BM25（Best Matching 25）算法基本思想

原创 扫地升 NLP工程化 2024年01月10日 10:00 浙江

BM25（Best Matching 25）是一种用于信息检索（Information Retrieval）和文本挖掘的算法，它被广泛应用于搜索引擎和相关领域。BM25 基于 TF-IDF（Term Frequency-Inverse Document Frequency）的思想，但对其进行了改进以考虑文档的长度等因素。

一.基本思想

以下是 BM25 算法的基本思想：

1. **TF-IDF** 的改进： BM25 通过对文档中的每个词项引入饱和函数（saturation function）和文档长度因子，改进了 TF-IDF 的计算。
2. 饱和函数： 在 BM25 中，对于词项的出现次数（TF），引入了一个饱和函数来调整其权重。这是为了防止某个词项在文档中出现次数过多导致权重过大。
3. 文档长度因子： BM25 考虑了文档的长度，引入了文档长度因子，使得文档长度对权重的影响不是线性的。这样可以更好地适应不同长度的文档。

二.计算方程

BM25 的具体计算公式如下：

$$\text{BM25}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{\text{len}(D)}{\text{avg\_len}}\right)}$$

其中：

- n是查询中的词项数。
- q_i是查询中的第i个词项。
- $\text{IDF}(q_i)$是逆文档频率，计算方式通常是$\log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$，其中$N$是文档总数，$n(q_i)$ 是包含词项q_i的文档数。
- $f(q_i, D)$是词项q_i在文档 D 中的出现次数（TF）。
- $\text{len}(D)$ 是文档 D 的长度。
- avg_len 是所有文档的平均长度。
- k_1 和 b 是调整参数，通常设置为 $k_1 = 1.5$ 和 $b = 0.75$。

BM25 算法的实现通常用于排序文档，使得与查询更相关的文档排名更靠前。在信息检索领域，BM25 已经成为一个经典的算法。

三.Python 实现

以下是一个简单的 Python 实现 BM25 算法的例子。请注意，实际应用中可能需要进行更复杂的文本预处理，例如去除停用词、词干化等。

```
import math
from collections import Counter

class BM25:
    def __init__(self, corpus, k1=1.5, b=0.75):
        self.k1 = k1
        self.b = b
        self.corpus = corpus
        self.doc_lengths = [len(doc) for doc in corpus]
        self.avg_doc_length = sum(self.doc_lengths) / len(self.doc_lengths)
        self.doc_count = len(corpus)
        self.doc_term_freqs = [Counter(doc) for doc in corpus]
        self.inverted_index = self.build_inverted_index()

    def build_inverted_index(self):
        inverted_index = {}
        for doc_id, doc_term_freq in enumerate(self.doc_term_freqs):
            for term, freq in doc_term_freq.items():
                if term notin inverted_index:
                    inverted_index[term] = []
                inverted_index[term].append((doc_id, freq))
        return inverted_index

    def idf(self, term):
        doc_freq = len(self.inverted_index.get(term, []))
        if doc_freq == 0:
            return0
        return math.log((self.doc_count - doc_freq + 0.5) / (doc_freq + 0.5) + 1.0)

    def bm25_score(self, query_terms, doc_id):
        score = 0
        doc_length = self.doc_lengths[doc_id]
        for term in query_terms:
            tf = self.doc_term_freqs[doc_id].get(term, 0)
            idf = self.idf(term)
            numerator = tf * (self.k1 + 1)
            denominator = tf + self.k1 * (1 - self.b + self.b * (doc_length / self.avg_doc_length
            score += idf * (numerator / denominator)
        return score

    def rank_documents(self, query):
        query_terms = query.split()
        scores = [(doc_id, self.bm25_score(query_terms, doc_id)) for doc_id in range(self.doc_cou
        sorted_scores = sorted(scores, key=lambda x: x[1], reverse=True)
        return sorted_scores

# Example usage
corpus = [
    "The quick brown fox jumps over the lazy dog",
    "A quick brown dog outpaces a swift fox",
```

```
    "The dog is lazy but the fox is swift",
    "Lazy dogs and swift foxes"
]

bm25 = BM25(corpus)
query = "quick brown dog"
result = bm25.rank_documents(query)

print("BM25 Scores for the query '{}':".format(query))
for doc_id, score in result:
    print("Document {}: {}".format(doc_id, score))
```

此代码创建了一个简单的 BM25 类，通过给定的语料库计算查询与文档的相关性得分。

Python项目 44

Python项目 · 目录

上一篇
Python中的cls语法

下一篇
Python中的魔法方法