文本相似度-bm25算法原理及实现

Python Web与Django大咖之路 2019-11-18

自从Elastcsearch升级至6.X后，原本的相关性评分算法从TF-IDF（词频-逆文档频率）变为了BM25，那么BM25到底有什么过人之处，下面就来探索。

原理

BM25算法，通常用来作搜索相关性平分。一句话概况其主要思想：对Query进行语素解析，生成语素qi；然后，对于每个搜索结果D，计算每个语素qi与D的相关性得分，最后，将qi相对于D的相关性得分进行加权求和，从而得到Query与D的相关性得分。
BM25算法的一般性公式如下：

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

其中，Q表示Query，qi表示Q解析之后的一个语素（对中文而言，我们可以把对Query的分词作为语素分析，每个词看成语素qi。）；d表示一个搜索结果文档；Wi表示语素qi的权重；R(qi，d)表示语素qi与文档d的相关性得分。

下面我们来看如何定义Wi。判断一个词与一个文档的相关性的权重，方法有多种，较常用的是IDF。这里以IDF为例，公式如下：

$$\mathrm{IDF}(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5},$$

其中，N为索引中的全部文档数，n(qi)为包含了qi的文档数。
根据IDF的定义可以看出，对于给定的文档集合，包含了qi的文档数越多，qi的权重则越低。也就是说，当很多文档都包含了qi时，qi的区分度就不高，因此使用qi来判断相关性时的重要度就较低。
我们再来看语素qi与文档d的相关性得分R（qi，d）。首先来看BM25中相关性得分的一般形式：

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

$$K = k_1 \cdot (1 - b + b \cdot \frac{dl}{avgdl})$$

其中，k1，k2，b为调节因子，通常根据经验设置，一般k1=2，b=0.75；fi为qi在d中的出现频率，qfi为qi在Query中的出现频率。dl为文档d的长度，avgdl为所有文档的平均长度。由于绝大部分情况下，qi在Query中只会出现一次，即qfi=1，因此公式可以简化为：

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

从K的定义中可以看到，参数b的作用是调整文档长度对相关性影响的大小。b越大，文档长度的对相关性得分的影响越大，反之越小。而文档的相对长度越长，K值将越大，则相关性得分会越小。这可以理解为，当文档较长时，包含qi的机会越大，因此，同等fi的情况下，长文档与qi的相关性应该比短文档与qi的相关性弱。
综上，BM25算法的相关性得分公式可总结为：

$$Score(Q, d) = \sum_{i}^{n} IDF(q_i) \cdot \frac{f_i \cdot (k_1 + 1)}{f_i + k_1 \cdot (1 - b + b \cdot \frac{dl}{avgdl})}$$

从BM25的公式可以看到，通过使用不同的语素分析方法、语素权重判定方法，以及语素与文档的相关性判定方法，我们可以衍生出不同的搜索相关性得分计算方法，这就为我们设计算法提供了较大的灵活性。

代码实现

```
import math
import jieba
from utils import utils

# 测试文本
text = '''
自然语言处理是计算机科学领域与人工智能领域中的一个重要方向。
它研究能实现人与计算机之间用自然语言进行有效通信的各种理论和方法。
自然语言处理是一门融语言学、计算机科学、数学于一体的科学。
因此，这一领域的研究将涉及自然语言，即人们日常使用的语言，
所以它与语言学的研究有着密切的联系，但又有重要的区别。
自然语言处理并不是一般地研究自然语言，
而在于研制能有效地实现自然语言通信的计算机系统，
特别是其中的软件系统。因而它是计算机科学的一部分。
'''

class BM25(object):

    def __init__(self, docs):
        self.D = len(docs)
        self.avgdl = sum([len(doc)+0.0 for doc in docs]) / self.D
        self.docs = docs
        self.f = []  # 列表的每一个元素是一个dict，dict存储着一个文档中每个词的出现
        self.df = {} # 存储每个词及出现了该词的文档数量
        self.idf = {} # 存储每个词的idf值
        self.k1 = 1.5
        self.b = 0.75
        self.init()

    def init(self):
        for doc in self.docs:
            tmp = {}
            for word in doc:
                tmp[word] = tmp.get(word, 0) + 1  # 存储每个文档中每个词的出现次
            self.f.append(tmp)
            for k in tmp.keys():
                self.df[k] = self.df.get(k, 0) + 1
        for k, v in self.df.items():
```

```
            self.idf[k] = math.log(self.D-v+0.5)-math.log(v+0.5)

    def sim(self, doc, index):
        score = 0
        for word in doc:
            if word not in self.f[index]:
                continue
            d = len(self.docs[index])
            score += (self.idf[word]*self.f[index][word]*(self.k1+1)
                      / (self.f[index][word]+self.k1*(1-self.b+self.b*d
                                                      / self.avgdl)))
        return score

    def simall(self, doc):
        scores = []
        for index in range(self.D):
            score = self.sim(doc, index)
            scores.append(score)
        return scores

if __name__ == '__main__':
    sents = utils.get_sentences(text)
    doc = []
    for sent in sents:
        words = list(jieba.cut(sent))
        words = utils.filter_stop(words)
        doc.append(words)
    print(doc)
    s = BM25(doc)
    print(s.f)
    print(s.idf)
    print(s.simall(['自然语言', '计算机科学', '领域', '人工智能', '领域']))
```

分段再分词结果

```
[['自然语言', '计算机科学', '领域', '人工智能', '领域', '中', '一个', '方向'],
['研究', '人', '计算机', '之间', '自然语言', '通信', '理论', '方法'],
['自然语言', '一门', '融', '语言学', '计算机科学', '数学', '一体', '科学'],
[],
```

```
['这一', '领域', '研究', '涉及', '自然语言'],
['日常', '语言'],
['语言学', '研究'],
['区别'],
['自然语言', '研究', '自然语言'],
['在于', '研制', '自然语言', '通信', '计算机系统'],
['特别', '软件系统'],
['计算机科学', '一部分']]
```

s.f

列表的每一个元素是一个dict，dict存储着一个文档中每个词的出现次数

```
[{'中': 1, '计算机科学': 1, '领域': 2, '一个': 1, '人工智能': 1, '方向': 1, '自然
{'之间': 1, '方法': 1, '理论': 1, '通信': 1, '计算机': 1, '人': 1, '研究': 1, '自
{'融': 1, '一门': 1, '一体': 1, '数学': 1, '科学': 1, '计算机科学': 1, '语言学':
{},
{'领域': 1, '这一': 1, '涉及': 1, '研究': 1, '自然语言': 1},
{'日常': 1, '语言': 1},
{'语言学': 1, '研究': 1},
{'区别': 1},
{'研究': 1, '自然语言': 2},
{'通信': 1, '计算机系统': 1, '研制': 1, '在于': 1, '自然语言': 1},
{'软件系统': 1, '特别': 1},
{'一部分': 1, '计算机科学': 1}]
```

s.df

存储每个词及出现了该词的文档数量

```
{'在于': 1, '人工智能': 1, '语言': 1, '领域': 2, '融': 1, '日常': 1, '人': 1, '这
```

s.idf

存储每个词的idf值

```
{'在于': 2.0368819272610397, '一部分': 2.0368819272610397, '一个': 2.036881927
```

s.simall(['自然语言', '计算机科学', '领域', '人工智能', '领域'])

```
['自然语言', '计算机科学', '领域', '人工智能', '领域']与每一句的相似度[5.0769919814
```

END

喜欢此内容的人还喜欢

Django 开发学习笔记

Python Web与Django大咖之路

北大医生：你们用焦虑养出来的娃，最后都送到我这里了

诗评万象

查理·芒格震撼演讲：如何低成本地从错误中爬出来？

金融精读