



## 中山大学数据科学与计算机学院

### 移动信息工程专业-人工智能

### 本科生实验报告

(2017-2018 学年秋季学期)

课程名称：Artificial Intelligence

教学班级	1511	专业（方向）	软件工程（移动信息）
学号	15352237	姓名	刘顺宇

## 一、实验题目

文本数据集的简单处理

## 二、实验内容

### 1. 算法原理

#### 1.1. 稀疏矩阵：

在矩阵中，若数值为 0 的元素数目远远多于非 0 元素的数目，并且非 0 元素分布没有规律时，则称该矩阵为稀疏矩阵。这时候我们使用三元顺序表来存储稀疏矩阵可以得到更高的算法效率，且节省了存储空间，因为我们只对非零元素进行操作，稀疏矩阵的计算速度更快。

#### 1.2. 在本次实验中，用了 c++ 语言，算法主要利用了 stl 模板库中的 map 和 vector，且定义了三元顺序表的结构体来解决问题。

1.2.1. map 的使用主要是为了方便实现 int 类型与 string 类型的映射。

1.2.2. vector 的使用主要是为了实现可变长的数组的定义。

1.2.3. 三元顺序表的结构体使用主要是为了更高效的进行对稀疏矩阵的运算。

#### 1.3. 算法详解（搭配流程图）：

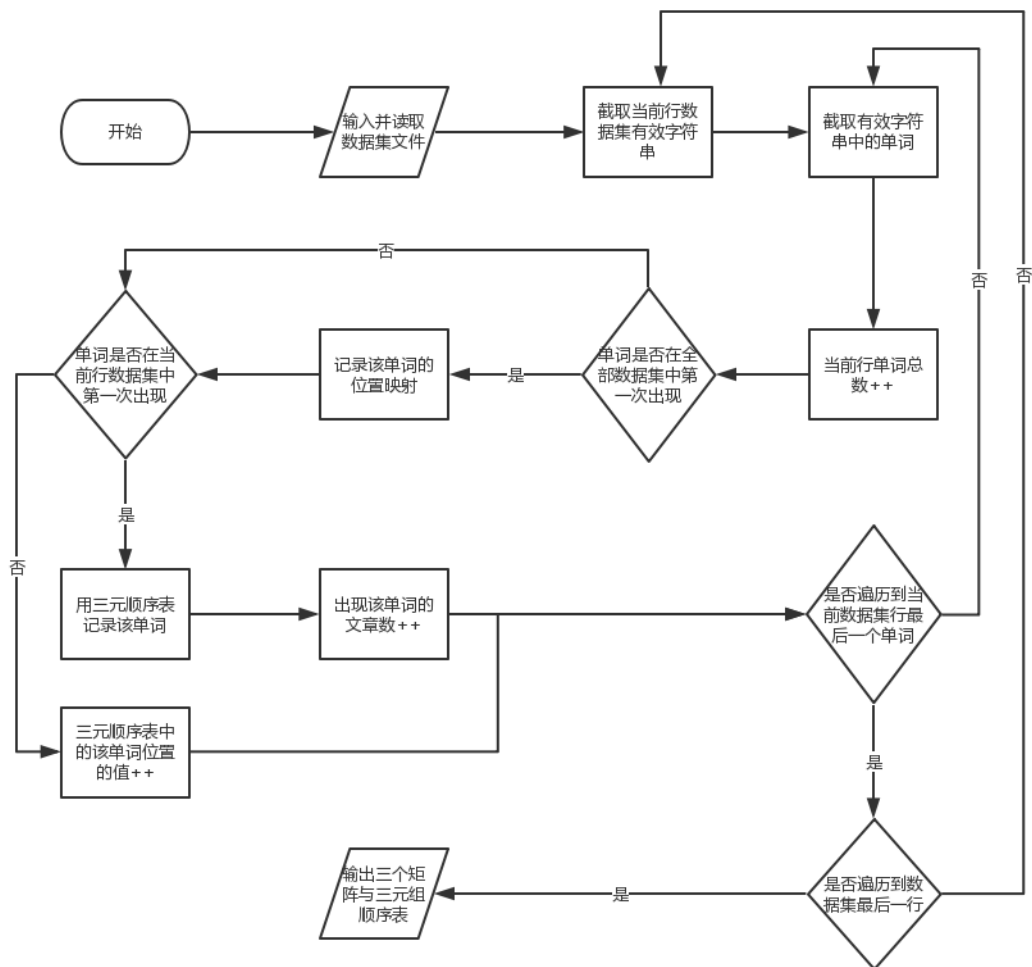
1.3.1. 截取当前行有效字符串：首先使用了 string 类型的 rfind() 反向查找函数，寻找到第二个 Tab 然后通过 substr 截取有效字符串。

1.3.2. 截取有效字符串中的单词：使用了 strtok 函数进行单词的截取

1.3.3. 判断单词是否在全部数据集中或是否在当前行数据集中，并同时确定该单词所处的列数，使用的是 map 映射函数，通过 find 来确定是否存在，存在则能确定位置，不存在则为其分配新的位置。

1.3.4. 在每一次判断完单词的存在性且确定其位置后将其放入三元组中，然后最后通过三元组与已知条件，通过遍历直接求解出三个所需矩阵。

## 2. 伪代码流程图



## 3. 关键代码截图（具体代码可看 cpp 文件，有详尽注释）

### 3.1. 截取单词

```

1 int pos = mydata_str.rfind("\t");//反向寻找第一个 Tab 出现的位置
2 mydata_str = mydata_str.substr(pos + 1, mydata_str.length() - pos - 1);//截取有效字符串
3 char* mydata_c = (char*)mydata_str.c_str();
4 const char* d = " "; //以空格作截取词
5 char* word;
6 word = strtok(mydata_c, d); //截取有效单词
  
```

### 3.2. 判断单词的位置并放入三元顺序表

```

1 //判断当前单词是否在全数据集第一次出现
2 map<string, int>::iterator it;
3 it = word_num.find(word_tem);
4 if(it == word_num.end()){
  
```



```
5 word_num[word_tem] = cnt_col; //记录新单词的列号
6 Data_matrix.col_word_cnt.push_back(0); //当前列单词出现的文章数置 0
7 Data_matrix.cn++;
8 cnt_col++;
9 }
10
11 //判断当前单词是否在当前行数据集第一次出现
12 it = word_pos.find(word_tem);
13 if(it == word_pos.end()){
14     word_pos[word_tem] = Data_matrix.data.size(); //记录新单词的列号
15     Data_matrix.data.push_back(Triple(cnt_row, word_num[word_tem], 1));
16     //将新单词位置放入三元顺序表
17     Data_matrix.col_word_cnt[word_num[word_tem]]++;
18     //当前列的单词出现的文章数++
19     Data_matrix.tn++;
20 }else{
21     Data_matrix.data[word_pos[word_tem]].value++;
22     //新单词所在三元顺序表 value 值++
23 }
```

### 3.3. 三元顺序表加法

```
1 //同时遍历两三元顺序表进行相加操作
2 while(i <= m1.tn && j <= m2.tn){
3     if(i == m1.tn && j == m2.tn) break; //遍历结束则退出循环
4     else if(i == m1.tn && j < m2.tn) m.data.push_back(m2.data[j]);
5     //只剩 B 三元顺序表项直接放入
6     else if(i < m1.tn && j == m2.tn) m.data.push_back(m1.data[i]);
7     //只剩 A 三元顺序表项直接放入
8     else{
9         //判断两三元顺序表项的行列号顺序放入
10         if(m1.data[i].row < m2.data[j].row){
11             m.data.push_back(m1.data[i]);
12             i++;
13             cnt++;
14         }else if(m1.data[i].row == m2.data[j].row){
15             if(m1.data[i].col < m2.data[j].col){
16                 m.data.push_back(m1.data[i]);
17                 i++;
18             }else if(m1.data[i].col == m2.data[j].col){
19                 //若两三元顺序表项行列号相同则相加值再放入
20                 Triple m_node(m1.data[i].row, m1.data[i].col, m1.data[i].value
21 + m2.data[j].value);
21                 m.data.push_back(m_node);
```



```
22         i++;
23         j++;
24     }else{
25         m.data.push_back(m2.data[j]);
26         j++;
27     }
28 }else{
29     m.data.push_back(m2.data[j]);
30     j++;
31 }
32 cnt++;
33 }
34 }
```

#### 4. 创新点&优化（如果有）

##### 4.1. 通过反向查找 Tab 快速截取有效字符串

```
1 int pos = mydata_str.rfind('\t');//反向寻找第一个 Tab 出现的位置
2 mydata_str = mydata_str.substr(pos + 1, mydata_str.length() - pos - 1);//截取
```

##### 4.2. 通过 map 判断单词是否出现过并记录位置

```
1 //判断当前单词是否在全部数据集第一次出现
2 map<string, int>::iterator it;
3 it = word_num.find(word_tem);
4 if(it == word_num.end()){
5     word_num[word_tem] = cnt_col;//记录新单词的列号
6     Data_matrix.col_word_cnt.push_back(0);//当前列单词出现的文章数置 0
7     Data_matrix.cn++;
8     cnt_col++;
9 }
```

##### 4.3. 算法主要通过三元顺序表存储矩阵，首先节省了空间开销，其次通过巧妙的运用 map 和 vector 来记录每个新单词的位置、每行的单词总数、每个单词所出现的文章总数，实现了只遍历一次文件就完成了三元顺序表的创建，然后便可以直接通过遍历三元顺序表来得到三个要求的矩阵。

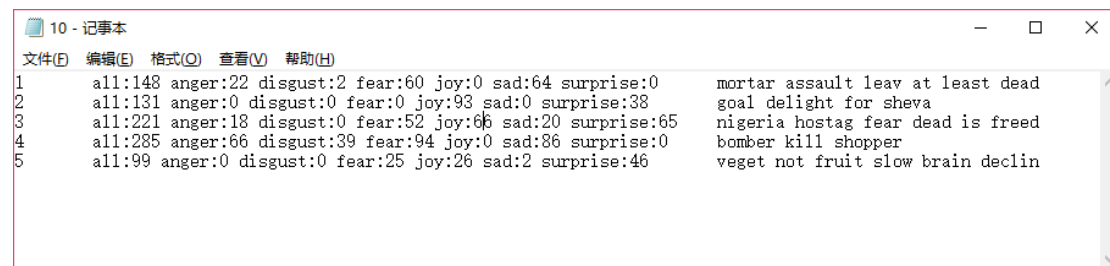
### 三、实验结果及分析

### 1. 实验结果展示示例（可图可表可文字，尽量可视化）

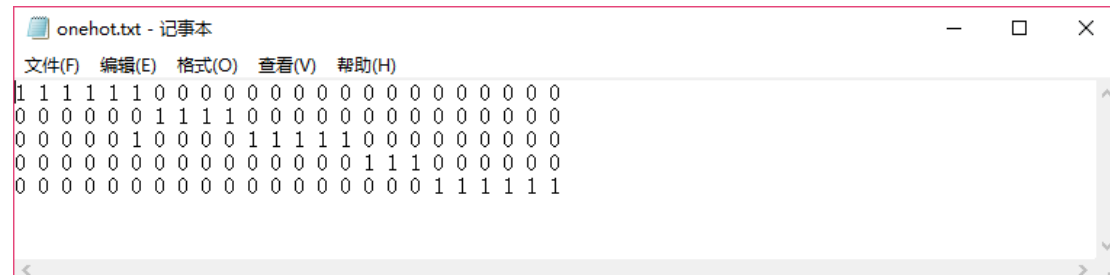
### 1.1. 程序运行时间



### 1.2. 由于数据量过大，用前五个数据集进行处理

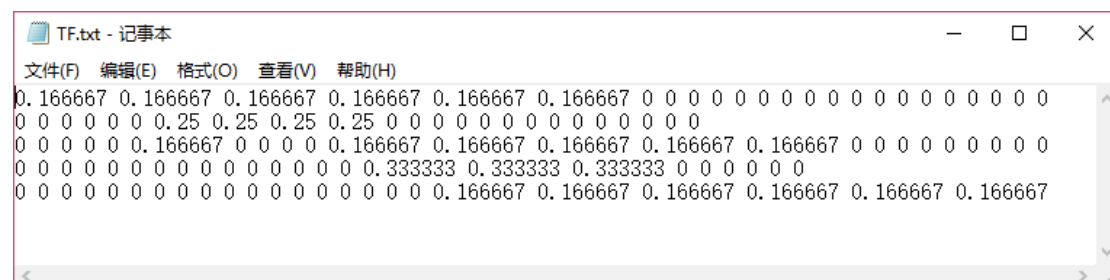


### 1.2.1. onehot 矩阵



分析：在第三个数据集的 dead 是全部数据集第二次出现，dead 第一次出现在第一行数据集中，所以应该在之前的列中已有 dead 的列，且由于 dead 是第五个出现的单词，所以应该为第五列。我们可以看到第三行第五列为 1，结果正确

### 1.2.2. TF 矩阵





分析：TF 矩阵为向量的每一个值标志对应的词语出现的次数归一化后的频率，我们算得第一行一共有 6 个单词，且每个单词出现一次，所以值都为 1/6，结果正确。

### 1.2.3. TFIDE 矩阵

```
TFIDF.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
0.386988 0.386988 0.386988 0.386988 0.386988 0.220321 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0.580482 0.580482 0.580482 0.580482 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0.220321 0 0 0 0 0.386988 0.386988 0.386988 0.386988 0.386988 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0.773976 0.773976 0.773976 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0.386988 0.386988 0.386988 0.386988 0.386988 0.386988
```

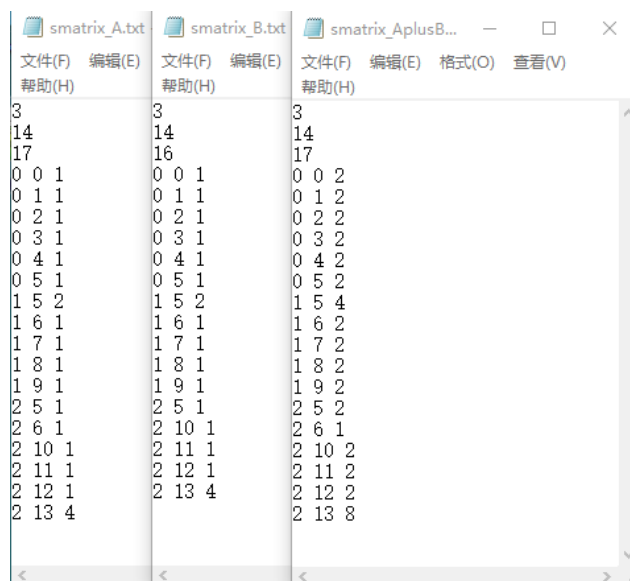
分析：TFIDE 矩阵为逆向文件频率再乘以词语出现的次数归一化后的频率，以第一行第六列的数据 0.220321 为例，归一化后的频率为 1/6，文章总数为 5，单词出现得文章数为 2，逆向文件频率为  $\log(5/2)/\log(2)$ ，再乘以 1/6，得到 0.2203213491，结果正确。

### 1.2.4. smatrix 三元顺序表

```
smatrix.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
5
24
25
0 0 1
0 1 1
0 2 1
0 3 1
0 4 1
0 5 1
1 6 1
1 7 1
1 8 1
1 9 1
2 5 1
2 10 1
2 11 1
2 12 1
2 13 1
2 14 1
3 15 1
3 16 1
3 17 1
4 18 1
4 19 1
4 20 1
4 21 1
4 22 1
4 23 1
```

分析：根据 onehot 矩阵与该三元表进行对比，onehot 矩阵中一共有 5 行，24 列，25 个有效值，且对应位置相同，与 smatrix 三元顺序表相符，结果正确。

## 1.2.5. 三元顺序表加法



smatrix_A.txt	smatrix_B.txt	smatrix_AplusB...
3	3	3
14	14	14
17	16	17
0 0 1	0 0 1	0 0 2
0 1 1	0 1 1	0 1 2
0 2 1	0 2 1	0 2 2
0 3 1	0 3 1	0 3 2
0 4 1	0 4 1	0 4 2
0 5 1	0 5 1	0 5 2
1 5 2	1 5 2	1 5 4
1 6 1	1 6 1	1 6 2
1 7 1	1 7 1	1 7 2
1 8 1	1 8 1	1 8 2
1 9 1	1 9 1	1 9 2
2 5 1	2 5 1	2 5 2
2 6 1	2 10 1	2 6 1
2 10 1	2 11 1	2 10 2
2 11 1	2 12 1	2 11 2
2 12 1	2 13 4	2 12 2
2 13 4		2 13 8

分析：我们可以看出位置相同的元素如 A 矩阵的(2,5)值为 1，B 矩阵的(2,5)值为 1，加在一起后为 2。A 矩阵中(2,6)元素在 B 矩阵中没有值，则直接按顺序加入三元顺序表中，结果正确。

## 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？

原分母表示出现了该单词的文章总数，但是如果该词语不在任何文章中，就会导致分母为零，因此一般情况下使用文章总数+1 作为分母。

### 2. IDF 数值有什么含义？TF-IDF 数值有什么含义？

- 2.1. IDF 数值：逆向文件频率，如果包含词条 t 的数据集越少，IDF 则越大，则说明词条 t 具有很好的类别区分能力。
- 2.2. TF-IDF 数值：某一特定数据集内的高词语频率，以及该词语在整个数据集集合中的低文件频率，可以产生出高权重的 TF-IDF。因此，TF-IDF 倾向于过滤掉常见的词语，保留重要的词语。

### 3. 为什么要用三元顺序表表达稀疏矩阵？

在矩阵中，若数值为 0 的元素数目远远多于非 0 元素的数目，并且非 0 元素分布没有规律时，则称该矩阵为稀疏矩阵。这时候我们使用三元顺序表来存储稀疏矩阵可以得到更高的算法效率，且节省了存储空间，因为我们只对非零元素进行操作，稀疏矩阵的计算速度更快。