实验报告

实验部分

题目及要求

信息检索。给定若干数量的文档和查询条件，请为每个查询条件返回前20条最相关的文档。
其中，每条文档与查询条件的相关性评级取值为{0, 1, 2, 3}，3为最相关，0为不相关。
返回结果将通过F1值与NDCG@20进行评价。详细题目见 `doc/实验题目.pdf`

小组成员

- PB16001749 伊昕宇
- PB16001848 方泽瑜

实验结果

PB16001749_伊昕宇_PB16001848_方泽瑜_lab1_submission_8.csv	0.8149518328235144	0.8766894921416802

数据文件在 `doc/submission.pdf`

项目部分

项目结构

源代码在 `SearchEngine/` 中，使用 **C++**，VS2019 开发环境

解决方案 `SearchEngine` 包括了三个项目：

- `DocumentSearch` 项目：主要用于输入文档 id 查询训练集文档以及 tfidf 值
- `SearchEngine` 项目：这个实验的主项目，根据查询搜索文档集返回前 20 个相关的文档
- `TestQueryResult` 项目：分析训练集结果的准确率值以及 DCG 等评价指标

`SearchEngine/` 中的 `DocumentSearch/` 、 `SearchEngine/` 、 `TestQueryResult/` 分别对应三个项目中的源文件。`SearchEngine.sln` 为解决方案文件。`dict/` 下为需要用到的词典，包含了 cppjieba 库的词典和我们自己的词典。

`doc/` 中包括了所有的文档。

项目依赖

- Intel Math Kernel Library
- cppjieba
- openmp

编译流程

SearchEngine 项目：

1. 打开 `SearchEngine/SearchEngine.sln`，在解决方案管理器中设置 SearchEngine 为启动项目
2. 打开 SearchEngine 项目属性页，选择 Release Win32
3. Intel Performance Libraries：勾选 Use Intel MKL，选择 sequential。或者直接显示链接 MKL 库，在下面有说明
4. VC++ 目录 —— 包含目录：添加 `<path>\cppjieba\deps` 和 `<path>\cppjieba\include`。如果显示链接 MKL，还需要添加 `...\mkl\include`
5. VC++ 目录 —— 库目录：如果显示链接 MKL，需要根据链接方式（如静态链接）添加 `mkl\lib\ia32_win`
6. C/C++ —— 预处理器：预处理器定义添加 `_CRT_SECURE_NO_WARNINGS`
7. C/C++ —— 语言：符合模式改为 `否`
8. C/C++ —— 语言：OpenMP支持改为 `是`
9. 链接器 —— 输入：如果显示链接 MKL 如静态链接，附加依赖项添加 `mkl_intel_c.lib;mkl_sequential.lib;mkl_core.lib`

其他两个项目均为用于训练和修正的辅助项目，配置方法类似，不需要他们也可以生成实验结果。

运行说明

SearchEngine 项目

设置 SearchEngine 项目为启动项目后直接在 Release x86 配置下编译运行即可，程序将自动读取同目录下的 `test_docs.csv` 并生成 tfidf 稀疏向量，之后读取 `test_querys.csv` 并进行自动查询，最后程序会在同目录下输出 `submission.csv` 作为实验结果。两个输入文件必须使用 utf-8 编码格式。SearchEngine 还会读取 `../dict/` 中的词典项目提供给 cppjieba 库和一些其他代码优化使用。

修改该项目下 main.cpp 中的开头的 `if 0` 为 `if 1` 将读取 `data_querys.csv` 和 `data_docs.csv` 并生成 `data_submission.csv` 作为测试。

DocumentSearch 项目

读取 `data_docs.csv` 以及词典文件进行 tf-idf 计算，并等待用户输入

用户可以输入 doc_id，将返回算法计算的 tfidf 值以及文档的具体信息，方便核对。

TestQueryResult 项目

读取 `data_compare.csv` 和 `data_submission.csv` 并生成和标准结果之间的对比分析，以及一些用于优化的参数。用户可以输入一个查询 id，将返回所有查错的文档和未查到的正确文档。

代码和算法部分

代码接口说明

Csv 类

```
static void read(const std::string &file_name,
                 int cols,
                 std::vector<std::vector<std::string>> &vec);
static void write(const std::string &file_name,
                  const std::vector<std::vector<std::string>> &vec);
```

用于读写 `.csv` 文件，这里因为文档中可能有英文逗号，只进行了简单的判断，标题和内容可能会有错误，不过整体不影响结果。

String_convert 类

```
static std::string string_to_utf8(const std::string &str);
static std::string utf8_to_string(const std::string &str);
```

借鉴了网络上的代码，负责 utf-8 编码和标准的 string 格式的互换。与主项目无关，主要用于在命令行上打印和输入方便调试

Documents 类

比较重要的接口如下：

```
using sparse_vector_type = std::pair<std::vector<int>,
                                     std::vector<double>>;
void calculate(std::vector<std::vector<std::string>> &&doc_vec);
const sparse_vector_type &get_doc_sparse_tfidf_vec(int doc_id) const { ... }
double get_doc_sparse_tfidf_vec_nrm2(int doc_id) const { ... }
```

- `calculate` 方法将本实验读 csv 得到的 4 列的 vector 作为输入，计算 tfidf 和其他相关的参数。
- `get_doc_sparse_tfidf_vec` 获取第 i 个文档的 tfidf 稀疏向量，采用 indx[] + x[] 的格式
- `get_doc_sparse_tfidf_vec_nrm2` 获取第 i 个文档的 tfidf 稀疏向量的模，方便后续计算

其他接口大部分为了方便调试和其他两个项目

Query 类

```
Query(const Documents &documents) : docs(documents) {}

// str 为问题字符串，n 为需要多少个结果，返回文档 id
Query_result query(const std::string &str, const std::string &qid = "", int n =
20) { ... }
Query_result query(std::string &&str, const std::string &qid = "", int n = 20) {
... }
```

- 构造函数接收 Documents 对象的引用，表示查询将在这个文档集中进行
- `query` 方法接受 utf8 格式的查询字符串，返回 Query_result 对象

Query_result 类

```
std::ostream &print_csv(std::ostream &os, const std::string &query_id);
```

打印结果

算法说明

使用 tfidf 作为基本算法，并作了如下优化：

- 标题权重增加 —— 把标题所占的 tfidf 权重增加
- 标题进行布尔匹配，结果线性组合 —— 额外进行了查询和标题的布尔匹配并作为结果分数的一项
- 大小写转换 —— 所有英文统一大小写
- 去除广告 空格 | —— 由于文档中有广告链接等干扰信息，特征是重复短文字并夹杂大量空格和 |
- tfidf太低的全文查找查询字符串 —— 对最终得分进行评估，前 20 个里如果普遍过低，则进行全文查找查询字符串。比如搜索中经常会出现某个文档中一段话，这时 tfidf 结果很低，使用全文搜索效果很好
- 提高网址匹配权重 —— url 也加入 tfidf
- 用户词典 —— 设置了我自己的停用词、近义词词典
- +分割匹配 —— 全文查找查询字符串时，对于 + 分割的查询字符串，分别临近查找
- log文档长度 —— 经过测试，训练集的文档中比较长的分数也比较好，内容比较优质。因此最终得分中包括了 log 文档长度一项。
- 近义词词典、学习词典 —— 利用训练集学习的词典，包括了手动生成的、人工的、面向测试集的词典，能略微提高分数

可调整的参数

`Documents.h`

```
static constexpr int TITLE_FACTOR = 70;
static constexpr int URL_FACTOR = 3;
```

这两项决定了计算 tfidf 时标题部分和 url 部分所占的比例

`Query.h`

```
static constexpr double TITLE_SIMILARITY_FACTOR = 1;        // tfidf的cos值系数为
1，和这个线性相加
static constexpr double TFIDF_ERROR_THRESHOULD = 1.2;       // value 低于这个值的
超过 n/2 则进行其他处理
```

如注释