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	15M2	专业（方向）	互联网
学号	15352218	姓名	林燕娜

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

关于数据集的处理：

① 词汇向量要求按出现的顺序排列，所以利用了 list 结构存储。

② 用 list<string>存储了每一行的内容，也用其存储了词汇向量；

③ 对于每一行的内容，利用 find 函数，查找里面是否有词汇向量中对应的词汇，以及出现的次数。用一个 Onehot 矩阵进行存储。并且非 0 元素，创建节点(row,col,1)，存储在一个列表中，就构成了一个稀疏三元表。

④ Onehot[i][j]指的是第 i 行内容中出现词汇向量中第 j 个单词的次数；Onehot 矩阵的最后一行是每一列非 0 元素的个数；Onehot 矩阵的最后一列是每一行的总和。

⑤ 得出 OneHot 矩阵：
Onehot 除去最后一行，最后一列，将非 0 元素改成 1，就是 OneHot 矩阵。

⑥ 得出 TF 矩阵：
Onehot 矩阵中（除去最后一行，最后一列），每个元素除以所在的行的最后一列的元素（该行的词汇数量总和），就是 TF 矩阵。

⑦ 得出 TF_IDF 矩阵：
每个词汇的 IDF，就是总行数/Onehot 矩阵最后一行的元素；
则 TF_IDF 就是 对应的 TF * log2(IDF)

稀疏三元表的加法：

与之前两个 set 合并的思想类似：

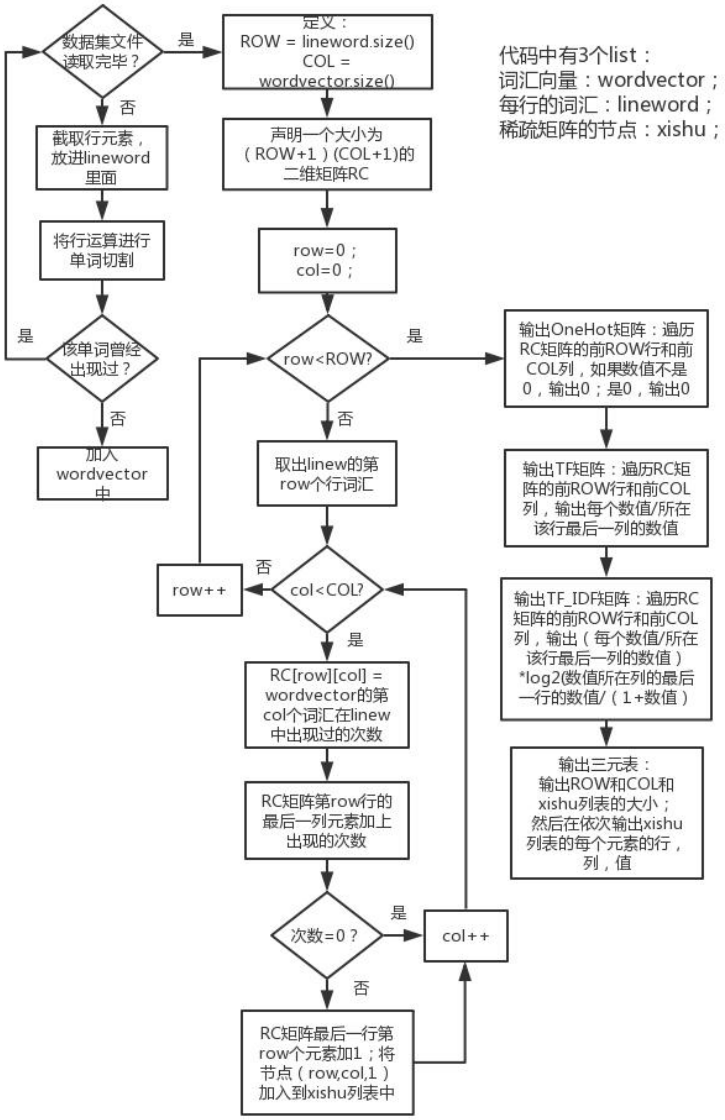
① 先将两个文件导入，将两个三元表 A,B 弄成 list<node>的形式，node 是（row,col,value）；

② 遍历两个三元表的元素，如果两个元素的 row 和 col 相同，说明是一个位置，将他们的 value 相加，加入到最后的结果 C 中，A,B 都跳到下一个元素。
如果位置不同，先比较 row，row 小的元素加到结果 C 中，对应的 A/B 跳到下一个元素。如果 row 相同，在比较 col，col 也同理。

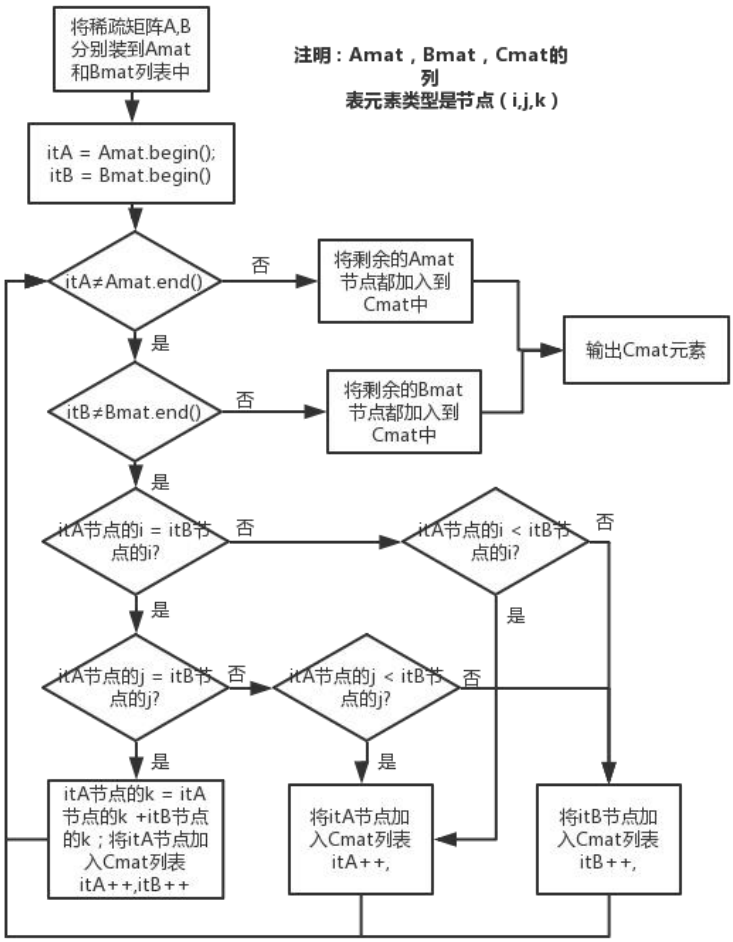
③ 当 A 或者 B 某个列表的元素已经遍历完后，将位遍历完的列表元素都加入到 C 中。

2. 流程图

➢ 数据集的处理、三个矩阵的输出、稀疏三元表的获得与输出：



➢ 稀疏三元表的加法：



3. 关键代码截图（带注释）

➢ 对原始文本的处理

（用 getline 得到每一行的内容，放进行词汇列表中；并且使用自己写的 createvector 函数对行词汇进行切割）

```
string a ;
while(getline(f,a)){
    size_t pos = a.rfind('\t');  //找出第二个制表符的位置
    a = ' '+a.substr(pos+1)+' '; // 在内容的前后加入空格，以便每个单词的分割
    lineword.push_back(a);       //将行内容加入行词汇列表中
    creatvector(a);              //将行内容进行分割，创建词汇列表
}
```

➢ 创建词汇向量列表

（根据传进来的每一行，对其进行单词切割，如果单词之前没出现过，加入词汇向量中）

```
//切割一个行词汇，构成词汇向量
void creatvector(string all){
    string tmp = "";
    for(int i = 0; i < all.size(); i++){//建立词汇向量
        if(all[i] == ' '){
            if(tmp == "") ;
            else{
                list<string>::iterator it = find(wordvector.begin(),wordvector.end(),tmp);
                if(it == wordvector.end()) //之前没出现过的单词，加入词汇向量中
                    wordvector.push_back(tmp);
            }
            tmp = "";
        }
        else tmp += all[i];//确保一个单词的完成性
    }
}
```

➢ 得到 Onehot 矩阵（不是 0 和 1，而是某行对应的某个单词出现次数）

（将每一行的内容和词汇向量列表做匹配，可以得到每一行的内容的词汇分布情况）

```
//对每一行词汇做处理，看是否出现词汇向量里的单词，以及出现的次数
for(it1 = lineword.begin();  it1 != lineword.end(); it1++){
    string  linew = *it1;
    for(it2 = wordvector.begin();  it2 != wordvector.end(); it2++){
        string vec = *it2;
        vec =' '+vec+ ' ';//为了避免is和this这种单词混淆，前后加空格以区分
        Onehot[row][col] = fun3(linew,vec);//fun3返回在row行中，出现单词vec的次数
        Onehot[row][COL] += Onehot[row][col];//最后一列是某行总数
        if(Onehot[row][col]){
            Onehot[ROW][col]++;
            node tmpp;
            tmpp.one = row;
            tmpp.two = col;
            tmpp.three = 1;
            xishu.push_back(tmpp);  //将非0元素弄成节点，加入到稀疏列表中
        }
        col++;
    }
    row++;
    col = 0;
}
```

➢ 得到子串出现次数函数

```
//确定在字符串中str中子串sub出现的次数num
int fun3(string str,string sub)
{
    int num = 0;
    while(1){
        size_t pos = str.find(sub);
        if (pos != -1 )  num++;
        else break;
        str = str.substr(pos+sub.size()-1); //去掉含有子串sub的部分，继续查找子串sub
    }
    return num;
}
```

➢ 稀疏矩阵的加法

（利用类似两个 set 的合并方法进行对两个稀疏三元表的加和）

```
//A,B加和
while(itA != Amat.end() && itB != Bmat.end()){//遍历A.B
    node tmpA = *itA;
    node tmpB = *itB;
    if(tmpA.one == tmpB.one && tmpA.two == tmpB.two){
        //如果两个元素位置相同，对应的值相加
        tmpA.three += tmpB.three;
        Cmat.push_back(tmpA);
        itA++;//A，B都跳到下一个元素
        itB++;
    }
    else if(tmpA.one < tmpB.one || tmpA.one == tmpB.one && tmpA.two < tmpB.two ){
        //如果A元素的i值比B的小，或者i值相同时，A元素的j比较小，A元素先进入列表
        Cmat.push_back(tmpA);
        itA++;
    }
    else{
        Cmat.push_back(tmpB);
        itB++;
    }
}
```

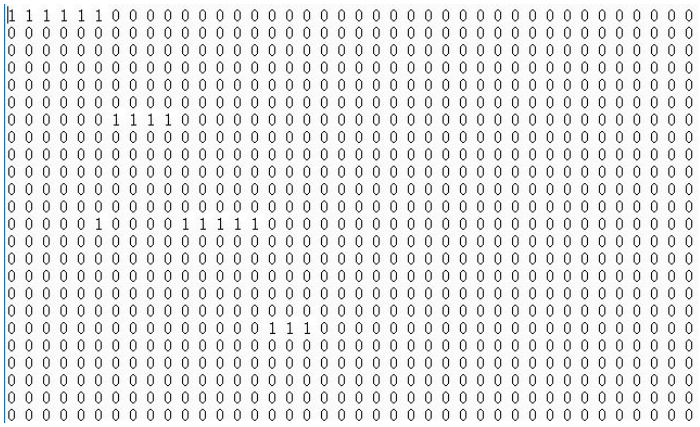
```
//如果A元素还有剩余，所有A元素加入到列表中，
while(itA != Amat.end()){
    node tmp = *itA;
    Cmat.push_back(tmp);
    itA++;
}
//如果B元素还有剩余,就将B的加入其中
while(itB != Bmat.end()){
    node tmp = *itB;
    Cmat.push_back(tmp);
    itB++;
}
```

三、 实验结果及分析

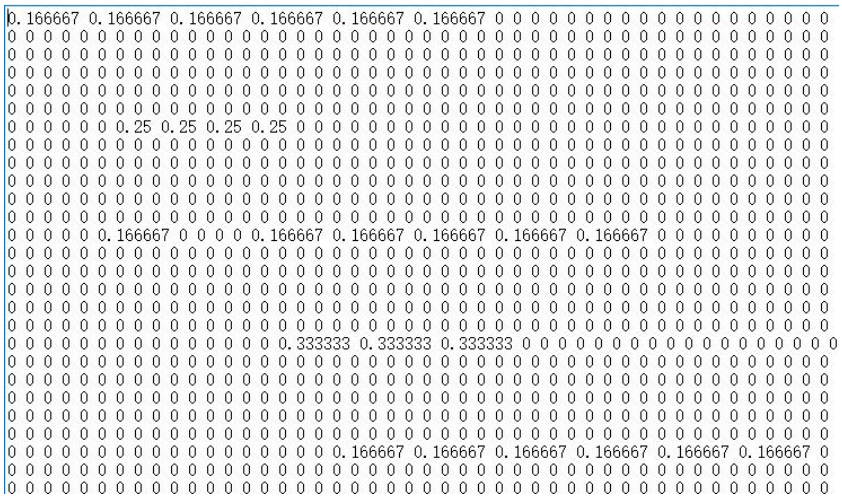
1. 实验结果展示示例（可图可表可文字，尽量可视化）

数据集的处理结果：

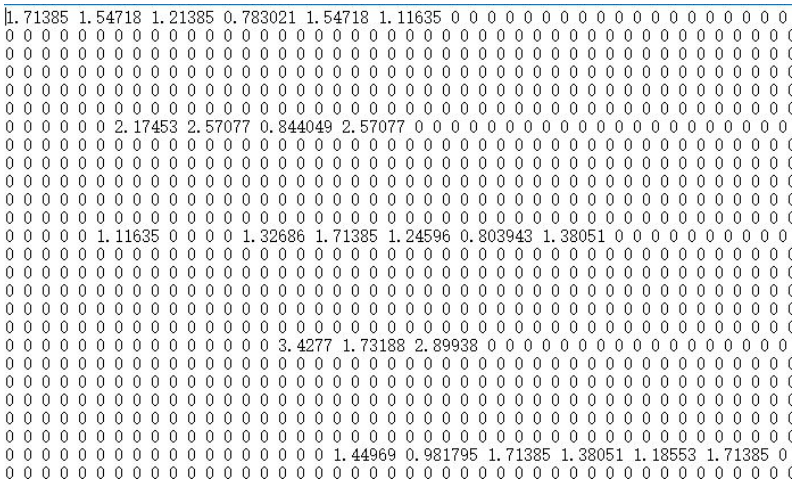
- OneHot 矩阵部分图



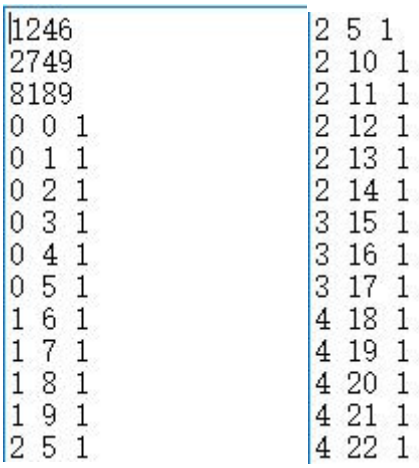
- TF 矩阵部分图



➢ TF_IDF 矩阵部分图



➢ 数据处理后的稀疏三元表部分图



自行给定小数据集的验证：

小数据集如下：

```
phone phone apple my
xilinc your my xilinc
apple phone expensive and using well
hello word xilinc
```

则每行的词汇分布如下：

行数\词汇	Phone	Apple	My	Xilinc	Your	Expensive	And	Using	Well	Hello	Word	行加和
1	2	1	1	0	0	0	0	0	0	0	0	4
2	0	0	1	2	1	0	0	0	0	0	0	4
3	1	1	0	0	0	1	1	1	1	0	0	6
4	0	0	0	1	0	0	0	0	0	1	1	3
列非0元素加和	2	2	2	2	1	1	1	1	1	1	1	

- OneHot 矩阵：

```
1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 0 0 0 0
1 1 0 0 0 1 1 1 1 0 0
0 0 0 1 0 0 0 0 0 1 1
```

将上面表格的非 0 元素改为 1，以第一行举例：2，1，1 改成 1，1，1；0 不变，所以 OneHot 矩阵的第一行应该为：1 1 1 0 0 0 0 0 0 0 0。与 txt 内容一致。而其他行也同理可验证其准确性。

- TF 矩阵

```
0.5 0.25 0.25 0 0 0 0 0 0 0 0
0 0 0.25 0.5 0.25 0 0 0 0 0 0
0.166667 0.166667 0 0 0 0.166667 0.166667 0.166667 0.166667 0 0
0 0 0 0.333333 0 0 0 0 0 0.333333 0.333333
```

将上面表格的每一行元素除以每一行的行加和词汇数，以第二行为例子，第二行的行加和是 4，而每个词汇对应出现的次数是：0 0 1 2 1 0 0 0 0 0 0，除以 4 后，可以得到：0 0 0.25 0.5 0.25 0 0 0 0 0 0 。与 txt 内容一致。其他行也同理可验证。

- TF_IDF 矩阵

```
0.5 0.25 0.25 0 0 0 0 0 0 0 0
0 0 0.25 0.5 0.5 0 0 0 0 0 0
0.166667 0.166667 0 0 0 0.333333 0.333333 0.333333 0.333333 0 0
0 0 0 0.333333 0 0 0 0 0 0.666667 0.666667
```

将总文章数 4 除以上面表格的列非 0 元素总和，再取 log2。得到每个词汇的 IDF 后，与 TF 矩阵的每个元素*对应的词汇的 IDF，得到 TF_IDF 矩阵。其中，前 4 个元素都一样，出现在两篇文章中，所以他们的 IDF 一致，都是 1，所以，前 4 列的 IF_IDF 应该与 TF 一致，明显，txt 的内容符合这一点。计算其他位置的值也与 txt 的内容一致，TF_IDF 矩阵正确。

- 数据集的稀疏三元表

```
4
11
15
0 0 1
0 1 1
0 2 1
1 2 1
1 3 1
1 4 1
2 0 1
2 1 1
2 5 1
2 6 1
2 7 1
2 8 1
3 3 1
3 9 1
3 10 1
```

观察表格中的非 0 元素，一共有 15 个，而且，每个非 0 元素的位置与稀疏表展现出来的也完全一致。所以，稀疏三元表正确。

稀疏三元表的加法：

```
3
7
11
0 0 1
0 1 1
0 5 1
0 6 1
1 2 1
1 3 1
1 4 1
2 0 2
2 1 1
2 3 1
2 5 1
```

```
3
7
5
0 1 1
0 5 1
1 0 1
1 6 1
2 0 1
```

给定了一下两个三元表。可以发现，两个三元表只有三个元素位置相同：第一个表的（0，1，1），（0，5，1），（2，0，2），以及第二个表的（0，1，1），（0，5，1），（2，0，1）。对应位置的元素相加之后，就成了（0，1，2），（0，5，2），（2，0，3）。其他不同位置的元素，按照大小排序，可以得到一下结果：

```
3
7
13
0 0 1
0 1 2
0 5 2
0 6 1
1 0 1
1 2 1
1 3 1
1 4 1
1 6 1
2 0 3
2 1 1
2 3 1
2 5 1
```

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？

答：IDF（Inverse Document Frequency），逆向文件频率。其中 DF 文件频率 = 出现词汇 A 的文章的数量/总文章数量，如果 DF 越大，说明词汇 A 在众多文章中出现，比较普遍，没有区分度，也就没那么重要。所以 DF 与词汇权重成负相关的关系。所以，取 DF 的逆，也就是倒数，又因为词频是近似服从指数分布，一些通用词出现的次数可能是低频词几十或者几百倍，直接采用文档频率取逆会导致稀缺词获得一个巨大的权重，会忽略其他词的影响，为了去掉这个影响，算法采用对文档频率取逆之后再取对数，所以 IDF = log（总文章数量/出现词汇 A 的文章的数量）。

在现实数据处理中，我们给出的所有文章可能都不包含词汇 A。则出现词汇 A 的文章的数量 = 0，IDF 没有意义。所以一般在 IDF 的分母+1，以免出现无意义的情况.而在本次实验中，词汇向量是由文章的词汇组成的，所以，每个词汇至少在一篇文章中出现,所以不会出现等于 0 无意义的情况。所以,本次代码中 ，使用的公式是不加 1 的。

2. IDF 数值有什么含义？TF-IDF 数值有什么含义？

答：IDF 越大 -> 出现词汇 A 的文章数量越少 -> 词汇 A 区分度高 -> 词汇 A 很重要。

TF_IDF，是频率表 TF * 文件频率 IDF。数值越大 -> 某词汇在某一篇文档中出现频率较高,并且在其他文档中很少出现 -> 能清晰的区别出某一篇文档,甚至作为某一篇文档的关键字 -> 该词汇非常重要。

3. 为什么要用三元顺序表表达稀疏矩阵？

答：稀疏矩阵是指矩阵中的非 0 元素远远小于 0 元素，矩阵非常庞大，但存储的信息有限，浪费空间。所以，利用**三元顺序表表达稀疏矩阵，可以大大的减少浪费的空间，提高效率。**