

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		姓名	Jw

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

One hot: One-hot 编码就是独热编码,就是用 N 位寄存器来对 N 个状态进行编码,每个状态都由它独立的寄存器位表示,并且在任意的时候,其中只有一位有效。可以这样理解,对于每一个特征,如果它有 m 个可能值,那么经过独热编码后,就变成了 m 个二元特征。并且,这些特征互斥,每次只有一个激活。因此,数据会变成稀疏的。这样做的好处是:

1. 解决分类器不好处理属性数据的问题。
2. 在一定程度上也起到了扩充特征的作用。

在本次实验中,用一个向量表示一篇文章,向量的长度为词汇表的大小,1 表示存在的单词,0 表示不存在,所以词汇就相当于前面说的状态。将结果保存在 onehot.txt 中。

TF(Term Frequency): TF 即词频,在一份给定的文章里,词频就是某一个词在该文章中出现的次数。这个数字通常需要归一化,以防止它偏向长的文章。(同一个词语在长文件里可能会比短文件有更高的词频,而不管该词语重要与否。)对于在某一特定文章里的词语 t_i 来说,它的重要性可表示为:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

以上式子中 $n_{i,j}$ 是该词在文章 d_j 中的出现次数,而分母则是在文章 d_j 中所有字词的出現次数之和。

在本次实验中,TF 矩阵就是将 one-hot 矩阵中的每一行的各个值都表示为该值/该行有值的个数。将结果保存在 TF.txt 中。

TF-IDF(term frequency-inverse document frequency):这是在 TF 的基础上加入 IDF 的字词重要性评估方法。字词的重要性随着它在文章中出现的次数成正比增加,但同时会随着它在语料库中出现的频率成反比下降。IDF 是**逆向文件频率**,可以由总文章数目 $|D|$ 除以包含该词语之文章数目 $|\{j:t_i \in d_j\}|$ 再将得到的商取对数,得到

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

然后

$$tfidf_{i,j} = tf_{i,j} \cdot idf_i$$

将结果保存在 TFIDF.txt 中。

稀疏矩阵三元顺序表: 前面说的 one-hot 矩阵是稀疏的,它的非 0 元素很少,而且分布没有规律,如果用二维数组来存储,会造成大量的空间浪费,所以我们对稀疏矩阵一般采取压缩存储的方法,即只存储其非零元素,其中一种常用方法就是使用三元顺序表。三元组是指形如 (x,y,z) 的集合,其中 x 和 y 表示该元素在第 x 行和第 y 列, z 为该元素的值。假设以顺序存储结构来表示三元组表,就称为顺序三元组表。一个例子如下:

	3	行数	
	8	列数	
	13	数值个数	
0	0	1	1
1	0	4	1
2	0	6	1
3	0	7	1
4	1	3	1
5	1	5	1
6	1	6	1
7	2	0	1
8	2	1	1
9	2	2	1
10	2	4	1
11	2	5	1
12	2	6	1
	行号 i	列号 j	数值 k

本次实验中需要将 one-hot 矩阵表示成三元顺序表并保存在 smatrix.txt 中

2. 伪代码

代码采用 python 来写

1.文本数据处理

Begin

file = 打开相应的数据文件

lines = file.readlines() /*读取文本中每一行数据,每一行是一个 list*/



```
for line in lines: /*遍历每一行*/
    for word in line: /*遍历每一行的每个单词
        IF word in Hash:
            Hash[word] = 这个 word 在全部文章中是第几个出现的
        If word 是第一次出现在当前文章:
            Num[word]+1
    Num[] = 相应的 IDF 值
/*获取每个矩阵, c1 是 one-hot, c2 是 one-hot 三元顺序表, c3 是 TF,
    C4 是 TFIDF
*/
[c1,c2,c3,c4] = getContent(lines,Hash,num)
输出 c1,c2,c3,c4 到相应文本中去
End
getContent(lines,Hash,num):
    初始化 c1,c2,c3,c4 为空
    c2[0],c2[1],c2[2] = 0 /*这三个是三元组的行数、列数、数值个数
    for line in lines: /*遍历每一行*/
        for word in Hash: /*遍历词汇表*/
            if word 在当前行中出现:
                更新 c2 中的行数、列数的最大值
                增加 c2 中的数值个数
                增加一组三元组到 c2
                c1 增加 1 到当前行的末尾
                c3 增加当前行 word 出现的次数到当前行的末尾
                c4 增加前行 word 出现的次数*IDF 到当前行的末尾
            else:
                c1、c2、c3 增加一个 0 到当前行末尾
        将 c3 和 c4 中的当前行的每个值 / 当前行的非 0 元素的个数
    return [c1,c2,c3,c4]
```

2.AplusB.

Begin

A = 读入的 A 三元顺序表

/*A[0],A[1],A[2]分别为行数、列数、数值个数, A[3]起是一三元组, 用 list 表示, 下同

*/

B = 读入的 B 三元顺序表

C[0] = max(A[0],B[0])



```
C[1] = max(A[1],B[1])
hash = [] /*存储最后一共有哪几个二维坐标*/
d = DICT()
/*字典，判断(x,y)是否出现过，d[(x,y)]为该坐标代表元素的最终值*/
for i in range(3,len(A)):
    key = (A[i][0],A[i][1]) /*以三元组前两位为关键字*/
    if key 没出现过:
        d[key] = A[i][3]
        hash.append((A[i][0],A[i][1]))
    else:
        d[key] += A[i][3]
对 B 的处理同 A，但是当 d[key]=0 时要删掉
对 hash 中的二元坐标 按第一位为第一关键字，第二位为第二关键字排序
C[3] = len(d)
按照 hash 中的坐标顺序一个一个将三元组加入到 C
return C
End
```

3. 关键代码截图（带注释）

1. 文件数据处理

数据读入及预处理:

```
def pre_handle(filepath):
    file = open(filepath, "r")
    lines = file.readlines()
    file.close()
    #提取每一篇文章的单词，一篇文章的单词作为一个list
    lines = list(map(lambda x: x[x.rindex('\n')+1:-1].split(), lines))
    #Hash统计每个单词是第几次出现的，num统计每个单词出现在多少篇文章中
    Hash, num = dict(), dict()
    cnt, row = 0, 0
    for line in lines:
        row += 1
        appearnum = set() #appearnum用来判断单词在当前文章中是否已经出现过
        for word in line:
            if word not in Hash:
                Hash[word] = cnt
                cnt = cnt + 1
                num[word] = 1
                appearnum.add(word)
            else:
                if word not in appearnum:
                    num[word] = num[word] + 1
                    appearnum.add(word)
        for word in num:
            num[word] = math.log(row/(1+num[word]), 2) #将num转变为IDF值
    Hash = sorted(Hash.items(), key=lambda x: x[1]) #将单词按照出现顺序排序
    return [lines, Hash, num]
```



计算各个矩阵:

```
def getContent(Lines,Hash,num):
    content1 = "" #onehot matrix
    content2 = [0,0,0] #onehot three triple matrix [maxrow,maxcol,valnum]
    content3 = "" #TF matrix
    content4 = "" #TF-IDF matrix
    row = -1
    for line in Lines: #遍历每一篇文章
        l_oh, l_TF, l_TFIDF = [], [], [] #分别记录one-hot,TF,TFIDF当前行的值
        col = -1; row += 1
        current_line_val = 0
        for word in Hash: #遍历词汇表
            col += 1
            word = word[0]
            if word in line: #如果该文章中包含单词word
                content2[0] = row + 1 if row + 1 > content2[0] else content2[0] #更新三元组顺序表的行
                content2[1] = col + 1 if col + 1 > content2[1] else content2[1] #更新三元组顺序表的列
                content2[2] += 1 #数值个数+1
                content2.append(["+str(row) + ", " + str(col) + ", " + "1"]) #增加一个三元组到顺序表中
                #one-hot矩阵新增一个1, TF, IDF矩阵都新增该词在该行出现的次数, 其中TFIDF先乘上一个IDF值
                l_oh.append(1)
                word_num = line.count(word)
                l_TF.append(word_num)
                l_TFIDF.append(word_num*num[word])
                #统计该文章中有多个非0值, 用于之后计算TF和IDF
                current_line_val += word_num
            else: #如果该文章中不包含单词word, 则one-hot, TF, TFIDF矩阵都新增一个0
                l_oh.append(0)
                l_TF.append(0)
                l_TFIDF.append(0)
        #将一篇文章的one-hot, TF, TFIDF,one-hot三元组结果转变成一行字符串加到相应的content中, 方便后面写入文件
        #TF和IDF要除以一个当前行的单词总数, 结果保留6位小数
        content1 += ' '.join(list(map(lambda x:"0" if x==0 else "1",l_oh))) + '\n' #one-hot
        content3 += ' '.join(list(map(lambda x:"0" if x==0 else str(round(x/current_line_val,12)),l_TF))) + '\n' #TF
        content4 += ' '.join(list(map(lambda x:"0" if x==0 else str(round(x/current_line_val,12)),l_TFIDF))) + '\n' #TFIDF
        content2 = '\n'.join(list(map(lambda x:x if isinstance(x,str) else "["+str(x)+"]",content2))) #one-hot 三元顺序表
    return [content1,content2,content3,content4]
```

2.A plus B

```
def plus(A,B):
    C = [0,0] #初始化行数和列数
    d = dict()
    hash = []
    for i in range(3,len(A)):
        l = A[i]
        key = (l[0],l[1]) #以(x,y)为关键字判断该坐标是否出现过
        if key not in d: #没有出现过则新增一个
            d[key] = l[2]
            hash.append((l[0],l[1]))
        else: #出现过则该坐标的值 加上新的值
            d[key] += l[2]

    #B的处理同A
    for i in range(3,len(B)):
        l = B[i]
        key = (l[0],l[1])
        if key not in d:
            d[key] = l[2]
            hash.append((l[0], l[1]))
        else:
            d[key] += l[2]
            if d[key] == 0: #注意这里是删除值为0的元素
                del d[key]
            hash.remove((l[0], l[1]))
    hash = sorted(hash, key=itemgetter(0,1))
    for item in hash:
        C[0] = max(C[0],item[0]+1)
        C[1] = max(C[1],item[1]+1)
    C[0] = "[" + str(C[0]) + "]";
    C[1] = "[" + str(C[1]) + "]";
    C.append "[" + str(len(d)) + "]"
    for key in hash:
        C.append "[" + str(key[0]) + ", " + str(key[1]) + ", " + str(d[key]) + "]"
    C = '\n'.join(C)
    return C
```

4. 创新&优化部分

本次实验是简单的数据处理，没有什么好优化的地方。

三、 实验结果及分析

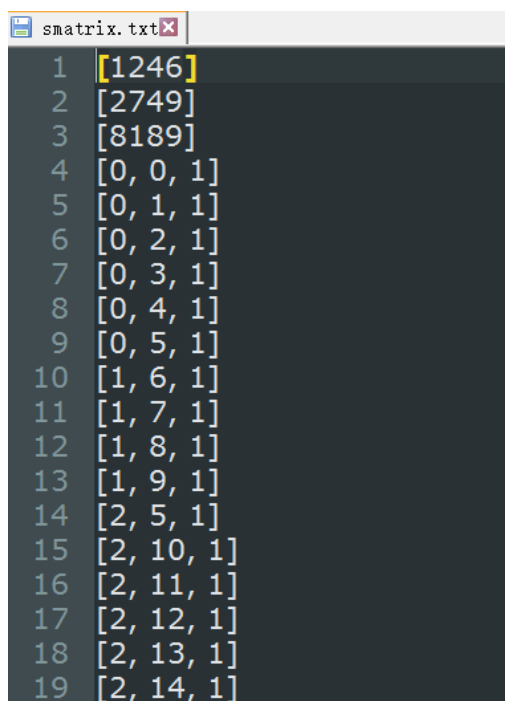
1. 实验结果展示示例（可图可表可文字，尽量可视化）

注：所有运算涉及到小数的最后结果均保留 12 位小数

One-hot: 由于结果数字太多，这里仅展示部分结果，完整结果请查看 [onehot.txt](#)



samtrix: 由于结果数字太多，这里仅展示部分结果，完整结果请查看 [samtrix.txt](#)



[illegible][illegible]

```
A: A.txt [X]
1 [3]
2 [7]
3 [11]
4 [0, 0, 1]
5 [0, 1, 1]
6 [0, 5, 1]
7 [0, 6, 1]
8 [1, 2, 1]
9 [1, 3, 1]
10 [1, 4, 1]
11 [2, 0, 1]
12 [2, 1, 1]
13 [2, 3, 1]
14 [2, 5, 1]
```

```
B: B.txt
1 [3]
2 [7]
3 [5]
4 [0, 1, 1]
5 [0, 5, 1]
6 [1, 0, 1]
7 [1, 6, 1]
8 [2, 0, 1]
9
```

```
C: C.txt [X]
```

1	[3]
2	[7]
3	[13]
4	[0, 0, 1]
5	[0, 1, 2]
6	[0, 5, 2]
7	[0, 6, 1]
8	[1, 0, 1]
9	[1, 2, 1]
10	[1, 3, 1]
11	[1, 4, 1]
12	[1, 6, 1]
13	[2, 0, 2]
14	[2, 1, 1]
15	[2, 3, 1]
16	[2, 5, 1]

2. 评测指标展示即分析（如果实验题目有特殊要求，否则使用准确率）

- 1.对于 one-hot, TF, TFIDF, 三元顺序表来说，主要看求出的矩阵的每一个元素的位置以及值是否准确。求出的结果与其他一些同学比较后基本一致
- 2.对于两个三元系数矩阵相加，看结果是否正确。

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：防止因为涉及到的词语不在语料库中而导致分母为 0，进而使计算出错的情况。

2. IDF 数值有什么含义？TF-IDF 数值有什么含义？

答：1. IDF 数值是逆向文件频率，是一个词语普遍重要性的度量。算法是 $\log(\text{总文档数}/(\text{出现该词语的文件数}+1))$ ，由该公式可以看出，一个词语出现的次数越小，它的 IDF 值越大，这说明该词语具有良好的类别区分能力。

2. TF-IDF 是综合了 TF 和 IDF 的优点，IDF 是可以很好地找被少数文件包含的词语。但如果某一类文档 C 中包含词条 t 的文档数为 m，而其它类包含 t 的文档总数为 k，显然所有包含 t 的文档数 $n=m+k$ ，当 m 大的时候，n 也大，按照 IDF 公式得到的 IDF 的值会小，就说明该词条 t 类别区分能力不强。但是实际上，如果一个词条在一个类的文档中频繁出现，则说明该词条能够很好代表这个类的文本的特征，这样的词条应该给它们赋予较高的权重，并选来作为该类文本的特征词以区别与其它类文档。这就是 IDF 的不足之处。这是应该增加词频即 TF 值，它是一个词语在某个文件出现的频率， $TF-IDF = TF * IDF$

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的 TF-IDF。因此，TF-IDF 倾向于过滤掉常见的词语，保留重要的词语

3. 为什么要用三元顺序表表达稀疏矩阵？

答：如果不用三元顺序表，而采用二维数组来表达稀疏矩阵，会造成大量的空间浪费，因为这个二维矩阵中有用的元素非常少。