数据科学与计算机学院

移动信息工程专业

本科生实验报告

(2017-2018学年秋季学期)

课程名称：人工智能

专业方向	班级	学号	姓名
移动互联网	1501	15352005	蔡景韬

一、实验题目

- 文本数据集简单处理

二、实验内容

1. 算法原理

- 得到总词条和词汇
  - 总词条
    - 总词条是指数据集中所有有用的词汇分行分列储存。
    - `vector<vector<string>>`存储总表。
    - 采用边读边储存的形式对总表进行存储。
    - 我们需要的信息在1个数字与7个字符串之后。
    - 每次都让文件流输出到一个int对象`file>>num`
      1. 输出成功，`file.good() == true`，此时标志着一行读取完毕，并读入第二行的数字部分，（将已储存的信息插入总表中），连续读入7个字符串并舍弃，为下一个判断分支准备。
      2. 输出失败，则此时文件流中的是我们需要的信息，首先`file.clear()`恢复文件流可用，然后读取字符串存储在`vector<string>`中。
  - 词汇
    - 词汇是指数据集中所有有用的单词不重复的储存。
    - `vector<string>`存储词汇。
    - 读入单个词时，就开始判断是否插入到词汇中。
    - 为了方便得出矩阵，增加`map<string,int>`的变量wordsMap，记录词在词汇中的id。
    - 读入一个词时，`wordsMap.find()`，若不在词汇表中，则插入词汇表，并在wordsMap中映射该词的id `wordsMap[str]=words.size()-1`。
  - 记录文章数`docNum=all.size()`与词汇数`wordSize=words.size()`。
- **OneHot**矩阵
  - OneHot矩阵是使用词汇的大小作为列数，文章数作为行数，当第i篇文章中出现第j个单词，则`OneHot[i][j]=1`。

- `OneHot[docNumMax][wordSizeMax]`二维数组存储矩阵。
- 整个OneHot矩阵置为0。
- 遍历总词单，将每一行的单词对应的id作为下标，索引OneHot矩阵，将其置为1

  `OneHot[i][ wordsMap[str] ] = 1`。
- 由于TF矩阵的需要，增加cntOneHot数组存储每个词在一行中出现的次数。

- **TF矩阵**
  - TF（*Term Frequency*）：向量的每一个值标志对应的词语出现的次数归一化后的概率（一行中逐个单词出现的概率）
  - `TF[docNumMax][wordSizeMax]`二维数组存储矩阵。
  - 整个TF矩阵置为0。
  - 遍历总词单，将每一行的单词对应的id作为下标，索引TF矩阵与cntOneHot，将每个词在一行中出现的概率赋值给TF矩阵 `cntOneHot[i][id] / all[i].size()`。
- **TF_IDF矩阵**
  - IDF（*逆向文件频率*）：表示单词在所有文章中出现的概率的倒数，并求对数。
  - `IDF[wordSizeMax]`二维数组存储矩阵。
  - 遍历总词条每一列，如果OneHot矩阵值为1，则该词出现次数+1，之后用文章总数除以次数+1 `log2((double)docNum / (double)(++cnt))`。
  - TF_IDF：表示单词在某篇文章中的标识度（可以用来区分这篇文章），等于词频TF * 反文件频率IDF。
  - 遍历TF矩阵，将TF矩阵值与IDF对应列值相乘即可得到TF_IDF矩阵 `TF_IDF[i][j] = TF[i][j]*IDF[j]`
- 三元表
  - OneHot矩阵为稀疏矩阵，使用三元表可以大大减少存储空间。
  - 使用TriTable类储存三元表

    ```
    struct triTable{
        int row,col,num;
        vector<triPoint> table;
    }TriTable;
    ```

    - triPoint是一个含有三个int类型成员的类
  - 遍历总词单，将每一行的单词对应的id与行数i成对压入集合（排序与去同），每行完成后将set中所有对(i,id)组成三元组(i,j,1)，插入三元表。
- 三元表加和
  - 三元表结构体同上文。
  - 使用权重值表示三元组在三元表中的前后位置，`num=i*10000+j`，num小的三元组排在三元表的靠前位置。
  - 读入A，B两个三元表，新建ans变量存储结果。两个迭代器同时遍历A、B两表，考虑三种情况：
    1. `权重值 aNum < bNum`：

       压入A当前位置的三元组；(A表迭代器推进)
    2. `权重值 aNum == bNum`：

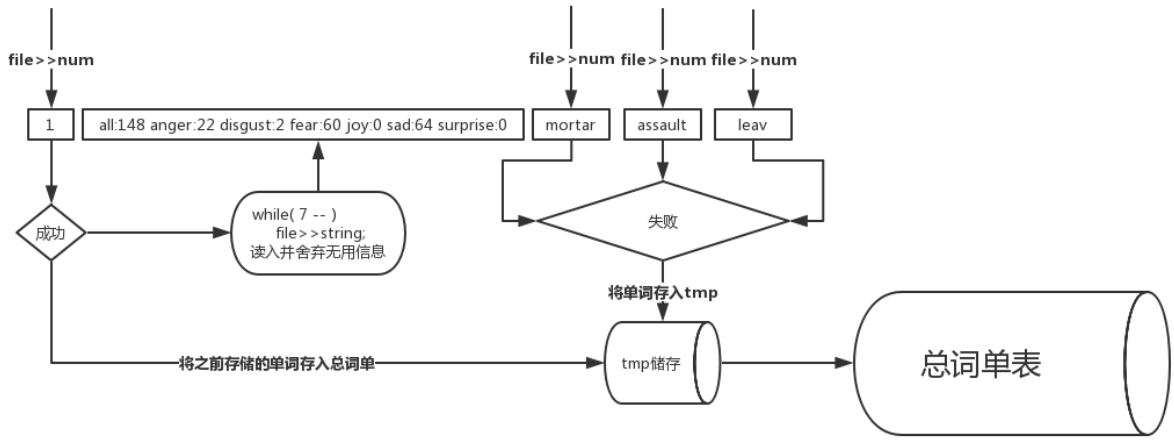
       压入A当前的三元组，其k值加上B表当前三元组k值；(A、B表迭代器同时推进)
    3. `权重值 aNum > bNum`：

压入B当前位置的三元组；(B表迭代器推进)

- 如果A、B两个迭代器未到结尾，则把剩下三元组全部压入。

2. 伪代码

- 处理数据集



- 词汇表

```
> 词汇表: words
> 词汇映射Map: wordsMap(单词映射到词汇表下标id)
> 输入一个单词str
> if( str在wordsMap中存在 ){
    将 str 插入 words;
    将 (str,words.size()-1) 插入 wordsMap;
  }
```

- **OneHot矩阵**

```
> 增加计数OneHot矩阵(单词在一行中出现的次数): cntOneHot
> 两个矩阵初始化置零
> 遍历总词单:
    for(i: 遍历所有行)
        for(j: 遍历所有列){
            OneHot[i][单词_ij对应的词汇表下标] = 1;
            cntOneHot[i][单词_ij对应的词汇表下标]++;
        }
```

- **TF矩阵**

```
> TF矩阵初始化置零
> 遍历总词单
    for(i: 遍历所有行)
        for(j: 遍历所有列)
            TF[i][单词_ij对应的词汇表下标]=这个单词出现的次数 / 该行词数;
```

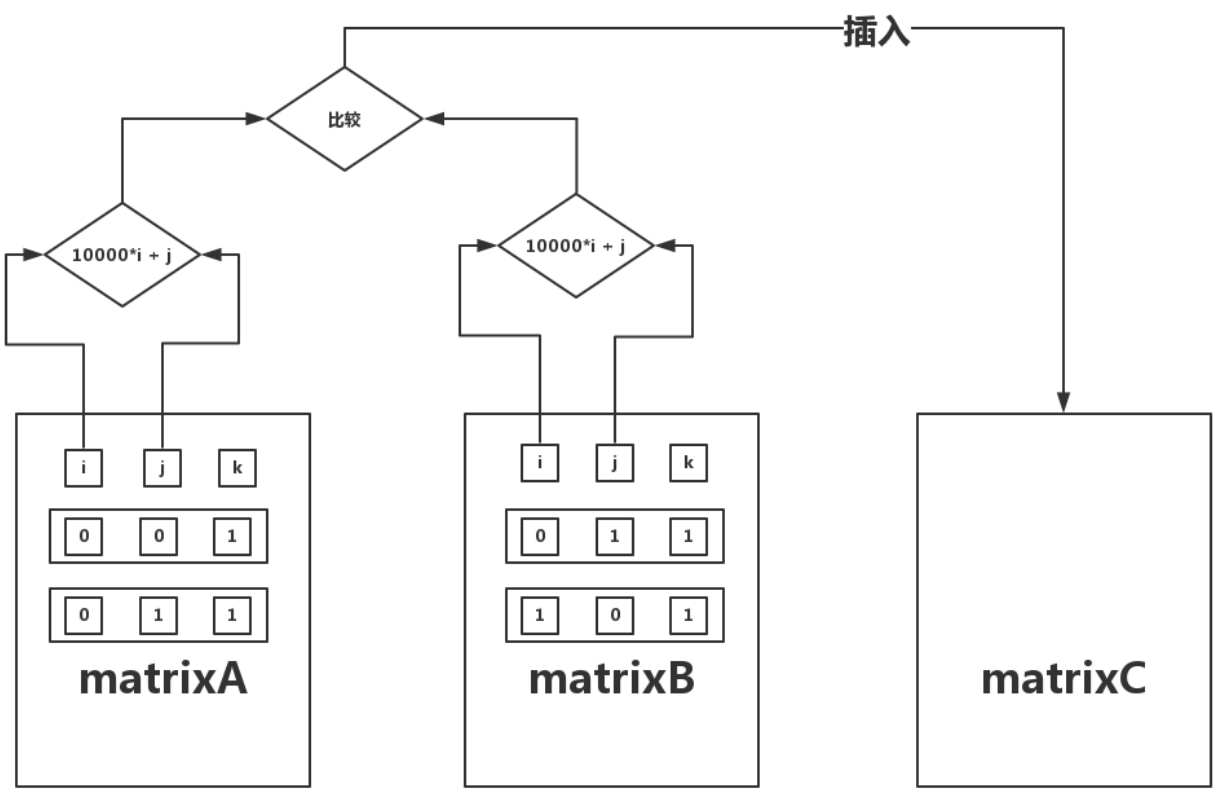
- **IDF矩阵**

```
> 遍历OneHot矩阵
    for(j: 遍历所有列){
         单词计数: cnt = 0
        for(i: 遍历所有行)
            if(OneHot[i][j],单词出现一次) cnt++;
        IDF[j] = log2(总文本数/++cnt);
    }
```

- 三元表

```
> 三元表数据结构:
    int row:行数,col:列数,num:三元组数
    vector<tirPoint(i,j,k)> table:三元表
> 遍历总词单
    for(i: 遍历所有行){
        声明集合: set<pair> idSet;
        for(j: 遍历所有列)
            将 单词_ij对应词汇表的下标 与 行数i 成对，插入idSet
        for(遍历idSet)
            将 idSet中每对pair 与 1 组成三元组,插入table
    }
```

- 三元表加和



3. 关键代码截图

- 处理数据集

```
while(!file.eof()){
    if(file>>num){ // return file.good()
        if(num!=1)  all.push_back(tmp);// 把已储存的一行单词存入总词单（第一行不用压入）
        tmp.clear() ; // 清空tmp内容
        int cnt=7;
        while(cnt--) file>>word; // 读取并舍弃无用内容
    }
    else{
        file.clear(); // 恢复文件流的状态
        file>>word;
        if(file.fail()) break;
        tmp.push_back(word);
        addWord(word); // 调用函数，判断并将单词插入词汇表
    }
}all.push_back(tmp); // 将最后一行压入
```

- `void addWord(string)`函数

```
if(wordsMap.find(str)==wordsMap.end()){ // 如果词汇不在词汇表中
    words.push_back(str);
    // 单词映射到词汇表对应的下标
    wordsMap.insert(pair<string,int>(str,words.size()-1));
}
```

- **OneHot**矩阵

```
memset(OneHot,0,sizeof(OneHot));
memset(cntOneHot,0,sizeof(cntOneHot)); // 计算单词在一行中出现的次数
for(int i=0; i<docNum; i++)
    for(int j=0; j<all[i].size(); j++){
        int id=wordsMap[all[i][j]];// 单词在词汇表的位置下标
        OneHot[i][id]=1; // 下标转化为OneHot矩阵的列数
        cntOneHot[i][id]++;
    }
```

- **TF**矩阵

```
memset(TF,0,sizeof(TF));
for(int i=0; i<docNum; i++)
    for(int j=0; j<all[i].size(); j++){
        int id=wordsMap[all[i][j]];// 单词在词汇表的位置下标
        TF[i][id]=(double)cntOneHot[i][id]/(double)all[i].size(); // 单词在一行的出现概率
    }
```

- **TF_IDF**

```
for(int j=0; j<wordSize; j++){
    int cnt=0;
    for(int i=0; i<docNum; i++)
            if(OneHot[i][j]) cnt++; // 记录每个单词在所有文本中的出现次数
    IDF[j]=log2((double)docNum/(double)(++cnt)); // 计算IDF矩阵
}
for(int i=0; i<docNum; i++)
    for(int j=0; j<wordSize; j++)
        TF_IDF[i][j] = TF[i][j]*IDF[j]; // 计算TF_IDF矩阵
```

- 三元表

```
TriTable.row = docNum;
TriTable.col = wordSize;
int cnt=0;
for(int i=0; i<docNum; i++){
    set<pair<int,int> > idSet; // 记录三元组的ij对
    for(int j=0; j<all[i].size(); j++){
        int id=wordsMap[all[i][j]];
        idSet.insert(pair<int,int>(i,id));
    }
    for(set<pair<int,int> >::iterator it=idSet.begin(); it!=idSet.end(); it++){
        // ij对和1组成三元组，压入表
        TriTable.table.push_back(triPoint(it->first,it->second,1));
        cnt++ ;
    }
}
TriTable.num=cnt; // 记录三元组数
```

- 三元表加和

```
while ( ait!=aEnd && bit!=bEnd ) { // A且B表都没到表尾
    // 对行数加权，并与列数求和，大小关系表示其在三元表中的前后关系
    aNum = ait->i*10000 + ait->j;
    bNum = bit->i*10000 + bit->j;
    // A当前三元组在前，先压入A当前三元组，并推进A
    if(aNum<bNum) ans.table.push_back(*ait++);
    // A、B两个三元组同行同列
    else if(aNum==bNum) {
        ans.table.push_back(triPoint(ait->i,ait->j,ait->k + bit->k));
        ait++,bit++;
    }
    // B当前三元组在前，先压入B当前三元组，并推进B
    else ans.table.push_back(*bit++);
}
// 如果A、B表任一个未到表位，将剩下的全部压入
while(ait!=aEnd) ans.table.push_back(*ait++) ;
while(bit!=bEnd) ans.table.push_back(*bit++) ;
ans.num = ans.table.size();
```

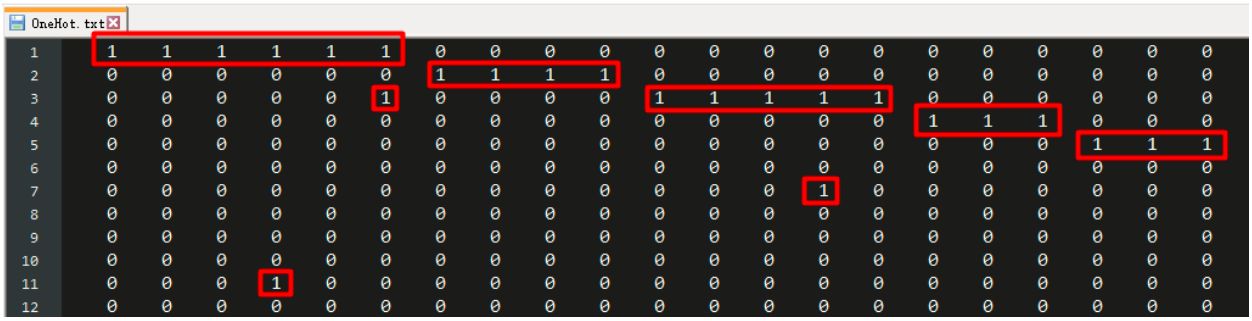
4. 创新点&优化

- 每次都让文件流输入到int类型，边读边处理，避免了读入一行再切割字符串的繁琐。
- 多加一个单词到词汇表下标的映射map，使得在求取OneHot与TF时，将时间复杂度从O(nm)降为O(n logm)。
- 发现使用freopen比fstream所需的时间要长，于是改用fstream。猜测是因为前者重定向标准输入输出，而后者是一个比标准输入输出更为简便的文件流，所以后者速度就快（就像cin、cout与scanf、printf？）。
- 在计算三元表求和时，将行数i乘以10000的权重，再加上列数j，将三元组在三元表的前后关系转化为数值的大小关系。

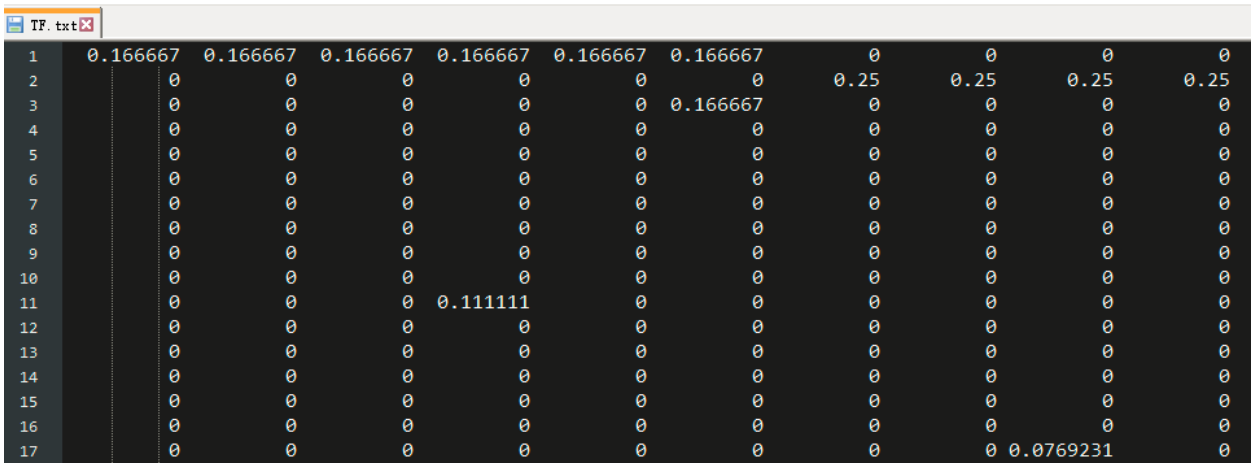
三、实验结果及分析

1. 实验结果展示示例

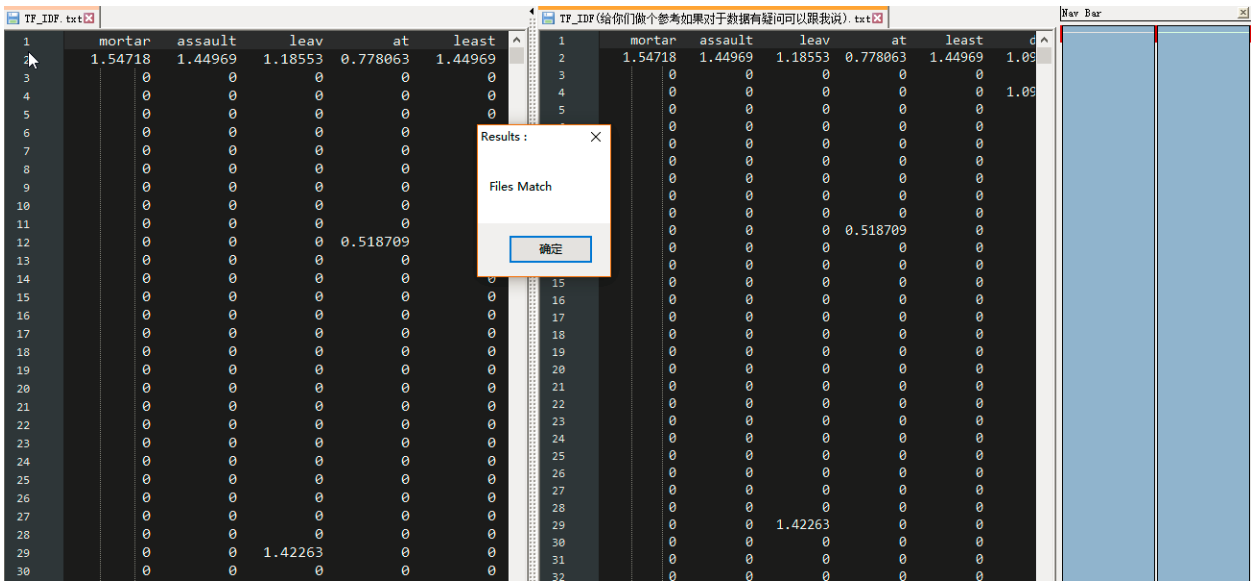
- **OneHot**矩阵



- **TF**矩阵



- **TF_IDF**矩阵（与同学结果的对拍）



四、思考题

1. IDF 的第二个计算公式中分母多了个 1 是为什么？

- 防止分母为0

2. IDF数值有什么含义？TF-IDF数值有什么含义？

- IDF（*逆向文件频率*）：表示单词在所有文本中出现的概率的倒数
  - 使用对数是为了取熵：取熵之后的熵值会比单纯的小数更明显更明确，也能表现出更大的信息量
  - 也有一种说法是：当文本数太大时，取对数能够缩小IDF的值
- TF是指一个单词在一个文本里出现的概率，而IDF是一个单词在所有文本中概率的倒数；那么TF_IDF表示的就是，如果TF_IDF数值大，则这个单词在某个文本里出现概率较高而在其他文本出现概率低，这样这个单词就可以作为该文本的标志词，便于区分文本。

3. 为什么要用三元顺序表表达稀疏矩阵？

- 为了压缩稀疏矩阵，减小存储空间。