9.6 特征提取,转换和选择及局部敏感哈希



9.6.1 特征提取(Feature Extraction)

9.6.2 特征转换(Feature Transformation)

9.6.3 特征选择(Feature Filtering)

9.6.4 局部敏感哈希(Locality-Sensitive Hashing (LSH))

9.6.1 特征提取



1. 特征提取操作

spark.ml包提供的提取操作包括以下几种。

（1）TF-IDF
（2）Word2Vec
（3）CountVectorizer
（4）FeatureHasher

9.6.1 特征提取



2．特征提取的例子

- “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。

- TF-IDF就是在数值化文档信息，衡量词语能提供多少信息以区分文档。其定义如下：

$$IDF(t,D) = log\frac{|D|+1}{DF(t,D)+1}$$

- TF-IDF 度量值表示如下：

$$TFIDF(t,d,D) = TF(t,d) \cdot IDF(t,D)$$

9.6.1 特征提取



在Spark ML库中，TF-IDF被分成两部分：

- TF (+hashing)
- IDF

•**TF**: HashingTF 是一个Transformer，在文本处理中，接收词条的集合然后把这些集合转化成固定长度的特征向量。这个算法在哈希的同时会统计各个词条的词频。

•**IDF**: IDF是一个Estimator，在一个数据集上应用它的fit()方法，产生一个IDFModel。 该IDFModel 接收特征向量（由HashingTF产生），然后计算每一个词在文档中出现的频次。IDF会减少那些在语料库中出现频率较高的词的权重。

9.6.1 特征提取



过程描述：

- 在下面的代码段中，我们以一组句子开始
- 首先使用分解器Tokenizer把句子划分为单个词语
- 对每一个句子（词袋），使用HashingTF将句子转换为特征向量
- 最后使用IDF重新调整特征向量（这种转换通常可以提高使用文本特征的性能）

9.6.1 特征提取



第1步：导入TF-IDF所需要的包。

```
>>> from pyspark.ml.feature import HashingTF, IDF, Tokenizer
>>> from pyspark.sql import SparkSession
```

第2步：创建一个集合，每一个句子代表一个文件。

```
# 创建一个 SparkSession（如果尚未创建）
>>> spark = SparkSession.builder.appName("example").getOrCreate()

# 创建数据集
>>> data = [
    (0, "I heard about Spark and I love Spark"),
  (0, "I wish Java could use case classes"),
    (1, "Logistic regression models are neat")
]
```

9.6.1 特征提取



```
# 将数据集转换为DataFrame
>>> sentence_data = spark.createDataFrame(data, ["label", "sentence"])

# 显示DataFrame
>>> sentence_data.show()
+-----+-------------------+
|label|           sentence|
+-----+-------------------+
|    0|I heard about Spa...|
|    0|I wish Java could...|
|    1|Logistic regressi...|
+-----+-------------------+
```

9.6.1 特征提取



第3步：用Tokenizer把每个句子分解成单词。

```
# 创建Tokenizer对象并设置输入列和输出列
>>> tokenizer = Tokenizer(inputCol="sentence", outputCol="words")

# 使用Tokenizer拆分句子列
>>> words_data = tokenizer.transform(sentence_data)

# 显示DataFrame，不截断显示
>>> words_data.show(truncate=False)

+-----+-----------------------------------+------------------------------------
|label|sentence                           |words                                |
+-----+-----------------------------------+------------------------------------
|0    |I heard about Spark and I love Spark|[i, heard, about, spark, and, i, love, spar
k]|
|0    |I wish Java could use case classes  |[i, wish, java, could, use, case, classes]
|1    |Logistic regression models are neat |[logistic, regression, models, are, neat]
+-----+-----------------------------------+------------------------------------
```

9.6.1 特征提取



第4步：用HashingTF的transform()方法把每个“词袋”哈希成特征向量。这里设置哈希表的桶数为2000。

```
# 创建HashingTF对象并设置输入列、输出列以及特征数量
>>> hashing_tf = HashingTF(inputCol="words", outputCol="rawFeatures", numFeat
ures=2000)

# 使用HashingTF将单词列转换为特征向量
>>> featurized_data = hashing_tf.transform(words_data)
```

9.6.1 特征提取



```
# 显示DataFrame，不截断显示
>>> featurized_data.select("words", "rawFeatures").show(truncate=False)

+-----------------------------------------+------------------------------
|words                                    |rawFeatures                                                
```

9.6.1 特征提取



第一行输出结果及其含义

输出结果	含义
2000	代表哈希表的桶数
[240,673,891,956,1286,1756]	代表“i”“heard”“about”“and”“love”“spark”6个单词的哈希值。注：哈希操作没有顺序性，所以索引240并不对应单词“i”
[1.0,1.0,1.0,1.0,2.0,2.0]	分别表示各单词的出现次数

9.6.1 特征提取



第5步：调用IDF方法来重新构造特征向量的规模，生成的变量idf是一个评估器，在特征向量上应用它的fit()方法，会产生一个IDFModel（名称为idfModel）。

```
# 创建IDF对象并设置输入列和输出列
>>> idf = IDF(inputCol="rawFeatures", outputCol="features")

# 拟合数据以创建IDF模型
>>> idf_model = idf.fit(featurized_data)
```

第6步：调用IDFModel的transform( )方法，得到每一个单词对应的TF-IDF度量值。

```
# 使用IDF模型转换数据
>>> rescaled_data = idf_model.transform(featurized_data)
```

9.6.1 特征提取



第6步：调用IDFModel的transform()方法，可以得到每一个单词对应的TF-IDF度量值。

```
# 使用IDF模型转换数据
>>> rescaled_data = idf_model.transform(featurized_data)
```

9.6.1 特征提取



```
# 显示DataFrame中的特征列和标签列
>>> rescaled_data.select("features", "label").show(truncate=False)
+-----------------------------------------------------------------------------------------------------------------+-----+
|features                                                                                                         |label|
+-----------------------------------------------------------------------------------------------------------------+-----+
|(2000,[240,673,891,956,1286,1756],[0.6931471805599453,0.6931471805599453,0.6931471805599453,0.6931471805599453,1.3862943611198906,0.5753641449035617])|0    |
|(2000,[80,342,495,1133,1307,1756,1967],[0.6931471805599453,0.6931471805599453,0.6931471805599453,0.6931471805599453,0.6931471805599453,0.2876820724517808 5,0.6931471805599453])|0    |
|(2000,[286,763,1059,1604,1871],[0.6931471805599453,0.6931471805599453,0.6931471805599453,0.6931471805599453,0.6931471805599453])|1    |
+-----------------------------------------------------------------------------------------------------------------+-----+
```

9.6.2 特征转换



1．特征转换操作

spark.ml包提供了大量的特征转换操作：

（1）Tokenizer
（2）StopWordsRemover
（3）NGram
（4）Binarizer
（5）PCA
（6）PolynomialExpansion
（7）Discrete Cosine Transform (DCT)
（8）StringIndexer
（9）IndexToString
（10）OneHotEncoder
（11）VectorIndexer
（12）Interaction
（13）Normalizer
（14）StandardScaler
（15）RobustScaler
（16）MinMaxScaler
（17）MaxAbsScaler
（18）Bucketizer
（19）ElementwiseProduct
（20）SQLTransformer
（21）VectorAssembler
(22) VectorSizeHint
（23）QuantileDiscretizer
(24) Imputer

9.6.2 特征转换



2．特征转换的例子

•在机器学习处理过程中，为了方便相关算法的实现，经常需要把标签数据（一般是字符串）转化成**整数索引**，或是在计算结束后将整数索引还原为相应的标签

•Spark ML包中提供了几个相关的转换器，例如：StringIndexer、IndexToString、OneHotEncoder、VectorIndexer，它们提供了十分方便的特征转换功能，这些转换器类都位于org.apache.spark.ml.feature包下

9.6.2 特征转换



- **StringIndexer**

- StringIndexer转换器可以把一列**类别型的特征**（或标签）进行编码，使其数值化，索引的范围从0开始，该过程可以使得相应的特征索引化，使得某些无法接受类别型特征的算法可以使用，并提高诸如决策树等机器学习算法的效率

- 索引构建的顺序为标签的频率，优先编码频率较大的标签，所以出现频率最高的标签为0号

- 如果输入的是数值型的，我们会把它转化成字符型，然后再对其进行编码

9.6.2 特征转换



首先，引入所需要使用的类。

```
>>> from pyspark.ml.feature import StringIndexer, IndexToString
>>> from pyspark.sql import SparkSession
```

9.6.2 特征转换



其次，构建1个DataFrame，设置StringIndexer的输入列和输出列的名字

```
# 创建一个 SparkSession（如果尚未创建）
>>> spark = SparkSession.builder.appName("example").getOrCreate()

# 创建数据集
>>> data = [
    (0, "a"), (1, "b"), (2, "c"), (3, "a"), (4, "a"), (5, "c")
]

# 将数据集转换为DataFrame
>>> df1 = spark.createDataFrame(data, ["id", "category"])

# 创建StringIndexer对象并设置输入列和输出列
>>> indexer = StringIndexer(inputCol="category", outputCol="categoryInde
x")
```

9.6.2 特征转换



最后，通过fit( )方法进行模型训练，用训练出的模型对原数据集进行处理，并通过indexed.show( )进行展示。

```
# 对DataFrame进行索引
>>> indexed_df = indexer.fit(df1).transform(df1)

# 显示DataFrame
>>> indexed_df.show()
+---+--------+-------------+
| id|category|categoryIndex|
+---+--------+-------------+
|  0|       a|          0.0|
|  1|       b|          2.0|
|  2|       c|          1.0|
|  3|       a|          0.0|
|  4|       a|          0.0|
|  5|       c|          1.0|
+---+--------+-------------+
```

9.6.2 特征转换



- **IndexToString**

- 与StringIndexer相反，IndexToString的作用是把标签索引的一列重新映射回原有的字符型标签。IndexToString一般都和StringIndexer配合使用。先用StringIndexer转化成标签索引，进行模型训练，然后在预测标签的时候再把标签索引转化成原有的字符标签。当然，Spark允许使用自己提供的标签。

9.6.2 特征转换



```
# 创建一个 IndexToString 转换器
>>> index_to_string = IndexToString(inputCol="categoryIndex", outputCol="original
Category")

# 使用转换器将索引列转换回原始类别列
>>> string_data = index_to_string.transform(indexed_df)

# 显示DataFrame中的"id"和"originalCategory"列
>>> string_data.select("id", "originalCategory").show()

+---+---------------+
| id|originalCategory|
+---+---------------+
|  0|              a|
|  1|              b|
|  2|              c|
|  3|              a|
|  4|              a|
|  5|              c|
+---+---------------+
```

9.6.2 特征转换



- **VectorIndexer**
- StringIndexer对单个类别型特征进行转换。如果特征都已经被组织在一个向量中，又想对其中某些单个分量进行处理，则可以利用VectorIndexer类进行转换。

- VectorIndexer 类的maxCategories参数，可以自动识别类别型特征，并将原始值转换为类别索引。它基于特征值的数量来识别需要被类别化的特征。那些取值数最多不超过maxCategories的特征，将会被类型化并转化为索引。

9.6.2 特征转换



首先引入所需要的类，并构建数据集。

```
>>> from pyspark.ml.feature import VectorIndexer
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.sql import SparkSession
# 创建Spark会话
>>> spark = SparkSession.builder.appName("example").getOrCreate()

# 创建示例数据
>>> data = [Vectors.dense(-1.0, 1.0, 1.0),
        Vectors.dense(-1.0, 3.0, 1.0),
        Vectors.dense(0.0, 5.0, 1.0)]

# 创建DataFrame并指定列名
>>> df = spark.createDataFrame([(vector,) for vector in data], ["features"])
```

9.6.2 特征转换



然后，构建VectorIndexer转换器，设置输入和输出列，并进行模型训练。

```
# 创建VectorIndexer对象并设置输入列、输出列以及最大不同特征值的数量
>>> vector_indexer = VectorIndexer(inputCol="features", outputCol="indexed
Features", maxCategories=2)

>>> indexerModel = vector_indexer.fit(df)
>>> indexedData = indexerModel.transform(df)

>>> indexedData.show()
+-------------+-------------+
|     features|indexedFeatures|
+-------------+-------------+
|[-1.0,1.0,1.0]|  [1.0,1.0,0.0]|
|[-1.0,3.0,1.0]|  [1.0,3.0,0.0]|
| [0.0,5.0,1.0]|  [0.0,5.0,0.0]|
+-------------+-------------+
```

9.6.2 特征转换



接下来，通过VectorIndexerModel的categoryMaps成员来获得被转换的特征及其映射，这里可以看到，共有两个特征被转换，分别是0号和2号。

```
# 获取分类特征信息的索引
>>> categorical_features = set(indexerModel.categoryMaps.keys())

# 打印分类特征信息
>>> print(f"Chose {len(categorical_features)} categorical features: {', '.join(map(str, categorical_features))}")
Chose 2 categorical features: 0, 2
```

9.6.2 特征转换



最后，把模型应用于原有的数据，并打印结果。

```
# 使用VectorIndexerModel对DataFrame进行特征列索引
>>> indexed = indexerModel.transform(df)

# 显示DataFrame
>>> indexed.show()
  +-------------+-------------+
  |     features|      indexed|
  +-------------+-------------+
|[-1.0,1.0,1.0]|[1.0,1.0,0.0]|
|[-1.0,3.0,1.0]|[1.0,3.0,0.0]|
| [0.0,5.0,1.0]|[0.0,5.0,0.0]|
+-------------+-------------+
```

9.6.3 特征选择



•特征选择（Feature Selection）指的是在特征向量中选择出那些“优秀”的特征，组成新的、更“精简”的特征向量的过程。它在高维数据分析中十分常用，可以剔除掉“冗余”和“无关”的特征，提升学习器的性能

9.6.3 特征选择



•1．特征选择操作

•（1）VectorSlicer

•VectorSlicer的作用类似于MATLAB/numpy中的“列切片”，它可以根据给定的索引（整数索引值或列名索引）选择出特征向量中的部分列，并生成新的特征向量。

•（2）RFormula

•RFormula提供了一种R语言风格的特征向量列选择功能，用户可以给其传入一个R表达式，它会根据该表达式，自动选择相应的特征列形成新的特征向量。

•（3）ChiSqSelector

•ChiSqSelector通过卡方选择的方法来进行特征选择，它的输入需要是一个已有标签的数据集，ChiSqSelector会针对每一个特征与标签的关系进行卡方检验，从而选择出那些统计意义上区分度最强的特征。

9.6.3 特征选择



- （4）UnivariateFeatureSelector

•UnivariateFeatureSelector可以看成是ChiSqSelector的扩展版，它也支持对连续型数据的特征选择，Spark会自动根据特征类型和标签类型来选择评估函数。当特征与标签都为类别型时，评估函数使用的也是chi-squared，此时UnivariateFeature Selector 的功能就相当于ChiSqSelector。当特征为连续型，标签为类别型时，评估函数为ANOVATest。当特征与标签类型都为连续型时，评估函数使用的是F-value。

- （5）VarianceThresholdSelector

•VarianceThresholdSelector将方差值小于人为给定的阈值的特征删除。该阈值默认为0，因此只有方差为0的特征，即所有值都相同的特征，会被删除。

9.6.3 特征选择



•2．选择操作的例子

•卡方选择是统计学上常用的一种有监督特征选择方法，它通过对特征和真实标签之间进行卡方检验，来判断该特征和真实标签的关联程度，进而确定是否对其进行选择。

•和spark.ml包中的大多数学习方法一样，spark.ml包中的卡方选择也是以“评估器+转换器”的形式出现的，主要由ChiSqSelector和ChiSqSelectorModel两个类来实现。

•首先，进行环境的设置，引入卡方选择器所需要使用的类。

```
>>> from pyspark.sql import SparkSession
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.sql import Row
>>> from pyspark.ml.feature import ChiSqSelector
```

9.6.3 特征选择



其次，创建实验数据，这是一个具有3个样本、4个特征维度的数据集，标签有1和0两种，我们将在此数据集上进行卡方选择。

```
# 创建Spark会话
>>> spark = SparkSession.builder.appName("example").getOrCreate()

# 创建数据集
>>> data = [
    (1, Vectors.dense(0.0, 0.0, 18.0, 1.0), 1),
    (2, Vectors.dense(0.0, 1.0, 12.0, 0.0), 0),
    (3, Vectors.dense(1.0, 0.0, 15.0, 0.1), 0)
]
```

9.6.3 特征选择



```
# 将数据集转换为DataFrame
>>> df = spark.createDataFrame([Row(id=id, features=vec, label=label) for id,
vec, label in data])
>>> df.show()
+---+-----------------+-----+
| id|         features|label|
+---+-----------------+-----+
|  1|[0.0,0.0,18.0,1.0]|    1|
|  2|[0.0,1.0,12.0,0.0]|    0|
|  3|[1.0,0.0,15.0,0.1]|    0|
+---+-----------------+-----+
```

9.6.3 特征选择



然后，用卡方选择进行特征选择器的训练，为了便于观察，我们设置只选择和标签关联性最强的一个特征（可以通过setNumTopFeatures(..)方法进行设置）。

```
# 创建ChiSqSelector对象并设置属性
>>> selector = ChiSqSelector(
    numTopFeatures=1,
    featuresCol="features",
    labelCol="label",
    outputCol="selected-feature"
)
```

9.6.3 特征选择



最后，用训练出的模型对原数据集进行处理，可以看见，第3列特征被选出作为最有用的特征列。

```
# 拟合ChiSqSelector模型
>>> selector_model = selector.fit(df)

# 使用模型进行特征选择
>>> result = selector_model.transform(df)

# 显示DataFrame，不截断显示
>>> result.show(truncate=False)
+---+-----------------+-----+---------------+
|id |features         |label|selected-feature|
+---+-----------------+-----+---------------+
|1  |[0.0,0.0,18.0,1.0]|1    |[18.0]         |
|2  |[0.0,1.0,12.0,0.0]|0    |[12.0]         |
|3  |[1.0,0.0,15.0,0.1]|0    |[15.0]         |
+---+-----------------+-----+---------------+
```

9.6.4 局部敏感哈希



•局部敏感哈希（Locality Sensitive Hashing）是一种被广泛应用在聚类、近似最近邻（Approximate Nearest Neighbor, ANN）、近似相似度连接（Approximate Similarity Join）等操作的哈希方法，基本思想是将那些在特征空间中相邻的点尽可能地映射到同一个哈希桶中。

•spark.ml包目前提供了两种LSH方法。

- •第一种是BucketedRandomProjectionLSH，使用欧式距离作为距离度量方法。
- •第二种是MinHash，使用Jaccard距离作为距离度量方法，显然，根据Jaccard相似度的性质，它只能够处理二元(0-1值)向量。