推荐系统实践-1. TFIDF

原创 RandySun Randy的技术笔记 2020-10-01

收录于话题
#推荐系统

2个

01
TF-IDF算法介绍

TF-IDF（Term Frequency–InverseDocument Frequency）是一种用于资讯检索与文本挖掘的常用加权技术。TF-IDF的主要思想是：如果某个词或短语在一篇文章中出现的频率TF高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。

上述引用总结就是：**一个词语在一篇文章中出现次数越多，同时在所有文档中出现次数越少，越能够代表该文章。**

TF(Term Frequency, 词频)表示词条在文本中出现的频率，这个数字通常会被归一化(一般是词频除以文章总词数), 以防止它偏向长的文件（同一个词语在长文件里可能会比短文件有更高的词频，而不管该词语重要与否）。TF用公式表示如下

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{1}$$

其中， $n_{i,j}$ 表示词条 t_i 在文档 d_j 中出现的次数， $TF_{i,j}$ 就是表示词条 t_i 在文档 d_j 中出现的频率。

但是， 一些通用的词语对于主题并没有太大的作用， 反倒是一些出现频率较少的词才能够表达文章的主题， 所以单纯使用是TF不合适的。权重的设计必须满足：一个词预测主题的能力越强，权重越大，反之，权重越小。所有统计的文章中，一些词只是在其中很少几篇文章中出现，那么这样的词对文章的主题的作用很大，这些词的权重应该设计的较大。IDF就是在完成这样的工作。

IDF(Inverse Document Frequency, 逆文件频率)表示关键词的普遍程度。如果包含词条 i 的文档越少， IDF越大，则说明该词条具有很好的类别区分能力。某一特定词语的IDF，可

以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到

$$IDF_i = \log \frac{|D|}{1 + |j : t_i \in d_j|} \tag{2}$$

其中，$|D|$ 表示所有文档的数量，$|j : t_i \in d_j|$ 表示包含词条 t_i 的文档数量，为什么这里要加 1 呢？主要是**防止包含词条 t_i 的数量为 0 从而导致运算出错的现象发生。**

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**，表达为

$$TF\text{-}IDF = TF \cdot IDF \tag{3}$$

02

基于python的TF-IDF实践

```python
# -*-coding:utf-8-*-
"""
Data: 2018-08
Author: Thinkgamer"""
import jieba
import math
import jieba.analyse

class TF_IDF:

    def __init__(self,file,stop_file):
        self.file = file
        self.stop_file = stop_file
        self.stop_words = self.getStopWords()

    # 获取停用词列表
    def getStopWords(self):
        swlist=list()
        for line in open(self.stop_file,"r",encoding="utf-8").readlines():
            swlist.append(line.strip())
        print("加载停用词完成...")
        return swlist
```

```
23
24      # 加载商品和其对应的短标题，使用jieba进行分词并去除停用词
25      def loadData(self):
26          dMap = dict()
27          for line in open(self.file,"r",encoding="utf-8").readlines():
28              id,title = line.strip().split("\t")
29              dMap.setdefault(id, [])
30              for word in list(jieba.cut(str(title).replace(" ",""), cut_all=Fa
31                  if word not in self.stop_words:
32                      dMap[id].append(word)
33          print("加载商品和对应的短标题，并使用jieba分词和去除停用词完成...")
34          return dMap
35
36      # 获取一个短标题中的词频
37      def getFreqWord(self,words):
38          freqWord = dict()
39          for word in words:
40              freqWord.setdefault(word,0)
41              freqWord[word] += 1
42          return freqWord
43
44      # 统计单词在所有短标题中出现的次数
45      def getCountWordInFile(self,word,dMap):
46          count = 0
47          for key in dMap.keys():
48              if word in dMap[key]:
49                  count += 1
50          return count
51
52      # 计算TFIDF值
53      def getTFIDF(self,words,dMap):
54          # 记录单词关键词和对应的tfidf值
55          outDic = dict()
56          freqWord = self.getFreqWord(words)
57          for word in words:
58              # 计算TF值，即单个word在整句中出现的次数
59              tf = freqWord[word]*1.0 / len(words)
60              # 计算IDF值，即log(所有的标题数/(包含单个word的标题数+1))
61              idf = math.log(len(dMap)/(self.getCountWordInFile(word,dMap)+1))
62              tfidf = tf * idf
```

```
            outDic[word] = tfidf
        # 给字典排序
        orderDic = sorted(outDic.items(), key=lambda x:x[1], reverse=True)
        return orderDic

    def getTag(self,words):
        # withWeight 用来设置是否打印权重
        print(jieba.analyse.extract_tags(words, topK=20, withWeight=True))

if __name__ == "__main__":
    # 数据集
    file = "data/id_title.txt"
    # 停用词文件
    stop_file = "data/stop_words.txt"
    tfidf=TF_IDF(file,stop_file)
    tfidf.getTag(open("data/one","r",encoding="utf-8").read(),)

    # dMap 中key为商品id，value为去除停用词后的词
    # dMap = tfidf.loadData()
    # for id in dMap.keys():
    #     tfIdfDic = tfidf.getTFIDF(dMap[id],dMap)
    #     print(id,tfIdfDic)
```

03

基于spark的TF-IDF实践

```

from pyspark.ml.feature import Word2Vec, Word2VecModel

from common.ContentPartition import ContentPartition
from common.SparkSessionBase import SparkSessionBase


class ContentVectorModel():

    content_vector_path = "hdfs://data1:8020/recommend/models/VECTOR_old.mode

    def __init__(self):
```

```
        self.spark = SparkSessionBase().create_spark_session()


    @staticmethod
    def get_model():
        word2vec_model = Word2VecModel.load(path=ContentVectorModel.content_v
        return word2vec_model



    # 获取训练数据集
    def _get_tran_data(self):
        basic_content = self.spark.sql(
            """
                SELECT cp.id publish_id,
                       cp.content_id,
                       cp.channel_id,
                       get_json_object(cc.content,'$.title') sentence
                from ods.content_publish cp,
                     ods.content_content cc
                WHERE cp.content_id = cc.id
                order by cp.id
            """
            )
        # spark读取文章内容并分词
        words_df = basic_content.rdd.mapPartitions(ContentPartition.segmentat
        return words_df


    def fit_model(self):
        train_data = self._get_tran_data()
        word2Vec = Word2Vec(vectorSize=1000, inputCol="words", outputCol="moc
        word2Vec_model = word2Vec.fit(train_data)
        word2Vec_model.write().overwrite().save(ContentVectorModel.content_ve
        return word2Vec_model


if __name__ == '__main__':
    csm = ContentVectorModel()
    csm.fit_model()
```

其中segmentation分词代码如下

```
    # 进行分词
    @staticmethod
    def segmentation(partition):
        # 加载保留词
        jieba.load_userdict(ContentPartition.get_preserve_dicpath())

        # 分词
        def cut_sentence(sentence):
            """对切割之后的词语进行过滤，去除停用词，保留名词，英文和自定义词库中的词
            stopwords_list = ContentPartition.get_stop_words()
            # print(sentence,"*"*100)
            seg_list = pseg.lcut(sentence)
            seg_list = [i for i in seg_list if i.flag not in stopwords_list]
            filtered_words_list = []
            for seg in seg_list:
                # print(seg)
                if len(seg.word) <= 1:
                    continue
                elif seg.flag == "eng":
                    if len(seg.word) <= 2:
                        continue
                    else:
                        filtered_words_list.append(seg.word)
                elif seg.flag.startswith("n"):
                    filtered_words_list.append(seg.word)
                elif seg.flag in ["x", "eng"]:  # 是自定一个词语或者是英文单词
                    filtered_words_list.append(seg.word)
            return filtered_words_list

        for row in partition:
            sentence = re.sub("<.*?>", "", row.sentence)  # 替换掉标签数据
            words = cut_sentence(sentence)
            yield row.publish_id, row.content_id, row.channel_id, words
```