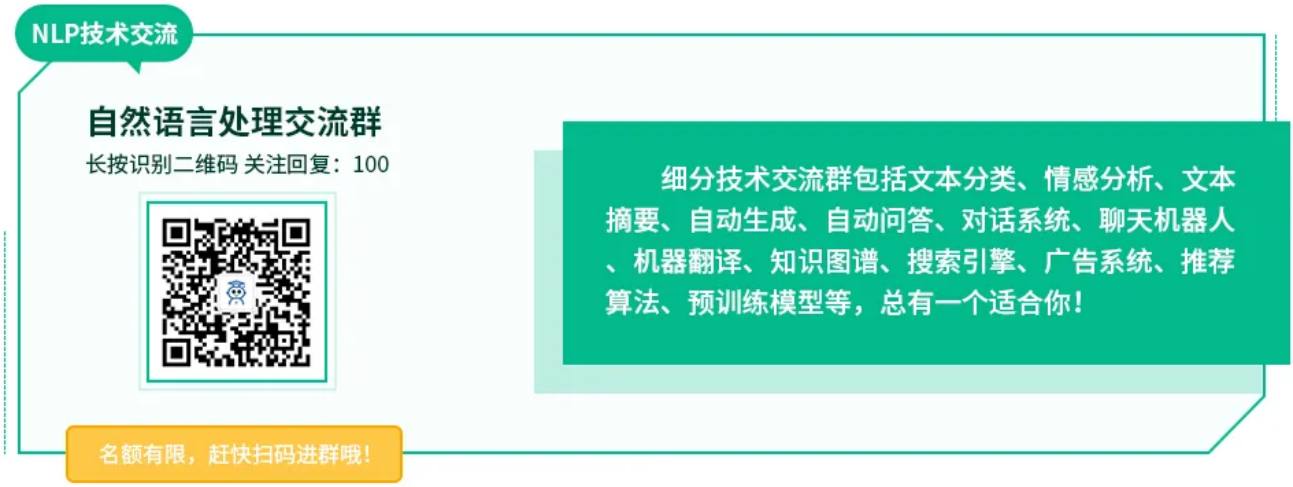
【短文本相似度】传统方法BM25解决短文本相似度问题

AINLP 2020-09-05

以下文章来源于机器学习算法与自然语言处理 ， 作者刘聪NLP

机器学习算法与自然语言处理

一个有情怀的公众号。机器学习、自然语言处理、算法等知识集中营、期待与你相遇~



作者 刘聪NLP

学校 | 中国药科大学 药学信息学硕士

知乎专栏 | 自然语言处理相关论文

之前介绍过TF-IDF计算短文本相似度，见刘聪NLP：传统方法TF-IDF解决短文本相似度问题，想着就把这一系列都介绍完吧，也算是自己的归纳总结，今天就介绍一下**如何使用BM25算法计算短文本相似度。**

> 上一篇短文本相似度算法研究文章中，我们举过这样一个场景，在问答系统任务（**问答机器人**）中，我们往往会人为地配置一些常用并且描述清晰的问题及其对应的回答，我们将这些配置好的问题称之为“**标准问**”。当用户进行提问时，常常将用户的问题与所有配置好的标准问进行相似度计算，找出与用户问题最相似的标准问，并返回其答案给用户，这样就完成了一次问答操作。

我们就以该场景，来介绍BM25的公式算法以及code代码。

其实，BM25算法是TF-IDF算法的优化，我们回顾一下，如何计算一个用户问题与一个标准问题的TF-IDF相似度？

$$Score\_TF\_IDF(Q,d)=\sum_{i}^{n}IDF_i*TF(q_i,d)$$

一个用户问题与一个标准问题的TF-IDF相似度，是将用户问题中每一词与标准问题计算得到的TF-IDF值求和。

BM25算法也是如此，计算公式如下：

$$Score(Q,d)=\sum_{i}^{n}W_iR(q_i,d)$$

其中，Q 为用户问题，d 为“标准问”库中的一个标准问题，n 为用户问题中词的个数，q_i 为用户问题中第 i 个词，W_i 为该词的权重，$R(q_i,d)$ 为该词与标准问题的相关性分数。

W_i 相当于TF-IDF算法中的IDF，$R(q_i,d)$ 相当于是TF-IDF算法中的TF；只不过BM25对这两个指标进行了优化，具体如下：

$$W_i=log(\frac{N-df_i+0.5}{df_i+0.5})$$

其中，N表示“标准问”库中标准问题的总个数，df_i 表示包含词汇 q_i 的标准问题的个数。

$$R(q_i,d)=\frac{f_i(k_1+1)}{f_i+K}*\frac{qf_i(k_2+1)}{qf_i+k_2}$$

$$K=k_1*(1-b+b*\frac{dl}{avg\_dl})$$

其中，k_1,k_2和b **是调协因子，一般分别设为2，1，0.75**；f_i 表示词汇 q_i 在标准问题中出现的次数；qf_i 表示词汇 q_i 在用户问题中出现的次数；dl 为标准问题的长度；avg_dl 为“标准问”库中所有标准问题的平均长度。

根据上述例子，我们进行coding，**BM25类定义具体如下**：

```
import numpy as np
from collections import Counter
```

```
class BM25_Model(object):
    def __init__(self, documents_list, k1=2, k2=1, b=0.5):
        self.documents_list = documents_list
        self.documents_number = len(documents_list)
        self.avg_documents_len = sum([len(document) for document in docume
        self.f = []
        self.idf = {}
        self.k1 = k1
        self.k2 = k2
        self.b = b
        self.init()

    def init(self):
        df = {}
        for document in self.documents_list:
            temp = {}
            for word in document:
                temp[word] = temp.get(word, 0) + 1
            self.f.append(temp)
            for key in temp.keys():
                df[key] = df.get(key, 0) + 1
        for key, value in df.items():
            self.idf[key] = np.log((self.documents_number - value + 0.5) /

    def get_score(self, index, query):
        score = 0.0
        document_len = len(self.f[index])
        qf = Counter(query)
        for q in query:
            if q not in self.f[index]:
                continue
            score += self.idf[q] * (self.f[index][q] * (self.k1 + 1) / (
                        self.f[index][q] + self.k1 * (1 - self.b + self.b
                                qf[q] * (self.k2 + 1) / (qf[q] + self.k2)

        return score

    def get_documents_score(self, query):
        score_list = []
        for i in range(self.documents_number):
            score_list.append(self.get_score(i, query))
        return score_list
```

其中，documents_list 表示需要输入的文本列表，内部每个文本需要事先分好词；documents_number表示文本总个数；avg_documents_len 表示所有文本的平均长度；f用于存储每个文本中每个词的出现的次数；idf用于存储每个词汇的权重值；init函数是类初始化函数，用于求解文本集合中的f和idf变量；get_score函数是获取一个文本与文本列表中一个文本的bm25相似度值；get_documents_score函数是获取一个文本与文本列表中所有文本的bm25相似度值。

定义好的BM25类如何去使用呢？具体如下：

首先，给出文本集合，也就是我们上文场景中提到的“标准问”库；

```
document_list = ["行政机关强行解除行政协议造成损失，如何索取赔偿？",
                 "借钱给朋友到期不还得什么时候可以起诉？怎么起诉？",
                 "我在微信上被骗了，请问被骗多少钱才可以立案？",
                 "公民对于选举委员会对选民的资格申诉的处理决定不服，能不能去法院
                 "有人走私两万元，怎么处置他？",
                 "法律上餐具、饮具集中消毒服务单位的责任是不是对消毒餐具、饮具进
```

然后，我们对其进行分词操作；

```
import jieba
document_list = [list(jieba.cut(doc)) for doc in document_list]
```

得到结果如下：

```
[['行政', '机关', '强行', '解除', '行政', '协议', '造成', '损失', '，', '如何
['借钱', '给', '朋友', '到期', '不', '还', '得', '什么', '时候', '可以', '起诉
['我', '在', '微信', '上', '被', '骗', '了', '，', '请问', '被', '骗', '多少
['公民', '对于', '选举', '委员会', '对', '选民', '的', '资格', '申诉', '的',
['有人', '走私', '两万元', '，', '怎么', '处置', '他', '？'],
['法律', '上', '餐具', '、', '饮具', '集中', '消毒', '服务', '单位', '的', '责
```

接下来，我们实例化TF-IDF类，生成一个对象；

```
bm25_model = BM25_Model(document_list)
```

我们默认 k_1 , k_2 和 b 使用默认值。

通过参数调用，观察示例化的对象中documents_list ， documents_number，avg_documents_len ， f 和idf变量具体存储了什么；

```
print(bm25_model.documents_list)
print(bm25_model.documents_number)
print(bm25_model.avg_documents_len)
print(bm25_model.f)
print(bm25_model.idf)
```

结果如下：

```
documents_list:
[['行政', '机关', '强行', '解除', '行政', '协议', '造成', '损失', '，', '如何
15.666666666666666
documents_number:
6
avg_documents_len:
15.666666666666666
tf:
[{'行政': 0.15384615384615385, '机关': 0.07692307692307693, '强行': 0.07692
{'借钱': 0.06666666666666667, '给': 0.06666666666666667, '朋友': 0.06666666
{'我': 0.058823529411764705, '在': 0.058823529411764705, '微信': 0.05882352
{'公民': 0.047619047619047616, '对于': 0.047619047619047616, '选举': 0.0476
{'有人': 0.125, '走私': 0.125, '两万元': 0.125, '，': 0.125, '怎么': 0.125,
{'法律': 0.05, '上': 0.05, '餐具': 0.1, '、': 0.1, '饮具': 0.1, '集中': 0.05
idf:
{'行政': 1.2992829841302609, '机关': 1.2992829841302609, '强行': 1.29928298
```

最后，我们给出一个用户问题，通过bm25算法，计算出“标准问”库中所有标准问的相似度值；

```
query = "走私了两万元，在法律上应该怎么量刑？"
query = list(jieba.cut(query))
scores = bm25_model.get_documents_score(query)
```

结果如下：

```
[-3.41951, -3.39528, 0.03410, -2.88660, 0.04016, -0.67311]
```

通过结果我们可以发现，第五个标准问“有人走私两万元，怎么处置他？”与用户问题“走私了两万元，在法律上应该怎么量刑？”最为相似，**符合我们的预期。**

解释一下：为什么会出现负值，并且正值都很小？

是因为，我们举例的标准问库较小，有一些词汇或标点，例如：“？”，出现在很多标准问句中，导致“？”的idf值小于0，在做累加时，导致相似度一些为负值，并且正值都很小。**因此，我们无论是使用TF-IDF，还是BM25时，都最好去掉停用词。**

以上，就是本人对BM25算法的总结，有错误地方还希望指出，欢迎大家交流。

代码可见：https://github.com/liucongg/ZhiHu_Code/tree/master/bm25_code

---

推荐阅读

这个NLP工具，玩得根本停不下来

征稿启示| 200元稿费+5000DBC（价值20个小时GPU算力）

完结撒花！李宏毅老师深度学习与人类语言处理课程视频及课件（附下载）

从数据到模型，你可能需要1篇详实的pytorch踩坑指南