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** 任课教师：**饶洋辉** **批改人(此处为 TA 填写)：**

教学班级	**1506**	专业（方向）	**移动信息工程（互联网）**
学号	**15352116**	姓名	**洪子淇**

一、 实验题目

1. 将数据集“semeval”的数据表示成 One-hot 矩阵，TF 矩阵，TF-IDF 矩阵，并分别保存为“onehot.txt”，“TF.txt”，“TFIDF.txt”三个文件。
2. 将数据集的 One-hot 矩阵表示成三元组矩阵，保存为“smatrix.txt”文件
3. 实现稀疏矩阵的加法运算。

二、 实验内容

1. 算法原理

a) 构建文本集

今次实验运用了 python 去写，在 python 中有一个 split（）函数，可以根据传入的参数对字符串进行分割，并且返回一个 list。根据 semeval 中每行文本数据的格式，可以先用‘\t’把字符串分割成三部分，而我们实验处理的是第三部分的数据，得到第三部分的数据之后，再以‘ ’（空格）将每一个 word 分开，由于最后一个 word 会带有‘\n’（换行符），可以将最后一个 word 用‘\n’分开之后取第一个再赋值回来给最后一个 word，那么就可以得到每一行我们所需要处理的文本数据。

由于文本数据的 word 会有所重复，所以需要对此进行去重处理。方法是构建一个字典集 dictext，遍历每一行的 word，如果字典里面不存在这个 word，就把它放入到一个 list 容器中(这是由于字典在索引的时候不能进行有序排列)。并且对字典里面的 word 的 value 进行初始化，方便接下来对它进行计数。

一行一行地遍历完这个文本之后，我们可以得到一个不重复且有序的文本集 ordertext，得到‘semeval’的行数，以及一个不重复可能无序的字典集 dictext。

b) 构建 TF 矩阵

TF 矩阵表示每一行中的每一个 word 出现的次数归一化的频率。

首先创建一个空的 list 容器 tfline（表示每一行的数据出现的次数），和一个空的 list 容器 count（纪录每一行单词的个数）重新开始对‘semeval’数据每一行进行遍历（开始），用字典映射到每行的 word 中，并对 word 的 value 值加 1（这就需要对字典集的 word 初始化为 0），并记录 word 的个数 c，再将 c 传到 count 容器中。得到不同的 word

出现的次数和 word 的总个数之后就可以归一化了。利用上文得到的有序的文本集 ordertext，对 ordertext 里面的每一个 word 进行遍历，利用字典映射的原理，可以得到 word 出现的次数，除以 word 的总个数之后，利用 tfline 存储每一行的向量，即得到 tf 矩阵的一行，然后将其 append 进入 tf 矩阵中。

c) **构建 One-hot 矩阵**

在 b）可以利用字典集直接得到 one-hot 矩阵的每一行 oline，在对 ordertext 进行遍历的时候，由于 word 的 value 值只有 0 或者大于 0 的值，可以直接利用 bool 函数，将 0 转为 false，将大于 0 的转为 true，然后再将其转化为 int 类型，就可以实现对大于 0 的 value 值都转化为 1 了。然后将每一行的向量 append 到 onehot 容器（list）中。

d) **计算 IDF 和构建 TFIDF 矩阵**

IDF 表示逆向文件频率，假设有 D 行数据，那么它表示对（D 除以 （出现该单词的行数 +1））取对数 。

在 a）中可以得到‘semeval’的行数，而出现单词的行数，可以利用 onehot 矩阵对每一行进行相加得到，便求出 idf 向量。而 TFIDF 矩阵是指 tf 矩阵和 idf 向量进行相乘，直接循环模拟即可。

e) **构建三元顺序表**

三元顺序表是一种表示稀疏矩阵的数据结构：第一行表示行数，第二行表示列数，第三行表示数值个数，下面的每一行表示了矩阵中不为 0 的元素的行号，列号以及数值。

可以通过先遍历矩阵得到矩阵的维度，以及不为 0 的元素个数，输出行号，列号以及元素个数之后，再重新遍历矩阵，对不为 0 的元素进行输出它所在的行列号以及对应数值。

f) **实现两个三元组顺序表的加法**

先对比两个矩阵的维度是否一致，如果不一致，直接输出“The dimension is different!”到文件“triadC.txt”中；如果维度相同，再对两个矩阵进行相加。主要有两种方法：

i. 创建一个新的矩阵，并且对矩阵进行初始化处理，将矩阵里的每一个数都初始化为 0，直接把 A 矩阵的元素加到矩阵对应的位置之后，再把 B 矩阵的元素加到矩阵之中。最后再把矩阵转化为三元组顺序表，复杂度是 n*m（n、m 分别表示行数、列数）

ii. 直接对三元顺序表 A 的每一个三元组和 B 进行比较，如果行号和列号相同，就把 B 的值加到 A 上面，并且把这个三元组从 B 中删除，循环遍历之后，A 和 B 中相同的行号和列号的三元组的相加值覆盖 A 中原来的值，而 B 中将会剩下和 A 中不同行号和列号的三元组，最后把 B 中的值插入到 A 中，最后将其按照顺序排序即可得 A 和 B 相加的三元组。复杂度：N*M（N 表示 A 中三元组的个数，M 表示 B 中三元组的个数，其中忽视了排序的复杂度。）

对比这两种加法的实现，对于维度比较小的三元组适合使用 i 方法，而对于很稀疏的两个矩阵的相加，则更适合使用 ii 方法。本实验，只实现 ii 方法。

2. 伪代码

a) 构建文本集

```
fileI = open(‘semeval’);   //open the dataset
dictext = {};              // a dictionary with the word
ordertext = [ ];           // a vertor with the ordering word
for line in fileI {        //scan every line in the fileI
   PICK_UP(line);          //get the wordsline from the line in the fileI
   for word in wordsline {  //scan every word in the wordsline
      if（word is not in the dictext）{
         push word in ordertext and dictext;
      }
   }
}
```

b) 构建 TF 矩阵

```
initialize the value of the word in dictext;    //set the value 0
TF = [ ];       //create the TF matrix
c = 0;
for word in wordsline {
   c++;         // record the number of the word in wordsline;
   dictext[word] += 1;                  //record the times of the word
   for O_word in ordertext {            //using the ordering in ordertext
      PUSH_tf( dictext[O_word] / c ); //push normalized value of word in TF_line
   }
   PUSH_TF(TF_line);                    //push every TF_line in TF matrix
}
```

c) 构建 One-hot 矩阵

```
//let the value in the TF matrix which greater than 0 become 1 and get onehot matrix
INT(BOOL(TF));
```

d) 计算 IDF 和构建 TFIDF 矩阵

```
IDF = [ ];      //record the times about the word appearing in a dataset(once in a line)
for col in onehot {
   sigma = 0;     //initialize the sigma
   for line in onehot {
      if (onehot[line][col] == 1) {
```

```
        sigma += 1;    //find the times
      }
    }
    PUSH_idf(D/(sigma + 1)); //record idf
}
MULTIPLY( TF,IDF );    //multiply the idf matrix with tf matrix
```

e) **构建三元顺序表**

```
count = 0;              //initialize
for row in matrix{
    for col in matrix{
        if (matrix[row][col] != 0){
            count += 1;                  //record the number of the nonzero
            PUSH_tri (row,col,matrix[row][col]);    //record the triad
        }
    }
}
PUSH_triad(row);
PUSH_triad(col);
PUSH_triad(count);
PUSH_triad(tri);                                  //create the triad
```

f) **实现两个三元顺序表的加法**

```
for triad_a in A{                  //scan the triad_a in A
    for triad_b in B{              //scan the triad_b in B
        if(triad_a_row == triad_b_row and triad_a_col== triad_b_col){
            triad_a_value += triad_b_value;          //add the value
            POP_b(triad_b);        //pop the same row and col in B
        }
    }
}
PUSH_addmatrix(A);             //push the triad_a in the addmatrix
PUSH_addmatrix(B);             // push the triad_b in the addmatrix
SORT(addmatrix);
```

3. 关键代码截图

a) **构建文本集，构建 One-hot，TF，IDF 和 TFIDF 矩阵**

构建文本集：

```
for line in fileI:
    l = line.split('\t')
    words = l[2].split(' ')
    words[len(words)-1] = (words[len(words)-1].split('\n'))[0]
    # text = text + words;
    for oneword in words:
        if oneword not in dictext:
            ordertext.append(oneword)
            dictext[oneword] = 0
    D += 1
```

构建 onehot 和 TF 矩阵：

```
for line in fileI:

    l = line.split('\t')
    words = l[2].split(' ')
    words[len(words)-1] = (words[len(words)-1].split('\n'))[0]
    for oneword in words:
        dictext[oneword] += 1
        c += 1
    count.append(c)
    for key in ordertext:                    onehot和TF矩阵中的每一行
        oline.append(int(bool(dictext[key])))
        tfline.append(dictext[key]/c)
    # print(oline)
    onehot.append(oline)   构建onehot，TF矩阵
    tf.append(tfline)

    oline = []
    tfline = []                    初始化
    c = 0
    for oneword in words:
        dictext[oneword] = 0
```

计算 idf，构建 TFIDF 矩阵：

```
idf = []
col = len(onehot[0])
for j in range(0,col):
    sigma = 0
    for i in range(0,D):
        if onehot[i][j]:
            sigma += 1
    # print(sigma)
    # base-e logarithmic
    idf.append(np.log2(D/(sigma+1)) )
# print(idf)
tfidf = []
for i in range(0,D):
    item = copy.copy(tf[i])
    for j in range(0,col): 得到idf后与tf矩阵相乘，注意
        item[j] *= idf[j]  python的浅拷贝问题
    tfidf.append(item)
```

b) 矩阵转换三元顺序表

```
fileO = open('smatrix.txt','w')
fileO.write('[' + str(row) + ']\n')
fileO.write('[' + str(col) + ']\n')
fileO.write('[' + str(count) + ']\n')

for i in range(0,row):
    for j in range(0,col):
        if int(data[i][j]) != 0:
            fileO.write('[' + str(i) + ', ' + str(j) + ', ' + str(data[i][j]) + ']\n')
```

c) 三元顺序表加法

```
for i in range(3,len(triadA)):
    for j in range(3,len(triadB)):
        if triadA[i][0] == triadB[j][0] and triadA[i][1] == triadB[j][1]:
            triadA[i][2] = str(float(triadA[i][2]) + float(triadB[j][2]))
            triadB.pop(j)
            break

for i in range(3,len(triadA)):
    triadA[i][2] = round(float(triadA[i][2]),4)
for i in range(3,len(triadB)):
    triadB[i][2] = round(float(triadB[i][2]),4)

for i in range(3,len(triadA)):
    if triadA[i][2] != 0:
        triadC.append(triadA[i])
for i in range(3,len(triadB)):
    triadC.append(triadB[i])
triadC.sort()
```

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）

部分展示（详见附件）

- 展示 onehot，TF 和 TFIDF 矩阵

```
[1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0,
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

onehot.txt

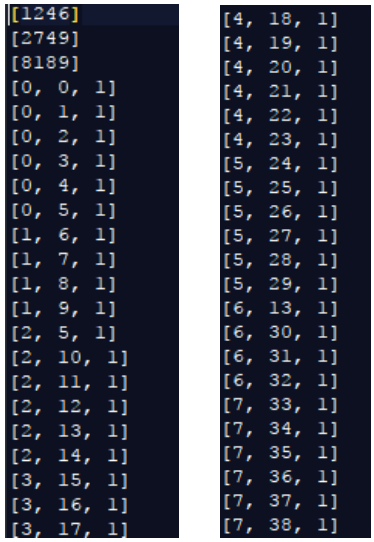
```
[0.16666666666666666, 0.16666666666666666, 0.16666666666666666, 0.16666666666666666,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.25, 0.25, 0.25, 0.25, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.16666666666666666, 0.0, 0.0, 0.0, 0.0, 0.166666666666666
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.333333
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.25, 0.0, 0.0, 0.
```

TF.txt

```
[1.547181392170667, 1.4496876420504743, 1.1855272252636149, 0.7780630848181322, 1.4496
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.0707720882560006, 2.3207720882560006, 0.8410562789373
[0.0, 0.0, 0.0, 0.0, 0.0, 1.0971081058138181, 0.0, 0.0, 0.0, 0.0, 1.2830209753838075,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.0943627
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.1978088141735819,
```

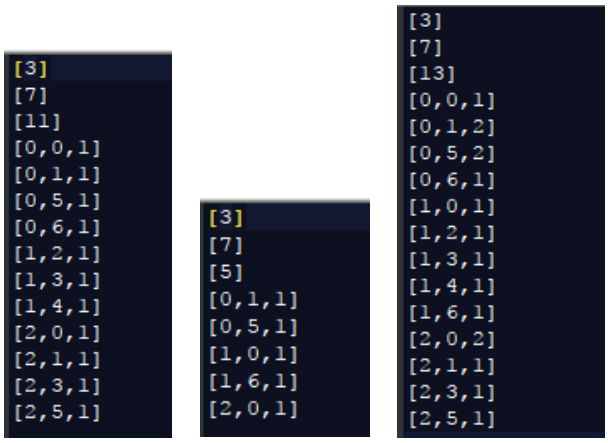
TFIDF.txt

- 展示 onehot 矩阵的三元组顺序表



smatrix.txt

- 展示 triadA 与 triadB 的加法：



triadA + triadB = triadC

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 D 的值可能与$|\{j:t_i \in d_j\}|$的值相等，取对数之后结果为 0，这样就对 TFIDF 的计算产生了较大的误差。TFIDF 矩阵的 0 值将不能被判定为这个单词没有出现在这一行。为了防止这个歧义，最好加上一个 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，公式是$\text{idfi} = \log(\frac{|D|}{|\{j:ti \in di\}|})$，|D|表示文章数，分母表示出现该单词的文章总数，当分母越小的时候，idf 值会越大，反映了当这个单词出现次数比较少时，可以很好区分文本的数据消息的不同，即是说，这个单词在文本消息上的区分度比较高，有

很大的权重。

3. 为什么要用三元顺序表表达稀疏矩阵?

答: 稀疏矩阵中，数值为 0 的元素个数远多于数值不为 0 的元素个数，对应会有一个稠密矩阵与之相反。稀疏矩阵本身不关注数值为 0 的元素，如果用二维矩阵的数据结构来存储稀疏矩阵的话，存储空间比较大，如果需要查询非 0 元素，需要用双重循环来查找，时间复杂度也比较高。如果直接用三元顺序表存储，前三行存储矩阵的维度以及非 0 元素的个数，下面存储了每个非 0 元素的行列号以及数值，存储空间比较小，利用率比较高，而且直接列出非 0 元素，当需要用到时，查找时间复杂度也比较低。

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