

## 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			
学号			

## 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) 文件读写:

① Fstream 提供了三个类, 用来实现 C++对文件的操作。Fstream 用于打开文件供读写、ifstream 用于从已有的文件中读取内容、ofstream 用于向文件写内容。

② 文件打开模式:

ios::in	读
ios::out	写
ios::app	从文件末尾开始写
ios::binary	二进制模式
ios::nocreate	打开一个文件时, 如果文件不存在, 不创建文件
ios::noreplace	打开一个文件时, 如果文件不存在, 创建文件
ios::trunc	打开以一个文件, 然后清空内容
ios::ate	打开一个文件时, 将位置移动到文件尾

③ 打开文件的方法:

<1> 调用构造函数, 指定文件名和打开模式:

```
ifstream f ("d:\\hello.txt",ios::nocreate);
```

<2> 使用 open 成员函数:

```
fstream f;
```

```
f.open ("d:\\hello.txt",ios::out);
```

注意: 路径的斜杠\\一定要双写。

④ 检查是否成功打开:

<1> 成功:

```
if (f) {...}; // 对 ifstream 和 ofstream 可用, 对 fstream 不可用
```

```
if (f.good()) {...};
```

<2> 失败:

```
if ( !f ) { ... };
```

```
if ( f.fail ( ) ) { ... };
```

⑤ 读写操作：

使用 << , >> 运算符。这只能进行文本文件的读写操作，用于二进制文件可能会产生错误。

⑥ 关闭文件：

使用成员函数 close: f.close ( ) 。

(2) 字符串切割：

用 string 类的两个函数 find 和 substr。find 函数的原型是：size\_t find ( const string& str, size\_t pos = 0 ) const; str 是子字符串，pos 是初始查找位置。找到的话返回第一次出现的位置，否则返回 string::npos。substr 函数的原型是：string substr ( size\_t pos = 0, size\_t n = npos ) const; 该函数会返回起始位置为 pos，长度为 n 的子字符串。

在该实验中，我都是使用这两个函数对文本进行切割，例如以空格为标记，使用 find 找到空格第一次出现的位置，然后从起始位置切割，然后继续寻找下一次空格的位置，然后从上一个空格的位置开始切割，以此类推。

(3) one-hot 矩阵：

One-hot 矩阵的每行代表一个文本，每列是不同的单词。One-hot 矩阵记录着文本中单词的出现情况，1 表示存在对应的单词，0 表示不存在。所以为了得到 one-hot 矩阵，首先用二维 vector 来表示矩阵，然后遍历所有文本，获得所有的不同单词（即获得不同单词集：Diff）。接下来再遍历每个文本，判断该文本中是否存在这些单词（即判断每行中是否存在 Diff），如果存在，则记录为 1，不存在则记录为 0。

(4) TF 矩阵：

TF：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。为了得到 TF 矩阵，首先用二维 vector 来表示矩阵，然后遍历所有文本，记录每个文本中每个单词出现的次数记录在二维向量 bridge 中。接下来再计算每个文本中单词的个数（在字符串且割时已经把每个文本的单词切割出来，放到向量 word 中，可直接由 word[row].size()

来得到每个文本的单词总数）。然后根据公式 
$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$
 计算即可得 TF 矩阵。

(5) TF-IDF 矩阵：

要求 TF-IDF 矩阵，首先求得 IDE，IDE 值的计算公式为：
$$idf_i = \log \frac{|D|}{|\{j : t_i \in d_j\}|}$$
。总的文章数 D 就等于文本的数量（在读取文件的同时已经切割好文本放到 result 中，所以直接通过 result.size ( ) 就可以得到文本的数量）。在已经得到 one-hot 的前提下，我们通过计算 one-hot 每列的数据的总和就可以得到某个单词在多少篇文章中出现过。获得了 D 和 t 之后就可以按照公式计算得到 IDF。在已经得到 TF 的前提下，我们通过

$$tfidf_{i,j} = tf_{i,j} \times idf_i$$
 就可以得到 TF-IDF。

(5) 稀疏矩阵三元顺序表：

稀疏矩阵的特征是在矩阵中，零元素的个数远远大于非零元素的个数，并且非零元素分布没有规律，所以我们对于稀疏矩阵一般采取压缩存储的方法，即存储非零元素。三元顺序表存储的内容是：稀疏矩阵的行数，稀疏矩阵的列数，稀疏矩阵中非零元素的个数，以及关于非零元素的信息（非零元素所在的行和列以及非零元素的值）。

## 2. 伪代码

### (1) lab1.cpp

```
//读取文件，将每行文本放入 result 中
void GetString(const char *filename){
    调用 open 函数打开文件；
    if（打开失败） return；
    else{
        While（一行一行地读取文件的内容,将每行文本的内容放入字符串 temp）{
            将 temp 以 tap 为标记进行切割，取切割所得的第三部分内容 s；
            将 s 放到向量 result 中；
        }
    }
    调用 close 函数关闭文件；
}
```

```
//获得不同的单词
void getDiff(){
    for（遍历 result 中的每行文本）{
        将文本以空格符为标记进行切割，切割所得的单词全部放入向量 v 中；
        将向量 v 放入二维向量 word 中；
        清空 v；
        在向量 Diff 中寻找每个切割所得的单词；
        if（在 Diff 中不能到找到该单词）将该单词放入 Diff 中；
    }
}
```

```
//获得每个文本中每个单词的个数
void getBridge(){
    for（遍历每个文本，即遍历 word）{
        for（遍历 Diff）{
            在 word[i]中寻找 Diff[j]；
            if（找不到） v.push_back(0)；
            else{
                if（第一次找到）v.push_back(1)；
                else v[j]++；
            }
        }
        将向量 v 放入二维向量 bridge 中；
        清空 v；
    }
}
```



```
//获得 one_hot 矩阵
//直接遍历 bridge 矩阵，大于 0 则记录为 1，等于 0 则记录为 0
void get_onehot(){
    for（遍历 bridge 的行）{
        for（遍历 bridge 的列）{
            if（bridge 的值>0）v.push_back(1);
            else if (bridge 的值==0) v.push_back(0);
        }
        将 v 放入二维向量 one-hot 中；
        清空 v；
    }
}
```

```
//获得 tf 矩阵
void get_tf(){
    for（遍历 bridge 的行）{
        通过 word[i].size()得到每行文本总的单词个数；
        for（遍历 bridge 的列）{
            if（bridge[i][j]>0）根据公式计算 tf,然后将 tf 的值放入向量 v 中；
            else v.push_back(0);
        }
        tf.push_back(v);
        v.clear();
    }
}
```

```
//获取 tf_idf 矩阵
void get_tfidf(){
    文章总数 D = result.size();
    for（遍历 one-hot）{
        将 one-hot 每列的数据相加得到 t;
    }
    t++;
    根据 IDF 的公式求 IDF:  $idf = \log_2(D/t)$ ，获得 IDF 的数值；
    根据 tf-idf 的公式计算:  $tfidf = 1.0 * idf[j] * tf[i][j]$ ;
}
```

```
typedef struct{
    int row;//非零数据的行
    int column;//非零数据的列
    int num;//非零数据的值
}triple; //三元顺序表
typedef struct{
```



```
int num_of_row;//稀疏矩阵的行数
int num_of_column;//稀疏矩阵的列数
int no_zero;//非零数据的个数
vector<triple> data;//记录非零数据的行、列和数值
}matric;

void get_smatric(){
    result.size 即文本总数就是 one-hot 矩阵的行数；
    Diff.size 即不同单词的个数就是 one-hot 矩阵的列数；
    for（遍历 one-hot 矩阵）{
        if（one-hot[i][j]!=0）{
            将该非零元素的行 i,列 j,以及数值 one-hot[i][j]记录在三元顺序表中；
            非零元素的个数 no_zero++;
        }
    }
}
```

## （2）AplusB.cpp

用两个结构体来实现三元顺序表：

```
typedef struct{
    int row;//非零数据的行
    int column;//非零数据的列
    int num;//非零数据的值
}triple; //三元顺序表

typedef struct{
    int num_of_row;//稀疏矩阵的行数
    int num_of_column;//稀疏矩阵的列数
    int no_zero;//非零数据的个数
    vector<triple> data;//记录非零数据的行、列和数值
}matric;
```

//将文件中的内容读取出来，放入三元顺序表中

```
matric GetMatric(const char *filename){
    while（一行一行地读取文本，将内容存在字符串 temp 中）{
        if(第一行) 将第一行的字符串数值转换成 int 类型，存储在 num_of_row 中；
        else if(第二行) 将字符串数值转换成 int 类型，存储在 num_of_column 中；
        else if(第三行) 将第三行的字符串数值转换成 int 类型，存储在 no_zreo 中；
        else{
            将每行文本以空格为标记进行切割；
            切割的第一个数值放在 tri.row 中；
            切割的第二个数值放在 tri.column 中；
            切割的第三个数值放在 tri.num 中；
        }
    }
}
```



```

    }
    将 tri 放入到三元顺序表 smatric 的向量 data 中;
}
}

//两个三元顺序表相加
matric AplusB(matric A, matric B){
    设立两个数组 visitA、visitB 来记录 A、B 中元素的访问情况;
    for(遍历 A){
        for (遍历 B) {
            if (在 B 中找到一个元素行列都和 A[i]相同并且没有被访问过){
                将行、列、相加的数值都放在 tri 中;
                将 tri 放入新的三元顺序表 smatric 的向量 data 中;
                在 visitA 和 visitB 中记录访问过这两个元素;
            }
            if (visitA[i]==0 即该元素在 B 中没找到符合条件的元素)
                将 A 中的该元素直接放入 smatric 中;
        }
    }
    for (遍历 visitB) {
        if(visitB[i]==0 即没被访问过) 将 B 中该元素直接放入 smatric 中;
    }
    对 smatric 按照行列的大小进行排序;
}

```

### 3. 关键代码截图（带注释）

(1) lab1.cpp:

《1》 读取文本文件中的内容

① 将文件打开:

```

ifstream ReadFile;
string temp; //将读取的文本放到temp中去
ReadFile.open(filename,ios::in);

```

② 读取文件中的每一行文本，并且把文本放到字符串 temp 中:

```

//一行一行地读取文件，将每行文本放到temp中
while(getline(ReadFile,temp)){

```

③ 以 tap 符为标记切割出我们需要的文本内容（第二个 tap 符之后的内容），然后将得到的文本内容放入 vector result 中:

```

int size = temp.size();
string pattern=" ";
//以tap符为标记来获得我们需要的内容【第二个tap之后的内容】
int pos;
pos = temp.find(pattern,0);
pos = temp.find(pattern,pos+1); //找到第二个tap
string s = temp.substr(pos+1,size); //将第三块内容切割出来
result.push_back(s); //将切割出来的文本放入result中

```



④ 关闭文本文件：

```
ReadFile.close();
```

《2》 将文本切割成单词，获得不同的单词：

① 遍历每一行文本（即遍历 result），将每一行文本以空格键为标记切割成一个个单词 s：

```
for(int i=0;i<result.size();i++){
    string str = result[i];
    int pos;
    string pattern = " "; //以空格符为标记将文本中切割成单词
    str+=pattern;
    int size = str.size();
    for(int j=0; j<size; j++){
        pos = str.find(pattern,j); //从位置j开始在str中找到空格的位置
        if(pos<size){
            string s = str.substr(j,pos-j); //截取从第j位开始的长度为pos-j的字符串(单词)
            v.push_back(s);
            vector<string>::iterator it;
            it = find(Diff.begin(),Diff.end(),s);
            //在存放不同单词的vector Diff中找切割出来的单词
            if(it==Diff.end()){ //如果在Diff中没有这个单词，则把这个单词加进去
                Diff.push_back(s);
            }
            j=pos+pattern.size()-1;
        }
    }
}
```

② 将每行文本切割的单词放入一个向量 v 中，记录完一行文本的单词后，再将 v 放入向量 word 中：

```
int size = str.size();
for(int j=0; j<size; j++){
    pos = str.find(pattern,j); //从位置j开始在str中找到空格的位置
    if(pos<size){
        string s = str.substr(j,pos-j); //截取从第j位开始的长度为pos-j的字符串(单词)
        v.push_back(s);
        vector<string>::iterator it;
        it = find(Diff.begin(),Diff.end(),s);
        //在存放不同单词的vector Diff中找切割出来的单词
        if(it==Diff.end()){ //如果在Diff中没有这个单词，则把这个单词加进去
            Diff.push_back(s);
        }
        j=pos+pattern.size()-1;
    }
}
word.push_back(v);
```

③ 将每行文本切割出来的每个单词与 Diff 中的单词进行比较，如果 Diff 中没有包含这个单词，则把这个单词放入 Diff 中：

```
vector<string>::iterator it;
it = find(Diff.begin(),Diff.end(),s);
//在存放不同单词的vector Diff中找切割出来的单词
if(it==Diff.end()){ //如果在Diff中没有这个单词，则把这个单词加进去
    Diff.push_back(s);
}
```

《3》 获得每个文本中每个单词的个数：

① 在每行文本的单词集中寻找 Diff 中的单词：

```
vector<string>::iterator it;
it = find(word[i].begin(),word[i].end(),Diff[j]); //在每行文本的单词中寻找Diff中的单词
```



- ② 用向量  $v$  来记录寻找的情况，如果没有找到则记录为 0:

```
if(it==word[i].end()){
    v.push_back(0); //找不到记录为0
```

- ③ 如果找到了  $\text{Diff}[j]$ ，则在该行文本的单词集中删除该单词。如果是第一次找到该单词的，则记录为 1；如果不是第一次找到的话，则将原来的记录++;

```
bool num=false;
while(it!=word[i].end()){
    it=word[i].erase(it);
    if(num==false){
        v.push_back(1); //第一次找到，记录为1
    }
    else{
        v[j]++; //不是第一次找到，则把原来记录的数字++
    }
    it = find(word[i].begin(),word[i].end(),Diff[j]);
    num=true;
}
```

- ④ 将每行文本中每个单词的个数的记录情况  $V$  放入到向量  $\text{bridge}$  中:

```
bridge.push_back(v); //将在每行文本中寻找Diff的结果v放入bridge中
```

#### 《4》 获得 one-hot 矩阵:

可以直接遍历向量  $\text{bridge}$  得到 one-hot 矩阵，因为 one-hot 矩阵是记录每行文本中是否出现了  $\text{Diff}$  数据集中的单词，而  $\text{bridge}$  是记录了出现的次数，所以只要把  $\text{bridge}$  中大于 0 的数据记录为 1，等于 0 的数据记录为 0 即可。

```
void get_onehot()
{
    int size_of_row=bridge.size();
    int size_of_column=bridge[0].size();
    vector<int> v;
    for(int i=0;i<size_of_row;i++){
        for(int j=0;j<size_of_column;j++){
            if(bridge[i][j]>=1){
                v.push_back(1);
            }
            else if(bridge[i][j]==0) v.push_back(0);
        }
        one_hot.push_back(v);
        v.clear();
    }
}
```

#### 《5》 获得 tf 矩阵:

- ①  $\text{tf}$  是向量的每一个值标志对应的单词出现的次数归一化的频率。

$$\text{tf}_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

- ② 通过  $\text{bridge}$  中的数据来计算  $\text{tf}$ 。 $\text{bridge}$  每行的数据的总和就是每行文本总的单词





个数, 即  $\sum_k n_{k,j}$  (也可以通过获得 word[i] 的 size 来知道每行文本的总的单词

个数), bridge 每行的每个数据对应的就是  $\sum_k n_{k,j}$ 。

```
for(int i=0;i<size_of_row;i++){
    int Denominator=0;
    for(int k=0;k<size_of_column;k++){
        Denominator+=bridge[i][k]; // 计算tf公式的分母(文本的单词总数)
        // 也可以通过word获得
    }
    // Denominator = word[i].size();
    for(int j=0;j<size_of_column;j++){
        if(bridge[i][j]>0){
            num=1.0*bridge[i][j]/Denominator;
            v.push_back(num);
        }
        else if(bridge[i][j]==0) v.push_back(0);
    }
    tf.push_back(v);
}
```

#### 《6》 获得 tf-idf 矩阵

- ① 要获得 tf-idf 矩阵, 首先得得到 IDF, IDF 的计算公式:

$$idf_i = \log \frac{|D|}{|\{j: t_i \in d_j\}|} \quad idf_i = \log \frac{|D|}{1 + |\{j: t_i \in d_j\}|}$$

D 可以通过有多少行文本来获得, 即通过 result.size() 来获得。

```
int D=result.size();
```

- ② 分子  $t_i$  可以通过计算 one-hot 每列的总和来得到一个单词出现在多少篇文本中:

```
for(int i=0;i<size_of_row;i++){
    t+=one_hot[i][j];
    // 通过计算将one_hot 每列的总数算出可知一个单词出现在多少篇文本中
}
t+=1;
```

$$idf_i = \log \frac{|D|}{1 + |\{j: t_i \in d_j\}|}$$

- ③ 按照公式 计算 idf:

```
double res = log2(1.0*D/t); // D - 文本总数, res是idf的数据
idf.push_back(res);
```

- ④ 按照公式  $tfidf_{i,j} = tf_{i,j} \times idf_i$  计算 tfidf:

```
vector<double> v;
int size_of_idf=idf.size();
for(int i=0;i<size_of_row;i++){
    for(int j=0;j<size_of_column;j++){
        double tfidf = 1.0*idf[j]*tf[i][j]; // tf_idf = idf*tf
        v.push_back(tfidf);
    }
    tf_idf.push_back(v);
    v.clear();
}
```



《7》 one-hot 的三元顺序表:

① 使用结构体来构建三元顺序表:

```
typedef struct{
    int row; // 非零数据的行
    int column; // 非零数据的列
    int num; // 非零数据的值
}triple; // 三元顺序表

typedef struct{
    int num_of_row; // 稀疏矩阵的行数
    int num_of_column; // 稀疏矩阵的列数
    int no_zero; // 非零数据的个数
    vector<triple> data; // 记录非零数据的行、列和数值
}matrix;
```

② 遍历 one-hot 矩阵, 当遇到非零数值时, 将数值的行、列和数值都记录起来:

```
triple tri;
for(int i=0; i<size_of_row; i++){
    for(int j=0; j<size_of_column; j++){
        if(one_hot[i][j]!=0){
            tri.row=i;
            tri.column=j;
            tri.num=one_hot[i][j];
            smatric.data.push_back(tri);
            smatric.no_zero++;
        }
    }
}
```

③ 三元顺序表还要记录表的大小(行、列)以及非零数据的个数:

```
smatric.num_of_row=result.size();
smatric.num_of_column=Diff.size();

triple tri;
for(int i=0; i<size_of_row; i++){
    for(int j=0; j<size_of_column; j++){
        if(one_hot[i][j]!=0){
            tri.row=i;
            tri.column=j;
            tri.num=one_hot[i][j];
            smatric.data.push_back(tri);
            smatric.no_zero++;
        }
    }
}
```

《8》 将 vector 中的数据写入到文本中, 注意路径名中的\要双写, 打开文件后一定要记得关闭文件。

```
ofstream in;
in.open("F:\\学习资料\\大三上\\人工智能\\实验\\实验1\\onehot.txt", ios::trunc);
int size_of_row=one_hot.size();
int size_of_column=one_hot[0].size();
for(int i=0; i<size_of_row; i++){
    for(int j=0; j<size_of_column; j++){
        in<<one_hot[i][j]<<" ";
        // cout << left << setw(10) << one_hot[i][j] << " ";
    }
    in<<"\\n";
    // cout << endl;
}
in.close();
```



## (2) AplusB.cpp

《1》 使用两个结构体来构件三元顺序表:

```
typedef struct{
    int row; // 非零数据的行
    int column; // 非零数据的列
    int num; // 非零数据的值
}triple; // 三元顺序表

typedef struct{
    int num_of_row; // 稀疏矩阵的行数
    int num_of_column; // 稀疏矩阵的列数
    int no_zero; // 非零数据的个数
    vector<triple> data; // 记录非零数据的行、列和数值
}matrix;
```

《2》 读取文件，并且将文件中的三元顺序表的内容放入三元顺序表 matrix 中:

- ① 一行一行地读取文件，并且将文件每一行的内容放到字符串 temp 中。文件内容的第一行是表的行数，第二行是表的列数，第三行是非零数据的个数，分别记录起来。

```
while(getline(ReadFile,temp)){
    if(circle==1) smatric.num_of_row = string_to_num(temp);
    else if(circle==2) smatric.num_of_column = string_to_num(temp);
    else if(circle==3) smatric.no_zero = string_to_num(temp);
```

- ② 文本内容第三行之后都是有关非零数值的信息（行、列、值）。首先以空格为标记将文本切割成三段，这三段分别是行、列、值，然后分别将他们记录起来:

```
else{
    triple tri;
    int size = temp.size();
    string pattern=" ";
    int posr,posc;
    posr = temp.find(pattern,0);
    string r = temp.substr(0,posr); // 行
    tri.row = string_to_num(r);
    temp.erase(0,posr+1);
    posc = temp.find(pattern,0);
    string c = temp.substr(0,posc); // 列
    tri.column = string_to_num(c);
    temp.erase(0,posc+1);
    tri.num = string_to_num(temp); // 非零数值个数
    smatric.data.push_back(tri);
}
circle++;
```

- ③ 因为切割出来的是字符串，而我们记录的内容是 int 型，所以我们需要函数将 string 类型的数值转换成 int 类型。

```
int string_to_num(string str)
{
    int size = str.size(),num=str[0]-'0';
    for(int i=1;i<size;i++){
        num+=(str[i]-'0')*10;
    }
    return num;
}
```



《3》 将两个三元顺序表 A、B 进行相加：

- ① 首先遍历三元顺序表 A 中非零数据的信息 data（即行、列），将 A 中的每个非零数据都和 B 中的非零数据进行对比，如果 B 存在行、列都和 A 中非零数据相同的数据并且没有相加过（visit=0），则将行、列相同的两个非零数据的数值相加，放到另一个三元顺序表 C 中。

```
for(int i=0;i<sizeA;i++){
    for(int j=0;j<sizeB;j++){
        triple tri;
        if(A.data[i].row==B.data[j].row && A.data[i].column==B.data[j].column&&visitB[j]==0){
            tri.row=A.data[i].row;
            tri.column=A.data[i].column;
            tri.num=A.data[i].num + B.data[j].num;
            smatric.data.push_back(tri);
            visitB[j]=1;
            visitA[i]=1;
        }
    }
}
```

- ② 如果 B 中所有的非零数据都和 A 中的某个非零数据进行对比了，但是没有找到行、列相同的数据，则将 A 中的非零数据放入 C 中。

```
if(visitA[i]==0){
    smatric.data.push_back(A.data[i]);
}
```

- ③ 遍历完 A 中的所有非零数据后，判断 B 中是否还存在没有相加的非零数据。如果有，也放入到 C 中。

```
for(int k=0;k<sizeB;k++){
    if(visitB[k]==0){
        smatric.data.push_back(B.data[k]);
    }
}
```

- ④ 最后，将 C 中的非零数据的信息进行排序，首先按照行数大小排序，行数相同则按照列数大小排序。

```
sort(smatric.data.begin(),smatric.data.end(),comparison);

bool comparison(triple A,triple B)
{
    if(A.row!=B.row) return A.row<B.row;
    else return A.column<B.column;
}
```

- ⑤ 只要行列相同的矩阵才能相加，所以相加得到矩阵 C 的行列和 A、B 相同。然后再通过获得存放非零数据的向量 data 的大小可得到非零数据的个数。

```
smatric.num_of_row=A.num_of_row;
smatric.num_of_column=A.num_of_column;

smatric.no_zero = smatric.data.size();
```



### 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) lab1.cpp

	onehot.txt	tf.txt	tfidf.txt
1	1	1	1
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0
10	0	0	0
11	0	0	0
12	0	0	0
13	0	0	0
14	0	0	0
15	0	0	0
16	0	0	0
17	0	0	0

	onehot.txt	tf.txt	tfidf.txt
1	0.166667	0.166667	0.166667
2	0.000000	0.25	0.25
3	0.000000	0.166667	0.166667
4	0.000000	0.333333	0.333333
5	0.000000	0.166667	0.166667
6	0.000000	0.166667	0.166667
7	0.000000	0.25	0.25
8	0.000000	0.166667	0.166667
9	0.000000	0.166667	0.166667
10	0.000000	0.166667	0.166667
11	0.000000	0.111111	0.111111
12	0.000000	0.166667	0.166667
13	0.000000	0.166667	0.166667
14	0.000000	0.166667	0.166667
15	0.000000	0.166667	0.166667
16	0.000000	0.2	0.2
17	0.000000	0.076923	0.076923
18	0.000000	0.125	0.125
19	0.000000	0.166667	0.166667
20	0.000000	0.166667	0.166667

	onehot.txt	tf.txt	tfidf.txt
1	1.54718	1.44969	1.18553
2	0.00000	2.07077	2.32077
3	0.00000	1.09711	1.28302
4	0.00000	3.09436	1.71794
5	0.00000	1.38051	0.970609
6	0.00000	1.28302	1.54718
7	0.00000	1.19781	2.32077
8	0.00000	1.32686	1.16019
9	0.00000	1.32686	1.16019
10	0.00000	1.32686	1.16019
11	0.00000	0.518709	0.518709
12	0.00000	1.09711	1.28302
13	0.00000	1.09711	1.28302
14	0.00000	1.09711	1.28302
15	0.00000	1.31653	1.31653
16	0.00000	0.258787	0.258787
17	0.00000	0.822831	0.822831
18	0.00000	0.675712	0.675712
19	0.00000	0.675712	0.675712
20	0.00000	0.675712	0.675712
21	0.00000	0.675712	0.675712



1246	340 515 1	1136 111 1
2749	340 579 1	1136 256 1
8189	340 580 1	1136 278 1
0 0 1	340 595 1	1136 676 1
0 1 1	340 1172 1	1136 2593 1
0 2 1	341 614 1	1137 404 1
0 3 1	341 840 1	1137 536 1
0 4 1	341 897 1	1137 642 1
0 5 1	341 922 1	1137 1938 1
1 6 1	341 984 1	1137 2594 1
1 7 1	341 1173 1	1137 2595 1
1 8 1	342 27 1	1138 70 1
1 9 1	342 55 1	1138 104 1
2 5 1	342 66 1	1138 2141 1
2 10 1	342 174 1	1138 2346 1
2 11 1	342 197 1	1138 2596 1
2 12 1	342 839 1	1139 62 1
2 13 1	342 931 1	1139 217 1
2 14 1	342 932 1	1139 2597 1
3 15 1	342 1174 1	1139 2598 1
3 16 1	343 3 1	1140 51 1
3 17 1	343 49 1	1140 91 1
4 18 1	343 416 1	1140 511 1
4 19 1	343 1175 1	1140 735 1
4 20 1	343 1176 1	1140 2464 1
4 21 1	343 1177 1	1140 2465 1
4 22 1	344 187 1	1141 36 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：逆向文件频率，是一个词语普遍重要性的度量。我们可以通过总文件数目除以出现该词语的文件数目，再将商取对数得到 IDF。如果所给的数据集（所有文件）中没有出现该词语的话，那么出现该词语的文件数目就为 0，这就会导致分母为 0，所以一般都用（出现该词语的文件数目+1）作为分母，这样就能防止分母为 0 的情况出现。

### 2. IDF 数值有什么含义？TF-IDF 数值有什么含义？

答：① IDF（逆向文件频率）是一个词语普遍重要性的度量。IDF 数值表示一个词语出现的文本频数，即所有文本中有多少个文本中含有该词语。含有该词语的文本数越多，则 IDF 的数值越小，说明了该词语区别不同类别文本的能力越小。

② TF-IDF 的主要思想是：如果某个词或短语在一篇文章出现的频率 TF（TF 表示词条在文本中出现的频率。）高，并且在其他文章中很少出现（IDF 高：IDF 表示一个词语出现的文本频数），则  $TF \cdot IDF$  也高，则认为此词语具有很好的类别区分能力，适合用来分类。所以 TF-IDF 的数值代表了一个词语是否适合用来分类，TF-IDF 的数值越大，则说明该词语具有越好的类别区分能力。

参考：<https://baike.baidu.com/item/tf-idf/8816134?fr=aladdin>



### 3. 为什么要用三元顺序表表达稀疏矩阵？

答：在稀疏矩阵中，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。若我们把稀疏矩阵所有的数据都存储起来，例如用二维数组来存储稀疏矩阵，这不仅会浪费时间，还需要更多的内存。所以我们用三元顺序表来表达稀疏矩阵，只存储其非零元素，节省时间和内存。