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	**15M1**	专业（方向）	**移动互联网**
学号	**15352048**	姓名	**陈潇**

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） 通过处理简单数据集，可以将文本以向量的形式表现出来，对于文本之间的相似性以及对文本中的单词出现的频率及重要性有一个大致的数据表示。

2） One-hot 矩阵可以用向量表现一个文本，其中的单词以 0 或 1 表示在词汇表中该词在该行出现或未出现。将文本分割为多个句子，就可以得到多行，即可以得到一个以【分割的句子数量】，以【词汇表的词汇数量】为列的稀疏矩阵。

3） TF 矩阵可以用向量表现一个文本，与 One-hot 矩阵相似，但得到的稀疏矩阵的每一个值在该行中出现次数的频率，即以出现次数除以该行词数的总数（可重复）。

4） TF-IDF 矩阵是在 TF 矩阵的基础上讲没一行乘以一个 idf 向量，该向量以 log 的形式，值为【文本总数除以该词出现在不同文本中的数量】。

5） 稀疏矩阵三元顺序表是讲稀疏矩阵转化为三元顺序表，按照出现顺序排序。

6） 矩阵加法运算是基于稀疏矩阵三元顺序表进行的运算操作，相同的【行和列】的值相加，不相同的添加为新的行和列。

2. 伪代码

```
int main()
{
    打开总文本文件;
    while(按行读入到字符串 s)
    {
        对 s 分割到需要的句子 s1;
        将 s1 加入到总句子集合;
        对 s1 进行单词分割;
        将分割后得到的单词加入到总单词集合;
        清除总单词集合中重复的单词;
        执行相应需要的矩阵操作;
    }
}
```

```
void One_Hot()
{
    打开输出文本;
    for(对总句子集合中的每一句 s)
    {
        对 s 进行单词分割;
        将分割后得到的单词加入到该行单词向量表 l 中;
        for(对总单词集合的每一个词 w)
        {
            for(对 l 中的每个词)
            {
                if(w 在 l 中)  输出文本输出 1;
                else  输出文本输出 0;
            }
        }
        清空 l;
        输出文本换行;
    }
}
```

```
void TF()
{
    打开输出文本;
    for(对总句子集合中的每一句 s)
    {
        对 s 进行单词分割;
        将分割后得到的单词加入到该行单词向量表 l 中;
        for(对总单词集合的每一个词 w)
        {
            for(对 l 中的每个词)
            {
                if(w 在 l 中)  输出文本输出 w 在 l 出现的次数/l 的容量;
                else  输出文本输出 0;
            }
        }
        清空 l;
        输出文本换行;
    }
}
```

```
void TF_IDF()
{
    打开输出文本;
    创建 map;
    对 map 循环操作，使得其中的对应关系为总单词表中的单词和该单词在不同文章中的
总出现次数;
    其余操作与 TF 类似;
    for(对总句子集合中的每一句 s)
    {
        对 s 进行单词分割;
        将分割后得到的单词加入到该行单词向量表 l 中;
        for(对总单词集合的每一个词 w)
        {
            for(对 l 中的每个词)
            {
                if(w 在 l 中)  输出文本输出 w 在 l 出现的次数/l 的容量 * log(总句子集合
的容量/在 map 中 w 的值);
                else  输出文本输出 0;
            }
        }
        清空 l;
        输出文本换行;
    }
}
```

```
void smatrix()
{
    打开输入文件;
    打开输出文件;
    while(按行把输入文件读入字符串 s)
    {
        for(对该行的每个字符 c)
        {
            if(c == '1')  输出文本输出 c 的行号和  (c 的位置+1)/2  和换行;
        }
    }
}
```

```
void AplusB()
{
    创建全局变量 map mapA，形式为<pair(double,double), double>;
    通过文本输入，指定名称以打开两个保存好的三元顺序矩阵文件
    打开两个输入文件 A，B;
    A，B 的前三行分别是行数、列数、项数，按规则分割后存为 double
    while(对 A 的每一行 s)
    {
        按','分割以获得前两个数值，存为 x, y, 最后一个数值存为 v1;
        存入 mapA;
    }
    while(对 B 的每一行 s)
    {
        按','分割以获得前两个数值，存为 x, y, 最后一个数值存为 v2;
        if(查询 mapA 中有该项) v1+=v2;
        else  存入 mapA;
    }
    按所需规格输出
}
```

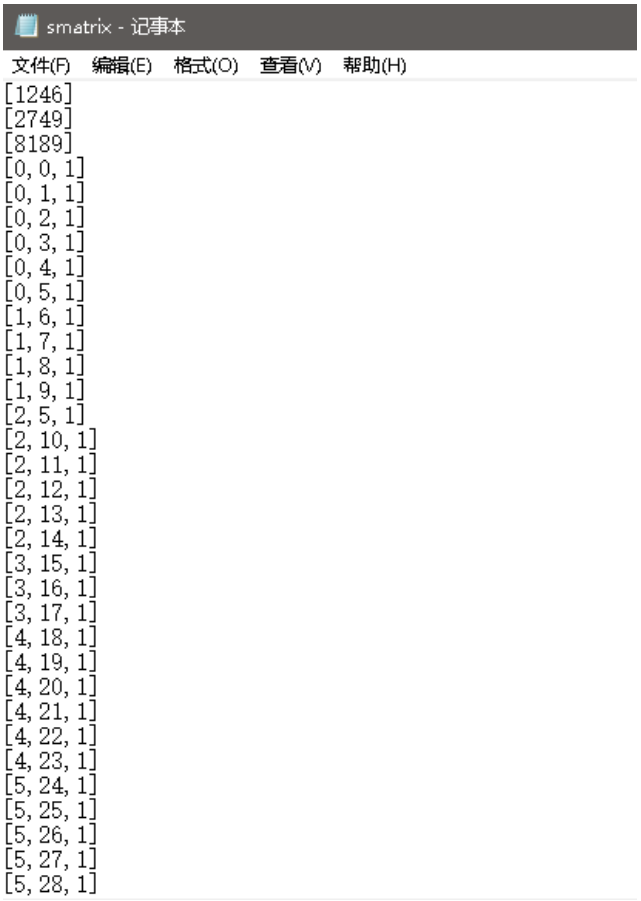
3. 关键代码截图（带注释）

```
//对于文本的每一行，先转换成 string 形式操作
    while(getline(data, s))
    {
//将 string 转 char
        char s1[s.length()+1];
        for(i = 0; i < s.length(); i++)
        {
            s1[i] = s[i];
        }
        s1[i] = '\0';
        const char *d = "\t";
        const char *d1 =" ";
       char *p, *p1;
       p = strtok(s1,d);
//分割两次以得到原数据中以/t 分割开的第三段字符串 p
        p=strtok(NULL,d);
        p=strtok(NULL,d);
//将每一句话放入 vector 中
        ArticleSet.push_back(p);
        p1 = strtok(p,d1);
//将每一个单词放入 vector 中，可重复，后续消除重复
        while(p1)
        {
            DataSet.push_back(p1);
            p1 = strtok(NULL,d1);
        }
    }
```

1） 我使用的是 C++的文件读取和字符串分割的方式，简而言之是将先分割出每一句话，再分割到每一个词，词是重复的，后续通过操作消除重复的单词。这也是我再三个矩阵操作中的分割单词的核心部分。

```
        mapA_it = mapA.find(pair<double, double>(x, y));
        if(mapA_it != mapA.end())
        {
            mapA[pair<double, double>(x, y)] += v;
        }
        else
        {
            mapA.insert( pair<pair<double, double>,
double>(pair<double, double>(x, y) , v) );
        }
```

2） 由于三个矩阵的变换输出部分比较相似，我认为矩阵加法部分是另一部分比较关键的代码，这里我的矩阵是从.txt 文件读取的按照课件中的标准格式的[X,X,X]三元顺序矩阵，所以一开始处理的部分是以逗号进行分割，然后读取字符串后转成 double 存储。我使用了一个全局变量 map mapA，形式是<pair(x, y), v>，也就是 pair 中再套了一个 pair，将得到 x,y,z 以这种<xy 坐标, v 值>的形式存储。原本以为需要再自己写一个 map 中的排序操作，但是没想到按照 map 的键值是 pair 的情况下依然也是按照 first 和 second 的值二元从小到大的默认排序，故相对方便了一些。最后得到的 mapA 中就是相加后得到的结果。



3） 稀疏矩阵三元顺序表的 semeval 样本输出（部分），行、列、总数经过比对是基本正确的。

4. 创新点&优化（如果有）

1） 优化：全程不采用数组等需要写内存的存储操作，仅用 vector 存储总文本的每一句话、所有单词等必要操作（一次），总耗时约 4 秒。

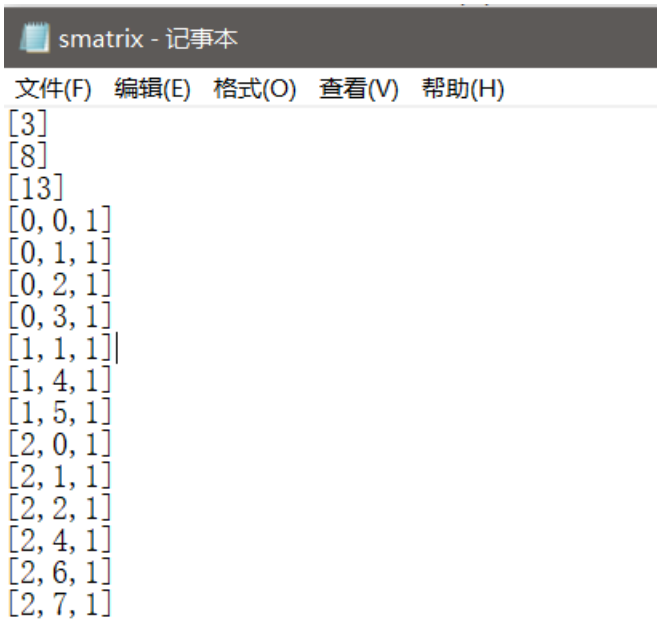
2） 优化：使用 map<pair <x, y>, y>的形式，利用 C++库特性自动进行了排序。

三、 实验结果及分析

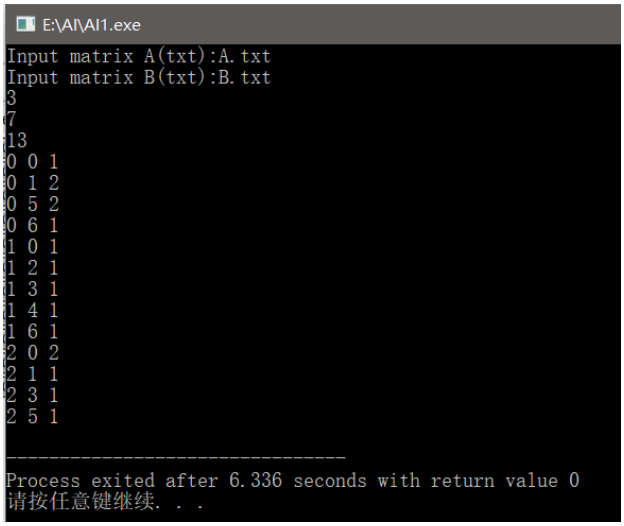
1. 实验结果展示示例（可图可表可文字，尽量可视化）

1） 采用课件所给的训练文本“苹果手机贵”的例子，得出的结果与课件相同。若使用 semeval 训练文本，则也会得到类似的“对角矩阵”的结果。原因是因为按照出现顺序排序的话，新增的单词出现的地方必然是在较靠后的句子中的后半部分，当训练文本的规模很大时，会出现一个类似于对角矩阵，且在下三角部分有零星的数据分布的情况（猜测，非定论）。

2） 另外两个矩阵的验证结果依然与课件中的相同。但是出现 TF 与 TF_IDF 不同的原因是 TF 表示该词在文章中出现的频率越高，则值越大，而 IDF 会降低在文章中反复出现的词的值。



3） 使用三元顺序表运算结果如图，能正确表示。



4） 课件所给例子的加法矩阵运算结果。

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？

多了个 1 是为了考虑更加广泛的应用情况：比如所给的词典超出了训练文本的范围，这就会出现训练文本中的某个单词在总的词典中没有出现过的情况（例如：给定一个新的训练文本，其中的词都是全新出现过的），那么在做矩阵的时候就会出现分母（词典中的词在文本中出现的词数）为 0 的情况使得算数异常，为了避免算数异常的情况，故要分母加 1。

2. IDF 数值有什么含义？TF-IDF 数值有什么含义？

IDF 的数值表示这个词在文章中出现的频率越高，则值越低（例如：我 I 是 is 等对于重要

信息没有帮助反而反复出现的单词，虽然频率高但是不重要）。通过这个 IDF 的数值与 TF-IDF 的数值相乘，就可以得到 TF 矩阵的综合加强版，过滤掉某些出现频率高的日常用语，更加便于重要数据的表现。

3. 为什么要用三元顺序表表达稀疏矩阵？

一方面，稀疏矩阵计算需要更多的内存和时间，且稀疏矩阵的特点就是包含了大量的无数据意义的 0；另一方面，稀疏矩阵难以直观显示，而用三元顺序表都可以规避以上两点问题。

|---------- 如有优化，重复 1，2 步的展示，分析优化后结果 ----------|

PS：可以自己设计报告模板，但是内容必须包括上述的几个部分，不需要写实验感想