知 首发于
菜菜菜叶子子
写文章 登录

Spark MLlib 文本特征提取（TF-IDF/Word2Vec/CountVectorizer）

刘玲源 · 1 年前

Spark MLlib 提供三种文本特征提取方法，分别为TF-IDF、Word2Vec以及CountVectorizer，其原理与调用代码整理如下：

TF-IDF

算法介绍：

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：

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

其中，|D|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。

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

调用：

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），我们使用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。然后，我们的特征向量可以在算法学习中。

Scala:

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

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)
```

Java：

```
importjava.util.Arrays;
importjava.util.List;
importorg.apache.spark.ml.feature.HashingTF;
importorg.apache.spark.ml.feature.IDF;
importorg.apache.spark.ml.feature.IDFModel;
importorg.apache.spark.ml.feature.Tokenizer;
importorg.apache.spark.ml.linalg.Vector;
importorg.apache.spark.sql.Dataset;
importorg.apache.spark.sql.Row;
importorg.apache.spark.sql.RowFactory;
importorg.apache.spark.sql.SparkSession;
importorg.apache.spark.sql.types.DataTypes;
importorg.apache.spark.sql.types.Metadata;
importorg.apache.spark.sql.types.StructField;
importorg.apache.spark.sql.types.StructType;
List<Row> data = Arrays.asList(
  RowFactory.create(0.0, "Hi I heard about Spark"),
  RowFactory.create(0.0, "I wish Java could use case classes"),
  RowFactory.create(1.0, "Logistic regression models are neat")
);
StructType schema = new StructType(new StructField[]{
new StructField("label", DataTypes.DoubleType, false, Metadata.empty()),
new StructField("sentence", DataTypes.StringType, false, Metadata.empty())
});
Dataset<Row> sentenceData = spark.createDataFrame(data, schema);
Tokenizer tokenizer = new Tokenizer().setInputCol("sentence").setOutputCol("words"
Dataset<Row> wordsData = tokenizer.transform(sentenceData);
int numFeatures = 20;
HashingTF hashingTF = new HashingTF()
  .setInputCol("words")
  .setOutputCol("rawFeatures")
  .setNumFeatures(numFeatures);
Dataset<Row> featurizedData = hashingTF.transform(wordsData);
// alternatively, CountVectorizer can also be used to get term frequency vectors
IDF idf = new IDF().setInputCol("rawFeatures").setOutputCol("features");
IDFModel idfModel = idf.fit(featurizedData);
Dataset<Row> rescaledData = idfModel.transform(featurizedData);
for (Row r : rescaledData.select("features", "label").takeAsList(3)) {
  Vector features = r.getAs(0);
  Double label = r.getDouble(1);
```

```
    System.out.println(features);
    System.out.println(label);
  }
```

Python：

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer

sentenceData = spark.createDataFrame([
    (0, "Hi I heard about Spark"),
    (0, "I wish Java could use case classes"),
    (1, "Logistic regression models are neat")
], ["label", "sentence"])
tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
wordsData = tokenizer.transform(sentenceData)
hashingTF = HashingTF(inputCol="words", outputCol="rawFeatures", numFeatures=20)
featurizedData = hashingTF.transform(wordsData)
# alternatively, CountVectorizer can also be used to get term frequency vectors

idf = IDF(inputCol="rawFeatures", outputCol="features")
idfModel = idf.fit(featurizedData)
rescaledData = idfModel.transform(featurizedData)
for features_label in rescaledData.select("features", "label").take(3):
    print(features_label)
```

Word2Vec

算法介绍：

Word2vec是一个Estimator，它采用一系列代表文档的词语来训练word2vecmodel。该模型将每个词语映射到一个固定大小的向量。word2vecmodel使用文档中每个词语的平均数来将文档转换为向量，然后这个向量可以作为预测的特征，来计算文档相似度计算等等。

在下面的代码段中，我们首先用一组文档，其中每一个文档代表一个词语序列。对于每一个文档，我们将其转换为一个特征向量。此特征向量可以被传递到一个学习算法。

调用：

Scala:

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

// 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)
```

Java:

```java
import java.util.Arrays;
import java.util.List;

import org.apache.spark.ml.feature.Word2Vec;
import org.apache.spark.ml.feature.Word2VecModel;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.RowFactory;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.types.*;

// Input data: Each row is a bag of words from a sentence or document.
List<Row> data = Arrays.asList(
  RowFactory.create(Arrays.asList("Hi I heard about Spark".split(" "))),
  RowFactory.create(Arrays.asList("I wish Java could use case classes".split(" "))
  RowFactory.create(Arrays.asList("Logistic regression models are neat".split(" ")
);
StructType schema = new StructType(new StructField[]{
  new StructField("text", new ArrayType(DataTypes.StringType, true), false, Metada
});
Dataset<Row> documentDF = spark.createDataFrame(data, schema);

// Learn a mapping from words to Vectors.
Word2Vec word2Vec = new Word2Vec()
  .setInputCol("text")
  .setOutputCol("result")
  .setVectorSize(3)
  .setMinCount(0);
Word2VecModel model = word2Vec.fit(documentDF);
Dataset<Row> result = model.transform(documentDF);
for (Row r : result.select("result").takeAsList(3)) {
  System.out.println(r);
}
```

Python:

```
from pyspark.ml.feature import Word2Vec

# Input data: Each row is a bag of words from a sentence or document.
documentDF = spark.createDataFrame([
    ("Hi I heard about Spark".split(" "), ),
    ("I wish Java could use case classes".split(" "), ),
    ("Logistic regression models are neat".split(" "), )
], ["text"])
# Learn a mapping from words to Vectors.
word2Vec = Word2Vec(vectorSize=3, minCount=0, inputCol="text", outputCol="result")
model = word2Vec.fit(documentDF)
result = model.transform(documentDF)
for feature in result.select("result").take(3):
    print(feature)
```

Countvectorizer

算法介绍：

Countvectorizer和Countvectorizermodel旨在通过计数来将一个文档转换为向量。当不存在先验字典时，Countvectorizer可作为Estimator来提取词汇，并生成一个Countvectorizermodel。该模型产生文档关于词语的稀疏表示，其表示可以传递给其他算

法如LDA。

在fitting过程中，countvectorizer将根据语料库中的词频排序选出前vocabsize个词。一个可选的参数minDF也影响fitting过程中，它指定词汇表中的词语在文档中最少出现的次数。另一个可选的二值参数控制输出向量，如果设置为真那么所有非零的计数为1。这对于二值型离散概率模型非常有用。

示例：

假设我们有如下的DataFrame包含id和texts两列：

id | texts

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

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

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

文本中的每一行都是一个文档类型的数组(字符串)。调用的CountVectorizer产生词汇(a,b,c)的CountVectorizerModel，转换后的输出向量如下：

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])

每个向量代表文档的词汇表中每个词语出现的次数。

调用：

Scala：

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

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()
```

Java:

```
import java.util.Arrays;
import java.util.List;

import org.apache.spark.ml.feature.CountVectorizer;
import org.apache.spark.ml.feature.CountVectorizerModel;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.RowFactory;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.types.*;

// Input data: Each row is a bag of words from a sentence or document.
List<Row> data = Arrays.asList(
  RowFactory.create(Arrays.asList("a", "b", "c")),
  RowFactory.create(Arrays.asList("a", "b", "b", "c", "a"))
);
StructType schema = new StructType(new StructField [] {
  new StructField("text", new ArrayType(DataTypes.StringType, true), false, Metada
});
Dataset<Row> df = spark.createDataFrame(data, schema);

// fit a CountVectorizerModel from the corpus
```

```
CountVectorizerModel cvModel = new CountVectorizer()
  .setInputCol("text")
  .setOutputCol("feature")
  .setVocabSize(3)
  .setMinDF(2)
  .fit(df);

// alternatively, define CountVectorizerModel with a-priori vocabulary
CountVectorizerModel cvm = new CountVectorizerModel(new String[]{"a", "b", "c"})
  .setInputCol("text")
  .setOutputCol("feature");

cvModel.transform(df).show();
```

Python:

```
from pyspark.ml.feature import CountVectorizer

# Input data: Each row is a bag of words with a ID.
df = spark.createDataFrame([
    (0, "a b c".split(" ")),
    (1, "a b b c a".split(" "))
], ["id", "words"])

# fit a CountVectorizerModel from the corpus.
cv = CountVectorizer(inputCol="words", outputCol="features", vocabSize=3, minDF=2.
model = cv.fit(df)
result = model.transform(df)
result.show()
```

「卖菜中..」

赞赏

还没有人赞赏，快来当第一个赞赏的人吧！

mllib Spark 机器学习

收藏 分享 举报

8

…

2 条评论

写下你的评论...

bit扛把子

您好，我在运行java版本的TF-IDF代码时，Dataset<Row> sentenceData = spark.createDataFrame(data, schema);出现错误，
Error:(43, 55) java: 不兼容的类型 需要: org.apache.spark.sql.DataFrame 找到: org.apache.spark.sql.Dataset<org.apache.spark.sql.Row>
DataSet和DataFrame这里用混了吗？求解释，困扰在下很久了，多谢！

1 年前

刘广森

问下楼主 Countvectorizer统计词频的时候 如果 特征 我提前就确定了 不让他自己统计词频找出特征 应该 怎么对文本 进行标注
比如 我有100个句子 在这100个句子中 我人工确定了 60个特征 对这 100个句子 分词后 想用 我自定义的特征来表示 该如何操作
如果 可以 想加下楼主qq 详细请教下

7 个月前

文章被以下专栏收录

菜菜菜叶子子

叶子子学习笔记

进入专栏

推荐阅读

Spark MLlib 数据预处理 - 特征变换（一）

Tokenizer（分词器）算法介绍：Tokenization将文本划分为独立个体（通常为单词）。RegexTokenizer基于正则表达式提供更多的划分选项。默认情况下，参数“pattern”为划分文本的分隔符。或者... 查看全文 >

刘玲源 · 1 年前 · 发表于 菜菜菜叶子子

Spark机器学习库（MLlib）中文指南

Spark机器学习库(MLlib)指南 MLlib是Spark里的机器学习库。它的目标是使实用的机器学习算法可扩展并容易使用。它提供如下工具： 1.机器学习算法：常规机器学习算法包括分类、回... 查看全文 >

刘玲源 · 1 年前 · 发表于 菜菜菜叶子子

Pipeline详解及Spark MLlib使用

本文中，我们介绍机器学习管道的概念。机器学习管道提供一系列基于数据框的高级的接口来帮助用户建立和调试实际的机器学习管道。管道里的主要概念MLlib提供标准的接口来使联合多个... 查看全文 >

刘玲源 · 1 年前 · 发表于 菜菜菜叶子子

中文文本分类模型，数据集，开源项目集合

CNN-RNN中文文本分类，基于tensorflowhttps://github.com/gaussic/text-classification-cnn-... 查看全文 >

灰灰 · 1 个月前 · 发表于 TensorFlowNews