文本检索-作业报告

北京大学 2021-2022 春季《Python 程序设计与数据科学导论》

姓名: 李思哲　　学号: 1900013061

学院: 北京大学　信息科学技术学院　　日期: 2022.5.6

1 任务描述

本次任务主要内容为实现文本检索：即给定一个文本数据集，用户输入检索词，运行程序返回与之相关的文章，同时结合网络通信，以客户端、服务器两部分实现检索任务。

任务中完成了以下功能：

1. 数据处理：处理数据集，提取词根并构建词典
2. 检索排序：基于 server-client 架构进行文档检索，主要实现客户端与服务器的通信
3. 排序优化：构建文档-词的 TF-IDF 矩阵，得到文档向量，优化排序
4. 检索结果评估：评估检索结果与文章类别的关系
5. 构建相似词语列表并对文档进行模糊匹配

2 功能实现情况

项目已上传至https://github.com/sizhelee/TextSearch,具体运行方式及数据处理代码可参考 README。

2.1 数据处理

数据处理的代码保存在`preprocess.py`，主要完成了 6 部分工作：

1. 将 Server 端的每一条新闻转化成 txt 格式存储，保存为`"./server_files/xxx.txt"`，便于进行服务器与客户端之间的通信
2. 读取英文数据集，划分单词，提取词根并构建词典，保存为`"./server_files/vocab.csv"`，便于读取文件时保持单词格式
3. 构建文档-词的 TF-IDF 向量
4. 根据文档-词的 TF-IDF 矩阵，利用每一行作为文档向量，并利用 PCA 进行降维得到文档向量，保存为`"./server_files/file_feats_1000.npy"`
5. 利用文档的特征向量，计算文档之间的相似度 (2225*2225)，保存为`"./server_files/similarity.npy"`，便于找到关联文档
6. 计算词向量：首先计算词-词共现矩阵得到词语之间的关联，作为原始词向量，并进行降维得到最终的词向量，保存为`"./server_files/word_vec.npy"`，便于为每个单词寻找相似词
7. 计算单词之间的相似度,为每个单词找到最相似的 5 个单词作为相似词,相似词信息保存为`"./server_files/synonym.`

2.2 检索排序

这一部分主要完成 S/C 框架的搭建以及最简单的检索排序算法。

S/C 框架采用 socket 作为接口，进行了简单的握手操作以保证鲁棒性，防止数据传输出现错误，具体信息传递方式如图 1。完成套接字的连接后，客户端将收到的查询词传递给服务器，服务器处理得到结果后，将结果文档的 id 传递给客户端，并逐个发送文档。对于每个文档，服务器首先生成报头，包括文档的长度信息，并将报头和文档内容发送给客户端。考虑到 buffer 的最大长度，客户端接收文件时每次接收比特数有限，故采用 ACK 报文表示客户端已完成当前文档的接收。服务器接收 ACK 后发送 NOF 表示当前不是最后一篇文档，或发送 FIN 表示已完成所有文档的传输。

最基础的检索排序算法利用检索词在文本和标题中的出现次数对文档进行打分和排序，得分高于设定阈值的文档进入结果集合。

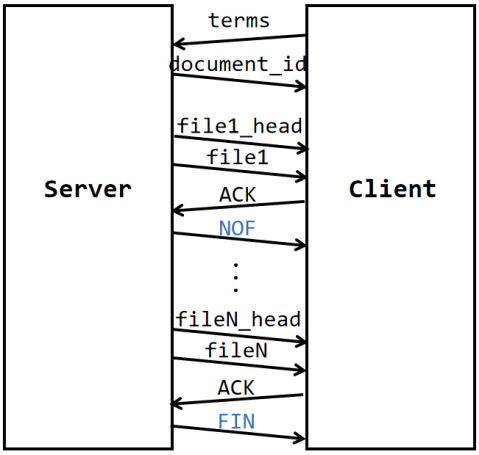

Figure 1: Server Client 信息传递框架

2.3 排序优化

排序优化的操作保存在`local_server.py/text_search`函数中，预处理文件可在初始化时直接读入。排序优化使用了 HITS 算法，具体步骤如下：

1. 采用上一步中提到的基础检索排序算法命中一组文章
2. 利用文章之间的相似度矩阵，找到有关联的（相似度高的）文章加入结果集合
3. 求出新的文章集合的主特征向量，函数保存在`utils/util.py/cal_feat_vec`中
4. 计算每篇文章与该特征向量的相似度，根据相似度对文章进行排序，相似度大于设定阈值的文章即为结果文章

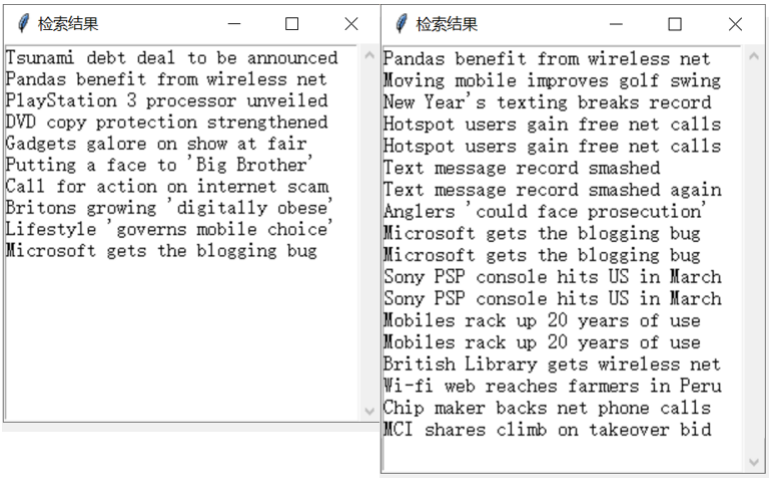

Figure 2: 两种检索架构的结果对比，输入“math”和“technology”

2.4 检索结果测评

对于检测结果，我们主要关注两个方面的表现，即 expertise（专业性）和 diversity（多样性）。我们希望检索返回的排名较前的文章都具有相同的主题，而检索结果中也应具有一定的多样性，即尽可能包含更多种类的文章，因此我们提出了上述两个评价指标其中 expertise 衡量前 K 篇文章中出现最多的类别的占比，diversity 表示涵盖所有文章类别所需的最少的文章数目。具体的计算方式为：

$$S_{exp} = max(times\_of\_each\_class)/K \tag{1}$$

$$S_{div} = (\frac{class\_num}{min(index\_contains\_all\_classes)})^t \tag{2}$$

其中，K 表示排名前 K 的文章，由检索得到结果的数量自动转换生成，t 为控制多样性得分的参数。

2.5 相似词及模糊匹配

相似词的计算在数据处理部分已经阐述。对于模糊匹配，我们在 UI 界面中增加的“模糊匹配”的按钮选项。选择模糊匹配模式，对每一个查询的单词寻找其相似词，更新查询词列表，并利用 HITS 方式进行检索。

3 检索算法

对比简单查找算法和 HITS 优化后的检索算法，我们发现优化算法倾向于返回更多的相关文章，其主要原因在于优化算法不仅考虑查询词在本文中出现的概率，也考虑到文章之间的相似性，具有全局的信息，因而能够得到更全面的相关关系。

此外，优化算法的另一优点在于可通过多个参数的调节，控制检索精度以及外延的广度。

4 性能评估及可视化

我们选取一些检索结果绘制词云图及 expertise 和 diversity 得分进行可视化，结果如图 3所示。

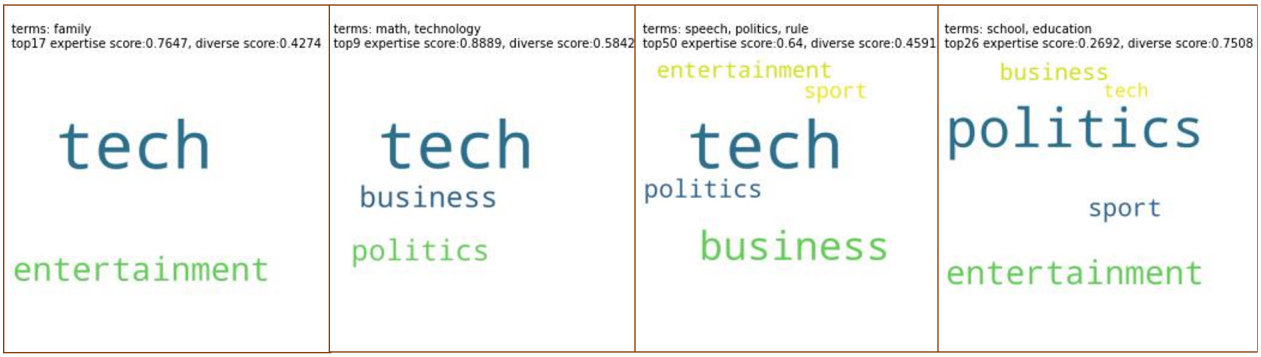

Figure 3: 搜索结果词云图可视化对比

我们发现对于大多数的检索结果，expertise 的得分在 0.6 之上，证明模型的检索结果与文本的分类相契合。与此同时，我们发现某种程度上 expertise 和 diversity 的得分呈现负相关，即说明需要通过参数的调节在多样性与专业性进行更好的权衡。从理论上分析，二者负相关的另一可能原因是 K 值的选取不恰当。选择适当的 K 值既可以保证前 K 篇文章具有较高专业性，同时，所有检索结果也具有丰富的多样性。当然，手动选择 K 值总能达到以上效果，而如何自动生成 K 值则有待进一步研究。

5 总结及展望

在本次作业中，我们利用基于 Tkinter 的 SC 架构和 HITS 算法实现了一个简易的文本检索模型，并在此基础上实现了模糊匹配功能，提出 expertise 和 diversity 两项测试指标进行性能评测。由于时间仓促，本次作业尚存在一些值得未来继续探讨的地方：（1）模型中包含较多参数，如何调节参数使得模型在专业性和多样性中达到最佳平衡（2）如何选取 K 值以最大化两项得分（3）采用更多种方式（例如 WordsVec，Bert Encoder 等）生成词向量及文章向量，利用预训练模型的优势得到更好的检索结果（4）优化 SC 结构中的信息传递方式，使得用户在点击浏览全文时才需下载文档全部内容