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	1501	专业（方向）	移动互联网
学号	15352015	姓名	曹广杰

Contents

一、 实验题目

文本数据集简单处理

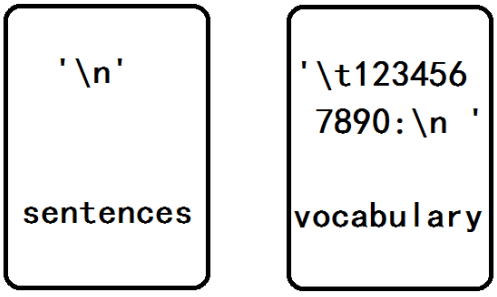
二、 实验内容

算法原理

在处理词汇之前，需要将整个文本进行分析与拆解。

1）将文本根据回车分解成为不同的句子，并储存起来；

2）将整篇文章根据分解符更加细致地分解成词汇，并储存起来；



将数据集“semeval”的数据表示成 One-hot 矩阵：

Onehot 矩阵主要测试不同的句子中是否含有某些词汇。

1. 在句子集合中遍历句子；
2. 在遍历循环中嵌套遍历词库；

当前我们已经有：句子的集合、词汇的集合。

所以我们只需要对每一个句子检查是否存在某个词即可，遍历句子的集合并在集合中遍历词汇的集合，将存在与否的信息作为矩阵元素输出到文件中即可。

将数据集“semeval”的数据表示成 TF 矩阵：

表示为 TF 矩阵，即为表示词语出现的次数归一化后的频率。

1. 在句子集合中遍历每一个句子；
2. 对每一个句子中出现的词汇重新构建词库，词汇可以重复；
3. 使用循环结构统计词出现的次数，归一化计算权值；

当前我们已经有：句子的集合、词汇的集合。

所以我们需要统计在不同的句子中每个词汇出现的次数，以及所有词汇出现的总次数。考虑到这一点就不能再使用之前的词汇集合了，因为那个集合中的元素绝无重复。需要对每一个句子重新构建词汇的集合，笔者使用 vector 数据结构用于储存某一个句子中所有的词汇出现的信息，无论重复与否。现在对不同的句子有不同的输出，在分析每一个句子并输出分析结果之前都需要对可重复出现词汇的集合初始化，以便多个句子中不会互相干扰。

完成这些准备之后，对所有句子分析操作得到的权值进行输出，输出为矩阵模式。

将数据集“semeval”的数据表示成 TF-IDF 矩阵：

实现 TF-IDF 矩阵其实还是基于 TF 矩阵的实现方法，在每一个系数之前乘以一个对数值。至此，系数的计算就不需要重新实现，沿用 TF 矩阵的求算方法即可。

而对于对数部分：log（文章总数目/含有该词的文章数），则需要重新统计。

1. 文章总数，实际就是句子数目，可以直接使用我们之前的句子集合的数目返回值；
2. 含有该词的文章数目，则需要对所有的文章重新统计，或者整理 onehot 矩阵；

为了保证数据运算的独立性，笔者这里使用重新统计的方法，格式为双重循环。外部循环为对于词库的遍历，内部遍历句子的集合，同时统计含有某一个词的文章数目。将词（字符串格式）与含有词的文章数（整数形式）联系起来可以使用 c 语言库中的 STL-map。

得到数据之后，将输出数据修正添加 IDF 矩阵的部分，即可。

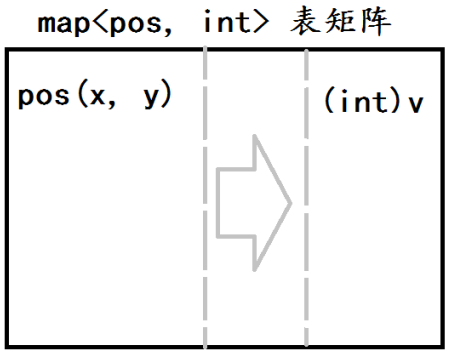
将数据集的 One-hot 矩阵表示成三元组矩阵：

处理 onehot 矩阵为三元组，需要在已有的 onehot 矩阵的基础上，通过逐个字符数据的读入，找出所有值为 1 的数据，在遍历过程中注意换行符以及初始化计算其所在坐标，综合输出即可。

实现系数矩阵加法运算：

实现三元组矩阵加法，实际上就是实现：位置坐标的查询、系数数值的相加以及在新数值组插入时保证有序。

那么我们处理的对象矩阵，表示它的容器应该是同时具有以下特点：有序、可以查询、可以插入新数值。这里 STL 库中的 map 数据结构同时具有以上所有特点，故而被优先考虑。Map 作为储存矩阵的数据结构，那么每一行三个数据，就分别代表该数字的位置信息以及数值信息，为了方便根据坐标位置查询数值，可以将坐标位置独立分离出来作为一个类，便于管理。



这样，在矩阵 a 与 b 相加时，对矩阵 b 中的每一行：

如果坐标与 a 的坐标重复，根据 map 实例访问参数，直接相加；

如果坐标信息在 a 中不存在，则使用 insert 函数，在矩阵 a 中有序插入一行。以上；

伪代码

实现构建句子库以及词库的伪代码如下：

```
char c;
while(从文件中读取字符c){
    根据'\n'分割为句子s_reg;
    句子库.push_back(s_reg);
}

词库.push_back(被拆分的词);
```

Onehot 矩阵：

```
#ifdef one_hot
ofstream fout("one-hot.txt");
for(遍历句子库){
    string s = 当前句子;
    for(遍历词库){
        输出查词结果;
    }fout << endl;
}
#endif
```

TF 矩阵：

```
#ifdef tf
ofstream fout("TF.txt");
for(遍历句子库){

    构建当前句子s的词库(含有重复词);
    构建映射(词 -> 归一化权值);

    for(遍历该句子词库){
        fout << 该词权值;
    }
    fout << endl;
}
#endif
```

TF-IDF 矩阵：

```
#ifdef TFidf
ofstream fout("TF_IDF.txt");
for(遍历词库){
    for(遍历句子库)
        统计记录文件数目(含有某词);

for(遍历句子库){
    for(遍历词库){
        if(没找到嘿嘿)  fout << 0 << ' ';
        else{
            fout << 由TF计算的权值
                 * log( (词库容量 / (1+含有该词的文件数目(43 line)) )
                 << ' ';
        }
    }fout << endl;
}
#endif
```

转化为三元组：

```
    fstream f("one-hot.txt", ios::in);
    ofstream fout("smatrix.txt");
    while((c = f.get()) != EOF){
//      只找1的数据;
        if(c == '1'){
            数据总量++;
//          不断更新位置信息;
            行计数 + 列计数;
            输出位置信息以及数值信息;
        }
        列数++;
//      另起一行的时候，更新位置信息;
        if(c == '\n'){
            得到最多有多少列;
            列数 = 0;
            行数++;
        }
    }
```

三元组矩阵加法：

```
map<位置, 数值, comp>
AplusB(map<位置, 数值, comp> a, map<位置, 数值, comp> b){
//  查询b中的映射元素,  一个都不要放过;
    for(遍历映射b){
        if(当前位置存在与a中){
//          哎呀找到了,  更新数值;
            a对应的数值 += b当前位置的值;
        }else{
//          map会自动排序,  所以不方;
            a.insert(遍历到的当前映射);
        }
    }
    return a;
}
```

关键代码截图

那师兄我就只放截图了哈，按规矩行事。放心放心，截图我也是挑着放的。

拆分文件为句子集：

```
    while((c=f.get()) != EOF){
//      prepare sentences from the paragraph;
        s_reg += c;
        s_tmp += c;
//      split the parageaph by milestones;
        if(c == '\n'){
            vcts.push_back(s_tmp);
            s_tmp = "";
        }
    }
```

拆分句子为词库：

```
    char *vocabulary;
    vocabulary = strtok(stxt, d);
    vector<string> vctvSame;
//  continue the spliting process until ...
    while(vocabulary){
//      vectorSame designed for calculating TF matrix;
        if(same)    vctvSame.push_back(vocabulary);
        else{
//          To make sure all elements are diff;
            vector<string>::iterator it
                = find(vctv.begin(), vctv.end(), vocabulary);
            if(it == vctv.end())
                vctv.push_back(vocabulary);
        }
//      With Null, won't change sentence to split;
        vocabulary = strtok(NULL, d);
    }
```

Onehot 矩阵就不放了，仅仅是一个二重循环而已；

TF 矩阵：

计算权值的部分：Cwgt 代表 calculate weight

```
//calculate weight with vector containing the same elements;
map<string, double> Cwgt(vector<string> v, bool IDF) {

//  sum 为出现的词汇总数;
    int sum = 0;
//  为了实现“ 词汇 => 数目”的索引，使用map结构;
    map<string, int> m;
    for (int i = 0; i < v.size(); i++) {
        m[v[i]] += 1;
        sum += 1;
    }
//  实现“ 词汇 => 归一化系数”的索引;
    map<string, double> mwgt;
    for (int i = 0; i < v.size(); i++) {
        string s = v[i];
        mwgt[s] = (double)m[s] / sum;
    } return mwgt; }
```

TF-IDF 矩阵：

TF-IDF 的实现，在计算含有某词的文本数目的时候，使用的是基于 onehot 的代码——而在计算权值的时候，则是照搬的 Tfmatrix 的实现代码，乏善可陈。

转化为三元组：

转化为三元组的实现中值使用了一重循环,搜寻到即输出,笔者觉得不必费心劳神审阅。

三元组矩阵加法：

```
    //比较困难的是数据结构的选择和处理，而非算法;
    map<pos, int, comp> AplusB(map<pos, int, comp> a, map<pos, int, comp> b){
    //  遍历b，对各个元素逐一检查修整;
        for(map<pos, int, comp>::iterator it=b.begin(); it!=b.end(); it++){
            pos pb = it->first;
            int value = it->second;
    //      若a中出现了该位置坐标，数值叠加;
            if(a[pb]){
                a[pb] += it->second;
            }else{
                a.insert(pair<pos, int>(pb, 0));//注意我插了个0;
                a[pb] += value;
            }
        }
        return a;
    }
```

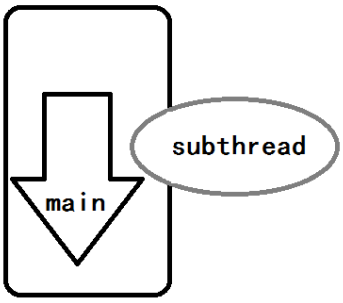
创新点&优化

有一件事一直让笔者不能释怀，那就是将稀疏矩阵转化为三元组的流程。常规流程是对于矩阵进行遍历，找到 1，则即时输出位置坐标继而是存在符号，可是其前面理应还需要输出三元组的行数，列数以及存在的非零数据总量，可惜的是这三者只有在最后遍历之后才可以得到数据。

```
3
8
13
0 0 1
0 1 1
0 2 1
0 3 1
1 1 1
1 4 1
1 5 1
2 0 1
2 1 1
2 2 1
2 6 1
2 4 1
2 7 1
```

如果想在文端就输出前 3 个数据，恐怕不得不将三元组的主题部分储存到一个凉快的地方以便为最后得到的数据输出让路，但是储存这一操作，本身就是一种资源的浪费。为了避免多余的储存操作，笔者在原有的单线程基础上进行了修改。

说来简单，就是把输出矩阵的部分强行分出一个线程，并通过信号量操作把这些个**子线程全都吊起来**，静静等待主线程输出完，并释放信号量，子线程才开始逐一输出。



关键代码如下（我就不贴伪代码了么么哒）：

```
//          subthreads for printing sparse;
            v.push_back(thread(print, l, r));
        }
        row++;
        if(c == '\n'){
            rowmax = row-1;
            row = 0;
            line++;
        }
    }

    fout << line << endl;
    fout << rowmax << endl;
    fout << datacnt << endl;
    mtx.unlock();
//  release the subthreads;
    for(auto& subthread : v){
        subthread.join();
    }
```

注意一点，子线程的输出时，传入参数一定要是局部变量，全局变量的话，所有线程都会随着全局变量的变化而输出同样的结果。

三、 实验结果及分析

实验结果展示示例

实验结果展示实例，我的天哪！实验结果展示实例！我怎么展示。。。

朋友们，为了证实我的算法是正确的，或者说大致是正确的，或者说应该是正确的，或者说可以得到预期的结果。我们先看个小栗子，还是以苹果肾 8 为例：

测试 onehot 矩阵的输出：

us.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

苹果 手机 好用 销售
市民 买 手机 手机
市民 觉得 苹果 手机 贵 好用

onehot.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

1 1 1 1 0 0 0 0
0 1 0 0 1 1 0 0
1 1 1 0 1 0 1 1

onehot.txt - 记事本

文件(F) 编辑(E) 格式(O)

1 1 1 1 0 0 0 0
0 1 0 0 1 1 0 0
1 1 1 0 1 0 1 1

瞧啊瞧啊，与 PPT 上的输出刚好符合。

测试 TF 矩阵是否合理：

TF.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

0.25 0.25 0.25 0.25 0 0 0 0
0 0.5 0 0 0.25 0.25 0 0
0.166667 0.166667 0.166667 0 0.166667 0 0.166667 0.166667

tf.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

0.25 0.25 0.25 0.25 0 0 0 0
0 0.5 0 0 0.25 0.25 0 0
0.166667 0.166667 0.166667 0 0.166667 0 0.166667 0.166667

测试 TF-IDF 矩阵的算法是否符合预期：

TFIDF.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

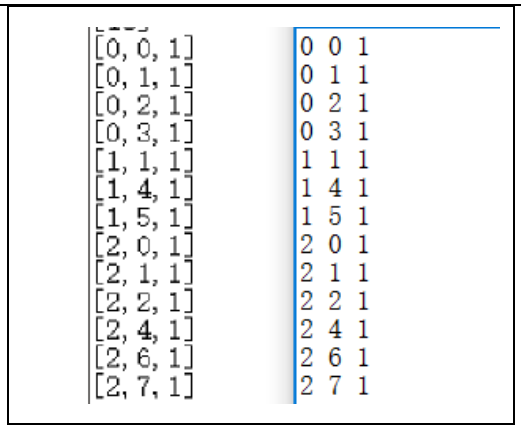
0 -0.0719205 0 0.101366 0 0 0 0
0 -0.143841 0 0 0 0.101366 0 0
0 -0.047947 0 0 0 0 0.0675775 0.0675775

tfidf.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

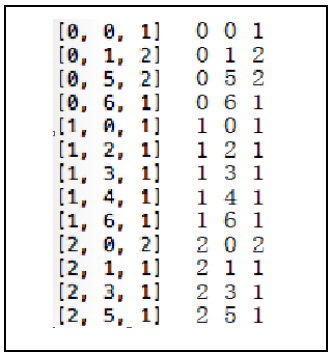
0 -0.0719205 0 0.101366 0 0 0 0
0 -0.143841 0 0 0 0.101366 0 0
0 -0.047947 0 0 0 0 0.0675775 0.0675775

测试三元组的输出符合预期：



[0, 0, 1]	0 0 1
[0, 1, 1]	0 1 1
[0, 2, 1]	0 2 1
[0, 3, 1]	0 3 1
[1, 1, 1]	1 1 1
[1, 4, 1]	1 4 1
[1, 5, 1]	1 5 1
[2, 0, 1]	2 0 1
[2, 1, 1]	2 1 1
[2, 2, 1]	2 2 1
[2, 4, 1]	2 4 1
[2, 6, 1]	2 6 1
[2, 7, 1]	2 7 1

测试三元组相加函数：



[0, 0, 1]	0 0 1
[0, 1, 2]	0 1 2
[0, 5, 2]	0 5 2
[0, 6, 1]	0 6 1
[1, 0, 1]	1 0 1
[1, 2, 1]	1 2 1
[1, 3, 1]	1 3 1
[1, 4, 1]	1 4 1
[1, 6, 1]	1 6 1
[2, 0, 2]	2 0 2
[2, 1, 1]	2 1 1
[2, 3, 1]	2 3 1
[2, 5, 1]	2 5 1

就酱。

评测指标展示及分析：

补充一点，对于实验的测评分析，笔者还是从效率的角度入手。

之前看到别人讨论关于实现的运行时间的时候，发现大多数人的运行时间都很慢，少则几秒，多则几十秒。然而我的输出都是秒出，这让我很方张，后来询问了室友知道，输出的时候，有一些情况：

1）输出时加空格。我是没有加空格的，这么大的数据量就不是给人看的，是给机器读写的，添加了空格读也费事，写也费时。

2）输出前存矩阵。我是没有存矩阵的，读到哪就输出到哪里，不会再使用更多的空间去储存矩阵，笔者觉得就这次的实验任务而言，这样就够了，为什么另开空间储存矩阵呢；

3）多线程输出与单线程输出的运行时间。笔者的测试结果是单线程要更快一些，由上文可知，笔者在转化三元组的时候使用了多线程设计，输出时间在 7s 上下，有些许波动。而单线程输出时候，秒出，输出时间处于 2s 以内。不出意外，这是因为执行了更多的任务，很显然在多线程执行中，需要实现线程的不断切换，怕就怕这个切换过程浪费了太多的资源，而当线程执行的任务更多时，线程切换时间所占的比例降低，这种情况下，多线程操作才显示出优势吧。

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？

关键词：分母，多 1。多一个 1，可以在整体上使得上升函数更加缓慢，然后…，没有了。从整体上看，就产生这样的影响，而如果从关键词上考虑，分母多 1，最大的可能就是

避免分母为 0，再有什么意思我也不知道了，恕才疏学浅。

2. IDF 数值有什么含义？TF-IDF 数值有什么含义？

首先，IDF 值与出现次数呈负相关关系。而 TF 矩阵中的归一化系数与之呈正相关关系，二者相互作用共同影响 TF-IDF 的表达。

首先，就 TF 矩阵来说，数据出现的次数越多，TF-IDF 中的值就越大。这就意味着，整个文本都在受这个词出现的次数的影响。

考虑 IDF 值，出现的文本越多，值反而越小，这是说这个词比较常见，不然那还能说明什么。

两个值相乘，综合起来看，就是查询词是否可以作为出现该词次数最多的文件的关键词。如果出现的文本数少，说明该词可能是中心词而不是随便就可以使用到的词，而在某一文本中出现的次数多则同样表示为该文本中这个词不可或缺嘤嘤嘤。

3. 为什么要用三元顺序表表达稀疏矩阵？

所谓稀疏矩阵，就是说不是零的元素——很稀疏，其他的都是零，但是在储存较多的数据时，这些无意义的重复的数据反而占了大部分空间，这就是舍本逐末的行为。为了强调非零数值的重要性，同时消除没有必要的空间浪费，使用三元顺序表代替稀疏矩阵表达大量的数据是经济又实惠的方法。