# ML库特征选择总结

## 特征提取

### TF-IDF

#### TF-IDF介绍

TF-IDF是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。TF-IDF加权的各种形式常被搜索引擎应用，作为文件与用户查询之间相关程度的度量或评级。

#### TF-IDF使用

1. 首先创建数据集，假设数据集的schema如下：

StructType schema = new StructType(new StructField[]{

new StructField("label", DataTypes.DoubleType, false, Metadata.empty()),

new StructField("sentence", DataTypes.StringType, false, Metadata.empty())

});

1. 其次创建Tokenizer并且设置输入列 输出列，目的将输入列按照空格等方式分割开形成新的Dataset

setInputCol("sentence").setOutputCol("words")

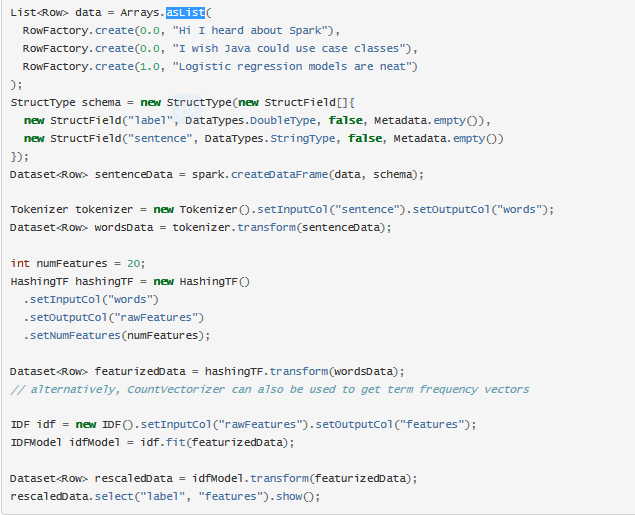
1. 接着创建HashingTF并且设置输入输出列及其

.setInputCol("words")

.setOutputCol("rawFeatures")

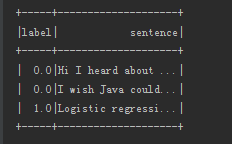
.setNumFeatures(numFeatures)

1. 接着可以使用HashingTF的transform得到TF数据集
2. 接着我们便可以调用IDF 然后调用fit得到模型
3. 然后模型调用其transform可以应用其他数据上面

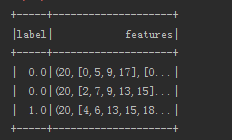


#### 效果图

初始数据如下：



结果数据如下：



### Word2Vec

#### word2vec介绍

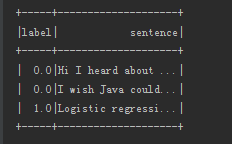
Word2Vec是用来计算单词的向量表示的方法，相似单词的表示在向量空间是更加相似的，我们通过训练的模型寻找与某个单词或者某个向量等最为接近的k个单词。

#### word2vec使用

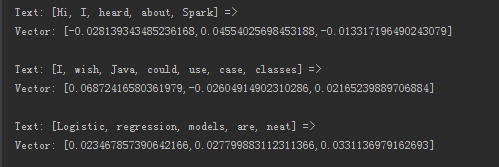


#### 效果图

初始数据如下：



数据结果如下：



### CountVectorizer

#### CountVectorizer介绍

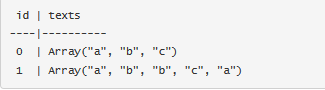
在没有字典库可用的情况下，通过这个类我们可以将一组文本文档集合转为令牌数量的向量

#### CountVectorizer使用

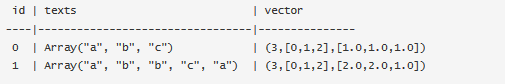


#### 1.3.3效果图

初始数据如下：



结果数据如下：



## 特征转换

### Tokenizer

#### Tokenize介绍

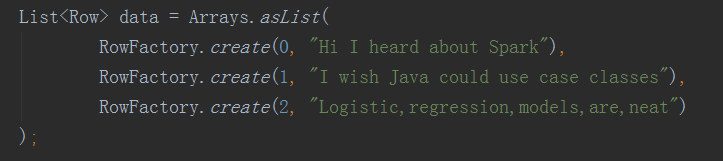
将一个句子按照一定的格式划分成单词，可以按照空格划分 按照逗号划分，根据自己需要而定。

#### Tokenize使用

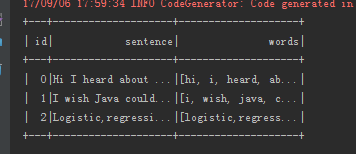


#### 2.1.3 效果图

初始数据集如下：



结果数据集如下：



### 2.2 StopWordsRemover

#### 2.2.1 StopWordsRemover介绍

这个类的作用是去掉一些停止词

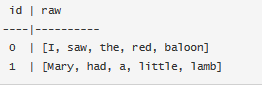
#### 2.2.2 StopWordsRemover使用



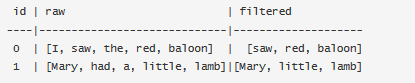
默认的是英语会自动加载英语的停止词然后根据输入进行过滤

#### 2.2.3 效果图

初始数据如下：



结果数据如下：



### 2.3 n-gram

#### 2.3.1 n-gram介绍

n-gram是将一个句子中的每个单词后面的n-1个单词组成然后包含该单词，组成一个由n个单词构成的一个n令牌

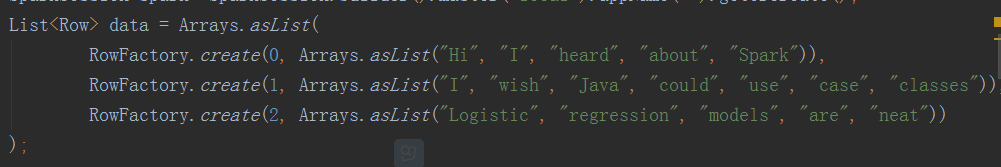
#### 2.3.2 n-gram使用

主要的是通过Ngram().setN()可以配置n-gram中的n是多少

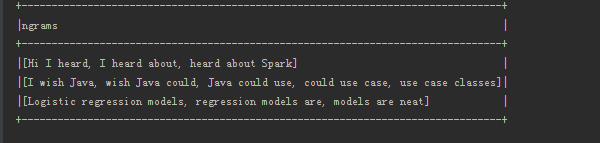


2.3.3 效果图

初始数据集：



结果数据集：



### 2.4 Binarizer

#### 2.4.1 Binarizer介绍

Binarizer用于二元分类，采用普遍使用的方法inputcol 和outputcol 另加一个threshold，大于threshold的将被分为一类 ，小于threshold的将被分为另一类

#### 2.4.2 Binarizer使用

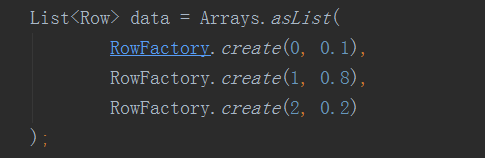
我们通过这些方法设置这个类的一些参数

Binarizer binarizer = new Binarizer()  
 .setInputCol("feature")  
 .setOutputCol("binarized\_feature")  
 .setThreshold(0.5);

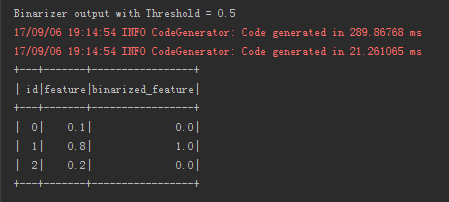


#### 2.4.3 效果图

初始数据如下：



结果数据如下：



### 2.5 PCA

#### 2.5.1 PCA介绍

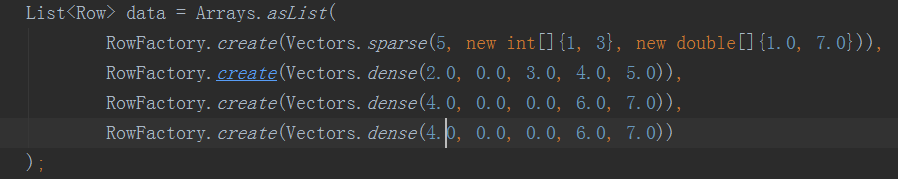
PCA（Principal Component Analysis）是一种常用的数据分析方法。PCA通过线性变换将原始数据变换为一组各维度线性无关的表示，可用于提取数据的主要特征分量，常用于高维数据的降维

#### 2.5.2 PCA使用

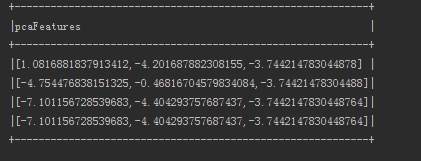


#### 2.5.3 效果图

初始数据集如下：



结果数据集：

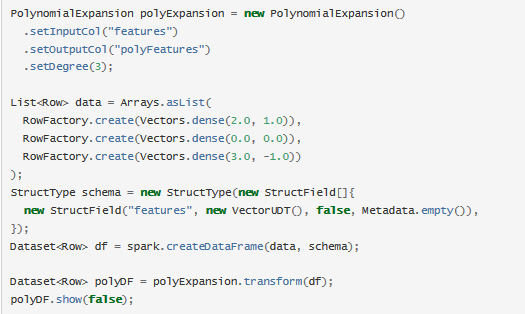


### 2.6 PolynomialExpansion

#### 2.6.1 PolynomialExpansion介绍

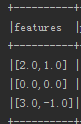
多项式扩展，通过设置setDegree方法设置多项式扩展的深度

#### 2.6.2 PolynomialExpansion使用

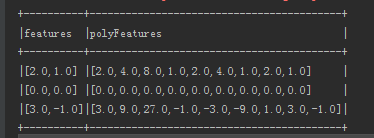


#### 2.6.3 效果图、

初始数据集如下：



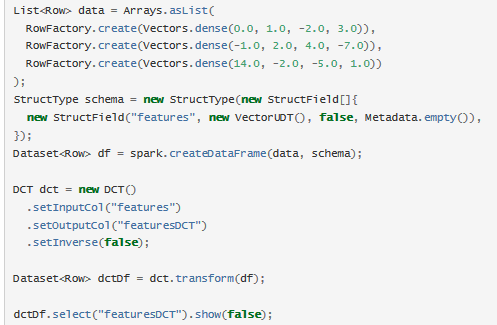
结果数据集如下：



### 2.7 Discrete Cosine Transform (DCT)

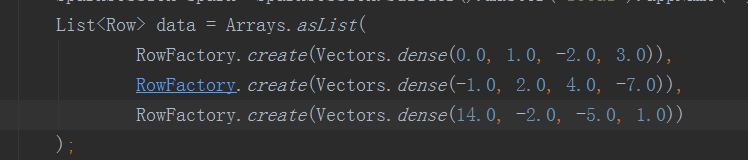
#### 2.7.1 DCT介绍

#### 2.7.2 DCT使用

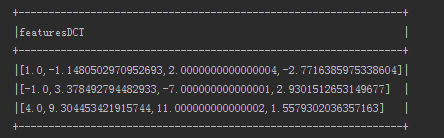


#### 2.7.3 示例

初始数据集如下：



结果数据如下：

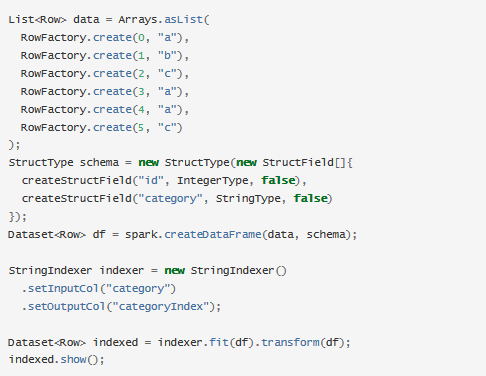


### 2.8 StringIndexer

#### 2.8.1 StringIndexer介绍

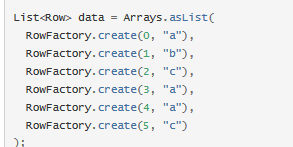
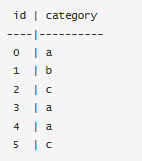
将一列转换为分类列，即将一列里面相同的字符串归纳为同一个标签值

#### 2.8.2 StringIndexer使用

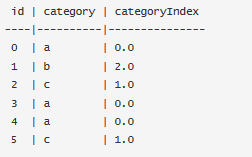


#### 2.8.3 效果图

初始数据如下：



结果数据如下：



### 2.9 IndexToString

#### 2.9.1 IndexToString介绍

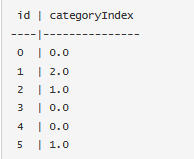
将归纳的索引转为原来对应的字符串

#### 2.9.2 IndexToString使用

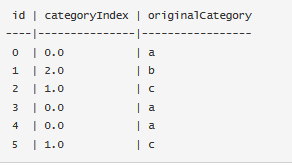


#### 2.9.3 效果图

初始数据：



结果数据：

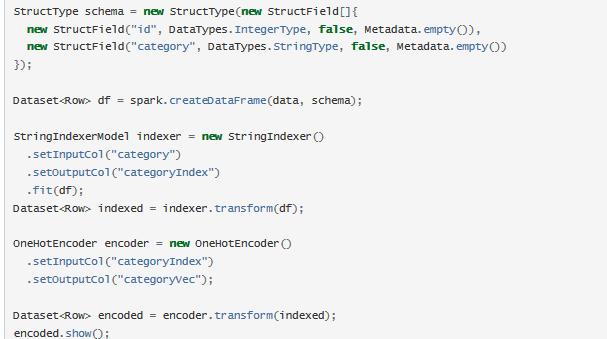


### 2.10 OneHotEncoder

#### 2.10.1 OneHotEncoder介绍

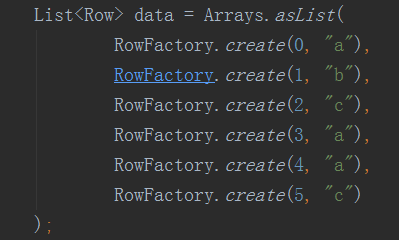
本类用于映射一个lable索引到一个二元向量

#### 2.10.2 OneHotEncoder使用

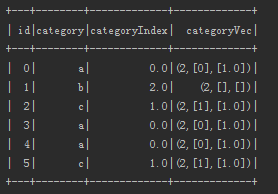


#### 2.10.3 效果图

初始数据图：



结果图如下：



### 2.11 VectorIndexer

#### 2.11.1 VectorIndexer介绍

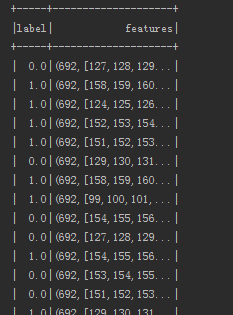
能够自动的划分哪些特征是分类型的并且将分类型的特征转为索引

#### 2.11.2 VectorIndexer使用

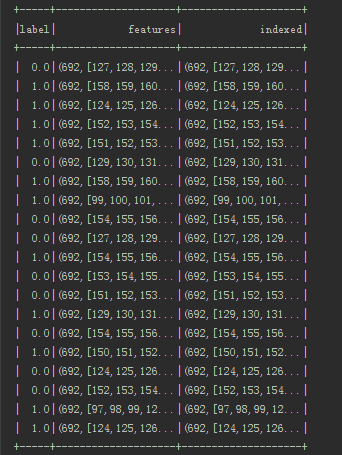


#### 2.11.3 效果图

初始数据如下，其中692的含义是一共有692个特征，后面的中括号里面是代表每一个样本包含了哪些特征：



结果数据如下：



### 2.12 Interaction

#### 2.12.1 Interaction介绍

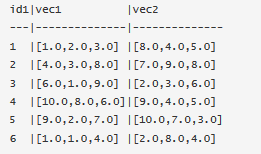
这个类的作用是类似于特征组合的效果，假设有我们将原始特征x1 x2封装到一个VectorAssembler，将x3,x4封装到另一个VectorAssembler中，那么通过Interaction我们可以得到x1x2 x1x3 x2x3 x2x4的组合

#### 2.12.2 Interaction使用

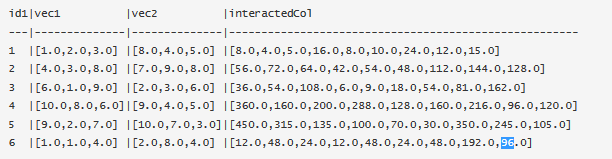


#### 2.12.3 效果图

原始数据集经过VectorAssembler处理后数据如下：



结果数据如下：



### 2.13 Normalizer

#### 2.13.1 Normalizer介绍

正则化的过程是将每个样本缩放到单位范数（每个样本的范数为1），spark里面默认范数为2，如果后面要使用如二次型（点积）或者其它核方法计算两个样本之间的相似性这个方法会很有用比如TF-IDF中使用L2正则。

Normalizer主要思想是对每个样本计算其p-范数，然后对该样本中每个元素除以该范数，这样处理的结果是使得每个处理后样本的p-范数（l1-norm,l2-norm）等于1。

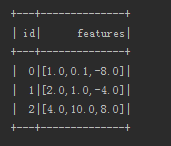
#### 2.13.2 Normalizer使用

通过setP方法我们可以设置范式级别

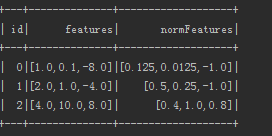


#### 2.13.3 效果图

原始数据如下：



结果数据如下：

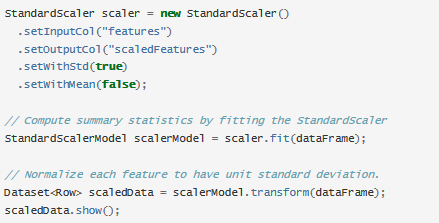


### 2.14 StandardScaler

#### 2.14.1 StandardScaler介绍

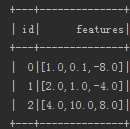
将数据标准化为均值为0标准差为1的数据，或者缩放到单位方差 或者用原始数据减去该列的均值

#### 2.14.2 StandardScaler使用

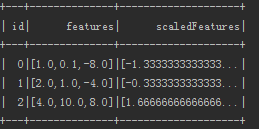


#### 2.14.3 效果图

原始如数如下：



只使用了均值的效果如下，第一列 1 2 4 平均值为2.333333 然后我们用1-2.3333 2-2.3333 4-2.33333 便得到了scaledFeature中的第一列数据：



### 2.15 MinMaxScaler

#### 2.15.1 MinMaxScaler介绍

将每一列数据获取该列最小值min(x) 最大值max(x) 然后进行下面操作进行标准化

max默认为1 min默认为0

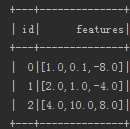
可以setMax setMin设置

#### 2.15.2 MinMaxScaler使用

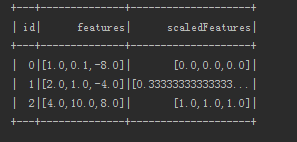


#### 2.15.3 效果图

原始如数如下：



结果数据如下：



### 2.16 MaxAbsScaler

#### 2.16.1 MaxAbsScaler介绍

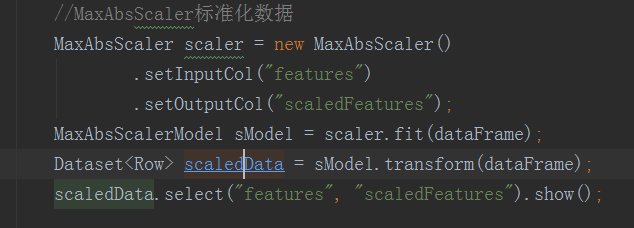
这个类用于数据的行处理，每个值分别除以每行中绝对值最大的数，例如一行数据是

1 2 -3

然后-3的绝对值最大为3

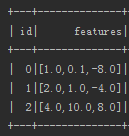
然后我们用1/3 2/3 -3/3 得到最后结果，每个数范围为[-1,1]

#### 2.16.2 MaxAbsScaler使用

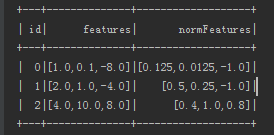


#### 2.16.3 效果图

初始数据如下：



结果数据如下:



### 2.17 Bucketizer

#### 2.17.1 Bucketizer介绍

这个类的作用是将连续特征划分到不同的区间段中也就是bin或者bucket

#### 2.17.2 Bucketizer使用

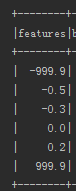
我们可以通过自己创建一个double数组然后放入分割点，比如下面的-0.5 0.0 0.5

然后会把数据小于-0.5的分为一类，然后[-0.5,0)分为一类，[0,0.5)分为一类，[0.5,正无穷)分为一类

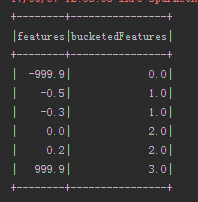


#### 2.17.3 效果图

初始数据：



结果数据：



### 2.18 ElementwiseProduct

#### 2.18.1 ElementwiseProduct介绍

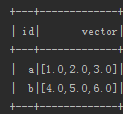
将一个向量乘以另一个向量对应位置相乘得打新的向量

#### 2.18.2 ElementwiseProduct使用

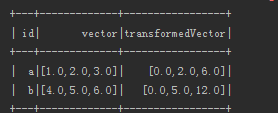


#### 2.18.3 效果图

原始数据：



结果数据：

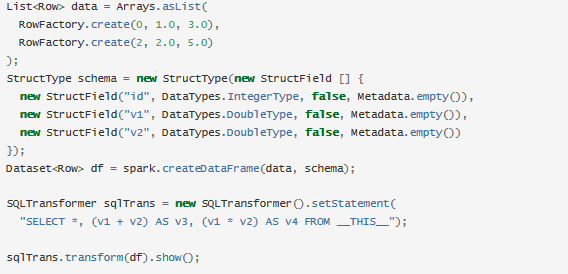


### 2.19 SQLTransformer

#### 2.19.1 SQLTransformer介绍

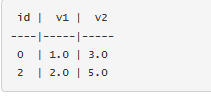
SQLTransformer工具用来转换由SQL定义的陈述。目前仅支持SQL语法如"SELECT ...FROM \_\_THIS\_\_ ..."，其中"\_\_THIS\_\_"代表输入数据的基础表。选择语句指定输出中展示的字段、元素和表达式，支持Spark SQL中的所有选择语句。用户可以基于选择结果使用Spark SQL建立方程或者用户自定义函数

#### 2.19.2 使用

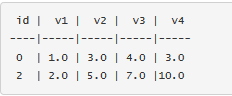


#### 2.19.3 效果图

输入数据：



输出数据：



### 2.20 VectorAssembler

#### 2.20.1介绍

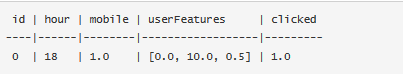
VectorAssembler是一个转换器，它将给定的若干列合并为一列向量。它可以将原始特征和一系列通过其他转换器得到的特征合并为单一的特征向量，来训练如逻辑回归和决策树等机器学习算法。VectorAssembler可接受的输入列类型：数值型、布尔型、向量型。输入列的值将按指定顺序依次添加到一个新向量中。

#### 2.20.2 使用

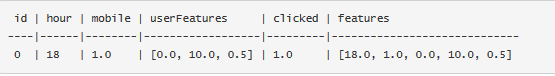


#### 2.20.3效果图

输入集合



输出集合



### 2.21 QuantileDiscretizer

#### 2.21.1介绍

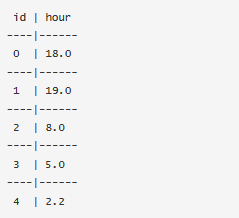
QuantileDiscretizer讲连续型特征转换为分级类别特征。分级的数量由numBuckets参数决定。分级的范围有渐进算法决定。渐进的精度由relativeError参数决定。当relativeError设置为0时，将会计算精确的分位点（计算代价较高）。分级的上下边界为负无穷到正无穷，覆盖所有的实数值。

#### 2.21.2使用

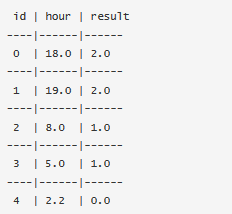


#### 2.21.3效果图

输入列集



输出列集



## 特征选择

### 3.1 VectorSlicer

#### 3.1.1介绍

VectorSlicer是一个转换器输入特征向量，输出原始特征向量子集。VectorSlicer接收带有特定索引的向量列，通过对这些索引的值进行筛选得到新的向量集，可以接受两种索引：

1整数索引既向量中特征的索引，使用setIndices设置，

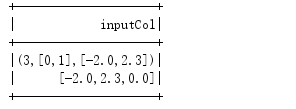
2字符串索引---代表向量中特征的名字，要求向量列有AttributeGroup，因为这根据Attribute来匹配名字字段。

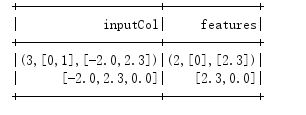
#### 3.1.2使用



#### 3.1.3效果图

输入集合：



输出集合：  


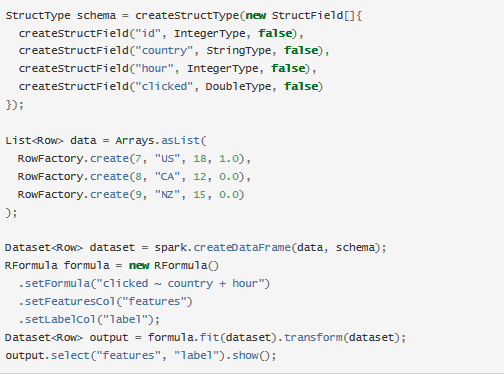
### 3.2 Rformula

#### 3.2.1介绍

RFormula通过R模型公式来选择列。支持R操作中的部分操作包括‘~’, ‘.’, ‘:’, ‘+’以及‘-‘

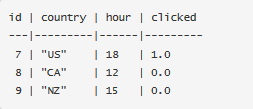
RFormula产生一个向量特征列以及一个double或者字符串标签列。如果用R进行线性回归，则对String类型的输入列进行one-hot编码、对数值型的输入列进行double类型转化。如果类别列是字符串类型，它将通过StringIndexer转换为double类型。如果标签列不存在，则输出中将通过规定的响应变量创造一个标签列。

#### 3.2.2使用

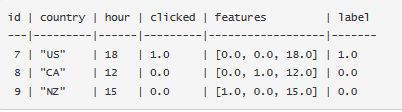


#### 3.2.3效果

输入列集



输出列集



### 3.3 ChiSqSelector

#### 3.3.1介绍

ChiSqSelector代表卡方特征选择。它适用于带有类别特征的标签数据。

ChiSqSelector根据独立卡方检验，然后选取类别标签主要依赖的特征。它类似于选取最有预测能力的特征。它支持三种特征选取方法：

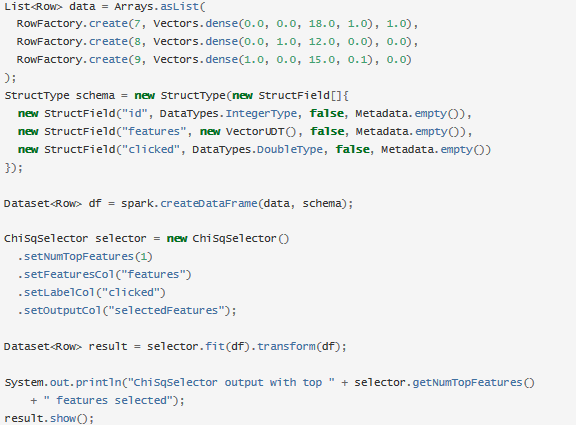
1、numTopFeatures：通过卡方检验选取最具有预测能力的Top(num)个特征；

2、percentile：类似于上一种方法，但是选取一小部分特征而不是固定(num)个特征；

3、fpr:选择P值低于门限值的特征，这样就可以控制false positive rate来进行特征选择；

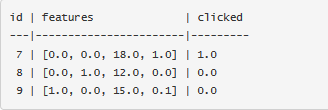
        默认情况下特征选择方法是numTopFeatures(50)，可以根据setSelectorType()选择特征选取方法。

#### 3.3.2使用

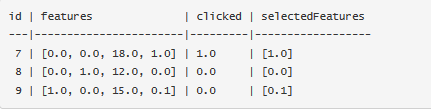


#### 3.3.3效果图

输入列集



输出列集



## 局部敏感哈希

LSH的基本思想是：将原始数据空间中的两个相邻数据点通过相同的映射或投影变换（projection）后，这两个数据点在新的数据空间中仍然相邻的概率很大，而不相邻的数据点被映射到同一个桶的概率很小。也就是说，如果我们对原始数据进行一些hash映射后，我们希望原先相邻的两个数据能够被hash到相同的桶内，具有相同的桶号。

那具有怎样特点的hash functions才能够使得原本相邻的两个数据点经过hash变换后会落入相同的桶内？这些hash function需要满足以下两个条件：

1）如果d(x,y) ≤ d1， 则h(x) = h(y)的概率至少为p1；

2）如果d(x,y) ≥ d2， 则h(x) = h(y)的概率至多为p2；

其中d(x,y)表示x和y之间的距离，d1 < d2， h(x)和h(y)分别表示对x和y进行hash变换。

### 4.1 MinHashLSH

#### 4.1.1 MinHashLSH介绍

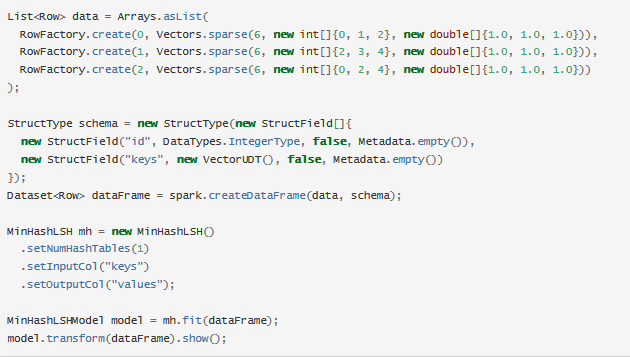
输入是自然数的集合，Jaccard distance被定义如下：



然后采用一个随机哈希函数g到这个集合中的每一个元素中，然后采用所有哈希值最小的

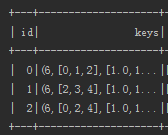


#### 4.1.2 MinHashLSH使用

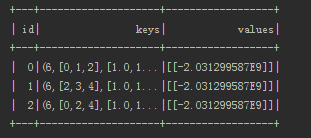


#### 4.1.3 效果图

初始数据如下：



结果数据如下：



### 4.2 Bucketed Random Projection for Euclidean Distance

#### 4.2.1 介绍

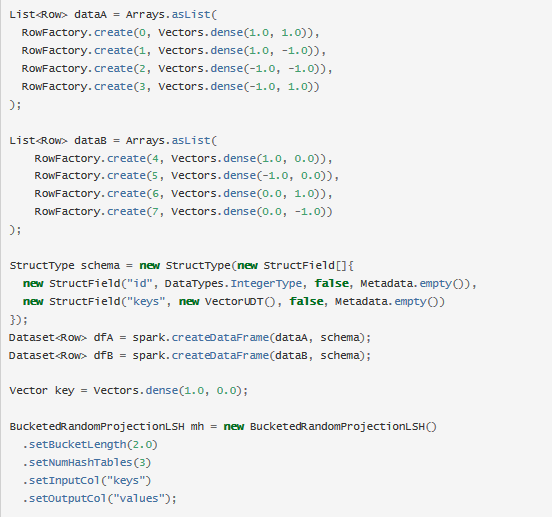
采用欧几里得方法：



将向量x映射到一个随机单位向量v上，并且根据映射结果放到相应的桶中

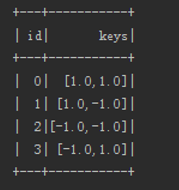


#### 4.2.2 使用

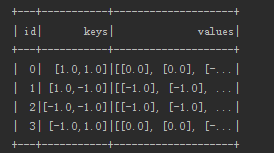


#### 4.2.3 效果图

初始数据：



BucketedRandomProjectionLSHModel调用 transform(dfA)后数据如下：



### 4.3 Approximate Similarity Join

#### 4.3.1 介绍

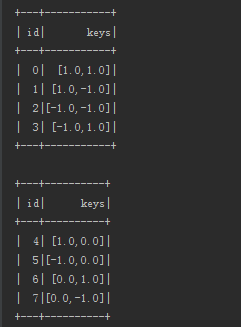
Approximate Similarity Join一般用于两个数据集的操作，返回的形式是一对一对形式，返回的结果是小于用户设置的阈值的数据

#### 4.3.2 使用

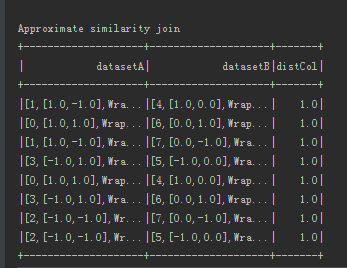


#### 4.3.3 效果图

两个数据集如下：



调用approxSimilarityJoin(dfA, dfB, 1.5)



### 4.4 Approximate Nearest Neighbor Search

#### 4.4.1 介绍

寻找与某个向量最近的k个向量出来，在已经训练的BucketedRandomProjectionLSHModel模型基础上操作，BucketedRandomProjectionLSHModel继承与LSHModel

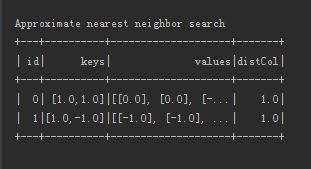
#### 4.4.2 使用



#### 4.4.3 效果

Vector key = Vectors.dense(1.0, 0.0);

然后调用approxNearestNeighbors(dfA, key, 2)，2是寻找这个key表示的向量近似最近邻查找两个出来，在数据集dfA里面查找



如果approxNearestNeighbors(dfA, key, 3)结果如下

