BM25Retriever检索器实现

原创 致Great ChallengeHub 2024年06月01日 23:57 北京

> 原理下一篇讲，先贴出代码

> https://github.com/gomate-community/GoMate/blob/main/gomate/modules/retrieval/bm25_retriever.py

```
import logging
import math
from multiprocessing import Pool, cpu_count
from typing import List,Dict
import jieba
import numpy as np
import tiktoken
from gomate.modules.retrieval.retrievers import BaseRetriever
jieba.setLogLevel(logging.INFO)

def tokenizer(text: str):
    return [word for word in jieba.cut(text)]
class BM25:
    def __init__(self, corpus, tokenizer=None):
        self.corpus_size = 0
        self.avgdl = 0
        self.doc_freqs = []
        self.idf = {}
        self.doc_len = []
        self.tokenizer = tokenizer

        if tokenizer:
            corpus = self._tokenize_corpus(corpus)
        nd = self._initialize(corpus)
        self._calc_idf(nd)

    def _initialize(self, corpus):
        nd = {}  # word -> number of documents with word
        num_doc = 0
        for document in corpus:
            self.doc_len.append(len(document))
            num_doc += len(document)

            frequencies = {}
            for word in document:
                if word not in frequencies:
                    frequencies[word] = 0
                frequencies[word] += 1
            self.doc_freqs.append(frequencies)

            for word, freq in frequencies.items():
                try:
                    nd[word] += 1
                except KeyError:
                    nd[word] = 1

            self.corpus_size += 1

        self.avgdl = num_doc / self.corpus_size
        return nd

    def _tokenize_corpus(self, corpus):
        pool = Pool(cpu_count())
        tokenized_corpus = pool.map(self.tokenizer, corpus)
        return tokenized_corpus

    def _calc_idf(self, nd):
        raise NotImplementedError()

    def get_scores(self, query):
        raise NotImplementedError()

    def get_batch_scores(self, query, doc_ids):
        raise NotImplementedError()

    def get_top_n(self, query, documents, n=5):

        assert self.corpus_size == len(documents), "The documents given don't match the index cor

        scores = self.get_scores(query)
        top_n = np.argsort(scores)[::-1][:n]
        return [{'text': documents[i], 'score': scores[i]} for i in top_n]


class BM25Okapi(BM25):
    def __init__(self, corpus, tokenizer=None, k1=1.5, b=0.75, epsilon=0.25):
        self.k1 = k1
        self.b = b
        self.epsilon = epsilon
        super().__init__(corpus, tokenizer)

    def _calc_idf(self, nd):
        """
        Calculates frequencies of terms in documents and in corpus.
        This algorithm sets a floor on the idf values to eps * average_idf
        """
        # collect idf sum to calculate an average idf for epsilon value
        idf_sum = 0
        # collect words with negative idf to set them a special epsilon value.
        # idf can be negative if word is contained in more than half of documents
        negative_idfs = []
        for word, freq in nd.items():
            idf = math.log(self.corpus_size - freq + 0.5) - math.log(freq + 0.5)
            self.idf[word] = idf
            idf_sum += idf
            if idf < 0:
                negative_idfs.append(word)
        self.average_idf = idf_sum / len(self.idf)

        eps = self.epsilon * self.average_idf
        for word in negative_idfs:
            self.idf[word] = eps

    def get_scores(self, query):
        """
        The ATIRE BM25 variant uses an idf function which uses a log(idf) score. To prevent negat
        this algorithm also adds a floor to the idf value of epsilon.
        See [Trotman, A., X. Jia, M. Crane, Towards an Efficient and Effective Search Engine] for
        :param query:
        :return:
        """
        score = np.zeros(self.corpus_size)
        doc_len = np.array(self.doc_len)
        for q in query:
            q_freq = np.array([(doc.get(q) or 0) for doc in self.doc_freqs])
            score += (self.idf.get(q) or 0) * (q_freq * (self.k1 + 1) /
                                               (q_freq + self.k1 * (1 - self.b + self.b * doc_len
        return score

    def get_batch_scores(self, query, doc_ids):
        """
        Calculate bm25 scores between query and subset of all docs
        """
        assert all(di < len(self.doc_freqs) for di in doc_ids)
        score = np.zeros(len(doc_ids))
        doc_len = np.array(self.doc_len)[doc_ids]
        for q in query:
            q_freq = np.array([(self.doc_freqs[di].get(q) or 0) for di in doc_ids])
            score += (self.idf.get(q) or 0) * (q_freq * (self.k1 + 1) /
                                               (q_freq + self.k1 * (1 - self.b + self.b * doc_len
        return score.tolist()
```

```
class BM25L(BM25):
    def __init__(self, corpus, tokenizer=None, k1=1.5, b=0.75, delta=0.5):
        # Algorithm specific parameters
        self.k1 = k1
        self.b = b
        self.delta = delta
        super().__init__(corpus, tokenizer)

    def _calc_idf(self, nd):
        for word, freq in nd.items():
            idf = math.log(self.corpus_size + 1) - math.log(freq + 0.5)
            self.idf[word] = idf

    def get_scores(self, query):
        score = np.zeros(self.corpus_size)
        doc_len = np.array(self.doc_len)
        for q in query:
            q_freq = np.array([(doc.get(q) or 0) for doc in self.doc_freqs])
            ctd = q_freq / (1 - self.b + self.b * doc_len / self.avgdl)
            score += (self.idf.get(q) or 0) * (self.k1 + 1) * (ctd + self.delta) / \
                     (self.k1 + ctd + self.delta)
        return score

    def get_batch_scores(self, query, doc_ids):
        """
        Calculate bm25 scores between query and subset of all docs
        """
        assert all(di < len(self.doc_freqs) for di in doc_ids)
        score = np.zeros(len(doc_ids))
        doc_len = np.array(self.doc_len)[doc_ids]
        for q in query:
            q_freq = np.array([(self.doc_freqs[di].get(q) or 0) for di in doc_ids])
            ctd = q_freq / (1 - self.b + self.b * doc_len / self.avgdl)
            score += (self.idf.get(q) or 0) * (self.k1 + 1) * (ctd + self.delta) / \
                     (self.k1 + ctd + self.delta)
        return score.tolist()


class BM25Plus(BM25):
    def __init__(self, corpus, tokenizer=None, k1=1.5, b=0.75, delta=1):
        # Algorithm specific parameters
        self.k1 = k1
        self.b = b
        self.delta = delta
        super().__init__(corpus, tokenizer)

    def _calc_idf(self, nd):
        for word, freq in nd.items():
            idf = math.log(self.corpus_size + 1) - math.log(freq)
            self.idf[word] = idf

    def get_scores(self, query):
        score = np.zeros(self.corpus_size)
        doc_len = np.array(self.doc_len)
        for q in query:
            q_freq = np.array([(doc.get(q) or 0) for doc in self.doc_freqs])
            score += (self.idf.get(q) or 0) * (self.delta + (q_freq * (self.k1 + 1)) /
                                               (self.k1 * (1 - self.b + self.b * doc_len / self.av
        return score

    def get_batch_scores(self, query, doc_ids):
        """
        Calculate bm25 scores between query and subset of all docs
        """
        assert all(di < len(self.doc_freqs) for di in doc_ids)
        score = np.zeros(len(doc_ids))
        doc_len = np.array(self.doc_len)[doc_ids]
        for q in query:
            q_freq = np.array([(self.doc_freqs[di].get(q) or 0) for di in doc_ids])
            score += (self.idf.get(q) or 0) * (self.delta + (q_freq * (self.k1 + 1)) /
                                               (self.k1 * (1 - self.b + self.b * doc_len / self.av
        return score.tolist()


class BM25RetrieverConfig:
    def __init__(self, tokenizer=None, k1=1.5, b=0.75, epsilon=0.25, delta=0.5, algorithm='Okapi'
        self.tokenizer = tokenizer
        self.k1 = k1
        self.b = b
        self.epsilon = epsilon
        self.delta = delta
        self.algorithm = algorithm

    def log_config(self):
        config_summary = """
      FaissRetrieverConfig:
       Tokenizer: {tokenizer},
       K1: {k1},
       B: {b},
       Epsilon: {epsilon},
       Delta: {delta},
       Algorithm: {algorithm},
      """.format(
            tokenizer=self.tokenizer,
            k1=self.k1,
            b=self.b,
            epsilon=self.epsilon,
            delta=self.delta,
            algorithm=self.algorithm,
        )
        return config_summary


class BM25Retriever(BaseRetriever):
    def __init__(self, config):
        self.tokenizer = config.tokenizer
        self.k1 = config.k1
        self.b = config.b
        self.epsilon = config.epsilon
        self.delta = config.delta
        self.algorithm = config.algorithm

    def fit_bm25(self, corpus):
        self.corpus=corpus
        if self.algorithm == 'Okapi':
            self.bm25 = BM25Okapi(corpus=corpus, tokenizer=self.tokenizer, k1=self.k1, b=self.b, e
        elif self.algorithm == 'BM25L':
            self.bm25 = BM25L(corpus=corpus, tokenizer=self.tokenizer, k1=self.k1, b=self.b, delta
        elif self.algorithm == 'BM25Plus':
            self.bm25 = BM25Plus(corpus=corpus, tokenizer=self.tokenizer, k1=self.k1, b=self.b, de
        else:
            raise ValueError('Algorithm not supported')

    def retrieve(self, query: str='',top_k:int=3) -> List[Dict]:
        tokenized_query = " ".join(self.tokenizer(query))
        search_docs = self.bm25.get_top_n(tokenized_query, self.corpus, n=top_k)
        return search_docs
```

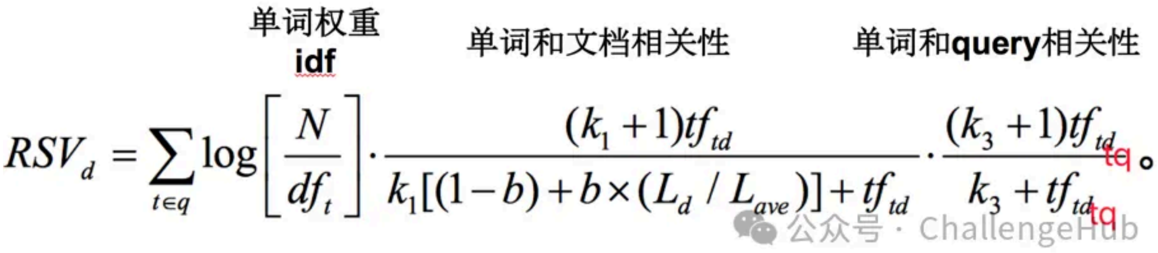
调用如下：

```
from gomate.modules.retrieval.bm25_retriever import BM25RetrieverConfig, BM25Retriever, tokenizer

if __name__ == '__main__':
    bm25_retriever_config = BM25RetrieverConfig(
        tokenizer=tokenizer,
        k1=1.5,
        b=0.75,
        epsilon=0.25,
        delta=0.25,
        algorithm='Okapi'
    )
    bm25_retriever = BM25Retriever(bm25_retriever_config)
```

```
corpus = [


]
new_files = [
    r'H:\Projects\GoMate\data\伊朗.txt',
    r'H:\Projects\GoMate\data\伊朗总统罹难事件.txt',
    r'H:\Projects\GoMate\data\伊朗总统莱希及多位高级官员遇难的直升机事故.txt',
    r'H:\Projects\GoMate\data\伊朗问题.txt',
]
for filename in new_files:
    with open(filename, 'r', encoding="utf-8") as file:
        corpus.append(file.read())
bm25_retriever.fit_bm25(corpus)
query = "伊朗总统莱希"
search_docs = bm25_retriever.retrieve(query)
print(search_docs)
```



$$RSV_d = \sum_{t\in q} \log\left[\frac{N}{df_t}\right] \cdot \frac{(k_1+1)tf_{td}}{k_1[(1-b)+b\times(L_d/L_{ave})]+tf_{td}} \cdot \frac{(k_3+1)tf_{td}}{k_3+tf_{td}}。$$

RAG 45

RAG · 目录

上一篇

tree2retriever:面向RAG场景的递归摘要树检索器实现

下一篇

【RAG入门必备技能】Faiss框架使用与FaissRetriever实现

内容由AI生成