lab4

1. 目录结构

```
lab4/
│  functions.py              // 主调用程序
│  main.py                   // 可运行程序
│
├─article                    // 存放文档集
│
├─model                      // 存放预训练模型
│    ├─bert-base-uncased     // BERT
│    │      config.json
│    │      model.safetensors
│    │      tokenizer.json
│    │      tokenizer_config.json
│    │      vocab.txt
│    │
│    └─roberta-base          // RoBERTa
│        │  merges.txt
│        │  vocab.json
│        │
│        └─roberta-base-model
│               config.json
│               pytorch_model.bin
│
└─result                     // 存放实验中产生的结果文件
       inverted_index.pkl    // 倒排记录表
       word_embedding_bert.pkl    // BERT 词嵌入
       word_embedding_roberta.pkl    // RoBERTa 词嵌入
```

2. 实现细节

2.1. task1

- `read_file(path)`：**读取文档**

  对于文档集中的 20738 个文档逐个读取，逐行读取文本，去除换行符和空行。

- `segment_word(articles)`：**分词**

  调用 jieba 库，对于每个文档的每个句子进行分词，对分词结果进行去除标点，数字和单字的处理。

- `build_inverted_index(segmentation, file_path)`：**建立倒排索引**

  每个词的倒排索引都由一个列表保存。在建立过程中，顺序访问每篇文章的分词结果，得到的倒排索引即已按照 docID 升序排列。最后以”词 – 倒排索引“的形式保存为 pickle 文件，每一行为一个词和它的倒排索引列表。

2.2. task2

- `embedding(segmentation, file_path)`：**词语 Embedding**

  合并所有文档的分词结果并去重，然后使用 BERT 预训练模型对每个词生成词嵌入向量，最后以”词 – 词嵌入向量“的形式保存为 pickle 文件，每一行为一个词和它的词嵌入向量。

- `SimilarK`：**存放相似词**

  采用最小堆的数据结构存放最相似的 Top K 个词。

- `find_similar_words(keyword, k, vocab)`：**寻找相似词**

  对于词典中的每个词，计算其与输入关键词的词嵌入向量之间的余弦相似度。使用 `SimilarK` 存放最相似的 Top K 个词。

2.3. task3

- `load_vocab(file_path)`：**加载实验中产生的结果文件**

  加载 task1 保存的倒排记录表或 task2 保存的词嵌入向量 pickle 文件。

- `exact_match(keyword, embedding, inverted_index)`：**and 检索**

  检索同时包含输入关键词以及与其语义相似度最高的 Top 2 个词的文章的 docID。具体实现为，首先寻找与输入关键词语义相似度最高的 Top 2 个词，再在倒排记录表中找到这三个词的倒排索引列表，并计算它们的文档频率。此处的文档频率（DF）其实并不是”频率“，由于对于每个词而言语料库中的文档总数是相同的，所以可以省去除以文档总数的步骤，计算公式如下：

  $$DF(t, D) = \text{语料库 } D \text{ 中包含词 } t \text{ 的文档数}$$

  所以对于一个词，它的文档频率等于它的倒排索引列表的长度。

然后处理多个词项的倒排索引列表的 and 合并。先将这些词项按照文档频率升序排列，然后每次从文档频率最小的两个词项开始合并。合并过程中，首先取出两个词项的倒排索引列表，然后对每个倒排索引列表分别维护一个定位指针，两个指针都是从前往后扫描，每次比较当前指针对应倒排索引，即 docID。如果两个 docID 相同，则将其加入检索结果，然后同时向后移动两个指针；如果两个 docID 不同，则向后移动指向较小 docID 的那个指针。

此处还实现了跳表，采用均匀放置策略，即对于长度为 L 的倒排索引列表，从列表首部开始每隔 $\sqrt{L}$ 长度放一个跳表指针。

- `respond(docIDs)`：**返回检索结果**

  如果检索到的 docID 数量为 0，则打印“抱歉，没有与此相关的检索结果”；否则，先将检索到的 docID 排序，然后依次打印出来，同时打印检索到的 docID 的总数。

- `if __name__ == '__main__':`：**main 函数**

  使用一个 while (true) 循环，首先接收用户输入的关键词，然后询问用户是否想要查看与输入词的语义相似度最高的 Top K 个相似词（是/否），如果用户输入“是”，则请用户设置 K 的值，然后调用 `find_similar_words` 函数进行寻找，寻找完成后返回寻找结果。接下来记录检索开始时间，并调用 `exact_match` 函数进行检索，检索完成后返回检索结果，记录检索结束时间，计算检索时间，打印本次检索用时，最后询问用户是否想要继续检索（是/否），如果用户输入“否”，则跳出循环，否则重复该流程。

3. 创新点

- 使用 RoBERTa 预训练模型对词语进行 Embedding，并基于余弦相似度衡量词语编码向量之间的相似性，在寻找相似词任务上取得了较好效果。
- 隐式实现跳表，节省了存储空间和 I/O 开销。均匀放置策略可以忽略查询词项的分布情况，并且在该任务中索引相对静态，可以很简便地实现均匀放置。
- 进行了查询优化，大幅降低了检索时间。

```
################################################################

robot：您好，请输入关键词
user：革命
robot：请问您是否想要查看与输入词的语义相似度最高的 Top K 个相似词（是/否）
user：是
robot：好的，请设置 K 的值
user：2
robot：与输入词的语义相似度最高的 Top 2 个相似词： 革命史 革命家
robot：本次检索用时 2.5915563106536865 秒
robot：好的，接下来将为您检索同时包含输入词及其语义相似度最高的 Top 2 个词的所有文档
robot：共 1 个检索结果： 11418.txt

robot：本次检索用时 2.5789809226989746 秒

robot：请问您是否想要继续检索（是/否）
user：是

################################################################
```