# 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 | **M1** | 专业（方向） | **移动信息工程（移动互联网）** |
|---|---|---|---|
| 学号 | **15352102** | 姓名 | **韩硕轩** |

## 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）打开一篇文章

（2）逐行读取到字符串中

（3）逐个分出单词

（4）判断单词是否是数字或情感，不是则计入词库，是则丢弃

（5）重复（3）（4）直到读完单词

（6）重复（2）（3）（4）（5）直到读完所有句子

（7）根据原始的统计矩阵输出 one_hot 矩阵

（8）根据原始的统计矩阵算出 TF 矩阵并输出

（9）根据原始的统计矩阵和 TF 矩阵算出 TFIDF 矩阵并输出

（10）根据原始的统计矩阵算出 smatrix 并输出

（11）输入两个三元顺序表

（12）对于第一个表中的每一项，搜索是否在第二个表中也出现了，然后处理

（13）输出第二个表即为加和答案

### 2. 伪代码

```
While (!文件读完了)
{
    读取下一行
    文本数++;
    While (还有未读单词)
    {
```

If (是数字或者情感) 丢弃

Else

If (之前出现过) 记入之前的位置 else 新开一个位置记录

读取下一个单词

}

}

For I from 1 to 文本数

For j from 1 to 单词数

输出 one_hot

For I from 1 to 文本数

For j from 1 to 单词数

输出 TF

For I from 1 to 文本数

For j from 1 to 单词数

输出 TFIDF

For I from 1 to 文本数

For j from 1 to 单词数

算出 smatrix，统计有多少个值

For I from 1 to 值的个数

输出 smatrix

文件读入 t1 和 t2 两个三元顺序表

For I 遍历 t1

{

If (t1[i]在 t2 中出现) t1[i]加入 t2 的对应位置

Else 在 t2 中新开一个位置加入 t1[i]

}

For I 遍历 t2

输出 t2 每一项

## 3. 关键代码截图（带注释）

（1）变量的定义和作用

```
int num, txt, cnt;                          //分别是不同的单词总数，句子数和三元组的值的个数
int ori[1500][15000];                       //原始的单词统计矩阵。其中第0行表示每个单词在多少个文本中出现过（算idf用）
                                            //第0列表示每句话有多少个单词（算tf用）
double TF[1500][15000], TFIDF[1500][15000]; //TF矩阵和TFIDF矩阵
string words[15000];                        //按照出现顺序记录每一个单词
triad sm, t1, t2, tsum;                     //分别是onehot矩阵的三元组，两个输入的三元组和结果三元组
```

（2）读一行句子和分割字符串

```
while (!fin.eof())                          //52行和54行是一行一行，即一句一句，读入文件的方法
{
    fin.getline(s, 500);
    txt++;
    const char *d = "    ";                 //一个space和一个tab
```

（3）分出词并做统计

```
p = strtok(s, d);
while (p)                                   //这个while循环用来分词
{
    string ss = p;
    if (isdigit(p[0]) || ss.find(":", 0) < string::npos)
    {
        p = strtok(NULL, d);
        continue;
    }                                       //去除数字和表示情感的词
    int pos = search(p);                    //判断是新词还是旧词
    ori[txt][0]++;
    if (pos > 0)                            //如果是旧词就加到原来的位置
    {
        ori[txt][pos]++;
        if (ori[txt][pos] == 1) ori[0][pos]++;
    }
    else                                    //如果是新词就开新的位置存下来
    {
        num++;
        ori[txt][num]++;
        ori[0][num]++;
        words[num] = p;
    }
    p = strtok(NULL, d);                    //分出下一个单词，81行和58行一起分词
}
```

（4）输出 one_hot

```
for (int i = 1; i <= txt; i++)
{
    for (int j = 1; j < num; j++)
        if (ori[i][j] > 0)                  //输出onehot矩阵，如果ori里面大于1则表示出现过，输出1，否则输出0
            onehot << 1 << " ";
        else
            onehot << 0 << " ";
    if (ori[i][num] > 0)
        onehot << 1 << endl;
    else
        onehot << 0 << endl;
}
```

（5）输出 TF

```
for (int i = 1; i <= txt; i++)              //这个双重循环输出TF矩阵，由于这个矩阵在后面还有用所以要存
{
    for (int j = 1; j < num; j++)
    {
        TF[i][j] = ori[i][j] * 1.0 / ori[i][0];
        tf << fixed << setprecision(4) << TF[i][j] << " ";
    }
    TF[i][num] = ori[i][num] * 1.0 / ori[i][0];
    tf << fixed << setprecision(4) << TF[i][num] << endl;
}
```

（6）输出 TFIDF

```
for (int i = 1; i <= txt; i++)                  //这个双重循环输出TFIDF矩阵
{
    for (int j = 1; j < num; j++)
    {
        TFIDF[i][j] = TF[i][j] * log(txt * 1.0 / (ori[0][j] + 1)) / log(2);
        tfidf << fixed << setprecision(4) << TFIDF[i][j] << " ";
    }
    TFIDF[i][num] = TF[i][num] * log(txt * 1.0 / (ori[0][num] + 1)) / log(2);
    tfidf << fixed << setprecision(4) << TFIDF[i][num] << endl;
}
```

（7）计算并输出 smatrix

```
for (int i = 1; i <= txt; i++)              //由ori数组得出smatrix
{
    for (int j = 1; j < num; j++)
    {
        if (ori[i][j] > 0)
        {
            sm.t[cnt][0] = i - 1;           //i和j都减一是因为从1开始统计，第0行和列用处见18行的注释
            sm.t[cnt][1] = j - 1;
            sm.t[cnt][2] = 1;
            cnt++;
        }
    }
}
```

（8）（读取两个三元顺序表不是关键代码）两个顺序表的相加

```
for (int i = 0; i<t1.nums; i++)                    //如果t1中的一个元素在t2中出现过，就直接加上去
{
    bool found = false;
    for (int j = 0; j<t2.nums; j++)
        if (t1.t[i][0] == t2.t[j][0] && t1.t[i][1] == t2.t[j][1])
        {
            t2.t[j][2] += t1.t[i][2];
            found = true;
            break;
        }
    if (!found)                                    //否则，在t2新开一个单元用来存t1的这个元素
    {
        t2.t[t2.nums][0] = t1.t[i][0];
        t2.t[t2.nums][1] = t1.t[i][1];
        t2.t[t2.nums][2] = t1.t[i][2];
        t2.nums++;
    }
}
```

（9）对结果顺序表排序

```
for (int i = 0; i<t2.nums - 1; i++)          //排序t2，因为它存的就是结果，使得按照行优先列次之的顺序输出
    for (int j = i + 1; j<t2.nums; j++)
        if (t2.t[i][0] > t2.t[j][0] || (t2.t[i][0] == t2.t[j][0] && t2.t[i][1]>t2.t[j][1]))
        {
            int tem;
            tem = t2.t[i][0]; t2.t[i][0] = t2.t[j][0]; t2.t[j][0] = tem;
            tem = t2.t[i][1]; t2.t[i][1] = t2.t[j][1]; t2.t[j][1] = tem;
            tem = t2.t[i][2]; t2.t[i][2] = t2.t[j][2]; t2.t[j][2] = tem;
        }
```

### 4. 创新点&优化（如果有）

没有降低算法复杂度的大优化

# 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）

我自己设计了小的数据集 semeval2 验证正确性：

```
1       all:148 anger:22 disgust:2 fear:60 joy:0 sad:64 surprise:0  mortar
assault leav at least dead
2       all:131 anger:0 disgust:0 fear:0 joy:93 sad:0 surprise:38   goal delight
for sheva
3       all:221 anger:18 disgust:0 fear:52 joy:66 sad:20 surprise:65
        nigeria hostag fear dead is freed freed freed
2       all:131 anger:0 disgust:0 fear:0 joy:93 sad:0 surprise:38   goal delight
for sheva
3       all:221 anger:18 disgust:0 fear:52 joy:66 sad:20 surprise:65
        nigeria hostag fear dead is freed
```

从原 semeval 中取了前三句话，将 2、3 两句复制了一遍，并将第三句话最后一个词 freed 重复了两遍。结果如下：

onehot.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

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

首先是 onehot。从结果中看出 4、5 两行和 2、3 两行完全一样，并且第三行最后被复制了的 freed 没有重复出现。

TF.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
0.1667 0.1667 0.1667 0.1667 0.1667 0.1667 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.2500 0.2500 0.2500 0.2500 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.1250 0.0000 0.0000 0.0000 0.0000 0.1250 0.1250 0.1250 0.1250 0.3750
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.2500 0.2500 0.2500 0.2500 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.1667 0.0000 0.0000 0.0000 0.0000 0.1667 0.1667 0.1667 0.1667 0.1667
```

然后是 TF 矩阵，可以看出 freed 的 TF 值是同一句中其他单词的三倍，而最后一句没有复制 freed 的则所有值都是 1/6。

TFIDF.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
0.2203 0.2203 0.2203 0.2203 0.0537 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.1842 0.1842 0.1842 0.1842 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0402 0.0000 0.0000 0.0000 0.0000 0.0921 0.0921 0.0921 0.0921 0.2764
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.1842 0.1842 0.1842 0.1842 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0537 0.0000 0.0000 0.0000 0.0000 0.1228 0.1228 0.1228 0.1228 0.1228
```

最后是 TFIDF，我在这里用计算验证。如果这里的计算是对的，那么前面的矩阵都没有问题。

先看第一个词，应该是$\frac{1}{6}\log_2\frac{5}{1+1}=0.2203$，结果正确，再比如第三行最后一个词，应该是$\frac{3}{8}\log_2\frac{5}{2+1}=0.2764$，也正确，其余同理。

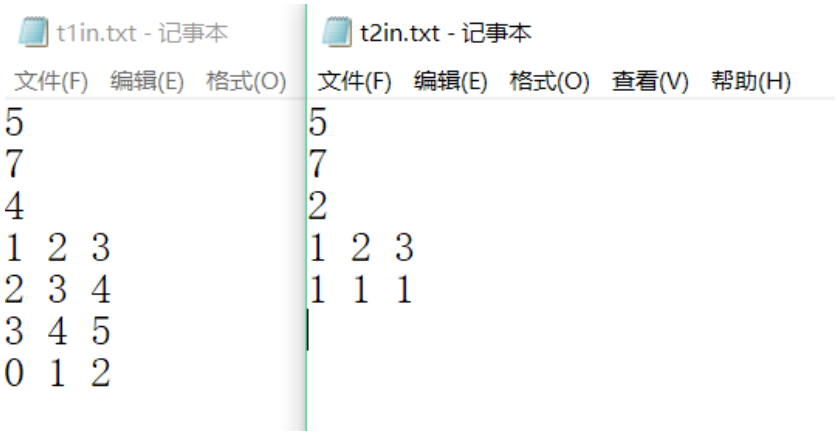
随后的三元顺序表经过和 onehot 的对比也是正确的：

smatrix.txt - 记事本

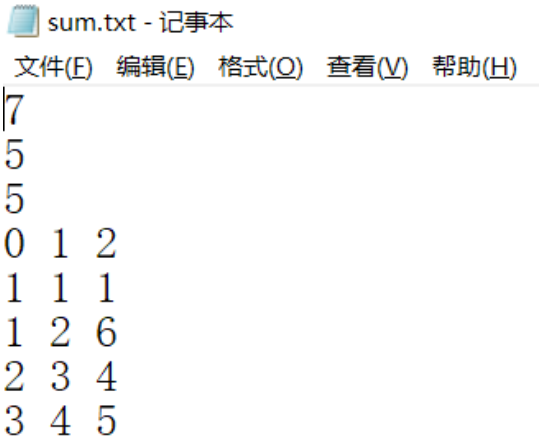
文件(F) 编辑(E) 格式(O

```
[3]
[15]
[24]
[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]
[3, 6, 1]
[3, 7, 1]
[3, 8, 1]
[3, 9, 1]
[4, 5, 1]
[4, 10, 1]
[4, 11, 1]
[4, 12, 1]
[4, 13, 1]
```

最后是三元顺序表加法。我写了如下两个表：




虽然比较小但是包含了行列相同的元素和没按行列优先顺序排序的情况。运行结果如下：




发现相同行列的元素被相加，其余都按照行优先列次之的顺序添加在表里面。

# 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：因为如果该单词没有在任何一个文本中出现（虽然本实验这种情况不会发生），那么分母就是 0。为了防止除以 0 才给分母加上 1。

**2. IDF 数值有什么含义？TF-IDF 数值有什么含义？**

答：IDF 是逆向文件频率，从公式上来看，IDF 值越小，表示一个词出现的次数越多。而 TF-IDF 的值越大，表明一个词不仅出现得多，而且不是那些常用的词汇，说明这个词越重要。

**3. 为什么要用三元顺序表表达稀疏矩阵？**

答：因为稀疏矩阵大多数的值都是 0，如果用矩阵去存会造成存储空间的浪费。用三元顺序表可以大大节省空间开销。