# X-BM25 实现说明文档

本文档详细说明了基于 Python 实现的 BM25 搜索算法的结构、功能和关键组件。该实现具有模块化、可重用的设计，支持英文和中文文本搜索，并且具备良好的扩展性。

## 概述

BM25 是一种常用于信息检索的概率排序算法，它根据文档与查询的相关性对文档进行评分。本实现包括以下几个主要组件：

- **抽象基类**： `AbstractBM25` 定义了核心 BM25 逻辑。
- **语言特定子类**： `EnglishBM25` 和 `ChineseBM25` 分别实现了英文和中文的分词及预处理逻辑。
- **工厂函数**： `create_bm25` 和便捷搜索函数 `bm25_search` 提供了简单的接口进行实例化和搜索操作。

## 代码结构

代码组织如下：

- **抽象基类**： `AbstractBM25` 定义了 BM25 算法的核心方法及其基本实现。
- **具体类**： `EnglishBM25` 和 `ChineseBM25` 实现了各自的语言特定分词逻辑。
- **工厂和工具函数**： `create_bm25` 和 `bm25_search` 提供了简便的接口来创建实例并执行搜索。

## 依赖

- `abc` 和 `abstractmethod`： 用于定义抽象基类。
- `typing.List`： 用于类型提示。
- `math`： 用于 BM25 评分计算中的对数运算。
- `jieba`： 中文分词库。
- `PyStemmer`： 英文词干提取库。
- `re`： 用于英文文本的正则表达式预处理。
- `json` 和 `pickle`： 用于保存和加载索引。
- `stopwords`： 自定义的停用词模块，支持英文和中文。

# 类详情

## AbstractBM25

`AbstractBM25` 是 BM25 算法的抽象基类，负责实现核心的 BM25 逻辑。

### 构造函数

```
def __init__(self, corpus: List[str], k1: float = 1.5, b: float = 0.75, stopwords: tuple = ()):
```

参数：

- `corpus` ： 文档字符串列表。
- `k1` ： 控制词频饱和度的参数，默认值为 1.5。
- `b` ： 控制文档长度归一化的参数，默认值为 0.75。
- `stopwords` ： 停用词元组，默认值为空。

行为：初始化文档集合，计算文档长度、平均长度，并构建词频（TF）和文档频率（DF）索引。

异常：若 `corpus` 为空，抛出 `ValueError` 。

### 抽象方法

```
@abstractmethod
def _tokenize(self, text: str) -> List[str]:
```

子类必须实现此方法，定义语言特定的分词逻辑。

### 核心方法

- `_tokenize_corpus(self)` ： 使用 `_tokenize` 对整个文档集合进行分词。
- `_build_index(self)` ： 构建 TF （ `self.tf` ）和 DF （ `self.df` ）索引。
  - `self.tf` ： 列表，记录每个文档中每个词的频率。
  - `self.df` ： 字典，记录每个词在多少个文档中出现过。
- `_score(self, query_tokens: List[str], doc_id: int) -> float` ： 计算查询与某文档的 BM25 分数。

公式：

$$\sum(\text{IDF} \times \text{TF\_part})$$

其中：

$$\text{IDF} = \log\left(\frac{N - \text{DF} + 0.5}{\text{DF} + 0.5} + 1\right)$$

$$\text{TF\_part} = \frac{\text{freq} \times (k1 + 1)}{\text{freq} + k1 \times (1 - b + b \times \frac{\text{doc\_len}}{\text{avg\_doc\_len}})}$$

- `search(self, query: str, top_k: int = 5) -> List[tuple]`：对查询进行分词，计算所有文档的分数，返回前 `top_k` 个结果（格式为 `(doc_id, score)` 元组）。
- `save(self, filepath: str)`：将索引保存为 JSON 或 Pickle 文件。
- `load(cls, filepath: str, corpus: List[str])`：类方法，从文件中加载索引并重建 BM25 实例。

# 语言特定子类

### `EnglishBM25`

继承自 `AbstractBM25`，实现英文特定的分词逻辑。

#### 构造函数

```
def __init__(self, corpus: List[str], k1: float = 1.5, b: float = 0.75, stopwords: tuple = STOPW
```

初始化英文词干提取器（使用 `PyStemmer`）并使用默认英文停用词。

#### 分词方法

```
def _tokenize(self, text: str) -> List[str]:
```

步骤：

1. 使用正则表达式去除非字母数字字符并转换为小写。
2. 将文本分割为单词。
3. 使用 `PyStemmer` 进行词干提取。
4. 过滤停用词。

返回：处理后的词列表。

### `ChineseBM25`

继承自 `AbstractBM25`，实现中文特定的分词逻辑。

**构造函数**

```
def __init__(self, corpus: List[str], k1: float = 1.5, b: float = 0.75, stopwords: tuple = STOPW
```

使用默认中文停用词。

**分词方法**

```
def _tokenize(self, text: str) -> List[str]:
```

步骤：

1. 使用 `jieba.cut` 进行中文分词。
2. 过滤停用词。

返回：处理后的词列表。

# 工厂和工具函数

### `create_bm25`

```
def create_bm25(corpus: List[str], language: str, k1: float = 1.5, b: float = 0.75, stopwords: t
```

用途：根据指定语言创建 `EnglishBM25` 或 `ChineseBM25` 实例。

参数：

- `language`： "english"/"en" 或 "chinese"/"cn"。
- `stopwords`： 可选的自定义停用词，默认使用语言特定的停用词。

异常：不支持的语言会抛出 `ValueError`。

### `bm25_search`

```
def bm25_search(corpus: List[str], query: str, language: str, top_k: int = 5, k1: float = 1.5, b
```

用途：一键执行 BM25 搜索操作。

返回：前 `top_k` 个结果的 `(doc_id, score, document_text)` 元组列表。

**load_bm25**

```python
def load_bm25(filepath: str, corpus: List[str]):
```

用途：加载索引。

返回：BM25 对象。

# BM25 公式

BM25 基于词频（TF）和文档频率（DF）对文档进行评分。评分公式如下：

$$\text{Score}(d, q) = \sum_{t \in q} \text{IDF}(t) \times \text{TF}_{t,d}$$

其中：

- $d$ 是文档，
- $q$ 是查询。
- $\text{IDF}(t)$ 是逆文档频率（Inverse Document Frequency），衡量词 $t$ 在语料库中的稀有性。
- $\text{TF}_{t,d}$ 是词 $t$ 在文档 $d$ 中的词频，经过归一化处理。

## 1. 逆文档频率（IDF）

逆文档频率（IDF）衡量词 $t$ 在语料库中的稀有性，公式为：

$$\text{IDF}(t) = \log \left( \frac{N - \text{DF}(t) + 0.5}{\text{DF}(t) + 0.5} + 1 \right)$$

其中：

- $N$ 是文档总数。
- $\text{DF}(t)$ 是词 $t$ 出现的文档数。

## 2. 词频部分（TF）

词频部分对词 $t$ 在文档 $d$ 中的频率进行归一化处理，公式如下：

$$\mathrm{TF}_{t,d} = \frac{f_{t,d} \times (k1 + 1)}{f_{t,d} + k1 \times \left(1 - b + b \times \frac{\mathrm{len}(d)}{\mathrm{avg\_len}}\right)}$$

其中：

- $f_{t,d}$ 是词 $t$ 在文档 $d$ 中的出现次数。
- $\mathrm{len}(d)$ 是文档 $d$ 的长度（即词数）。
- $\mathrm{avg\_len}$ 是整个语料库的平均文档长度。
- $k1$ 和 $b$ 是调整参数，用于控制词频的饱和度和文档长度的影响。

## 3. 总评分

最终的 BM25 分数是所有查询词的 IDF 和 TF 的加权和：

$$\mathrm{Score}(d, q) = \sum_{t \in q} \left( \log \left( \frac{N - \mathrm{DF}(t) + 0.5}{\mathrm{DF}(t) + 0.5} + 1 \right) \times \frac{f_{t,d} \times (k1 + 1)}{f_{t,d} + k1 \times \left(1 - b + b \times \frac{\mathrm{len}(d)}{\mathrm{avg\_len}}\right)} \right)$$

# 使用方法

```python
from bm25 import load_bm25, create_bm25
import os

# 测试代码
if __name__ == "__main__":
    output_dir = 'test_index_outputs'
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    # 英文测试
    english_corpus = [
        "this is a sample document about machine learning",
        "machine learning is fascinating and useful",
        "this document discusses deep learning techniques",
        "another sample about artificial intelligence"
    ]
    english_query = "machine learning"

    # 创建并保存为 JSON
    bm25_en_json = create_bm25(english_corpus, 'english')
    bm25_en_json.save(os.path.join(output_dir, 'bm25_en.json'))

    # 从 JSON 加载并搜索
    loaded_bm25_en_json = load_bm25(os.path.join(output_dir, 'bm25_en.json'), english_corpus)
    print("英文查询（JSON加载）:", english_query)
    results_json = loaded_bm25_en_json.search(english_query, top_k=3)
    for doc_id, score in results_json:
        print(f"文档ID: {doc_id}, 得分: {score:.4f}, 文本: {english_corpus[doc_id]}")

    # 创建并保存为 Pickle
    bm25_en_pkl = create_bm25(english_corpus, 'english')
    bm25_en_pkl.save(os.path.join(output_dir, 'bm25_en.pkl'))

    # 从 Pickle 加载并搜索
    loaded_bm25_en_pkl = load_bm25(os.path.join(output_dir, 'bm25_en.pkl'), english_corpus)
    print("\n英文查询（Pickle加载）:", english_query)
    results_pkl = loaded_bm25_en_pkl.search(english_query, top_k=3)
    for doc_id, score in results_pkl:
        print(f"文档ID: {doc_id}, 得分: {score:.4f}, 文本: {english_corpus[doc_id]}")
```

```
print("\n")

# 中文测试
chinese_corpus = [
    "这是一个关于机器学习的样本文档",
    "机器学习既迷人又实用",
    "本文档讨论深度学习技术",
    "另一个关于人工智能的样本"
]
chinese_query = "机器学习"

# 创建并保存为 JSON
bm25_cn_json = create_bm25(chinese_corpus, 'chinese')
bm25_cn_json.save(os.path.join(output_dir, 'bm25_cn.json'))

# 从 JSON 加载并搜索
loaded_bm25_cn_json = load_bm25(os.path.join(output_dir, 'bm25_cn.json'), chinese_corpus)
print("中文查询（JSON加载）:", chinese_query)
results_json = loaded_bm25_cn_json.search(chinese_query, top_k=3)
for doc_id, score in results_json:
    print(f"文档ID: {doc_id}, 得分: {score:.4f}, 文本: {chinese_corpus[doc_id]}")

# 创建并保存为 Pickle
bm25_cn_pkl = create_bm25(chinese_corpus, 'chinese')
bm25_cn_pkl.save(os.path.join(output_dir, 'bm25_cn.pkl'))

# 从 Pickle 加载并搜索
loaded_bm25_cn_pkl = load_bm25(os.path.join(output_dir, 'bm25_cn.pkl'), chinese_corpus)
print("\n中文查询（Pickle加载）:", chinese_query)
results_pkl = loaded_bm25_cn_pkl.search(chinese_query, top_k=3)
for doc_id, score in results_pkl:
    print(f"文档ID: {doc_id}, 得分: {score:.4f}, 文本: {chinese_corpus[doc_id]}")
```

# 依赖

- Python 3.10+
- jieba（用于中文分词）
- PyStemmer（用于英文词干提取）

# 许可证

MIT 许可证