跃 （7天内未访问）

- 1流失 （7天以上无访问）

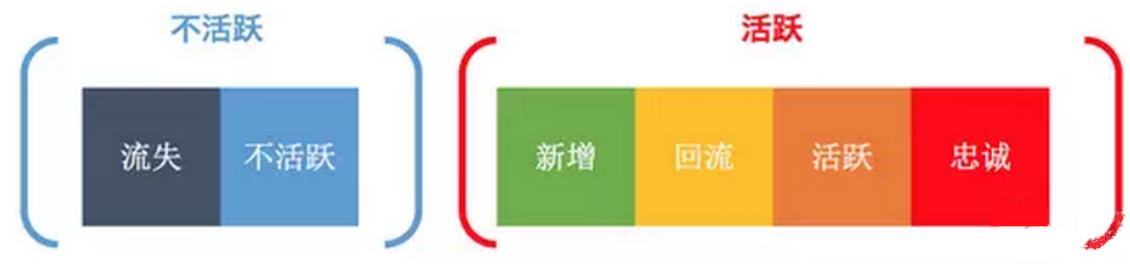
## 2.2代码实现

以下是完整代码，详细步骤后续分解。

|  |
| --- |
| **package** cn.itcast.up.ml  **import** cn.itcast.up.base.BaseModel2 **import** cn.itcast.up.common.**HDFSUtils import** org.apache.spark.ml.clustering.{KMeans, KMeansModel} **import** org.apache.spark.ml.feature.VectorAssembler **import** org.apache.spark.sql.\_  **import** scala.collection.immutable   */\*\*  \* Author itcast  \* Date 2019/10/28 9:22  \* Desc 用户活跃度模型-RFE  \* Recency:最近一次访问时间,用户最后一次访问距今时间  \* Frequency:访问频率,用户一段时间内访问的页面总数,  \* Engagements:页面互动度,用户一段时间内访问的独立页面数,也可以定义为页面 浏览量、下载量、 视频播放数量等  \*/* **object RFEModel extends** BaseModel2{  **def** main(args: **Array**[String]): Unit = {  execute()  }  */\*\*  \* 获取标签id(即模型id,该方法应该在编写不同模型时进行实现)  \* @return  \*/* **override def** getTagID(): Int = 45   */\*\*  \* 开始计算  \* inType=HBase##zkHosts=192.168.10.20##zkPort=2181##  \* hbaseTable=tbl\_logs##family=detail##selectFields=global\_user\_id,loc\_url,log\_time  \* @param fiveDF MySQL中的5级规则 id,rule  \* @param hbaseDF 根据selectFields查询出来的HBase中的数据  \* @return userid,tagIds  \*/* **override def** compute(fiveDF: DataFrame, hbaseDF: DataFrame): DataFrame = {  **import** *spark*.**implicits**.\_  **import** scala.collection.**JavaConversions**.\_  **import** org.apache.spark.sql.**functions**.\_  *//fiveDF.show()  //fiveDF.printSchema()   //hbaseDF.show(10,false)  //hbaseDF.printSchema()  /\* +---+----+ | id|rule| +---+----+ | 46| 1| | 47| 2| | 48| 3| | 49| 4| +---+----+  root  |-- id: long (nullable = false)  |-- rule: string (nullable = true)  +--------------+-------------------------------------------------------------------+-------------------+ |global\_user\_id|loc\_url |log\_time | +--------------+-------------------------------------------------------------------+-------------------+ |424 |http://m.eshop.com/mobile/coupon/getCoupons.html?couponsId=3377 |2019-09-09 13:09:21| |619 |http://m.eshop.com/?source=mobile |2019-07-29 15:07:41| |898 |http://m.eshop.com/mobile/item/11941.html |2019-08-14 09:23:44| |642 |http://www.eshop.com/l/2729-2931.html |2019-08-11 03:20:17| |130 |http://www.eshop.com/ |2019-08-12 11:59:28| |515 |http://www.eshop.com/l/2723-0-0-1-0-0-0-0-0-0-0-0.html |2019-07-23 14:39:25| |274 |http://www.eshop.com/ |2019-07-24 15:37:12| |772 |http://ck.eshop.com/login.html |2019-07-24 07:56:49| |189 |http://m.eshop.com/mobile/item/9673.html |2019-07-26 19:17:00| |529 |http://m.eshop.com/mobile/search/\_bplvbiwq\_XQS75\_btX\_ZY1328-se.html|2019-07-25 23:18:37| +--------------+-------------------------------------------------------------------+-------------------+ only showing top 10 rows  root  |-- global\_user\_id: string (nullable = true)  |-- loc\_url: string (nullable = true)  |-- log\_time: string (nullable = true)  \*/   //0.定义常量字符串,避免后续拼写错误* **val** recencyStr = "recency"  **val** frequencyStr = "frequency"  **val** engagementsStr = "engagements"  **val** featureStr = "feature"  **val** scaleFeatureStr = "scaleFeature"  **val** predictStr = "predict"   *//1.按用户id进行聚合获取用户活跃度-RFE  //Recency:最近一次访问时间,用户最后一次访问距今时间,当前时间 - log\_time  //Frequency:访问频率,用户一段时间内访问的页面总数,count(loc\_url)  //Engagements:页面互动度,用户一段时间内访问的独立页面数,也可以定义为页面 浏览量、下载量、 视频播放数量等,distinct count(loc\_url)* **val** recencyAggColumn: **Column** = *datediff*(*date\_sub*(*current\_timestamp*(),60), *max*('log\_time)) as recencyStr  **val** frequencyAggColumn: **Column** = *count*('loc\_url) as frequencyStr  **val** engagementsAggColumn: **Column** = *countDistinct*('loc\_url) as engagementsStr   **val** RFEResult = hbaseDF.groupBy('global\_user\_id)  .agg(recencyAggColumn, frequencyAggColumn, engagementsAggColumn)  *//RFEResult.show(10)  /\* +--------------+-------+---------+-----------+ |global\_user\_id|recency|frequency|engagements| +--------------+-------+---------+-----------+ | 296| 12| 380| 227| | 467| 12| 405| 267| | 675| 12| 370| 240| | 691| 12| 387| 244| | 829| 12| 404| 269| | 125| 12| 375| 246| | 451| 12| 347| 224| | 800| 12| 395| 242| | 853| 12| 388| 252| | 944| 12| 394| 252| +--------------+-------+---------+-----------+  \*/   // R:0-15天=5分，16-30天=4分，31-45天=3分，46-60天=2分，大于61天=1分  // F:≥400=5分，300-399=4分，200-299=3分，100-199=2分，≤99=1分  // E:≥250=5分，230-249=4分，210-229=3分，200-209=2分，1=1分* **val** recencyScore: **Column** = *when*(*col*(recencyStr).between(0, 15), 5)  .when(*col*(recencyStr).between(16, 30), 4)  .when(*col*(recencyStr).between(31, 45), 3)  .when(*col*(recencyStr).between(46, 60), 2)  .when(*col*(recencyStr).gt(60), 1)  .as(recencyStr)   **val** frequencyScore: **Column** = *when*(*col*(frequencyStr).geq(400), 5)  .when(*col*(frequencyStr).between(300, 399), 4)  .when(*col*(frequencyStr).between(200, 299), 3)  .when(*col*(frequencyStr).between(100, 199), 2)  .when(*col*(frequencyStr).leq(99), 1)  .as(frequencyStr)   **val** engagementsScore: **Column** = *when*(*col*(engagementsStr).geq(250), 5)  .when(*col*(engagementsStr).between(200, 249), 4)  .when(*col*(engagementsStr).between(150, 199), 3)  .when(*col*(engagementsStr).between(50, 149), 2)  .when(*col*(engagementsStr).leq(49), 1)  .as(engagementsStr)   **val** RFEScoreResult =RFEResult.select('global\_user\_id as "userId", recencyScore, frequencyScore, engagementsScore)  .where('userId.isNotNull and *col*(recencyStr).isNotNull and *col*(frequencyStr).isNotNull and *col*(engagementsStr).isNotNull)  *//RFEScoreResult.show(10)    //3.聚类  //为方便后续模型进行特征输入，需要部分列的数据转换为特征向量，并统一命名，VectorAssembler类就可以完成这一任务。  //VectorAssembler是一个transformer，将多列数据转化为单列的向量列* **val** vectorDF: DataFrame = **new** VectorAssembler()  .setInputCols(*Array*(recencyStr, frequencyStr, engagementsStr))  .setOutputCol(featureStr)  .transform(RFEScoreResult)*//.na.drop  //vectorDF.show(10)  /\* +------+-------+---------+-----------+-------------+ |userId|recency|frequency|engagements| feature| +------+-------+---------+-----------+-------------+ | 296| 5| 4| 4|[5.0,4.0,4.0]| | 467| 5| 5| 5|[5.0,5.0,5.0]| | 675| 5| 4| 4|[5.0,4.0,4.0]| | 691| 5| 4| 4|[5.0,4.0,4.0]| | 829| 5| 5| 5|[5.0,5.0,5.0]| | 125| 5| 4| 4|[5.0,4.0,4.0]| | 451| 5| 4| 4|[5.0,4.0,4.0]| | 800| 5| 4| 4|[5.0,4.0,4.0]| | 853| 5| 4| 5|[5.0,4.0,5.0]| | 944| 5| 4| 5|[5.0,4.0,5.0]| +------+-------+---------+-----------+-------------+  \*/   //https://cloud.tencent.com/document/product/851/35146  //最小最大归一化将每个特征调整到一个特定的范围,通常是（0,1）  //(X - X.min)/(X.max - X.min)  //归一化数据可以使各个特征维度对目标函数的影响权重一致，提高迭代的求解的收敛速度  /\*val scaler = new MinMaxScaler()  .setInputCol(featureStr)  .setOutputCol(scaleFeatureStr)   var scalerModel: MinMaxScalerModel = scaler.fit(vectorDF)  val scalerDF = scalerModel.transform(vectorDF)  scalerDF.show(10)\*/  /\* +------+-------+---------+-----------+-------------+-------------+ |userId|recency|frequency|engagements| feature| scaleFeature| +------+-------+---------+-----------+-------------+-------------+ | 296| 5| 4| 4|[5.0,4.0,4.0]|[0.5,0.0,0.0]| | 467| 5| 5| 5|[5.0,5.0,5.0]|[0.5,1.0,1.0]| | 675| 5| 4| 4|[5.0,4.0,4.0]|[0.5,0.0,0.0]| | 691| 5| 4| 4|[5.0,4.0,4.0]|[0.5,0.0,0.0]| | 829| 5| 5| 5|[5.0,5.0,5.0]|[0.5,1.0,1.0]| | 125| 5| 4| 4|[5.0,4.0,4.0]|[0.5,0.0,0.0]| | 451| 5| 4| 4|[5.0,4.0,4.0]|[0.5,0.0,0.0]| | 800| 5| 4| 4|[5.0,4.0,4.0]|[0.5,0.0,0.0]| | 853| 5| 4| 5|[5.0,4.0,5.0]|[0.5,0.0,1.0]| | 944| 5| 4| 5|[5.0,4.0,5.0]|[0.5,0.0,1.0]| +------+-------+---------+-----------+-------------+-------------+  \*/* **val** kMeans: **KMeans** = **new** KMeans()  .setK(4)  .setSeed(10) *//可重复的随机* .setMaxIter(2)*//最大迭代次数* .setFeaturesCol(featureStr) *//特征列* .setPredictionCol(predictStr)*//预测结果列   //4.训练模型* **var** model: **KMeansModel** = **null  val** path = "/model/RFEModel/"  **if** (**HDFSUtils**.*getInstance*().exists(path)){  model = **KMeansModel**.*load*(path)  }**else**{  model = kMeans.fit(vectorDF)  model.save(path)  }   *//5.预测* **val** result: DataFrame = model.transform(vectorDF)  result.show(10)  */\* +------+-------+---------+-----------+-------------+-------+ |userId|recency|frequency|engagements| feature|predict| +------+-------+---------+-----------+-------------+-------+ | 296| 5| 4| 4|[5.0,4.0,4.0]| 0| | 467| 5| 5| 5|[5.0,5.0,5.0]| 2| | 675| 5| 4| 4|[5.0,4.0,4.0]| 0| | 691| 5| 4| 4|[5.0,4.0,4.0]| 0| | 829| 5| 5| 5|[5.0,5.0,5.0]| 2| | 125| 5| 4| 4|[5.0,4.0,4.0]| 0| | 451| 5| 4| 4|[5.0,4.0,4.0]| 0| | 800| 5| 4| 4|[5.0,4.0,4.0]| 0| | 853| 5| 4| 5|[5.0,4.0,5.0]| 1| | 944| 5| 4| 5|[5.0,4.0,5.0]| 1| +------+-------+---------+-----------+-------------+-------+  \*/   //6.测试时看下聚类效果* **val** ds: **Dataset**[Row] = result  .groupBy(predictStr)  .agg(*max*(*col*(recencyStr) + *col*(frequencyStr) + *col*(engagementsStr)), *min*(*col*(recencyStr) + *col*(frequencyStr) + *col*(engagementsStr)))  .sort(*col*(predictStr).asc)  ds.show()  */\* +-------+------------------------------------------+------------------------------------------+ |predict|max(((recency + frequency) + engagements))|min(((recency + frequency) + engagements))| +-------+------------------------------------------+------------------------------------------+ | 0| 13| 13| | 1| 14| 14| | 2| 15| 15| | 3| 14| 14| +-------+------------------------------------------+------------------------------------------+  \*/   //问题: 每一个簇的ID是无序的,但是我们将分类簇和rule进行对应的时候,需要有序  //7.按质心排序,质心大,该类用户价值大  //[(质心id, 质心值)]* **val** center: immutable.IndexedSeq[(Int, Double)] = **for**(i <- model.clusterCenters.indices) **yield** (i, model.clusterCenters(i).toArray.sum)  **val** sortedCenter: immutable.IndexedSeq[(Int, Double)] = center.sortBy(\_.\_2).reverse  sortedCenter.foreach(*println*)  */\* (2,15.0) (3,14.0) (1,14.0) (0,13.0)  \*/  //[(质心id, rule值)]* **val** centerIdAndRule: immutable.IndexedSeq[(Int, Int)] = **for**(i <- sortedCenter.indices) **yield** (sortedCenter(i).\_1, i+1)  centerIdAndRule.foreach(*println*)  */\* (2,1) (3,2) (1,3) (0,4)  \*/* **val** predictRuleDF: DataFrame = centerIdAndRule.toDF(predictStr,"rule")  predictRuleDF.show()  */\* +-------+----+ |predict|rule| +-------+----+ | 2| 1| | 3| 2| | 1| 3| | 0| 4| +-------+----+  \*/   //8.将rule和5级规则进行匹配* **val** ruleTagDF: DataFrame = fiveDF.as[(Long,String)].map(t=>(t.\_2,t.\_1)).toDF("rule","tag")  ruleTagDF.show()  */\* +----+---+ |rule|tag| +----+---+ | 1| 46| | 2| 47| | 3| 48| | 4| 49| +----+---+  \*/* **val** predictRuleTagDF: DataFrame = predictRuleDF.join(ruleTagDF,"rule")  predictRuleTagDF.show()  */\* +----+-------+---+ |rule|predict|tag| +----+-------+---+ | 1| 2| 46| | 2| 3| 47| | 3| 1| 48| | 4| 0| 49| +----+-------+---+  \*/  //Map[predict, tag]* **val** map: Map[Long, Long] = predictRuleTagDF.as[(String,Long,Long)].map(t=>(t.\_2,t.\_3)).collect().toMap  **val** predict2tag = **functions**.*udf*((predict:Long)=>{  **val** tag = map(predict)  tag  })   *//8.predict->tag* **val** newDF: DataFrame = result.select($"userId",predict2tag($"predict").as("tagIds"))  newDF.show(10) */\* +------+------+ |userId|tagIds| +------+------+ | 296| 49| | 467| 46| | 675| 49| | 691| 49| | 829| 46| | 125| 49| | 451| 49| | 800| 49| | 853| 48| | 944| 48| +------+------+  \*/* newDF  } } |

# 3.用户活跃度模型：RFE

用户活跃、用户价制度分析的在数据分析师的日常工作中经常会碰到，如何根据公司的业务情况对本公司的用户做活跃和价制度的划分是一种常规化的分析工作。



用户包含各种类型，反应了不同群体的特征和想法。在使用整个产品的周期中，我们应定义更全面的指标：

1、流失用户：有一段时间没有再打开产品，那么我们就视为流失用户，根据产品的属性，可以按30天，60天，90天等划分。

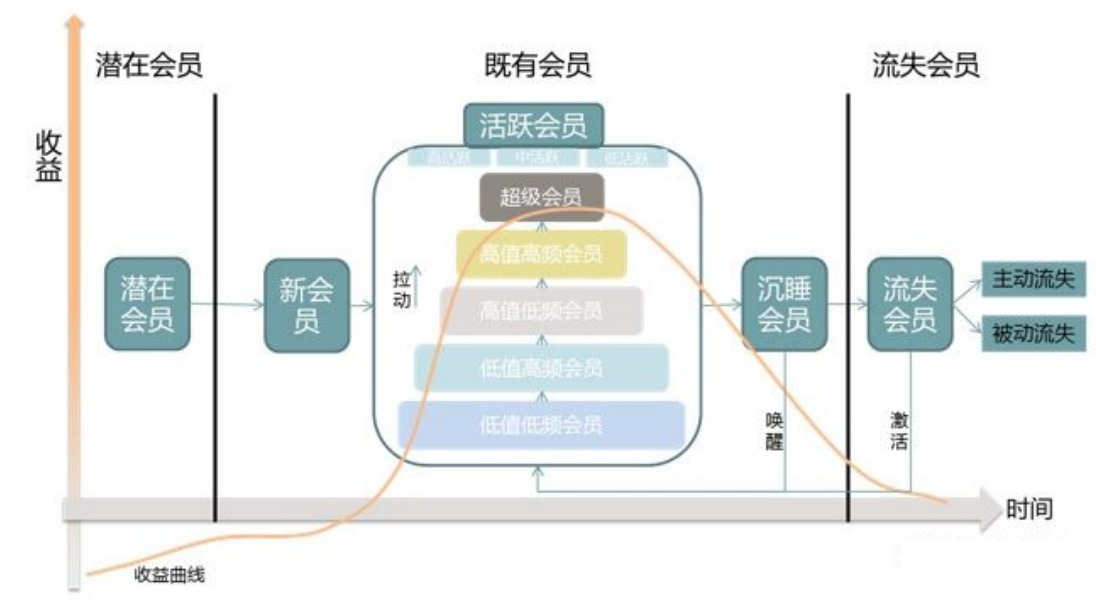
2、不活跃用户：有一段时间没有打开产品，为了和流失区分开来，需要选择无交集的时间范围。比如流失用户是60天以上没打开产品，那么不活跃则是0～60天没打开。

3、回流用户：有一段时间没用产品，之后突然回来再次使用，则称为回流用户。回流用户是活跃用户，且是由流失用户或不活跃用户唤回而来。

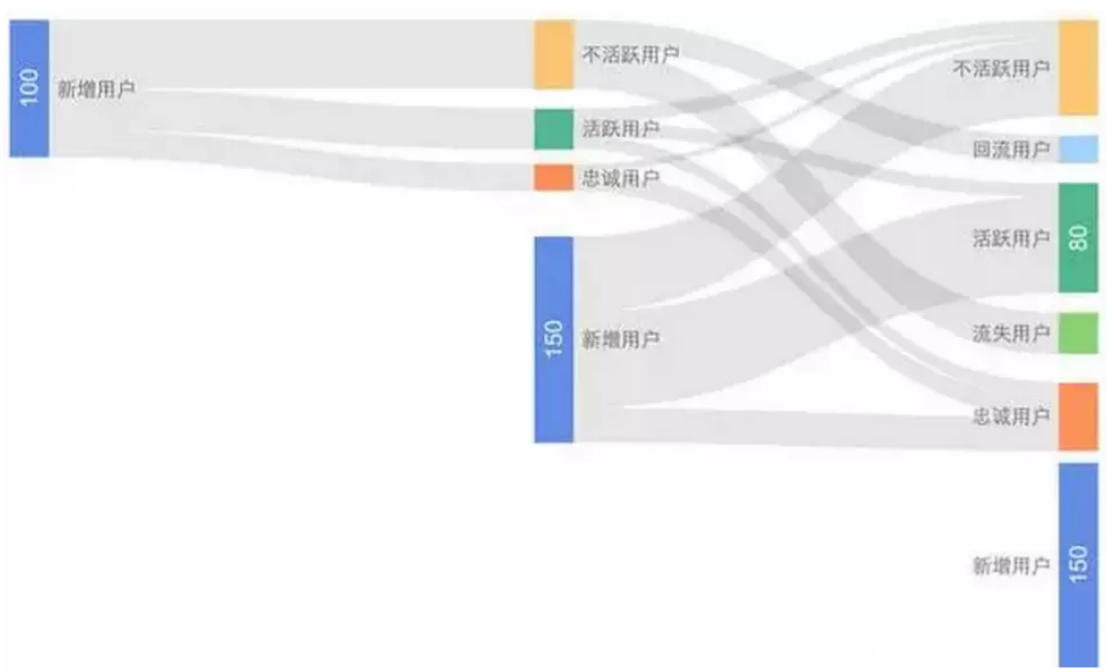
4、活跃用户：一段时间内打开过产品。

5、忠诚用户：也可以叫超级活跃用户，长期持续使用产品，比如连续四周，或者一个月内15天等。

在用户生命周期中，对每个用户进行群体划分，有针对性地做分群分层运营，可以更高地提高营收转化。（用户生命周期指用户从注册账户建立起业务关系到完全终止关系的全过程，它动态地描述了用户在不同阶段的大致特征）。



桑基（Sankey）图：详细的监控活跃数据的变化。



RFE模型可以说是RFM模型的变体。 RFE模型基于用户的普通行为（非转化或交易行为）产生，它跟RFM类似都是使用三个维度做价值评估。

## 3.1RFE 模型

RFE 模型是根据会员最近一次访问时间R（ Recency）、访问频率 F（Frequency）和页面互动度 E（Engagements）计算得出的RFE得分。

a.最近一次访问时间 R（ Recency）： 会员最近一次访问或到达网站的时间。

b.访问频率 F（ Frequency）：用户在特定时间周期内访问或到达的频率。

c.页面互动度 E（ Engagements）：互动度的定义可以根据不同企业或行业的交互情况而定，

例如可以定义为页面浏览时间、浏览商品数量、视频播放数量、点赞数量、转发数量等。

在RFE模型中，由于不要求用户发生交易，因此可以做未发生登录、 注册等匿名用户的行为价值分析， 也可以做实名用户分析。该模型常用来做用户活跃分群或价值区分， 也可用于内容型（例如论坛、新闻、资讯等）企业的会员分析。

RFM和 RFE模型的实现思路相同， 仅仅是计算指标发生变化。 对于RFE的数据来源， 可以从企业自己监控的用户行为日志获取，也可以从第三方网站分析工具获得。

## 3.2RFE 实践应用

在得到用户的RFE得分之后， 跟 RFM 类似也可以有两种应用思路：

1、基于三个维度值做用户群体划分和解读，对用户的活跃度做分析。

RFE得分为 313 的会员说明其访问频率低， 但是每次访问时的交互都非常不错， 此时

重点要做用户回访频率的提升，例如通过活动邀请、 精准广告投放、会员活动推荐等提升回访

频率。

2、基于RFE的汇总得分评估所有会员的活跃度价值，并可以做活跃度排名； 同时，该得分还可以作为输入维度跟其他维度一起作为其他数据分析和挖掘模型的输入变量，为分析建模提供基础。

比如：

**- 6忠诚 （1天内访问2次及以上，每次访问页面不重复）**

**- 5活跃 （2天内访问至少1次）**

**- 4回流 （3天内访问至少1次）**

**- 3新增 （注册并访问）**

**- 2不活跃 （7天内未访问）**

**- 1流失 （7天以上无访问）**

在RFE模型中， 由于不要求用户发生交易， 因此可以做未发生登录、 注册等匿名用户的行为价值分析， 也可以做实名用户分析。 该模型常用来做用户活跃分群或价值区分。

## 3.3标签模型开发

在标签管理平台新建标签（业务标签和属性标签），参考【客户价值模型RFM】构建标签模型，预测用户标签值，结合标签规则，给每个用户打标签。

### 3.3.1新建标签

新建业务（4级）标签：用户活跃度标签，相关字段信息如下：

标签名称：用户活跃度

标签分类：电商-某商城-行为属性

更新周期：1天

业务含义：用户活跃度分为**非常活跃、活跃、不活跃及非常不活跃四类**

标签规则：

inType=hbase

zkHosts=bigdata-cdh01.itcast.cn

zkPort=2181

hbaseTable=tbl\_tag\_logs

family=detail

selectFieldNames=global\_user\_id,loc\_url,log\_time

程序入口：

cn.itcast.tags.models.ml.RfeModel

算法名称：

KMEANS

算法引擎：

tags-model\_2.11.jar

模型参数：

--driver-memory 512m --executor-memory 512m --num-executors 1 --

executor-cores 1

新建属性（5级）标签，相关字段信息如下：

1）、属性值【非常活跃】

标签名称：非常活跃

标签含义：用户活跃度为非常活跃

标签规则：1

2）、属性值【活跃】

标签名称：活跃

标签含义：用户活跃度为活跃

标签规则：2

3）、属性值【不活跃】

标签名称：不活跃

标签含义：用户活跃度为不活跃

标签规则：3

4）、属性值【非常不活跃】

标签名称：非常不活跃

标签含义：用户活跃度为非常不活跃

标签规则：4

新建标签（业务标签和属性标签）插入SQL语句如下：

INSERT INTO `tbl\_basic\_tag` VALUES ('370', '用户活跃度', null,

'inType=hbase\nzkHosts=bigdatacdh01.

itcast.cn\nzkPort=2181\nhbaseTable=tbl\_tag\_logs\nfamily=detail\ns

electFieldNames=global\_user\_id,loc\_url,log\_time', null, '4', '316',

'2019-12-08 06:24:27', '2019-12-08 06:24:27', null, null);

INSERT INTO `tbl\_basic\_tag` VALUES ('371', '非常活跃', null, '1', null,

'5', '370', '2019-12-08 06:24:58', '2019-12-08 06:24:58', null, null);

INSERT INTO `tbl\_basic\_tag` VALUES ('372', '活跃', null, '2', null, '5',

'370', '2019-12-08 06:25:07', '2019-12-08 06:25:07', null, null);

INSERT INTO `tbl\_basic\_tag` VALUES ('373', '不活跃', null, '3', null,

'5', '370', '2019-12-08 06:25:14', '2019-12-08 06:25:14', null, null);

INSERT INTO `tbl\_basic\_tag` VALUES ('374', '非常不活跃', null, '4', null,

'5', '370', '2019-12-08 06:25:22', '2019-12-08 06:25:22', null, null);

### 3.3.2模型开发

继承基类AbstractModel ，实现标签计算方法doTag ，涉及到行为日志表数据中字段

值：

1 column=detail:global\_user\_id, timestamp=1574240607242,value=424

1 column=detail:loc\_url, timestamp=1574240607242,

value=http://m.eshop.com/m obile/coupon/getCoupons.html?couponsId=3377

1 column=detail:log\_time, timestamp=1574240607242, value=2019-08-13 03:03:55

RFE三者的打分规则如下:

计算R值：

最近一次访问时间，距离今天的天数 - max -> datediff

计算F值：

所有访问浏览量（PV） - count

计算E值：

**所有访问页面量（不包含重复访问页面）（UV） - count distinct**

R:0-15天=5分，16-30天=4分，31-45天=3分，46-60天=2分，大于61天=1分

F:≥400=5分，300-399=4分，200-299=3分，100-199=2分，≤99=1分

E:≥250=5分，200-249=4分，150-199=3分，149-50=2分，≤49=1分

具体代码如下所示：

/\*

45 用户活跃度

46 非常活跃 1

47 活跃 2

48 不活跃 3

49 非常不活跃 4

\*/

### 主函数写法

### 从业务中计算R、F、E值

*/\*\*  
 \* Desc 用户活跃度模型-RFE  
 \* Recency:最近一次访问时间,用户最后一次访问距今时间  
 \* Frequency:访问频率,用户一段时间内访问的页面总次数,  
 \* Engagements:页面互动度,用户一段时间内访问的独立页面数,也可以定义为页面 浏览量、下载量、 视频播放数量等  
 \*/***object** FREModel **extends** BaseModel{  
  
 **def** main(args: Array[String]): Unit = {  
 execute()  
 }  
 */\*\*  
 \* 获取标签id(即模型id,该方法应该在编写不同模型时进行实现)  
 \** **@return** *\*/* **override def** getTagID(): Int = 45  
  
 */\*\*  
 \* 开始计算  
 \** **@param fiveDF** *MySQL中的5级规则 id,rule  
 \** **@param hbaseDF** *根据selectFields查询出来的HBase中的数据  
 \** **@return** *userid,tagIds  
 \*/* **override def** compute(fiveDF: DataFrame, hbaseDF: DataFrame): DataFrame = {  
 *//fiveDF.show(10,false)  
 //fiveDF.printSchema()  
 //hbaseDF.show(10,false)  
 //hbaseDF.printSchema()  
 /\*  
 +---+----+  
|id |rule|  
+---+----+  
|46 |1 |  
|47 |2 |  
|48 |3 |  
|49 |4 |  
+---+----+  
  
root  
 |-- id: long (nullable = false)  
 |-- rule: string (nullable = true)  
  
+--------------+--------------------------------------+  
|global\_user\_id|loc\_url |log\_time   
+--------------+------------------+-------------------+  
|424 |http://m.eshop.com/mobile/coupon/getCoupons.html?couponsId=3377 |2019-08-13 03:03:55|  
|619 |http://m.eshop.com/?source=mobile |2019-07-29 15:07:41|  
|898 |http://m.eshop.com/mobile/item/11941.html |2019-08-14 09:23:44|  
|642 |http://www.eshop.com/l/2729-2931.html |2019-08-11 03:20:17|  
|130 |http://www.eshop.com/ |2019-08-12 11:59:28|  
+--------------+--------------+-----------------------+  
only showing top 10 rows  
  
root  
 |-- global\_user\_id: string (nullable = true)  
 |-- loc\_url: string (nullable = true)  
 |-- log\_time: string (nullable = true)  
 \*/*

### 根据规则计算RFE打分值Score

*//https://blog.csdn.net/liam08/article/details/79663018* **import** org.apache.spark.sql.functions.\_  
 **import** *spark*.implicits.\_  
  
 *//0.定义常量字符串,避免后续拼写错误* **val** recencyStr = **"recency"  
 val** frequencyStr = **"frequency"  
 val** engagementsStr = **"engagements"  
 val** featureStr = **"feature"  
 val** scaleFeatureStr = **"scaleFeature"  
 val** predictStr = **"predict"** *//Recency:最近一次访问时间,用户最后一次访问距今时间  
 //Frequency:访问频率,用户一段时间内访问的页面总次数,  
 //Engagements:页面互动度,用户一段时间内访问的独立页面数,也可以定义为页面 浏览量、下载量、 视频播放数量等* **val** recencyAggColumn: Column = *datediff*(*date\_sub*(*current\_timestamp*(),60),*max*(**"log\_time"**)) as recencyStr  
 **val** frequencyAggColumn: Column = *count*(**"loc\_url"**) as frequencyStr  
 **val** engagementsAggColumn: Column = *countDistinct*(**"loc\_url"**) as engagementsStr  
  
 *//1.根据用户id进行分组,求出RFE* **val** tempDF: DataFrame = hbaseDF

.groupBy(**"global\_user\_id"**)  
 .agg(recencyAggColumn, frequencyAggColumn, engagementsAggColumn)  
 *//tempDF.show(10,false)  
 /\*  
+--------------+-------+---------+-----------+  
|global\_user\_id|recency|frequency|engagements|  
+--------------+-------+---------+-----------+  
|296 |18 |380 |227 |  
|467 |18 |405 |267 |  
|675 |18 |370 |240 |  
|691 |18 |387 |244 |  
|829 |18 |404 |269 |  
|125 |18 |375 |246 |  
|451 |18 |347 |224 |  
|800 |18 |395 |242 |  
|853 |18 |388 |252 |  
|944 |18 |394 |252 |  
+--------------+-------+---------+-----------+  
 \*/*

### 打分Score预测

*//2.打分  
 // R:0-15天=5分，16-30天=4分，31-45天=3分，46-60天=2分，大于61天=1分  
 // F:≥400=5分，300-399=4分，200-299=3分，100-199=2分，≤99=1分  
 // E:≥250=5分，230-249=4分，210-229=3分，200-209=2分，1=1分* **val** recencyScore: Column = *when*(*col*(recencyStr).between(0, 15), 5)  
 .when(*col*(recencyStr).between(16, 30), 4)  
 .when(*col*(recencyStr).between(31, 45), 3)  
 .when(*col*(recencyStr).between(46, 60), 2)  
 .when(*col*(recencyStr).gt(60), 1)  
 .as(recencyStr)  
  
 **val** frequencyScore: Column = *when*(*col*(frequencyStr).geq(400), 5)  
 .when(*col*(frequencyStr).between(300, 399), 4)  
 .when(*col*(frequencyStr).between(200, 299), 3)  
 .when(*col*(frequencyStr).between(100, 199), 2)  
 .when(*col*(frequencyStr).leq(99), 1)  
 .as(frequencyStr)  
  
 **val** engagementsScore: Column = *when*(*col*(engagementsStr).geq(250), 5)  
 .when(*col*(engagementsStr).between(200, 249), 4)  
 .when(*col*(engagementsStr).between(150, 199), 3)  
 .when(*col*(engagementsStr).between(50, 149), 2)  
 .when(*col*(engagementsStr).leq(49), 1)  
 .as(engagementsStr)  
  
 **val** FREScoreDF: DataFrame = tempDF.select(**$"global\_user\_id"**.as(**"userId"**),recencyScore,frequencyScore,engagementsScore)  
 .where(**'userId**.isNotNull and *col*(recencyStr).isNotNull and *col*(frequencyStr).isNotNull and *col*(engagementsStr).isNotNull)  
 *//FREScoreDF.show(10,false)  
/\*  
only showing top 10 rows  
+------+-------+---------+-----------+  
|userId|recency|frequency|engagements|  
+------+-------+---------+-----------+  
|296 |4 |4 |4 |  
|467 |4 |5 |5 |  
|675 |4 |4 |4 |  
|691 |4 |4 |4 |  
|829 |4 |5 |5 |  
|125 |4 |4 |4 |  
|451 |4 |4 |4 |  
|800 |4 |4 |4 |  
|853 |4 |4 |5 |  
|944 |4 |4 |5 |  
+------+-------+---------+-----------+  
only showing top 10 rows  
 \*/*

### 使用聚类算法KMeans训练模型与预测

*//再使用SparkMLlib中的归一化工具进行数据缩放到[0,1]之间也可以  
  
//3.特征向量化***val** vecotrDF: DataFrame = **new** VectorAssembler()  
 .setInputCols(*Array*(recencyStr, frequencyStr, engagementsStr))  
 .setOutputCol(featureStr)  
 .transform(FREScoreDF)

### K值选择

*/\* //4.1准备待使用的k  
 val ks: List[Int] = List(2,3,4,5,6,7,8)  
 //4.2准备一个集合存放k对应的SSE  
 //集合内误差平方和:Within Set Sum of Squared Error, WSSSE/SSE  
 val K2SSE: mutable.Map[Int, Double] = mutable.Map[Int,Double]()  
 //4.3依次计算每个k对应的SSE  
 for(k <- ks){  
 val model: KMeansModel = new KMeans()  
 .setK(k)  
 .setMaxIter(10)  
 .setSeed(10)  
 .setFeaturesCol(featureStr)  
 .setPredictionCol(predictStr)  
 .fit(vecotrDF)  
 val SSE: Double = model.computeCost(vecotrDF)  
 K2SSE.put(k,SSE)  
 }  
 //4.4输出k的对应的SSE的值,根据肘部法则+运营/产品的业务要求确定最终的K值  
 K2SSE.foreach(println)\*/*

### 获取聚类类簇中心点，进行数据转换

*//5.模型的保存和加载  
 //使用最优K进行模型训练,训练完之后可以对模型进行保存,方便下次直接使用* **var** model: KMeansModel = **null  
 val** path:String = **"/model/RFEModel2"  
 if** (HDFSUtils.*getInstance*().exists(path)){  
 *//如果模型目录存在应该直接加载使用  
 println*(**"模型目录存在直接加载"**)  
 model= KMeansModel.*load*(path)  
 }**else**{  
 *//如果模型目录不存在应该进行训练并保存  
 println*(**"模型目录不存在,开始进行训练"**)  
 model = **new** KMeans()  
 .setK(4)  
 .setMaxIter(10)  
 .setSeed(10)  
 .setFeaturesCol(featureStr)  
 .setPredictionCol(predictStr)  
 .fit(vecotrDF)  
 model.save(path)  
 }  
  
 *//6.预测* **val** result: DataFrame = model.transform(vecotrDF)  
 result.show(10,**false**)  
 */\*  
+------+-------+---------+-----------+-------------+-------+  
|userId|recency|frequency|engagements|feature |predict|  
+------+-------+---------+-----------+-------------+-------+  
|296 |4 |4 |4 |[4.0,4.0,4.0]|0 |  
|467 |4 |5 |5 |[4.0,5.0,5.0]|2 |  
|675 |4 |4 |4 |[4.0,4.0,4.0]|0 |  
|691 |4 |4 |4 |[4.0,4.0,4.0]|0 |  
|829 |4 |5 |5 |[4.0,5.0,5.0]|2 |  
|125 |4 |4 |4 |[4.0,4.0,4.0]|0 |  
|451 |4 |4 |4 |[4.0,4.0,4.0]|0 |  
|800 |4 |4 |4 |[4.0,4.0,4.0]|0 |  
|853 |4 |4 |5 |[4.0,4.0,5.0]|1 |  
|944 |4 |4 |5 |[4.0,4.0,5.0]|1 |  
+------+-------+---------+-----------+-------------+-------+  
only showing top 10 rows  
 \*/*

### 类簇点分类数据与标签合并，进行打分

*//7.获取聚类编号和聚类中心  
 //接下来我们要对聚类之后的用户打Tag,不能仅仅根据聚类的序号打Tag  
 //因为聚类序号并不能代表客户的价值等级,和我们5级规则并不对应  
 //所以我们应该求出每个聚类的聚类中心,排序,聚类中心的值越大说明该用户群体价值越高  
  
 //model.clusterCenters.indices获取所有聚类中心的索引编号  
 //model.clusterCenters(i)根据索引取聚类中心  
 //model.clusterCenters(i).toArray.sum求该聚类中心的RFM的和  
 //[(聚类索引/编号, 聚类中心的RFM的和)]* **val** indexAndRFM: immutable.IndexedSeq[(Int, Double)] = **for**(i <- model.clusterCenters.indices) **yield** (i,model.clusterCenters(i).toArray.sum)  
 *//val tuples: immutable.IndexedSeq[(Int, Double)] = model.clusterCenters.indices.map(i=>(i,model.clusterCenters(i).toArray.sum))* indexAndRFM.foreach(*println*)  
 */\*  
(0,12.0)  
(1,13.0)  
(2,14.0)  
(3,13.0)  
 \*/  
 //println("==================")  
 //根据聚类中心的RFM的和排好序的[(聚类索引/编号, 聚类中心的RFM的和)]* **val** sortedIndexAndRFM: immutable.IndexedSeq[(Int, Double)] = indexAndRFM.sortBy(\_.\_2).reverse  
 *//sortedIndexAndRFM.foreach(println)  
  
 //7.将上面的排好序的聚类编号和聚类中心与5级规则进行对应* **val** indexAndRFMDS: Dataset[(Int, Double)] = sortedIndexAndRFM.toDS()  
 **val** fiveDS: Dataset[(Long, String)] = fiveDF.as[(Long,String)]  
 **val** tempRDD: RDD[((Int, Double), (Long, String))] = indexAndRFMDS.*rdd*.repartition(1).zip(fiveDS.*rdd*.repartition(1))  
 tempRDD.collect().foreach(*println*)  
 */\*  
((2,14.0),(46,1))  
((3,13.0),(47,2))  
((1,13.0),(48,3))  
((0,12.0),(49,4))  
 \*/* **val** ruleDF: DataFrame = tempRDD.map(t=>(t.\_1.\_1,t.\_2.\_1)).toDF(**"predict"**,**"tagIds"**)  
 ruleDF.show()  
 */\*  
+-------+------+  
|predict|tagIds|  
+-------+------+  
| 2| 46|  
| 3| 47|  
| 1| 48|  
| 0| 49|  
+-------+------+  
 \*/*

### 写入Hbase

**val** ruleMap: collection.Map[Int, Long] = ruleDF.as[(Int,Long)].*rdd*.collectAsMap()  
  
**val** predict2Tag = *udf*((predict:Int)=>{  
 ruleMap(predict)  
})  
  
**val** newDF: DataFrame = result.select(**$"userId"**,predict2Tag(**'predict**).as(**"tagIds"**))  
newDF.show()  
*/\* \*/*newDF

# 4.价格敏感度模型-PSM

## 4.0PSM模型引入

### 4.0.1大数据杀熟

<http://www.sohu.com/a/240094402_374686>

<http://www.qianjia.com/zhike/html/2019-10/11_13453.html>

<https://cloud.tencent.com/developer/article/1450512>

### 4.0.2PSM模型在网游中的运用

PSM（Price Sensitivity Measurement）模型是在70年代由Van Westendrop所创建，其目的在于衡量目标用户对不同价格的满意及接受程度，了解其认为合适的产品价格，从而得到产品价格的可接受范围。PSM的定价是从消费者接受程度的角度来进行的，既考虑了消费者的主观意愿，又兼顾了企业追求最大利益的需求。此外，其价格测试过程完全基于所取购买对象的自然反应，没有涉及到任何竞争对手的信息。虽然缺少竞品信息是PSM的缺陷所在，由于每个网络游戏均自成一个虚拟的社会体系，一般来说，其中每个道具或服务的销售均没有竞品（除非开发组自己开发了类似的道具或服务，产生了内部竞争），从这个角度来说，PSM模型比较适合用于网游中的道具或服务的定价。此外，该模型简洁明了，操作简单，使用非常方便。

#### （1）PSM模型实施具体步骤

第一步：通过定性研究，设计出能够涵盖产品可能的价格区间的价格梯度表。该步骤通常对某一产品或服务追问被访者4个问题，并据此获得价格梯度表。梯度表的价格范围要涵盖所有可能的价格点，最低和最高价格一般要求低于或高出可能的市场价格的三倍以上。

(1)便宜的价格：对您而言什么价格该道具/服务是很划算，肯定会购买的？

(2)太便宜的价格：低到什么价格，您觉得该道具/服务会因为大家都可以随便用，而对自己失去吸引力（或对游戏造成一些不良影响）？

(3)贵的价格：您觉得“有点高，但自己能接受”的价格是多少？

(4)太贵的价格：价格高到什么程度，您肯定会放弃购买？

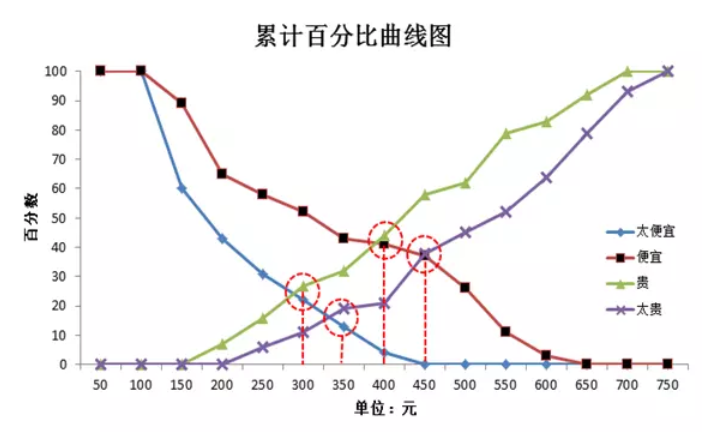
第二步：取一定数量有代表性的样本，被访者在价格梯度表上做出四项选择：有点低但可以接受的价格，太低而不会接受的价格，有点高但可以接受的价格，太高而不会接受的价格。

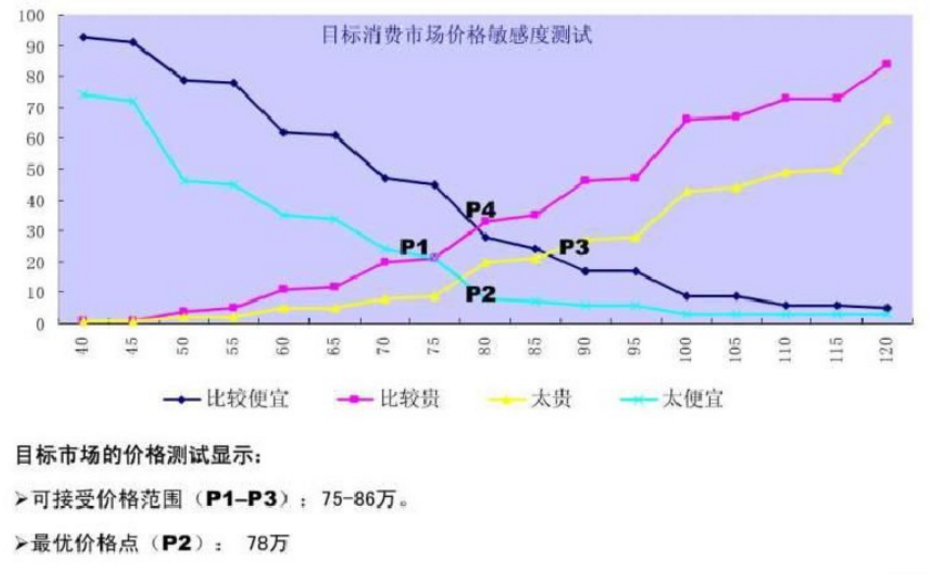
第三步：对所获得的样本数据绘制累计百分比曲线图，四条曲线的交点得出产品的合适价格区间以及最优定价点和次优定价点。

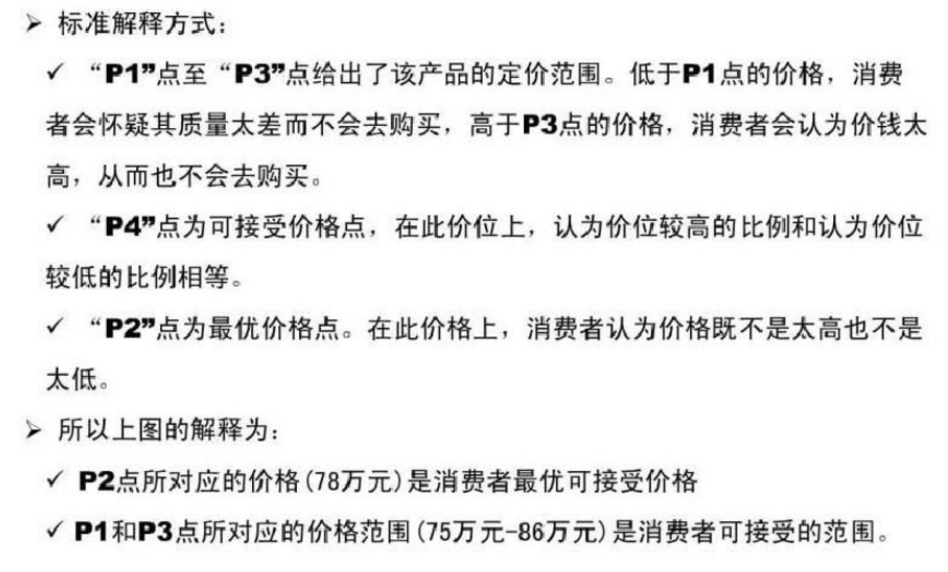
如下图， 对“便宜”和“太便宜”向下累计百分数（因为价格越低消费者越觉得便宜，即认为某价格便宜的消费者也会认为低于此价格的价格便宜），“贵”和“太贵”向上累计百分数（因为价格越高消费者越觉得贵，即认为某价格贵的消费者也会认为高于此价格的价格贵），能够得到四条累计百分比曲线。

“太便宜”和“贵”的交点意味着此价格能够让最多的人觉得“不会便宜到影响购买意愿，即使可能有点贵也是能够接受的”，“便宜”和“太贵”的交点意味着此价格能够让最多的人觉得“不会贵到不能接受，还是挺划算的”，因此这两个交点分别为价格区间的下限和上限。低于前者，消费者会因为担心“过于大众不能体现优越感，或会给游戏带来不好影响（如游戏平衡性）”而不愿购买；高于后者，消费者会认为价钱太高而不能接受。

一般来说，“太便宜”和“太贵”的交点作为最优价格点，因为在此处觉得“不过于便宜也不过于昂贵”的消费者最多。但是也有人认为“便宜”和“贵”的交点是最优价格，因为该交点取得了“划算，肯定会买”及“贵，但能接受”的平衡点，是能让最多消费者满意的价格。







#### （2）PSM模型的缺陷及解决

* 存在的问题

第一：只考虑到了消费者的接受率，忽视了消费者的购买能力，即只追求最大的目标人群数。但事实上，即使消费者觉得价格合理，受限于购买力等因素，也无法购买。

第二：研究中消费者可能出于各种因素（比如让价格更低能让自己收益，出于面子问题而抬高自己能接受的价格等）有意或无意地抬高或压低其接受的价格。由于消费者知道虚拟世界中的产品（道具或服务）没有成本，其压低价格的可能性较高。

第三：没有考虑价格变化导致的购买意愿（销量）变化。

* 解决方案

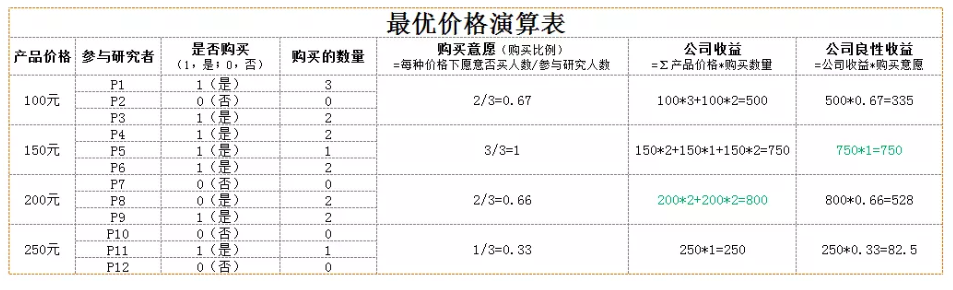
第一：为了避免购买力的影响， 问卷或访谈研究中要强调“定这个价格，以自己目前的情况是否会购买”，而非仅仅去客观判断该产品值多少钱。

第二：为了解决玩家抬高或压低价格的问题，可以增大样本量，预期随机误差可以相互抵消。

第三：仅仅从曲线获得最优价格，受到玩家压低或抬高价格的影响较大。由于该误差可能是系统误差，对此，可以用所获得的价格区间设计不同的价格方案，然后设计组间实验设计，每个参与研究的消费者只接触其中一种或几种价格方案，并对该价格方案下是否购买及购买数量做出决策，通过计算那种价格方案下玩家消费金钱量最高来分析出最佳价格方案。如下表。

第四：通过前一条中提到的组间实验设计，可以计算出不同价格下玩家购买意愿的变化，从而得知价格调整会对整体收益带来的影响。此外，价格接受比例还可以作为消费者对某价格满意度的指标，用于计算某价格下企业该产品的良性收益。

注意，我们的上述对策部分基于统计学和实验心理学理论，部分基于我们工作中的实践，欢迎大家讨论和优化。



### 4.0.3真实案例：用KANO模型和PSM价格敏感度确定产品功能和定价

今天和大家分享的是关于产品功能上线和合理定价的问题。记得一个产品姐姐在分享的时候说过一句话受益匪浅“一个产品从出生到长大，要先保证能用，再保证好用。”是的，在产品功能选择上，什么功能是保证用户能用的，什么功能是为了让用户好用的，每类功能会带给用户什么样的体验。这个问题一直也是我们在做产品过程中不断整理和思考的，用户说什么就做什么，功能丰富且齐全，但最终还是做死了的例子不胜枚举。那么一个新的功能到底上不上，这个功能/服务如果收费，价格应该是怎样的。今天就来给大家分享一下关于我之前项目遇到的这些问题和我们的解决办法。

#### （1）功能到底要不要上——KANO模型

KANO模型大家应该都很熟悉。我不太喜欢把能百度一下的“个人简介”贴上来凑字。简单说就是N个功能摆在这里，我们如何判定该功能符合基本型需求、期望型需求还是兴奋型需求？KANO模型就可以帮助我们解决这个问题。但是这里有必要说一下，评定一个需求的优先级绝对不是仅仅根据这个就直接判定的。其他吧啦吧啦的问题也是要考虑在其中的。

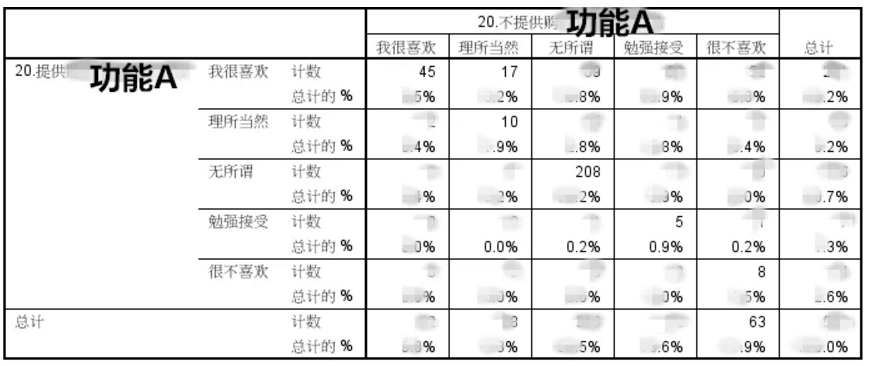
直接分享我之前的项目，如果有小伙伴不太了解KANO模型是什么，建议大家可以直接百度，我就不在这班门弄斧了。

项目背景：因为临近一个关键节日，我司产品希望能增加功能A、功能B、功能C、功能D、功能E，那么如何判定哪一些功能是可以排在前面，哪些可以稍微排后？我们设置了调研问卷来让用户对这5个功能进行评价。具体问卷形式如下图。

（为保护我司隐私，对具体内容进行隐藏。啊哈哈，我价值观很正的！）



这样一份问卷收到之后呢，你将会得到这样一个格式的数据。数据已经过处理。



这是一份“如果提供功能A五选项的选择人数\*如果不提供功能A五选项的选择人数”的交叉表。分别是交叉选择人数和人数在该选项总人数的占比。

这个图表怎么用嘞。这下我就必须要找一个解释图上来了。请见下图：



大家看到了这里面AQIRMO之类的标记了。后续的所有指数都是这些类别的求和。比如：所有分布在Q的格子里的值相加，就是Q的总体系数。所有R的格子里的值相加，就是R的总体系数。和我上面一张数据图一一对应。解释的够清楚了昂~

下面解释一下这些神奇AQIRMO都是什么意思：

A–魅力属性

O–期望属性（一维属性）

M–必备属性

I  –无差异属性（次要属性）

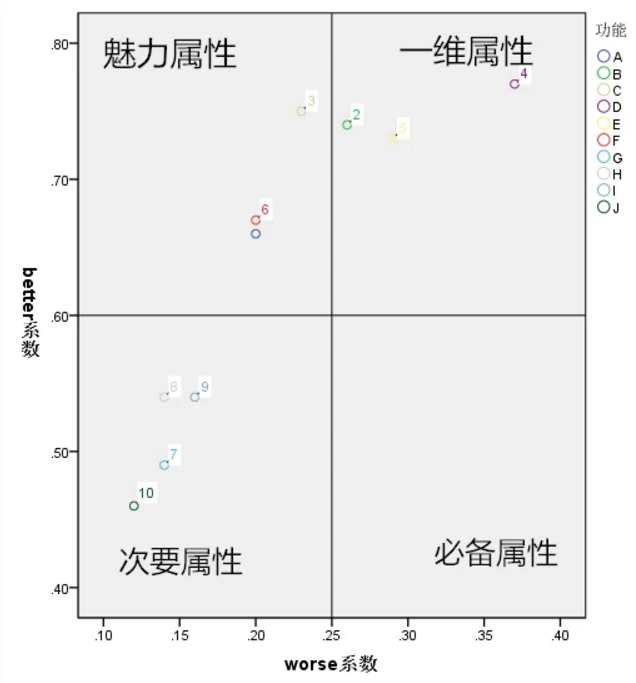
R –反向属性

Q–可疑结果

得分最高的属性就是这个功能的最后属性归属。

最后，增加了这个功能或者没有这个功能又会对用户满意度造成什么影响呢？这就是better-worse系数

增加后的满意系数（better）：（魅力属性+期望属性）/(魅力属性+期望属性+必备属性+无差异因素）

* 消除后的不满意系数（worse）：(期望属性+必备属性）/(魅力属性+期望属性+必备属性+无差异因素×（-1）
* 注：系数越接近于1或-1，说明对提供后产生满意或不提供后产生不满的影响越大
* 就酱紫，我们得到了这样的一些系数。就可以画一个逼格比较高的象限图啦。横坐标worse系数，纵坐标better系数。把四类属性分布于各个象限中。这个是用SPSS做的散点图。
* 
* 综上所述，我们就确认了功能ABCDE的各个属性归类和优先级啦~昂。功能的优先级确定就酱紫结束了。

#### （2）确定了功能/服务，该如何为其定价

不是每个功能的出现都是为了实现用户更好的操作而存在的，比如购物车、收藏夹之类的功能。还有一些功能的存在是为了能够赚钱的！是不是很直接！是不是说到了很多人的心里去！比如说卖东西寄快递，卖家愿意给你送货上门，为你提供这个功能虽然是为了用户体验更好，说到底还是起码不赚钱但不亏本的。那么快递费定价多少合适？（我就是举个例子，不要告诉我快递费多少钱是快递公司说的算）

就我司这次项目中，有一些功能是需要付一些费用的，那么要付多少钱能够保本，多少钱可以盈利，多少钱用户就觉得你有病了呢？比起拍脑袋猜，或者设置不同价格用市场来验证，我更建议在最开始的时候能确定一个价格范围值。那么用户接受的价格到底在哪个区间呢？这里就给大家隆重介绍——PSM价格敏感度测试。

还是上面的话，能百度的基本不贴字。简单说PSM就是帮助你能够获取到一个功能或者服务的用户可接受价格区间，并确定最佳价格。为了保护我司的数据。我来举个朋友卖水果的定价案例。一盒现切的水果拼盘，定价在多少最合适呢？市场容忍度是多少呢？首先，我们做了用户调研。问卷结构见下图。



（不用P图的感觉真好）这里就是问卷的结构啦。用户对每一档价格进行4个选项的评价，价格从最低到最高，最好极值设置的高一点点避免天花板和地板效应。区间取得尽量细一点，这样后面得到的结果会比较精准。

回收到问卷之后，我们将会得到这样一份数据，再来一波图。



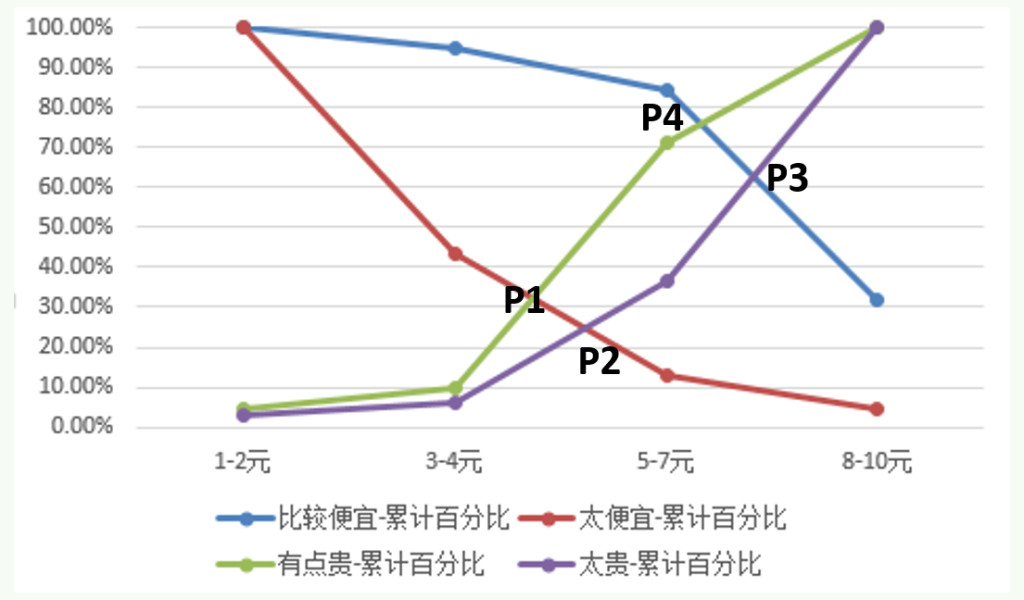
这里需要注意的有几点，因为我脑子对累计求和之类的东西总是绕不清，所以我写的详细一点。

我们得到了各类价格区间的“比较便宜、太便宜、有点贵可以忍、太贵了放弃”的频率值。然后每个选项都计算累计总和，比如：比较便宜右边，是从下往上的累计总和。为什么是从下至上求和呢？

因为，如果觉得8-10块都是比较便宜的话，5-1块钱当然都会觉得便宜啊。同理，如果觉得1-2块钱都贵的话，3-10块肯定都是觉得贵啊。

这里就是需要注意的点啦！很便宜和太便宜都是从下往上求和的，而有点贵和太贵了都是从上往下求和的。

求好了和值。计算当前这一行的和值的累计百分比（本行累计和值/累计总和）就阔以啦。举例：比较便宜列，累计和值=6，百分比=6/20；累计和值=13，百分比=13/20.以此类推。酱紫是不是就有4列百分比了？然后肿么做呢？——画图！



数据是我为了演示随便填的，图画的有点丑。好好统计出来的数据应该画的比较好看。

P1–比较贵\*太便宜曲线交叉点

P2–太贵\*太便宜曲线交叉点

P3–比较便宜\*太贵交叉点

P4–比较便宜\*比较贵交叉点

可接受价格范围：

P1–P3两点之间的价格都是用户可接受价格

低于P1——用户会认为太便宜而怀疑质量问题，高于P3——用户会觉得太贵而放弃

可接受价格点：P4——在此点，用户觉得价位太高的比例和价位太低的比例相等

最优价格点：P2——在此点，用户觉得价格既不会太贵也不会太便宜

就是这样的计算之后，我们将会得到最优价格。并且在这样一个合理的价格范围里去不断调试。这相对于直接拍脑袋来说，真的是靠谱多了呢~

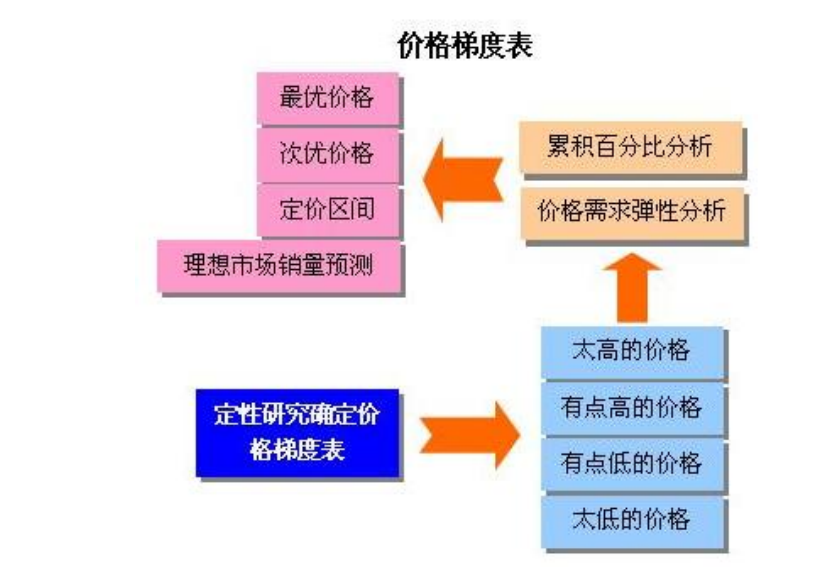
#### （3）参考资料：

1. 东京理工大学教授狩野纪昭(Noriaki Kano)和Fumio Takahashi《质量的保健因素和激励因素》(Motivator and Hygiene Factor in Quality)
2. 周达，梁英瑜，贺成功.基于KANO模型的顾客需求分析——以校园咖啡吧商品及服务项目筛选为例
3. PSM价格敏感度测试——2006.7.20 （百度文库）

# 4.价格敏感度模型：PSM

PSM模型（Price Sensitivity Measurement），即价格敏感度模型，是当前价格测试模型

中的一个常用模型，其特点为所有价格测试过程完全基于被访者的自然反应，没有任何竞争对手甚至自身产品的任何信息。**通过该模型，可以得到产品的最优价格和合理的价格间。**



为什么携程大数据“杀熟”的套路屡屡不改：

http://www.sohu.com/a/240094402\_374686

大数据杀熟”是否真的存在？：

http://www.qianjia.com/zhike/html/2019-10/11\_13453.html

啥是【大数据杀熟】，到底怎么回事？：

https://cloud.tencent.com/developer/article/1450512

## 4.1PSM 模型引入

PSM价格敏感度分析方法是在 70 年代由 Van Westendrop 所创建。其特点为所有价格测试过程完全基于被访者的自然反应，没有任何竞争对手甚至自身产品的任何信息。

**PSM模型即价格敏感度测试模型**，是目前在价格测试的诸多模型中，最简单、最实

用。为大多数市场研究公司所认可。通过PSM模型，不仅可以得出最优价格，而且得出合理的价格区间。

PSM模型的要点在于通过定性研究，设计出能够涵盖产品可能的价格区间的价格梯度表，然后在有代表性的样本中，请被访者在此价格梯度表上做出四项选择：有点高但可以接受的价格，有点低但可以接受的价格，太高而不会接受的价格，太低而不会接受的价格。对样本的这几个价格点，分别求其上向和下向累积百分比，以此累积百分比作价格需求弹性曲线，四条曲线的交点标出了产品的合适价格区间，最优定价点以及次优定价点。

我们如果有一款新产品，想要定价，我们就可以使用这个模型，找到对于消费者（被访问者）来说的最优价格和合理的售卖区间。

那这个模型怎么使用呢？都有些啥？

首先找到目标用户，咨询他们对于该产品的价格问题，如下4个，我们会得到4个价格：

1）、太高的价格（价格太高，不会购买）

2）、有点儿高的价格（价格有点儿高，但还可以接受，会买）

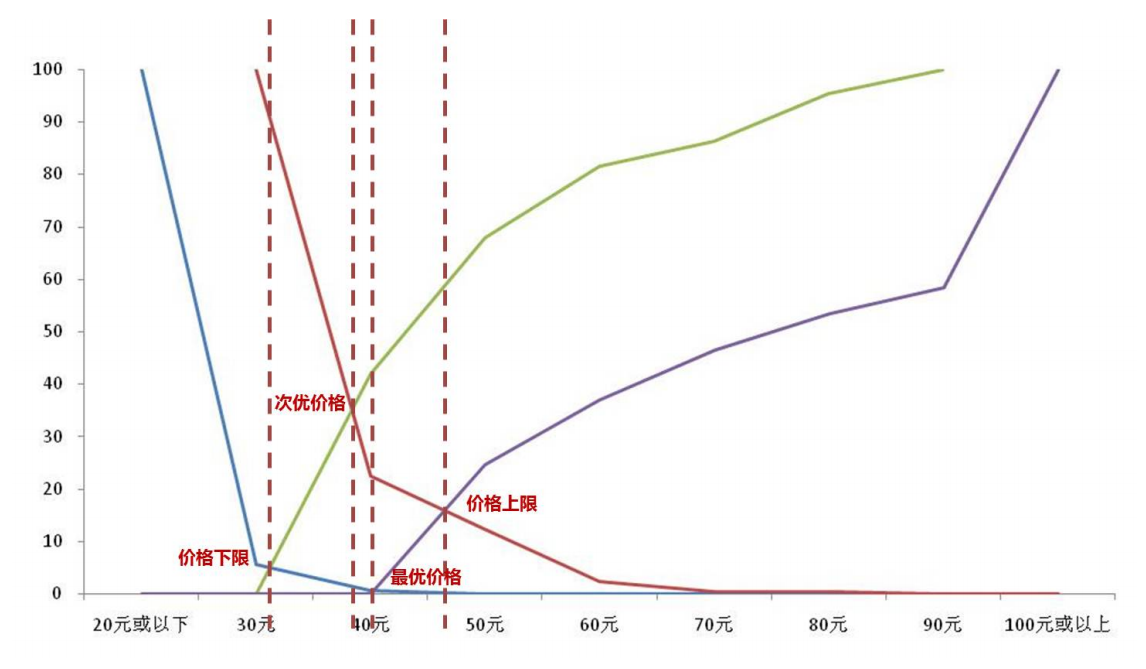
3）、有点儿低的价格（价格实惠，会买）

4）、太低的价格（价格太低，担心其他问题，不会买）



然后对上面的统计信息，对太低和较低价格求向下累计，对较贵和太贵价格求向上累计，可以绘制4条价格弹性曲线，它们的交叉点就是合适的价格区间。

最后绘制的是类似下图的曲线，其中“太低价格”和“较贵价格”的交点是价格下限，“较低价格”和“太贵价格”是价格上限，“太低价格”和“太贵价格”的交点是最优价格，“较低价格”和“较贵价格”的交点是次优价格。



## 4.2PSM 计算

/\*

50 消费敏感度 (4级标签）

51 极度敏感 1

52 比较敏感 2

53 一般敏感 3

54 不太敏感 4

55 极度不敏感 5

\*/

价格敏感度模型Price Sensitivity Meter，有时在实际业务中，会把用户分为3-5类，比如分为**极度敏感、较敏感、一般敏感、较不敏感、极度不敏感**。然后将每类的聚类中心值与实际业务所需的其他指标结合，最终确定人群类别，判断在不同需求下是否触达或怎样触达。

比如电商要通过满减优惠推广一新品牌的麦片，此时可优先选择优惠敏感且对麦片有消费偏好的用户进行精准推送，至于优惠敏感但日常对麦片无偏好的用户可暂时不进行推送或减小推送力度，优惠不敏感且对麦片无偏好的用户可选择不进行推送。

可见，在实际操作中，技术指标评价外，还应结合业务需要，才能使模型达到理想效果。

PSM模型中相关概念名词说明如下：

PSM模型中相关概念名词说明如下：

价格敏感度模型

ra: receivableAmount 应收金额

da: discountAmount 优惠金额

pa: practicalAmount 实收金额

tdon 优惠订单数

ton 总订单总数

ada 平均优惠金额

ara 平均每单应收

tda 优惠总金额

tra 应收总金额

**tdonr 优惠订单占比(优惠订单数 / 订单总数)**

**adar 平均优惠金额占比(平均优惠金额 / 平均每单应收金额)**

**tdar 优惠总金额占比(优惠总金额 / 订单总金额)**

**psm = 优惠订单占比 + 平均优惠金额占比 + 优惠总金额占比**

psmScore = tdonr + adar + tdar

简单来说可以使用占比和的值判断用户属于哪个类别,但是聚类效果完全取决于区间界限设置,所以还是用聚类算法

**>=1 极度敏感**

**0.4~1 比较敏感**

**0.1~0.3 一般敏感**

**0 不太敏感**

**<0 极度不敏感**

## 4.3模型标签开发

在标签管理平台新建标签（业务标签和属性标签），构建标签模型，预测用户标签值，结合标签规则，给每个用户打标签。

### 4.3.1新建标签

新建业务（4级）标签：消费敏感度标签，相关字段信息如下：

标签名称：消费敏感度

标签分类：电商-某商城-商业属性

更新周期：1天

业务含义：消费敏感度分为极度敏感、比较敏感、一般敏感、不太敏感及极度不敏感五类

标签规则：

inType=hbase

zkHosts=bigdata-cdh01.itcast.cn

zkPort=2181

hbaseTable=tbl\_tag\_orders

family=detail

selectFieldNames=memberid,ordersn,orderamount,couponcodevalue

程序入口：

cn.itcast.tags.models.ml.PsmModel

算法名称：

KMEANS

算法引擎：

tags-model\_2.11.jar

模型参数：

--driver-memory 512m --executor-memory 512m --num-executors 1 --

executor-cores 1

新建属性（5级）标签，相关字段信息如下：

1）、属性值【极度敏感】

标签名称：极度敏感

标签含义：用户消费敏感度为极度敏感

标签规则：1

2）、属性值【比较敏感】

标签名称：比较敏感

标签含义：用户消费敏感度为比较敏感

标签规则：2

3）、属性值【一般敏感】

标签名称：一般敏感

标签含义：用户消费敏感度为一般敏感

标签规则：3

4）、属性值【不太敏感】

标签名称：不太敏感

标签含义：用户消费敏感度为不太敏感

标签规则：4

5）、属性值【极度不敏感】

标签名称：极度不敏感

标签含义：用户消费敏感度为极度不敏感

标签规则：5

新建标签（业务标签和属性标签）插入SQL语句如下：

INSERT INTO `tbl\_basic\_tag` VALUES ('375', '消费敏感度', null,

'inType=hbase\nzkHosts=bigdatacdh01.

itcast.cn\nzkPort=2181\nhbaseTable=tbl\_tag\_orders\nfamily=detail\

nselectFieldNames=memberid,ordersn,orderamount,couponcodevalu', null,

'4', '315', '2019-12-08 06:26:33', '2019-12-08 06:26:33', null, null);

INSERT INTO `tbl\_basic\_tag` VALUES ('376', '极度敏感', null, '1', null,

'5', '375', '2019-12-08 06:27:42', '2019-12-08 06:27:42', null, null);

INSERT INTO `tbl\_basic\_tag` VALUES ('377', '比较敏感', null, '2', null,

'5', '375', '2019-12-08 06:27:56', '2019-12-08 06:27:56', null, null);

INSERT INTO `tbl\_basic\_tag` VALUES ('378', '一般敏感', null, '3', null,

'5', '375', '2019-12-08 06:28:05', '2019-12-08 06:28:05', null, null);

INSERT INTO `tbl\_basic\_tag` VALUES ('379', '不太敏感', null, '4', null,

'5', '375', '2019-12-08 06:28:13', '2019-12-08 06:28:13', null, null);

INSERT INTO `tbl\_basic\_tag` VALUES ('380', '极度不敏感', null, '5', null,

'5', '375', '2019-12-08 06:28:24', '2019-12-08 06:28:24', null, null);

### 4.3.2模型开发

继承基类AbstractModel ，实现标签计算方法doTag ，涉及到订单表数据中字段值：

10 column=detail:memberid, timestamp=1574240145469, value=13823431

10 column=detail:ordersn, timestamp=1574240145469,

value=gome\_792756751164275

10 column=detail:orderamount, timestamp=1574240145469,

value=2479.45

10 column=detail:couponcodevalue, timestamp=1574240145469,

value=0.00

PSM计算公式：

psm = 优惠订单占比 + 平均优惠金额占比 + 优惠总金额占比

tdonr adar tdar

tdonr 优惠订单占比(**优惠订单数 / 订单总数**)

adar 平均优惠金额占比(**平均优惠金额 / 平均每单应收金额**)

tdar 优惠总金额占比(**优惠总金额 / 订单总金额**)

psm的计算有很多除法,除数有可能为0,SparkSQL对于除数为0的记录直接返回null

### (1)从业务中获取五级标签及业务数据

*/\*\*  
 \* Author itcast  
 \* Date 2019/11/5 10:04  
 \* Desc 价格敏感度模型Price Sensitivity Meter  
 \* 有时在实际业务中，会把用户分为3-5类，  
 \* 比如分为极度敏感、较敏感、一般敏感、较不敏感、极度不敏感。  
 \* 然后将每类的聚类中心值与实际业务所需的其他指标结合，最终确定人群类别，判断在不同需求下是否触达或怎样触达。  
 \* 比如电商要通过满减优惠推广一新品牌的麦片，  
 \* 此时可优先选择优惠敏感且对麦片有消费偏好的用户进行精准推送，  
 \* 至于优惠敏感但日常对麦片无偏好的用户可暂时不进行推送或减小推送力度，  
 \* 优惠不敏感且对麦片无偏好的用户可选择不进行推送。  
 \* 可见，在实际操作中，技术指标评价外，还应结合业务需要，才能使模型达到理想效果。  
 \* //价格敏感度模型  
 \* //ra:receivableAmount 应收金额  
 \* //da:discountAmount 优惠金额  
 \* //pa:practicalAmount 实收金额  
 \*  
 \* //tdon 优惠订单数  
 \* //ton 总订单总数  
 \*  
 \* //ada 平均优惠金额  
 \* //ara 平均每单应收  
 \*  
 \* //tda 优惠总金额  
 \* //tra 应收总金额  
 \*  
 \* //tdonr 优惠订单占比(优惠订单数 / 订单总数)  
 \* //adar 平均优惠金额占比(平均优惠金额 / 平均每单应收金额)  
 \* //tdar 优惠总金额占比(优惠总金额 / 订单总金额)  
 \*  
 \* //psm = 优惠订单占比 + 平均优惠金额占比 + 优惠总金额占比  
 \* //psmScore = tdonr + adar + tdar  
 \* //简单来说可以使用占比和的值判断用户属于哪个类别,但是聚类效果完全取决于区间界限设置,所以还是用聚类算法  
 \* //>=1 极度敏感  
 \* //0.4~1 比较敏感  
 \* //0.1~0.3 一般敏感  
 \* //0 不太敏感  
 \* //<0 极度不敏感  
 \*/***object** PSMModel **extends** BaseModel{  
 **def** main(args: Array[String]): Unit = {  
 execute()  
 }  
 */\*\*  
 \* 获取标签id(即模型id,该方法应该在编写不同模型时进行实现)  
 \** **@return** *\*/* **override def** getTagID(): Int = 50  
  
 */\*\*  
 \* 开始计算  
 \** **@param fiveDF** *MySQL中的5级规则 id,rule  
 \** **@param hbaseDF** *根据selectFields查询出来的HBase中的数据  
 \** **@return** *userid,tagIds  
 \*/* **override def** compute(fiveDF: DataFrame, hbaseDF: DataFrame): DataFrame = {  
 fiveDF.show(10,**false**)  
 fiveDF.printSchema()  
*//* hbaseDF.show(100,**false**)  
 hbaseDF.printSchema()  
  
 */\*  
+---+----+  
|id |rule|  
+---+----+  
|51 |1 |  
|52 |2 |  
|53 |3 |  
|54 |4 |  
|55 |5 |  
+---+----+  
  
root  
 |-- id: long (nullable = false)  
 |-- rule: string (nullable = true)  
  
+---------+-------------------+-----------+---------------+  
|memberId |orderSn |orderAmount|couponCodeValue|  
+---------+-------------------+-----------+---------------+  
|13823431 |ts\_792756751164275 |2479.45 |0.00 |  
|4035167 |D14090106121770839 |2449.00 |0.00 |  
|4035291 |D14090112394810659 |1099.42 |0.00 |  
|4035041 |fx\_787749561729045 |1999.00 |0.00 |  
|13823285 |D14092120154435903 |2488.00 |0.00 |  
|4034219 |D14092120155620305 |3449.00 |0.00 |  
|138230939|top\_810791455519102|1649.00 |0.00 |  
|4035083 |D14092120161884409 |7.00 |0.00 |  
|138230935|D14092120162313538 |1299.00 |0.00 |  
|13823231 |D14092120162378713 |499.00 |0.00 |  
+---------+-------------------+-----------+---------------+  
only showing top 10 rows  
  
root  
 |-- memberId: string (nullable = true)  
 |-- orderSn: string (nullable = true)  
 |-- orderAmount: string (nullable = true)  
 |-- couponCodeValue: string (nullable = true)  
 \*/*

### (2)根据规则计算psm各项指标

**import** org.apache.spark.sql.functions.\_  
 **import** *spark*.implicits.\_  
  
  
 *//0.定义常量  
 // psmScore = tdonr + adar + tdar* **val** psmScoreStr: String = **"psm"  
 val** featureStr: String = **"feature"  
 val** predictStr: String = **"predict"** *//1.计算指标  
 //ra:receivableAmount 应收金额* **val** raColumn:Column = (**'orderAmount** + **'couponCodeValue**) as **"ra"** *//da:discountAmount 优惠金额* **val** daColumn:Column = **'couponCodeValue** as **"da"** *//pa:practicalAmount 实收金额* **val** paColumn:Column = **'orderAmount** as **"pa"** *//订单状态,1表示优惠订单即couponCodeValue != 0* **val** state:Column = functions  
 .*when*(**'couponCodeValue** =!= 0.0D, 1)*//不等于* .when(**'couponCodeValue** === 0.0d, 0)  
 .as(**"state"**)  
  
 *//tdon 优惠订单数=1有优惠+0没有优惠+0没有优惠+1有优惠* **val** tdon:Column = *sum*(**'state**) as **"tdon"** *//ton 总订单总数=count(无论出现0还是出现1，都记为一次)* **val** ton:Column = *count*(**'state**) as **"ton"** *//tda 优惠总金额==>***da表示优惠金额****val** tda: Column = *sum*(**'da**) as **"tda"** *//tra 应收总金额==>***ra表示应收金额****val** tra: Column = *sum*(**'ra**) as **"tra"** *//ada 平均优惠金额 === ('tda优惠总金额 / 'tdon优惠订单数)*

*//tdon优惠订单数可能为0  
 //ara 平均每单应收 === ('tra应收总金额 / 'ton总订单总数)* **val** tempDF: DataFrame = hbaseDF.select(**'memberId**, raColumn, daColumn, paColumn, state)  
 .groupBy(**'memberId**)  
 .agg(tdon, ton, tda, tra)  
 *//tempDF.show(10,false)  
 /\*  
 +---------+----------+------------+------+------------------+  
|memberId |tdon |ton |tda |tra |  
+---------+----------+------------+------+------------------+  
|4033473 |3 |142 |500.0 |252430.92 |  
|13822725 |4 |116 |800.0 |180098.34 |  
|13823681 |1 |108 |200.0 |169946.1 |  
|138230919|3 |125 |600.0 |240661.56999999998|  
|13823083 |3 |132 |600.0 |234124.17 |  
|13823431 |2 |122 |400.0 |181258.22 |  
|4034923 |1 |108 |200.0 |167674.89 |  
|4033575 |4 |125 |650.0 |255866.40000000002|  
|13822841 |0 |113 |0.0 |205931.91 |  
|13823153 |6 |133 |1200.0|251898.57 |  
+---------+----------+------------+------+------------------+  
only showing top 10 rows  
 \*/*

### (3)psm计算模块

*//3.计算psm  
 //tdonr 优惠订单占比(优惠订单数tdon / 订单总数ton)  
 //adar 平均优惠金额占比(平均优惠金额('tda / 'tdon)/ 平均每单应收金额('tra / 'ton))  
 //tdar 优惠总金额占比(优惠总金额tda / 订单总金额tra)  
 //psm = 优惠订单占比 + 平均优惠金额占比 + 优惠总金额占比  
 //psm的计算有很多除法,除数有可能为0,SparkSQL对于除数为0的记录直接返回null  
 //注意:SparkMLlib在计算的时候不能有null记录,所以应该将null记录过滤掉  
 //注意:SparkSQL的DSL语法中对于Null值的判断得使用isNotNull方法  
 //用"null" null 都不行  
 //注意:对于tdon优惠订单数为0,我们这里演示的知识点是机器学习需要处理null值,及如何处理  
 //而对于tdon优惠订单数为0的用户实际上是对价格不敏感的用户,应该要保留  
 //那么就可以将tdon优惠订单数为0的用户的用户的tdon优惠订单数置为一个很小的值,如0.00001* **val** psmScoreColumn:Column = (**'tdon** / **'ton**) + ((**'tda** / **'tdon**)/((**'tra** / **'ton**))) + (**'tda**/**'tra**) as **"psm"  
 val** psmScoreDF: DataFrame = tempDF.select(**'memberId**,psmScoreColumn).filter(**'psm**.isNotNull)  
 psmScoreDF.show(10,**false**)  
 */\*  
+---------+-------------------+  
|memberId |psm |  
+---------+-------------------+  
|4033473 |0.11686252330855691|  
|13822725 |0.16774328728519597|  
|13823681 |0.13753522440350205|  
|138230919|0.1303734438365045 |  
|13823083 |0.1380506927739941 |  
|13823431 |0.15321482374431458|  
|4034923 |0.13927276336831218|  
|4033575 |0.11392752155030905|  
|13823153 |0.15547466292943982|  
|4034191 |0.11026694172505715|  
+---------+-------------------+  
only showing top 10 rows  
 \*/*

### (4)特征工程形成特征向量

*//4.特征向量化***val** vectorDF: DataFrame = **new** VectorAssembler()  
 .setInputCols(*Array*(psmScoreStr))  
 .setOutputCol(featureStr).transform(psmScoreDF)

### (5)聚类算法模型训练

**var** model: KMeansModel = **null  
val** path = **"/model/PSMModel2/"***//5.聚类***if** (HDFSUtils.*getInstance*().exists(path)){  
 *//model保存过,直接加载* model= KMeansModel.*load*(path)  
}**else**{  
 *//model不存在,先训练再保存* model = **new** KMeans()  
 .setK(5)  
 .setMaxIter(10)  
 .setSeed(10)  
 .setFeaturesCol(featureStr)  
 .setPredictionCol(predictStr)  
 .fit(vectorDF)*//训练* model.save(path)  
}

### (6)模型预测

*//6.预测***val** result: DataFrame = model.transform(vectorDF)  
result.show(10,**false**)

### (7)获取聚类编号和聚类中心，进行数**据转换**

*//7.获取聚类编号和聚类中心  
 //接下来我们要对聚类之后的用户打Tag,不能仅仅根据聚类的序号打Tag  
 //因为聚类序号并不能代表客户的价值等级,和我们5级规则并不对应  
 //所以我们应该求出每个聚类的聚类中心,排序,聚类中心的值越大说明该用户群体价值越高  
  
 //model.clusterCenters.indices获取所有聚类中心的索引编号  
 //model.clusterCenters(i)根据索引取聚类中心  
 //model.clusterCenters(i).toArray.sum求该聚类中心的RFM的和  
 //[(聚类索引/编号, 聚类中心的RFM的和)]* **val** indexAndRFM: immutable.IndexedSeq[(Int, Double)] = **for**(i <- model.clusterCenters.indices) **yield** (i,model.clusterCenters(i).toArray.sum)  
 *//val tuples: immutable.IndexedSeq[(Int, Double)] = model.clusterCenters.indices.map(i=>(i,model.clusterCenters(i).toArray.sum))* indexAndRFM.foreach(*println*)  
 */\*  
(0,12.0)  
(1,13.0)  
(2,14.0)  
(3,13.0)  
 \*/  
 //println("==================")  
 //****根据聚类中心的RFM的和排好序的[(聚类索引/编号, 聚类中心的RFM的和)]*****val** sortedIndexAndRFM: immutable.IndexedSeq[(Int, Double)] = indexAndRFM.sortBy(\_.\_2).reverse  
 sortedIndexAndRFM.foreach(*println*)

### (8)类簇点分类数据与标签对应合并，进行打分

*//8.将上面的排好序的聚类编号和聚类中心与5级规则进行对应* **val** indexAndRFMDS: Dataset[(Int, Double)] = sortedIndexAndRFM.toDS()  
 **val** fiveDS: Dataset[(Long, String)] = fiveDF.as[(Long,String)]  
 **val** tempRDD: RDD[((Int, Double), (Long, String))] = indexAndRFMDS.*rdd*.repartition(1).zip(fiveDS.*rdd*.repartition(1))  
 tempRDD.collect().foreach(*println*)  
 */\*  
((2,14.0),(46,1))  
((3,13.0),(47,2))  
((1,13.0),(48,3))  
((0,12.0),(49,4))  
 \*/* **val** ruleDF: DataFrame = tempRDD.map(t=>(t.\_1.\_1,t.\_2.\_1)).toDF(**"predict"**,**"tagIds"**)  
 ruleDF.show()  
 */\*  
+-------+------+  
|predict|tagIds|  
+-------+------+  
| 2| 46|  
| 3| 47|  
| 1| 48|  
| 0| 49|  
+-------+------+  
 \*/*

### (9)写入Hbase中

**val** ruleMap: collection.Map[Int, Long] = ruleDF.as[(Int,Long)].*rdd*.collectAsMap()  
  
**val** predict2Tag = *udf*((predict:Int)=>{  
 ruleMap(predict)  
})  
  
**val** newDF: DataFrame = result.select(**$"memberId"**.as(**"userId"**),predict2Tag(**'predict**).as(**"tagIds"**))  
newDF.show()  
*/\*  
 \*/*newDF

## 4.4完成代码

**package** cn.itcast.up.mltest  
  
**import** cn.itcast.up.base.BaseModel  
**import** cn.itcast.up.common.HDFSUtils  
**import** org.apache.spark.ml.clustering.{KMeans, KMeansModel}  
**import** org.apache.spark.ml.feature.VectorAssembler  
**import** org.apache.spark.rdd.RDD  
**import** org.apache.spark.sql.{Column, DataFrame, Dataset}  
  
**import** scala.collection.\_  
  
**object** psm01 **extends** BaseModel {  
 **def** main(args: Array[String]): Unit = {  
 execute()  
 }  
  
 */\*\*  
 \* 获取标签id(即模型id,该方法应该在编写不同模型时进行实现)  
 \*  
 \** **@return** *\*/* **override def** getTagID(): Int = 50  
  
 */\*\*  
 \* 开始计算  
 \*  
 \** **@param fiveDF** *MySQL中的5级规则 id,rule  
 \** **@param hbaseDF** *根据selectFields查询出来的HBase中的数据  
 \** **@return** *userid,tagIds  
 \*/* **override def** compute(fiveDF: DataFrame, hbaseDF: DataFrame): DataFrame = {  
 fiveDF.show(10, **false**)  
 hbaseDF.show(10, **false**)  
 *// +---------+-------------------+-----------+---------------+  
 // |memberId |orderSn |orderAmount|couponCodeValue|  
 // +---------+-------------------+-----------+---------------+  
 // |13823431 |ts\_792756751164275 |2479.45 |0.00 |  
 // |4035167 |D14090106121770839 |2449.00 |0.00 |  
 // |4035291 |D14090112394810659 |1099.42 |0.00 |  
 // |4035041 |fx\_787749561729045 |1999.00 |0.00 |  
 // |13823285 |D14092120154435903 |2488.00 |0.00 |  
 // |4034219 |D14092120155620305 |3449.00 |0.00 |  
 // |138230939|top\_810791455519102|1649.00 |0.00 |  
 // psm = 优惠订单占比 + 平均优惠金额占比 + 优惠总金额占比  
 // tdonr adar tdar  
 // tdonr 优惠订单占比(优惠订单数 / 订单总数)  
 // adar 平均优惠金额占比(平均优惠金额 / 平均每单应收金额)  
 // tdar 优惠总金额占比(优惠总金额 / 订单总金额)  
 //>=1 极度敏感  
 //0.4~1 比较敏感  
 //0.1~0.3 一般敏感  
 //0 不太敏感  
 //<0 极度不敏感  
 // root  
 // |-- memberId: string (nullable = true)  
 // |-- orderSn: string (nullable = true)  
 // |-- orderAmount: string (nullable = true) 订单总金额，等于商品总金额＋运费  
 // |-- couponCodeValue: string (nullable = true) couponCodeValue 优惠码优惠金额* **import** *spark*.implicits.\_  
 **import** org.apache.spark.sql.functions  
 **import** org.apache.spark.sql.functions.\_  
 *//0.定义常量  
 // psmScore = tdonr + adar + tdar* **val** psmScoreStr: String = **"psm"  
 val** featureStr: String = **"feature"  
 val** predictStr: String = **"predict"** *//1.计算指标  
 // psm = 优惠订单占比 + 平均优惠金额占比 + 优惠总金额占比  
 // tdonr adar tdar  
 // tdonr 优惠订单占比(优惠订单数 / 订单总数)  
 // adar 平均优惠金额占比(平均优惠金额 / 平均每单应收金额)  
 // tdar 优惠总金额占比(优惠总金额 / 订单总金额)  
 //ra:receivableAmount 应收金额* **val** raColumn: Column = (**'orderAmount** + **'couponCodeValue**) as **"ra"** *//da:discountAmount 优惠金额* **val** daColumn: Column = **'couponCodeValue** as **"da"** *//pa:practicalAmount 实收金额* **val** paColumn: Column = **'orderAmount** as **"pa"** *//订单状态,1表示优惠订单即couponCodeValue != 0* **val** state: Column = functions  
 .*when*(**'couponCodeValue** =!= 0.0D, 1) *//不等于* .when(**'couponCodeValue** === 0.0d, 0)  
 .as(**"state"**)  
 *//tdon 优惠订单数* **val** tdon: Column = *sum*(**'state**) as **"tdon"** *//ton 总订单总数* **val** ton: Column = *count*(**'state**) as **"ton"** *//tda 优惠总金额* **val** tda: Column = *sum*(**'da**) as **"tda"** *//tra 应收总金额* **val** tra: Column = *sum*(**'ra**) as **"tra"  
 val** tempDF: DataFrame = hbaseDF.select(**'memberId**, raColumn, daColumn, paColumn, state)  
 .groupBy(**'memberId**)  
 .agg(tdon, ton, tda, tra)  
 tempDF.show(10, **false**)  
 *// psm = 优惠订单占比 + 平均优惠金额占比 + 优惠总金额占比  
 // tdonr adar tdar  
 // tdonr 优惠订单占比(优惠订单数 / 订单总数)  
 // ('tdon / 'ton) + (('tda / 'tdon)/(('tra / 'ton))) + ('tda/'tra) as "psm"  
 // adar 平均优惠金额占比(平均优惠金额 / 平均每单应收金额)  
 // tdar 优惠总金额占比(优惠总金额 / 订单总金额)  
 // +---------+----------+------------+------+------------------+  
 // |memberId |tdon |ton |tda |tra |  
 // +---------+----------+------------+------+------------------+  
 // |4033473 |3 |142 |500.0 |252430.92 |  
 // |13822725 |4 |116 |800.0 |180098.34 |  
 // |13823681 |1 |108 |200.0 |169946.1 |* **val** psmScoreColumn: Column = (**'tdon** / **'ton**) + ((**'tda** / **'tdon**) / ((**'tra** / **'ton**))) + (**'tda** / **'tra**) as **"psm"  
 val** psmScoreDF: DataFrame = tempDF.select(**'memberId**, psmScoreColumn).filter(**'psm**.isNotNull)  
 psmScoreDF.show(10, **false**)  
 *// +---------+-------------------+  
 // |memberId |psm |  
 // +---------+-------------------+  
 // |4033473 |0.11686252330855691|  
 // |13822725 |0.16774328728519597|  
 // |13823681 |0.13753522440350205|  
 // |138230919|0.1303734438365045 |  
 // |13823083 |0.1380506927739941 |  
 //4.特征向量化* **val** vectorDF: DataFrame = **new** VectorAssembler()  
 .setInputCols(*Array*(psmScoreStr))  
 .setOutputCol(featureStr).transform(psmScoreDF)  
 **var** model: KMeansModel = **null  
 val** path = **"/model/PSMModel2/"** *//5.聚类* **if** (HDFSUtils.*getInstance*().exists(path)) {  
 *//model保存过,直接加载* model = KMeansModel.*load*(path)  
 } **else** {  
 *//model不存在,先训练再保存* model = **new** KMeans()  
 .setK(5)  
 .setMaxIter(10)  
 .setSeed(10)  
 .setFeaturesCol(featureStr)  
 .setPredictionCol(predictStr)  
 .fit(vectorDF) *//训练* model.save(path)  
 }  
 *//6.预测* **val** result: DataFrame = model.transform(vectorDF)  
 result.show(10, **false**)  
 *//7.获取聚类编号和聚类中心  
 //接下来我们要对聚类之后的用户打Tag,不能仅仅根据聚类的序号打Tag  
 //因为聚类序号并不能代表客户的价值等级,和我们5级规则并不对应  
 //所以我们应该求出每个聚类的聚类中心,排序,聚类中心的值越大说明该用户群体价值越高  
  
 //model.clusterCenters.indices获取所有聚类中心的索引编号  
 //model.clusterCenters(i)根据索引取聚类中心  
 //model.clusterCenters(i).toArray.sum求该聚类中心的RFM的和  
 //[(聚类索引/编号, 聚类中心的RFM的和)]* **val** indexAndRFM: immutable.IndexedSeq[(Int, Double)] = **for** (i <- model.clusterCenters.indices) **yield** (i, model.clusterCenters(i).toArray.sum)  
 *//val tuples: immutable.IndexedSeq[(Int, Double)] = model.clusterCenters.indices.map(i=>(i,model.clusterCenters(i).toArray.sum))* indexAndRFM.foreach(*println*)  
 */\*  
(0,12.0)  
(1,13.0)  
(2,14.0)  
(3,13.0)  
 \*/  
 //println("==================")  
 //根据聚类中心的RFM的和排好序的[(聚类索引/编号, 聚类中心的RFM的和)]* **val** sortedIndexAndRFM: immutable.IndexedSeq[(Int, Double)] = indexAndRFM.sortBy(\_.\_2).reverse  
 sortedIndexAndRFM.foreach(*println*)  
 *//8.将上面的排好序的聚类编号和聚类中心与5级规则进行对应* **val** indexAndRFMDS: Dataset[(Int, Double)] = sortedIndexAndRFM.toDS()  
 **val** fiveDS: Dataset[(Long, String)] = fiveDF.as[(Long, String)]  
 **val** tempRDD: RDD[((Int, Double), (Long, String))] = indexAndRFMDS.*rdd*.repartition(1).zip(fiveDS.*rdd*.repartition(1))  
 tempRDD.collect().foreach(*println*)  
 */\*  
((2,14.0),(46,1))  
((3,13.0),(47,2))  
((1,13.0),(48,3))  
((0,12.0),(49,4))  
 \*/* **val** ruleDF: DataFrame = tempRDD.map(t=>(t.\_1.\_1,t.\_2.\_1)).toDF(**"predict"**,**"tagIds"**)  
 ruleDF.show()  
 */\*  
+-------+------+  
|predict|tagIds|  
+-------+------+  
| 2| 46|  
| 3| 47|  
| 1| 48|  
| 0| 49|  
+-------+------+  
 \*/* **val** ruleMap: collection.Map[Int, Long] = ruleDF.as[(Int,Long)].*rdd*.collectAsMap()  
  
 **val** predict2Tag = *udf*((predict:Int)=>{  
 ruleMap(predict)  
 })  
  
 **val** newDF: DataFrame = result.select(**$"memberId"**.as(**"userId"**),predict2Tag(**'predict**).as(**"tagIds"**))  
 newDF.show()  
 */\*  
  
 \*/* **null** }  
}

# 5.评论文本特征处理技术

## 5.1文本的特殊性

用户画像中的评论信息或文章信息或咨询信息都是文本数据，该类数据处理复杂主要有两个原因。

1. 文本和语言都隐含有结构信息，使用原始的文本很难铺捉到(例如：含义、上下文、不同词性的词语，句法结构等)，因此单纯的类似于传统结构化特征的方法是有问题的。
2. 文本数据的维度有效维度一般是非常巨大并且都是无限大的，如果所有的中文字符，特殊字符、俗语的总数有多少，然后加上其他语言和所有可以在互联网上找到的文本，即使在较小的数据集上，文本数据按照的维度也轻易超过千万甚至上亿个词汇。

**如何从数据中提取合适的特征？**

首先提出的就是**NLP自然语言处理领域**研究文本处理的技术，包括特征提取、建模、和机器学习。

主要涉及的技术由**提取特征的文本处理，分词和过滤技术**，降低输入数据的维度，提高提取的特征的信息含量和有用性。

## 5.2SparkMllib特征提取操作实践

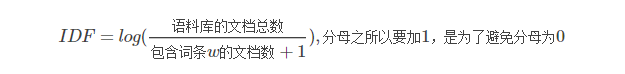
### 5.2.1 TF-IDF

词频－逆向文件频率（TF-IDF）是一种在文本挖掘中广泛使用的特征向量化方法，它可以体现一个文档中词语在语料库中的重要程度。

      词语由t表示，文档由d表示，语料库由D表示。词频TF(t,,d)是词语t在文档d中出现的次数。文件频率DF(t,D)是包含词语的文档的个数。



如果我们只使用词频来衡量重要性，很容易过度强调在文档中经常出现而并没有包含太多与文档有关的信息的词语，比如“a”，“the”以及“of”。如果一个词语经常出现在语料库中，它意味着它并没有携带特定的文档的特殊信息。逆向文档频率数值化衡量词语提供多少信息：



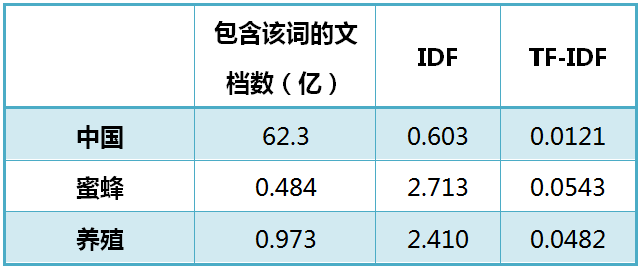
其中，|D|表示是语料库中的文档总数。由于采用了对数，如果一个词出现在所有的文件，其IDF值变为0。



某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。因此，TF-IDF倾向于过滤掉常见的词语，保留重要的词语。

可以看到，TF-IDF与一个词在文档中的出现次数成正比，与该词在整个语言中的出现次数成反比。所以，自动提取关键词的算法就很清楚了，就是计算出文档的每个词的TF-IDF值，然后按降序排列，取排在最前面的几个词。

接下来看一个例子，假定该文长度为1000个词，"中国"、"蜜蜂"、"养殖"各出现20次，则这三个词的"词频"（TF）都为0.02。然后，搜索Google发现，包含"的"字的网页共有250亿张，假定这就是中文网页总数。包含"中国"的网页共有62.3亿张，包含"蜜蜂"的网页为0.484亿张，包含"养殖"的网页为0.973亿张。则它们的逆文档频率（IDF）和TF-IDF如下：



从上表可见，"蜜蜂"的TF-IDF值最高，"养殖"其次，"中国"最低。（如果还计算"的"字的TF-IDF，那将是一个极其接近0的值。）所以，如果只选择一个词，"蜜蜂"就是这篇文章的关键词。

除了自动提取关键词，TF-IDF算法还可以用于许多别的地方。比如，信息检索时，对于每个文档，都可以分别计算一组搜索词（"中国"、"蜜蜂"、"养殖"）的TF-IDF，将它们相加，就可以得到整个文档的TF-IDF。这个值最高的文档就是与搜索词最相关的文档。

接下来我们就通过SparkMllib实现TF-IDF

import org.apache.spark.sql.SparkSession

import org.apache.spark.ml.feature.{HashingTF, IDF, Tokenizer}

object TfIdf\_1 {

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

val spark: SparkSession = SparkSession.builder()

.appName("SparkMlilb")

.master("local[2]")

.getOrCreate()

spark.sparkContext.setLogLevel("WARN")

val sentenceData = spark.createDataFrame(Seq(

(0, "Hi I heard about Spark"),

(0, "I wish Java could use case classes"),

(1, "Logistic regression models are neat")

)).toDF("label", "sentence")

val tokenizer = new Tokenizer().setInputCol("sentence").setOutputCol("words")

val wordsData = tokenizer.transform(sentenceData)

val hashingTF = new HashingTF()

.setInputCol("words").setOutputCol("rawFeatures").setNumFeatures(20)

val featurizedData = hashingTF.transform(wordsData)

// alternatively, CountVectorizer can also be used to get term frequency vectors

val idf = new IDF().setInputCol("rawFeatures").setOutputCol("features")

val idfModel = idf.fit(featurizedData)

val rescaledData = idfModel.transform(featurizedData)

rescaledData.select("features", "label").take(3).foreach(println)

}

}

### 5.2.2Word2Vec

word2vec，字面意思，将word转化为vector，word是顺序有意义的实体，比如文档中单词、用户依次点击的商品。

word2vec是NLP领域的重要算法，它的功能是将word用K维的dense vector来表达，训练集是语料库，不含标点，以空格断句。因此可以看作是种特征处理方法。

word2vec两种实现方式：

Skip-gram：用一个词语作为输入，来预测它周围的上下文。同义词p(word1|word2)

CBOW ：用一个词语的上下文作为输入，来预测这个词语本身。完形填空p(word1|word2,word3)

import org.apache.spark.sql.SparkSession

import org.apache.spark.ml.feature.Word2Vec

object Word2Vec\_3 {

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

val spark: SparkSession = SparkSession.builder()

.appName("SparkMlilb")

.master("local[2]")

.getOrCreate()

spark.sparkContext.setLogLevel("WARN")

// Input data: Each row is a bag of words from a sentence or document.

val documentDF = spark.createDataFrame(Seq(

"Hi I heard about Spark".split(" "),

"I wish Java could use case classes".split(" "),

"Logistic regression models are neat".split(" ")

).map(Tuple1.apply)).toDF("text")

// Learn a mapping from words to Vectors.

val word2Vec = new Word2Vec()

.setInputCol("text")

.setOutputCol("result")

.setVectorSize(3)

.setMinCount(0)

val model = word2Vec.fit(documentDF)

val result = model.transform(documentDF)

result.select("result").take(3).foreach(println)

}

}

结果：

[[0.03173386193811894,0.009443491697311401,0.024377789348363876]]

[[0.025682436302304268,0.0314303718706859,-0.01815584538105343]]

[[0.022586782276630402,-0.01601201295852661,0.05122732147574425]]

### 补充输出

//如何计算一个词汇的表达  
vecModel.getVectors.show(false)  
// +----------+----------------------------------------------------------------+  
// |word |vector |  
// +----------+----------------------------------------------------------------+  
// |are |[-0.12636634707450867,0.09880797564983368,-0.10397180169820786] |  
// |neat |[-0.04065164178609848,0.028278442099690437,-0.16341333091259003]|  
// |博客 |[0.11578879505395889,0.10333182662725449,-0.0694052129983902] |  
// |logistic |[0.12220428884029388,-0.031169431284070015,-0.14288558065891266]|  
// |classes |[-0.05294015631079674,0.14731089770793915,-0.002295464975759387]|  
// |小数据 |[-0.019351687282323837,-0.13164418935775757,0.1438964307308197] |  
// |i |[-0.15043501555919647,-0.04881952330470085,0.03254816308617592] |  
// |java |[0.1396487057209015,0.08163177222013474,0.1590203195810318] |  
// |regression|[0.036457594484090805,0.05805594101548195,-0.022408021613955498]|  
// |could |[0.14943106472492218,-0.11199285835027695,-0.04010562226176262] |  
// |use |[0.16599826514720917,0.06475517898797989,0.09221957623958588] |  
// |models |[0.11906852573156357,0.13384437561035156,0.143197700381279] |  
// |西瓜 |[0.15318696200847626,0.06039959564805031,0.07782142609357834] |  
// |case |[-0.05575539544224739,0.01002994179725647,-0.033811263740062714]|  
// |wish |[0.08216803520917892,0.0029408682603389025,-0.07986190915107727]|  
// |大数据 |[-0.16320903599262238,-0.1453019678592682,0.11392915993928909] |  
// +----------+----------------------------------------------------------------+  
println("寻找相似度高的词汇出来")  
//a dataframe with columns "word" and "similarity" of the word and the cosine similarities between the synonyms and the given word vector.  
vecModel.findSynonyms("大数据", 10).show(false)  
// +-------+--------------------+  
// |word |similarity |  
// +-------+--------------------+  
// |i |0.8884649872779846 |  
// |小数据 |0.8008750677108765 |  
// |case |0.2331354320049286 |  
// |are |-0.11841480433940887|  
// |could |-0.26949864625930786|  
// |java |-0.2958204448223114 |  
// |classes|-0.3376331329345703 |  
// |neat |-0.3824171721935272 |  
// |models |-0.3992846608161926 |  
// |use |-0.5257618427276611 |  
// +-------+--------------------+

### 5.2.3CountVectorizer

CountVectorizer并CountVectorizerModel旨在帮助将一组文本文档转换为标签计数的向量。当apriori字典不可用时，CountVectorizer可以用作Estimator提取词汇表，并生成一个CountVectorizerModel。该模型为词汇表上的文档生成稀疏表示，然后可以将其传递给其他算法，如LDA。

在拟合过程中，CountVectorizer将选择vocabSize按语料库中的术语频率排序的顶部单词。可选参数minDF还通过指定术语必须出现在文档中的最小数量（或<1.0）来影响拟合过程。另一个可选的二进制切换参数控制输出向量。如果设置为true，则所有非零计数都设置为1.这对于模拟二进制而非整数计数的离散概率模型特别有用。

**Examples**

假设我们有以下的DataFrame，带有列id和文本：

id | texts

----|----------

0 | Array("a", "b", "c")

1 | Array("a", "b", "b", "c", "a")

在文本中类型是 Array[String].

调用CountVectorizer的拟合会生成带有词汇表（a，b，c）的CountVectorizerModel。 然后转换后的输出列“vector”包含：

id | texts | vector

----|---------------------------------|---------------

0 | Array("a", "b", "c") | (3,[0,1,2],[1.0,1.0,1.0])

1 | Array("a", "b", "b", "c", "a") | (3,[0,1,2],[2.0,2.0,1.0])

import org.apache.spark.ml.feature.{CountVectorizer, CountVectorizerModel}

import org.apache.spark.sql.SparkSession

object CountVector\_3 {

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

val spark: SparkSession = SparkSession.builder()

.appName("SparkMlilb")

.master("local[2]")

.getOrCreate()

spark.sparkContext.setLogLevel("WARN")

val df = spark.createDataFrame(Seq(

(0, Array("a", "b", "c")),

(1, Array("a", "b", "b", "c", "a"))

)).toDF("id", "words")

// fit a CountVectorizerModel from the corpus

val cvModel: CountVectorizerModel = new CountVectorizer()

.setInputCol("words")

.setOutputCol("features")

.setVocabSize(3)

.setMinDF(2)

.fit(df)

// alternatively, define CountVectorizerModel with a-priori vocabulary

val cvm = new CountVectorizerModel(Array("a", "b", "c"))

.setInputCol("words")

.setOutputCol("features")

cvModel.transform(df).select("features").show()

}

}

结果:

+--------------------+

| features|

+--------------------+

|(3,[0,1,2],[1.0,1...|

|(3,[0,1,2],[2.0,2...|

+--------------------+

## 5.3底层MLLIB文本评论数据提取TFIDF特征

import org.apache.spark.SparkContext

/\*\*

\* A simple Spark app in Scala

\*/

object TFIDFExtraction {

var rareTokens = Set("")

val stopwords = Set(

"the","a","an","of","or","in","for","by","on","but", "is", "not", "with", "as", "was", "if",

"they", "are", "this", "and", "it", "have", "from", "at", "my", "be", "that", "to"

)

val regex = """[^0-9]\*""".r

def main(args: Array[String]) {

val sc = new SparkContext("local[2]", "First Spark App")

val path = "../data/20news-bydate-train/\*"

val rdd = sc.wholeTextFiles(path)

// count the number of records in the dataset

println(rdd.count)

/\*

...

11314

\*/

println(rdd.first())

val newsgroups = rdd.map { case (file, text) => file.split("/").takeRight(2).head }

println(newsgroups.first())

val countByGroup = newsgroups.map(n => (n, 1)).reduceByKey(\_ + \_).collect.sortBy(-\_.\_2).mkString("\n")

println(countByGroup)

/\*

(rec.sport.hockey,600)

(soc.religion.christian,599)

(rec.motorcycles,598)

(rec.sport.baseball,597)

(sci.crypt,595)

(rec.autos,594)

(sci.med,594)

(comp.windows.x,593)

(sci.space,593)

(sci.electronics,591)

(comp.os.ms-windows.misc,591)

(comp.sys.ibm.pc.hardware,590)

(misc.forsale,585)

(comp.graphics,584)

(comp.sys.mac.hardware,578)

(talk.politics.mideast,564)

(talk.politics.guns,546)

(alt.atheism,480)

(talk.politics.misc,465)

(talk.religion.misc,377)

\*/

val text = rdd.map { case (file, text) => text }

val whiteSpaceSplit = text.flatMap(t => t.split(" ").map(\_.toLowerCase))

println(whiteSpaceSplit.distinct.count)

// 402978

// split text on any non-word tokens

val nonWordSplit = text.flatMap(t => t.split("""\W+""").map(\_.toLowerCase))

println(nonWordSplit.distinct.count)

// 130126

// inspect a look at a sample of tokens

println(nonWordSplit.distinct.sample(true, 0.3, 50).take(100).mkString(","))

// filter out numbers

val filterNumbers = nonWordSplit.filter(token => regex.pattern.matcher(token).matches)

println(filterNumbers.distinct.count)

// 84912

println(filterNumbers.distinct.sample(true, 0.3, 50).take(100).mkString(","))

/\*

reunion,wuair,schwabam,eer,silikian,fuller,sloppiness,crying,crying,beckmans,leymarie,fowl,husky,rlhzrlhz,ignore,

loyalists,goofed,arius,isgal,dfuller,neurologists,robin,jxicaijp,majorly,nondiscriminatory,akl,sively,adultery,

urtfi,kielbasa,ao,instantaneous,subscriptions,collins,collins,za\_,za\_,jmckinney,nonmeasurable,nonmeasurable,

seetex,kjvar,dcbq,randall\_clark,theoreticians,theoreticians,congresswoman,sparcstaton,diccon,nonnemacher,

arresed,ets,sganet,internship,bombay,keysym,newsserver,connecters,igpp,aichi,impute,impute,raffle,nixdorf,

nixdorf,amazement,butterfield,geosync,geosync,scoliosis,eng,eng,eng,kjznkh,explorers,antisemites,bombardments,

abba,caramate,tully,mishandles,wgtn,springer,nkm,nkm,alchoholic,chq,shutdown,bruncati,nowadays,mtearle,eastre,

discernible,bacteriophage,paradijs,systematically,rluap,rluap,blown,moderates

\*/

// examine potential stopwords

val tokenCounts = filterNumbers.map(t => (t, 1)).reduceByKey(\_ + \_)

val oreringDesc = Ordering.by[(String, Int), Int](\_.\_2)

println(tokenCounts.top(20)(oreringDesc).mkString("\n"))

/\*

(the,146532)

(to,75064)

(of,69034)

(a,64195)

(ax,62406)

(and,57957)

(i,53036)

(in,49402)

(is,43480)

(that,39264)

(it,33638)

(for,28600)

(you,26682)

(from,22670)

(s,22337)

(edu,21321)

(on,20493)

(this,20121)

(be,19285)

(t,18728)

\*/

// filter out stopwords

val tokenCountsFilteredStopwords = tokenCounts.filter { case (k, v) => !stopwords.contains(k) }

println(tokenCountsFilteredStopwords.top(20)(oreringDesc).mkString("\n"))

/\*

(ax,62406)

(i,53036)

(you,26682)

(s,22337)

(edu,21321)

(t,18728)

(m,12756)

(subject,12264)

(com,12133)

(lines,11835)

(can,11355)

(organization,11233)

(re,10534)

(what,9861)

(there,9689)

(x,9332)

(all,9310)

(will,9279)

(we,9227)

(one,9008)

\*/

// filter out tokens less than 2 characters

val tokenCountsFilteredSize = tokenCountsFilteredStopwords.filter { case (k, v) => k.size >= 2 }

println(tokenCountsFilteredSize.top(20)(oreringDesc).mkString("\n"))

/\*

(ax,62406)

(you,26682)

(edu,21321)

(subject,12264)

(com,12133)

(lines,11835)

(can,11355)

(organization,11233)

(re,10534)

(what,9861)

(there,9689)

(all,9310)

(will,9279)

(we,9227)

(one,9008)

(would,8905)

(do,8674)

(he,8441)

(about,8336)

(writes,7844)

\*/

// examine tokens with least occurrence

val oreringAsc = Ordering.by[(String, Int), Int](-\_.\_2)

println(tokenCountsFilteredSize.top(20)(oreringAsc).mkString("\n"))

/\*

(lennips,1)

(bluffing,1)

(preload,1)

(altina,1)

(dan\_jacobson,1)

(vno,1)

(actu,1)

(donnalyn,1)

(ydag,1)

(mirosoft,1)

(xiconfiywindow,1)

(harger,1)

(feh,1)

(bankruptcies,1)

(uncompression,1)

(d\_nibby,1)

(bunuel,1)

(odf,1)

(swith,1)

(lantastic,1)

\*/

// filter out rare tokens with total occurence < 2

rareTokens = tokenCounts.filter{ case (k, v) => v < 2 }.map { case (k, v) => k }.collect.toSet

val tokenCountsFilteredAll = tokenCountsFilteredSize.filter { case (k, v) => !rareTokens.contains(k) }

println(tokenCountsFilteredAll.top(20)(oreringAsc).mkString("\n"))

/\*

(sina,2)

(akachhy,2)

(mvd,2)

(hizbolah,2)

(wendel\_clark,2)

(sarkis,2)

(purposeful,2)

(feagans,2)

(wout,2)

(uneven,2)

(senna,2)

(multimeters,2)

(bushy,2)

(subdivided,2)

(coretest,2)

(oww,2)

(historicity,2)

(mmg,2)

(margitan,2)

(defiance,2)

\*/

println(tokenCountsFilteredAll.count)

// 51801

// create a function to tokenize each document

// check that our tokenizer achieves the same result as all the steps above

println(text.flatMap(doc => tokenize(doc)).distinct.count)

// 51801

// tokenize each document

var tokens = text.map(doc => tokenize(doc))

println(tokens.first.take(20))

println(text.flatMap(doc => tokenize(doc)).distinct.count)

import org.apache.spark.mllib.linalg.{ SparseVector => SV }

import org.apache.spark.mllib.feature.HashingTF

import org.apache.spark.mllib.feature.IDF

// set the dimensionality of TF-IDF vectors to 2^18

val dim = math.pow(2, 18).toInt

val hashingTF = new HashingTF(dim)

val tf = hashingTF.transform(tokens)

// cache data in memory

tf.cache

val v = tf.first.asInstanceOf[SV]

println(v.size)

// 262144

println(v.values.size)

// 706

println(v.values.take(10).toSeq)

// WrappedArray(1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 2.0, 1.0, 1.0)

println(v.indices.take(10).toSeq)

// WrappedArray(313, 713, 871, 1202, 1203, 1209, 1795, 1862, 3115, 3166)

val idf = new IDF().fit(tf)

val tfidf = idf.transform(tf)

val v2 = tfidf.first.asInstanceOf[SV]

println(v2.values.size)

// 706

println(v2.values.take(10).toSeq)

// WrappedArray(2.3869085659322193, 4.670445463955571, 6.561295835827856, 4.597686109673142, ...

println(v2.indices.take(10).toSeq)

// WrappedArray(313, 713, 871, 1202, 1203, 1209, 1795, 1862, 3115, 3166)

val minMaxVals = tfidf.map { v =>

val sv = v.asInstanceOf[SV]

(sv.values.min, sv.values.max)

}

val globalMinMax = minMaxVals.reduce { case ((min1, max1), (min2, max2)) =>

(math.min(min1, min2), math.max(max1, max2))

}

println(globalMinMax)

// (0.0,66155.39470409753)

// test out a few common words

val common = sc.parallelize(Seq(Seq("you", "do", "we")))

val tfCommon = hashingTF.transform(common)

val tfidfCommon = idf.transform(tfCommon)

val commonVector = tfidfCommon.first.asInstanceOf[SV]

println(commonVector.values.toSeq)

// WrappedArray(0.9965359935704624, 1.3348773448236835, 0.5457486182039175)

// test out a few uncommon words

val uncommon = sc.parallelize(Seq(Seq("telescope", "legislation", "investment")))

val tfUncommon = hashingTF.transform(uncommon)

val tfidfUncommon = idf.transform(tfUncommon)

val uncommonVector = tfidfUncommon.first.asInstanceOf[SV]

println(uncommonVector.values.toSeq)

val hockeyText = rdd.filter { case (file, text) => file.contains("hockey") }

// note that the 'transform' method used below is the one which works on a single document

// in the form of a Seq[String], rather than the version which works on an RDD of documents

val hockeyTF = hockeyText.mapValues(doc => hashingTF.transform(tokenize(doc)))

val hockeyTfIdf = idf.transform(hockeyTF.map(\_.\_2))

// compute cosine similarity using Breeze

import breeze.linalg.\_

val hockey1 = hockeyTfIdf.sample(true, 0.1, 42).first.asInstanceOf[SV]

val breeze1 = new SparseVector(hockey1.indices, hockey1.values, hockey1.size)

val hockey2 = hockeyTfIdf.sample(true, 0.1, 43).first.asInstanceOf[SV]

val breeze2 = new SparseVector(hockey2.indices, hockey2.values, hockey2.size)

val cosineSim = breeze1.dot(breeze2) / (norm(breeze1) \* norm(breeze2))

println(cosineSim)

// 0.06700095047242809

val graphicsText = rdd.filter { case (file, text) => file.contains("comp.graphics") }

val graphicsTF = graphicsText.mapValues(doc => hashingTF.transform(tokenize(doc)))

val graphicsTfIdf = idf.transform(graphicsTF.map(\_.\_2))

val graphics = graphicsTfIdf.sample(true, 0.1, 42).first.asInstanceOf[SV]

val breezeGraphics = new SparseVector(graphics.indices, graphics.values, graphics.size)

val cosineSim2 = breeze1.dot(breezeGraphics) / (norm(breeze1) \* norm(breezeGraphics))

println(cosineSim2)

// 0.001950124251275256

// compare to sport.baseball topic

val baseballText = rdd.filter { case (file, text) => file.contains("baseball") }

val baseballTF = baseballText.mapValues(doc => hashingTF.transform(tokenize(doc)))

val baseballTfIdf = idf.transform(baseballTF.map(\_.\_2))

val baseball = baseballTfIdf.sample(true, 0.1, 42).first.asInstanceOf[SV]

val breezeBaseball = new SparseVector(baseball.indices, baseball.values, baseball.size)

val cosineSim3 = breeze1.dot(breezeBaseball) / (norm(breeze1) \* norm(breezeBaseball))

println(cosineSim3)

// 0.0013298577308832765

sc.stop()

}

def tokenize(line: String): Seq[String] = {

line.split("""\W+""")

.map(\_.toLowerCase)

.filter(token => regex.pattern.matcher(token).matches)

.filterNot(token => stopwords.contains(token))

.filterNot(token => rareTokens.contains(token))

.filter(token => token.size >= 2)

.toSeq

}

}

## 5.4用户画像关键词ML文本数据TFIDF特征

TFIDF关键词抽取，将抽取后的数据转化为libsvm的数据

package cn.itcaat.sparkmllibtest.features.documentType  
  
import org.apache.spark.SparkContext  
import org.apache.spark.mllib.classification.NaiveBayes  
import org.apache.spark.mllib.evaluation.MulticlassMetrics  
import org.apache.spark.mllib.feature.{HashingTF, IDF}  
import org.apache.spark.mllib.linalg.{SparseVector => SV}  
import org.apache.spark.mllib.regression.LabeledPoint  
import org.apache.spark.mllib.util.MLUtils  
//import org.apache.spark.ml.feature.HashingTF  
//import org.apache.spark.ml.feature.IDF  
  
/\*\*  
 \* A simple Spark app in Scala  
 \*/  
object DocumentClassification {  
  
 def main(args: Array[String]) {  
 //1-准备环境  
 val sc = new SparkContext("local[2]", "First Spark App")  
 val path = "D:\\BigData\\Workspace\\SparkMachineLearningTest\\SparkMllib\_BigDataSH16\\src\\main\\scala\\data\\20news-bydate-train\\\*"  
 // val path = "../data/20news-bydate-train/\*"  
 //2-使用sc的wholeTextFile的类型读取数据，使用的是hadoop的CombineInputFormat的数据，会将小文件合并  
 val rdd = sc.wholeTextFiles(path)  
 val text = rdd.map { case (file, text) => text }  
 val newsgroups = rdd.map { case (file, text) => file.split("/").takeRight(2).head }  
 val newsgroupsMap = newsgroups.distinct.collect().zipWithIndex.toMap  
 val dim = math.pow(2, 18).toInt  
 val hashingTF = new HashingTF(dim)  
  
 //分词  
 var tokens = text.map(doc => TFIDFExtraction.tokenize(doc))  
 val tf = hashingTF.transform(tokens)  
 tf.cache  
 val v = tf.first.asInstanceOf[SV]  
  
 //逆文档频率  
 val idf = new IDF().fit(tf)  
 val tfidf = idf.transform(tf)  
 val zipped = newsgroups.zip(tfidf)  
 println(zipped.first())  
 val train = zipped.map { case (topic, vector) => {  
 //libsvm的数据类型===LabeledPoint标签向量===形成libsvm的数据  
 LabeledPoint(newsgroupsMap(topic), vector)  
 }  
 }  
 //保存成libsvm的数据格式  
 //TODO uncomment to generate libsvm format  
 MLUtils.saveAsLibSVMFile(train, "./output/20news-by-date-train-libsvm")  
  
 train.cache  
 val model = NaiveBayes.train(train, lambda = 0.1)  
  
// val testPath = "../data/20news-bydate-test/\*"  
 val testRDD = sc.wholeTextFiles(path = path)  
 val testLabels = testRDD.map { case (file, text) =>  
 val topic = file.split("/").takeRight(2).head  
 newsgroupsMap(topic)  
 }  
 val testTf = testRDD.map { case (file, text) => hashingTF.transform(TFIDFExtraction.tokenize(text)) }  
 val testTfIdf = idf.transform(testTf)  
 val zippedTest = testLabels.zip(testTfIdf)  
 val test = zippedTest.map { case (topic, vector) => {  
 println(topic)  
 println(vector)  
 LabeledPoint(topic, vector)  
 }  
 }  
  
 //TODO uncomment to generate libsvm format  
 MLUtils.saveAsLibSVMFile(test, "./output/20news-by-date-test-libsvm")  
  
  
 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()  
 println(accuracy)  
 //0.7928836962294211  
 val metrics = new MulticlassMetrics(predictionAndLabel)  
 println(metrics.accuracy)  
 println(metrics.weightedFalsePositiveRate)  
 println(metrics.weightedPrecision)  
 println(metrics.weightedFMeasure)  
 println(metrics.weightedRecall)  
 //0.7822644376431702  
  
 val rawTokens = rdd.map { case (file, text) => text.split(" ") }  
 val rawTF = rawTokens.map(doc => hashingTF.transform(doc))  
 val rawTrain = newsgroups.zip(rawTF).map { case (topic, vector) => LabeledPoint(newsgroupsMap(topic), vector) }  
 val rawModel = NaiveBayes.train(rawTrain, lambda = 0.1)  
 val rawTestTF = testRDD.map { case (file, text) => hashingTF.transform(text.split(" ")) }  
 val rawZippedTest = testLabels.zip(rawTestTF)  
 val rawTest = rawZippedTest.map { case (topic, vector) => LabeledPoint(topic, vector) }  
 val rawPredictionAndLabel = rawTest.map(p => (rawModel.predict(p.features), p.label))  
 val rawAccuracy = 1.0 \* rawPredictionAndLabel.filter(x => x.\_1 == x.\_2).count() / rawTest.count()  
 println(rawAccuracy)  
 // 0.7661975570897503  
 val rawMetrics = new MulticlassMetrics(rawPredictionAndLabel)  
 println(rawMetrics.weightedFMeasure)  
 // older value 0.7628947184990661  
 // dec 2016 : 0.7653320418573546  
  
 sc.stop()  
 }  
  
}

## 5.5NativeBayes情感分类

### 5.5.1SparkMllib版本

import org.apache.spark.mllib.classification.NaiveBayes

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

import org.apache.spark.mllib.linalg.DenseVector

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

//import org.apache.spark.ml.classification.NaiveBayes

//import org.apache.spark.mllib.feature.{HashingTF, IDF}

import org.apache.spark.ml.feature.{HashingTF, IDF, Tokenizer}

import org.apache.spark.ml.linalg.{SparseVector => SV2}

import org.apache.spark.mllib.linalg.{SparseVector => SV}

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

import org.apache.spark.rdd.RDD

import org.apache.spark.sql.SparkSession

/\*\*

\* A simple Spark app in Scala

\*/

object DocumentClassificationArchive {

case class MyClass(label: Double, text: String)

def main(args: Array[String]) {

val sc = new SparkContext("local[2]", "First Spark App")

val spConfig = (new SparkConf).setMaster("local").setAppName("SparkApp")

val spark = SparkSession

.builder()

.appName("SparkRatingData").config(spConfig)

.getOrCreate()

import spark.implicits.\_

val path = "../data/20news-bydate-sample/\*"

val rdd = sc.wholeTextFiles(path)

val text = rdd.map { case (file, text) => text }

val newsgroups = rdd.map { case (file, text) => file.split("/").takeRight(2).head }

println(newsgroups.first())

val newsgroupsMap = newsgroups.distinct.collect().zipWithIndex.toMap

sc.broadcast(newsgroupsMap)

val textModified = rdd.map { case (file, text) =>

{

val labelText = file.split("/").takeRight(2).head

val label = newsgroupsMap.get(labelText).get

val y = (label.toDouble, text)

y

}

}

val df = textModified.map(

r => MyClass(r.\_1, r.\_2)).toDF("label","text")

df.show()

val tokenizer = new Tokenizer().setInputCol("text").setOutputCol("words")

val wordsData = tokenizer.transform(df)

wordsData.show()

val hashingTF = new HashingTF()

.setInputCol("words").setOutputCol("rawFeatures").setNumFeatures(20)

val featurizedData = hashingTF.transform(wordsData).select("label", "rawFeatures")

val idf = new IDF().setInputCol("rawFeatures").setOutputCol("features")

val idfModel = idf.fit(featurizedData)

val rescaledData = idfModel.transform(featurizedData)

//println(zippedTest.first())

//val rescaledLabelPoint = zippedTest

val trainingData = rescaledData.select("label","features").rdd.map( row => {

println(row.getDouble(0))

println(row)

println(row(1))

LabeledPoint(row.getDouble(0), {

val x = row(1).asInstanceOf[SV2]

val xDense = x.toDense

val a = xDense.toArray

val b = new DenseVector(a)

b

})

})

val testData = getRescaledTestData(sc, spark)

val model = NaiveBayes.train(trainingData, lambda = 0.01)

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

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

print(accuracy)

val metrics = new MulticlassMetrics(predictionAndLabel)

println(metrics.accuracy)

println(metrics.weightedFalsePositiveRate)

println(metrics.weightedPrecision)

println(metrics.weightedFMeasure)

println(metrics.weightedRecall)

sc.stop()

}

def getRescaledTestData(sc : SparkContext, spark : SparkSession) : RDD[LabeledPoint] = {

val path = "../data/20news-bydate-test/\*"

val rdd = sc.wholeTextFiles(path)

val text = rdd.map { case (file, text) => text }

val newsgroups = rdd.map { case (file, text) => file.split("/").takeRight(2).head }

println(newsgroups.first())

val newsgroupsMap = newsgroups.distinct.collect().zipWithIndex.toMap

sc.broadcast(newsgroupsMap)

val textModified = rdd.map {

case (file, text) => {

val labelText = file.split("/").takeRight(2).head

val label = newsgroupsMap.get(labelText).get

val y = (label.toDouble, text)

y

}

}

import spark.implicits.\_

val df = textModified.map(

r => MyClass(r.\_1, r.\_2)).toDF("label","text")

val tokenizer = new Tokenizer().setInputCol("text").setOutputCol("words")

val wordsData = tokenizer.transform(df)

val hashingTF = new HashingTF()

.setInputCol("words").setOutputCol("rawFeatures").setNumFeatures(20)

val featurizedData = hashingTF.transform(wordsData)

val idf = new IDF().setInputCol("rawFeatures").setOutputCol("features")

val idfModel = idf.fit(featurizedData)

val rescaledData = idfModel.transform(featurizedData)

val rescaledLabelPoint = rescaledData.select("label", "features").rdd.

map( row => LabeledPoint(row.getDouble(0), {

val x = row(1).asInstanceOf[SV2]

val xDense = x.toDense

val a = xDense.toArray

val b = new DenseVector(a)

b

}))

return rescaledLabelPoint

}

}

### 5.5.2SparkMl版本

import org.apache.spark.SparkConf

import org.apache.spark.ml.classification.NaiveBayes

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

import org.apache.spark.sql.SparkSession

object DocumentClassificationLibSVM {

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

val spConfig = (new SparkConf).setMaster("local").setAppName("SparkApp")

val spark = SparkSession

.builder()

.appName("SparkRatingData").config(spConfig)

.getOrCreate()

val data = spark.read.format("libsvm").load("./output/20news-by-date-train-libsvm/part-combined")

val Array(trainingData, testData) = data.randomSplit(Array(0.7, 0.3), seed = 1L)

// Train a NaiveBayes model.

val model = new NaiveBayes()

.fit(trainingData)

val predictions = model.transform(testData)

predictions.show()

val evaluator = new MulticlassClassificationEvaluator()

.setLabelCol("label")

.setPredictionCol("prediction")

.setMetricName("accuracy")

val accuracy = evaluator.evaluate(predictions)

println("Test set accuracy = " + accuracy)

spark.stop()

}

}

## 5.6用户画像主题词之Word2Vec模型

### 5.6.1SparkMllib版本

import org.apache.spark.SparkContext

import org.apache.spark.mllib.linalg.{SparseVector => SV}

/\*\*

\* A simple Spark app in Scala

\*/

object Word2VecMllib {

def main(args: Array[String]) {

val sc = new SparkContext("local[2]", "Word2Vector App")

val path = "./data/20news-bydate-train/alt.atheism/\*"

val rdd = sc.wholeTextFiles(path)

val text = rdd.map { case (file, text) => text }

val newsgroups = rdd.map { case (file, text) => file.split("/").takeRight(2).head }

val newsgroupsMap = newsgroups.distinct.collect().zipWithIndex.toMap

val dim = math.pow(2, 18).toInt

var tokens = text.map(doc => TFIDFExtraction.tokenize(doc))

import org.apache.spark.mllib.feature.Word2Vec

val word2vec = new Word2Vec()

//word2vec.setSeed(42) // we do this to generate the same results each time

val word2vecModel = word2vec.fit(tokens)

word2vecModel.findSynonyms("philosophers", 5).foreach(println)

sc.stop()

}

}

### 5.6.2SparkMl版本

import org.apache.spark.{SparkConf}

import org.apache.spark.ml.feature.Word2Vec

import org.apache.spark.sql.SparkSession

object Word2VecMl {

case class Record(name: String)

def main(args: Array[String]) {

val spConfig = (new SparkConf).setMaster("local").setAppName("SparkApp")

val spark = SparkSession

.builder

.appName("Word2Vec Sample").config(spConfig)

.getOrCreate()

import spark.implicits.\_

val rawDF = spark.sparkContext

.wholeTextFiles("./data/20news-bydate-train/alt.atheism/\*")

val temp = rawDF.map( x => {

(x.\_2.filter(\_ >= ' ').filter(! \_.toString.startsWith("(")) )

})

val textDF = temp.map(x => x.split(" ")).map(Tuple1.apply)

.toDF("text")

print(textDF.first())

val word2Vec = new Word2Vec()

.setInputCol("text")

.setOutputCol("result")

.setVectorSize(3)

.setMinCount(0)

val model = word2Vec.fit(textDF)

val result = model.transform(textDF)

result.select("result").take(3).foreach(println)

val ds = model.findSynonyms("philosophers", 5).select("word")

ds.rdd.saveAsTextFile("./output/alien-synonyms" + System.nanoTime())

ds.show()

spark.stop()

}

}

# 6.总结