Lab4 倒排索引与相似词查询

10235501419 李佳亮

本次实验使用python语言，实现对20738个文档的词语检索。本次实验结合了跳表优化的倒排索引，用BERT预训练模型模型对词进行向量编码，以及用FAISS构建的向量索引来实现相似向量快速检索，实现Top 2相似词的文档查询功能。

实验过程

1. 分词

本部分将用jieba分词库对20738个文档进行分词，同时去除标点、数字、单字。分词后得到一个字典对象，key为文档编号，value为文档分词后得到的列表。

考虑到每次运行代码时，原始文档集是静态不变的，第一次运行（冷启动时）我们把分词结果的字典缓存到`tokenized_docs.json`文件中，这样后续运行时可以直接加载该文件（热启动），大大减少了检索系统的初始化时间。

2. 语义向量构建类（基于BERT编码）

下面，我们实现一个类`ChineseWordEncoder`，它使用Hugging Face社区上的`hfl/chinese-roberta-wwm-ext`预训练模型作为中文语义编码模型。该类实现一个方法`get_vector`，它接收一个中文词语并将其编码为语义向量后返回。

> **Note**
>
> BERT不是一个从词到向量的静态一一映射，而是一个计算模型。BERT对上下文敏感，它会根据输入的具体词序列来实时计算语义向量。因此，同一个词，在不同的上下文中，BERT给出的语义向量也不同。但本次实验中，我们送入BERT模型的都是孤立的词，这样的话BERT也是把它tokenize为`[CLS] + word + [SEP]`这一结构走一次模型做推理得到的。`[CLS]`是BERT在设计上是用来表示整个输入序列语义的，因此最后返回的是第`0`个token的向量，也就是`[CLS]`向量。

3. 相似向量快速检索（基于FAISS）

我们在文档中所有词汇中查找某个词的相似词汇时，需要把所有出现过的词都送入BERT编码成向量。为了防止程序后面多次运行时做重复的编码工作，我们要设计一套逻辑来保存这些词的语义向量。

- 首先，我们把所有文档中出现过的词按某一特定顺序排列，保存在`word_list.json`中。
- 接着，用BERT来逐个将`word_list.json`中的词进行编码。由于词的个数接近十万，对其进行向量化需要消耗大量的时间与空间，为了防止过程中出现中断，我们考虑在对词进行向量化的过程中将词向量保存在本地。我们用SQLite创建一个数据库文件`word_vectors.db`来存储词向量，他本质就是一个静态的键值对映射（中文词汇->该词的BERT编码向量），相较于用json保存，它更节省空间，可以按需加载与插入。【这一步不是必要的，在后续中其实用不到；如果是实际工程上数据集动态不断变动，用数据库存储会很方便更新】
- 最后我们使用FAISS构建一个快速相似向量检索引擎。给定向量组和查询向量，FAISS对象可以快速查询Top K个与之相似度最高的向量索引。
  变量`vecs`按`word_list.json`中的词的顺序存储有所有词编码的向量。
  - 用`faiss.IndexFlatL2(vecs.shape[1])`初始化一个以L2范数（欧几里得距离）为标准的扁平索引对象。

- 调用`add(vecs)`方法向扁平索引对象添加向量组，扁平索引对象将在这个向量组中进行相似向量的检索。
- 输入一个向量组`vectors`和数字`k`，`search(vectors, k)`方法可以返回一个二元组`(dist, indices)`，其中`indices`保存向量组中每个向量最近的k个向量的索引，`dist`保存向量组中每个向量最近的k个向量与之的距离，形状均为`(|vectors|, k)`。由于word_list中词的顺序和FAISS对象中词向量的顺序相同，`indices`返回的索引就可以通过word_list来映射回词。

为了避免多次运行时FAISS索引的重复构建，我们调用`faiss.write_index`把它也保存在本地的`faiss.index`中。

4. 倒排索引类

接下来构建起倒排索引类。类中维护一个字典对象`index`，他是词语到倒排表的映射。维护一个FAISS索引对象`faiss_index`，它用于查询Top K相似的向量。

`build`方法把3中的工作包装了起来，即接收1中分词后得到的字典、2中的语义向量构建类对象，做构建倒排索引、构建word_list、用BERT将所有词编码为向量、构建起FAISS索引对象的工作。

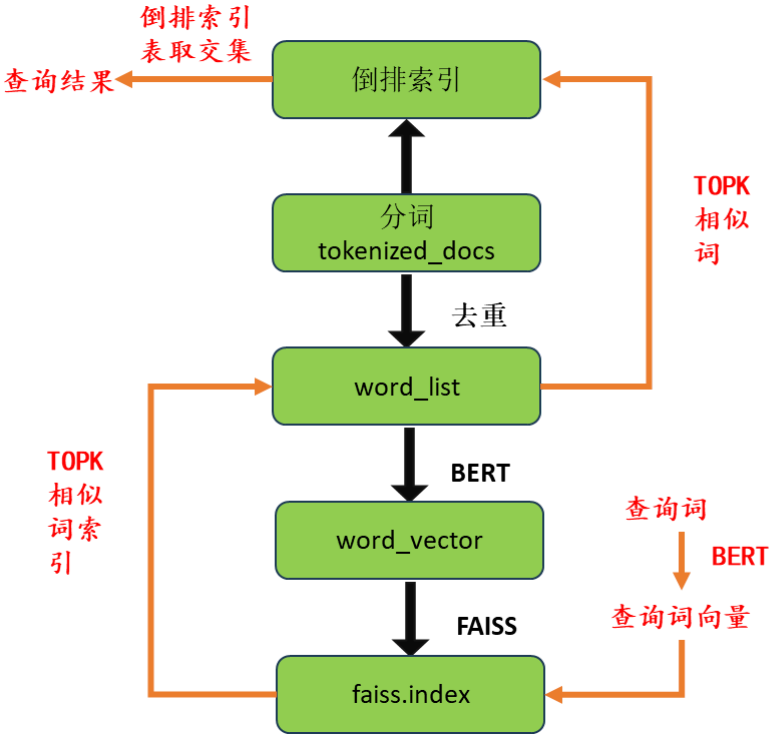
`intersect`方法接收两个列表，以跳表方式对其取交集。采用根号L策略。其逻辑为：

- 初始化：
  - `list1` 和 `list2` 是升序的posting list。
  - `i`，`j` 分别为两个列表的当前遍历位置。初始化为列表首位。
  - `skip1`，`skip2` 为跳跃步长，取列表长度的平方根。
- 循环：
  - 如果`list[i] == list[j]`，说明找到了交集元素，加入 `result`，两个指针都向前推进一位。
  - 若当前 `list1[i]` 小于 `list2[j]`，意味着 `list1[i]` 不可能出现在交集中。
    - 尝试跳跃前进 `skip1` 步，加速跳过多个无效元素。
    - 若跳跃位置超过`list2[j]`，则回退，只向前推进一步。
  - 若当前 `list1[i]` 大于 `list2[j]`，同理。
- 最后即可返回交集结果，注意指针不要超过列表边界。

定义成员函数`top_k_similar`，它使用FAISS索引对象搜索与给定词向量相似度最高的`k+1`个词向量索引与距离（因为一个词向量一定与自身最相似），通过word_list将索引映射回词，并返回`k`个二元组`(word, distance)`即相似度最高的`k`个词语与其相似度距离。

在查询函数`search`中，我们可以在倒排索引中获得Top 2词汇的倒排索引表，以及查询词本身的倒排索引表。接着我们用跳表来做这个交集操作。注意，交集操作中我们可以首先对两个较小的列表先取交集，再把结果和第三个列表取交集。

构建逻辑以及查询逻辑可以用下面这张图简单表示。（真实的代码结构与之有出入，但是逻辑是图中所示）



实验结果

若文档过多，仅展示了前10篇文档。

```
=== 🔎 查询关键词：互联网 ===
查询词包含文档数：439，示例:[8, 93, 106, 122, 126, 163, 180, 221, 235, 251]
Top2相似词：
  1. 互联网络 (L2距离28.31)，出现于2篇文档，示例:[15433, 19630]
  2. 互联网站 (L2距离28.79)，出现于1篇文档，示例:[9514]
交集文档数：0，示例:[]
查询耗时：0.256秒
=== 🔎 查询关键词：经济 ===
查询词包含文档数：3040，示例:[2, 5, 13, 28, 29, 30, 32, 44, 47, 65]
Top2相似词：
  1. 非经济 (L2距离40.02)，出现于1篇文档，示例:[17892]
  2. 经济账 (L2距离40.52)，出现于2篇文档，示例:[10483, 18906]
交集文档数：0，示例:[]
查询耗时：0.091秒
=== 🔎 查询关键词：美国 ===
查询词包含文档数：1752，示例:[28, 39, 62, 67, 77, 110, 126, 136, 140, 152]
Top2相似词：
  1. 美等国 (L2距离36.43)，出现于1篇文档，示例:[29]
  2. 美两国 (L2距离39.21)，出现于4篇文档，示例:[4373, 16508, 18173, 18192]
交集文档数：0，示例:[]
查询耗时：0.080秒
=== 🔎 查询关键词：消费 ===
查询词包含文档数：265，示例:[90, 140, 250, 311, 381, 395, 415, 513, 538, 547]
Top2相似词：
  1. 消费观 (L2距离43.97)，出现于1篇文档，示例:[10196]
  2. 消费量 (L2距离44.27)，出现于8篇文档，示例:[3433, 4317, 10131, 10196, 11781, 11782, 17302, 18685]
交集文档数：0，示例:[]
查询耗时：0.154秒
=== 🔎 查询关键词：军队 ===
查询词包含文档数：396，示例:[2, 5, 91, 106, 144, 156, 178, 308, 453, 528]
Top2相似词：
  1. 军人 (L2距离31.02)，出现于124篇文档，示例:[453, 549, 605, 643, 655, 673, 762, 800, 1627, 1778]
  2. 部队 (L2距离31.51)，出现于477篇文档，示例:[1, 2, 15, 91, 144, 178, 183, 316, 371, 404]
交集文档数：10，示例:[453, 3818, 3887, 3919, 3960, 7723, 11399, 13479, 16498, 20654]
查询耗时：0.097秒
```

功能扩展

用栈的思路实现了中缀表达式向后缀表达式的转换以及后缀表达式的计算，支持复杂布尔表达式的查询。

```
输入表达式: NOT 经济 AND (互联网 AND 消费)
后缀表达式: ['经济', 'NOT', '互联网', '消费', 'AND', 'AND']
匹配文档数: 4
文档ID样例: [19658, 18827, 15702, 18847]
输入表达式: 经济 AND 互联网 AND 消费
后缀表达式: ['经济', '互联网', 'AND', '消费', 'AND']
匹配文档数: 21
文档ID样例: [18179, 1672, 18700, 6960, 14388, 5430, 15418, 16700, 3260, 1472]
输入表达式: 互联网 AND 消费
后缀表达式: ['互联网', '消费', 'AND']
匹配文档数: 25
文档ID样例: [18179, 1672, 18827, 18700, 18847, 6960, 14388, 5430, 15418, 16700]
```

总结

本次实验简单实现了一个具备“语义相似词扩展能力”的文档检索系统。

倒排表交集用根号L策略的跳表进行了优化，实现加速效果。

探索学习了BERT和FAISS的简单使用：用BERT对词语进行向量编码，用Facebook开源的 FAISS工具库构建向量索引，进行快速的相似向量检索，实现了TOP K相似词的查询。

用栈实现了布尔表达式的解析与求值，支持复杂布尔表达式的查询。

附录（GitHub链接）

- Main.ipynb - 实验代码
- cache/ - 保存的本地缓存文件