分词系统工程报告

课程：自然语言理解
姓名：郑波
学号： 2013211644
班级 ： 计 算 机 科 学 与 技 术 13-2 班

日期： 2015 年 10 月 24 日

一、研究背景

随着计算机网络的飞速发展和普及， 信息检索日益重要。 在这种环境下， 搜索引擎逐渐成为技术人员的开发热点，比如 Google、百度、微软必应、网易有道等为代表的大型搜索引擎成为人们生活工作的不可或缺的网络工具。 由此可见以文本为主要对象的自然语言处理和信息检索的重要性日益显著。

就信息检索来说， 最重要的并不是找到所有的结果， 而是得到最相关最符合用户需求的结果。对于自然语言来说，这就涉及到了分词，分词的准确与否，常常直接影响检索的准确性。 与英美语系不同， 中文的词与词之间是没有自然界限的，如果要使搜索引擎准确无误地理解用户的搜索需求， 返回用户最希望获得的资料，一个准确度好，效率高的分词功能十分必要。

中文分词不像西文那样有明显的空格分隔符，所以对中文的分词十分困难。在现成的中文自动分词方法中， 基于词典或者以词典切分为主结合统计模型的分词方法占主导地位， 而中文分词的主要困难不在于词典中词条的匹配， 而是在于切分歧义的消解和未登录词的识别，这两大难题一直没有得到非常有效的解决。

中文分词技术伴随着搜索引擎技术的发展而发展， 国内外专家和学者对于中文分词技术中的困难和解决方案已经做了广泛的研究和探讨。 在此基础上， 提高中文分词技术的准确率和适应性， 在科研理论和实际应用中， 都具有广泛的意义。

二、 模型方法

本工程使用了基于单字状态的隐马尔可夫模型分词方法

首先，讲一下隐马尔可夫模型，它实际上就是一个五元组 (O,S,π ,A,B)

其中：

O：一个可观察序列,对于一个分词实例就是一个汉字字串， 其中 $?_1?_2\cdots?_?$ 代表字串的每个字。（之后的 k 就代表单字总数）

S：隐藏状态的集合 $?_1?_2\cdots?_?$ 对于一个分词实例就是每一个字的状态的集合，在本工程中，集合为这四种状态 {S(单字成词),B（词头） ,M（词中） ,E（词尾）}。（之后的 n 就代表状态总数）

π ：向量 π 是初始状态空间的概率，就是每个状态 $?_?$ 于词库中的初始概率分布

A：A 是一个 n×n 维的状态转移矩阵，该矩阵的 A（i,j）的值表示对每个词 i 状态到 j 状态的转换概率（事先对于状态进行编号） 。对于本工程使用的状态集来说，有些状态转移是不可能的（比如词尾 词中，词头 单字成词），所以大简化了矩阵的求解。

B：B 矩阵是一个 n×k 维的观察概率分布矩阵，对于 B（i,j），就表示在当前状态 $?_?$ 的情况下可观察单字是 $?_?$ 的概率 ,比如在对“ 我是中国人 ”字串分析的时候， B（0,0）表示当是单字成词的情况下，词库中是“ 我 ”的概率。

那么有了这个五元组，我们就能进行下一步计算了。

明确目标：给每个词打上最有可能的词状态标注， 根据词的状态标注切分字串。（单字成词切开， 遇到词尾切开），于是问题就变成了对最大概率词状态序列的求解，这正是隐马尔可夫模型要解决的三个问题之一。

求解算法： Viterbi（维特比算法）

定义两个 n×k 维矩阵 $?_1?_2$，其中：

$$T_1[i,j] = \max_k \left(T_1[k,j-1] \cdot A_{ki} \cdot B_{ij}\right),$$

$$T_2[i,j] = \arg\max_k \left(T_1[k,j-1] \cdot A_{ki} \cdot B_{ij}\right)$$

就是说 $T_1(i,j)$ 保存前一词的所有可能状态到当前词的状态 j 的最大概率， T_2保存了这个最大概率时对应的前一个词的状态， 这样，当对最后一个词分析完成并求得最大概率后（最后一个最大概率就是全局最优概率） ，就能通过 T_2矩阵逆推到第一个词的状态，最终确定所有词的状态，并进行切分操作

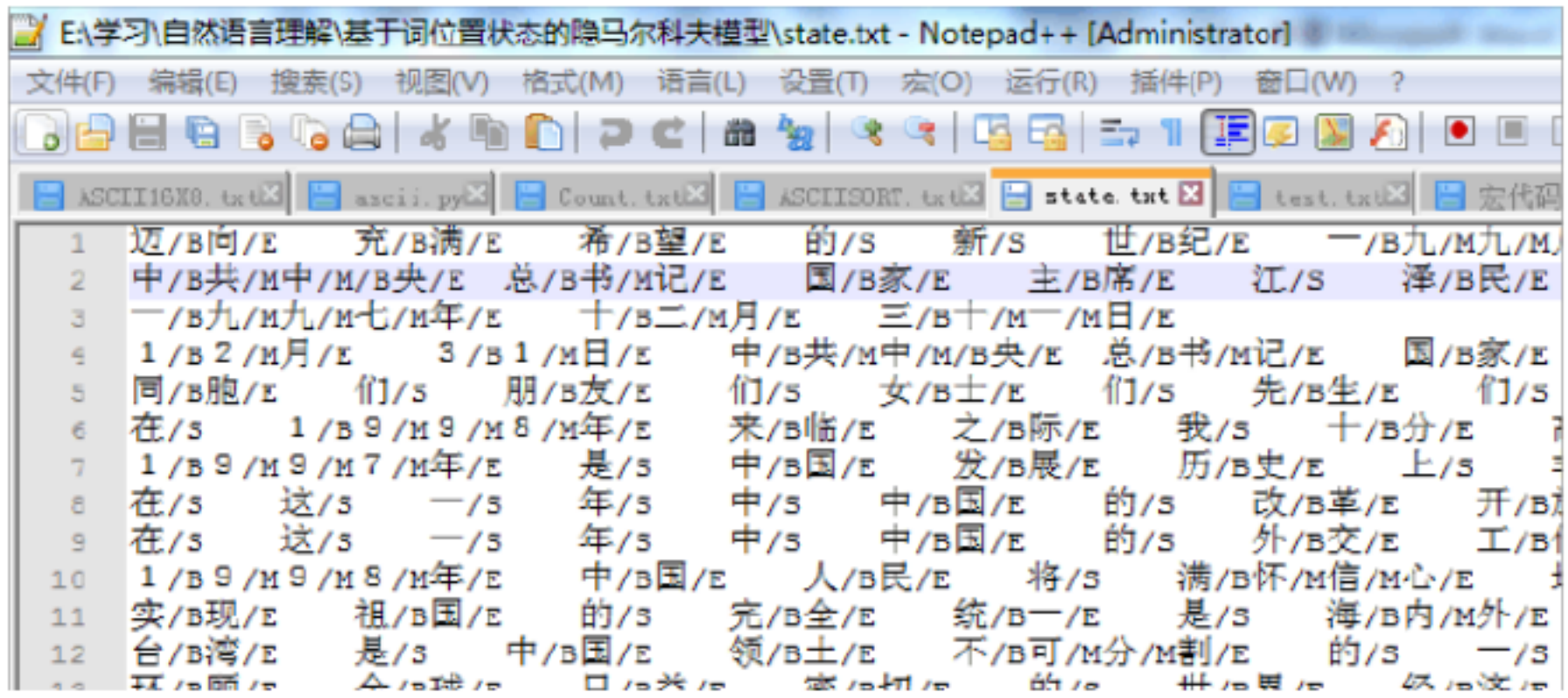
三、系统设计

根据本工程的模型要求，先要求出隐马尔可夫模型的五元组。 O 是可观察序列，已给，只需读入内存即可

S：S 是状态集合， 我们这里设定为 {S,B,M,E}, 所以先要对语料库进行处理， 使语料库的每个字的后面紧跟该字的状态，比如“ 中 /B 国/M 人/E ”,“ 研/B 究/E ”,处理代码如下（本工程全部采用 Python 语言实现）：

```
src=open("OnlyWord.txt",'r')
dst=open("state.txt",'w')
str=""
while True:
    line=src.readline()
    if line:
        Rst=line.split()
        for i in Rst:
            lenth=len(i)
            if lenth==1:
                i=i+'/S'
                str=str+'S'
            else:
                i=i.replace(i[0],i[0]+'/B')
                str=str+'B'
                for k in range(0,lenth-2):
                    i=i[:3*(k+1)+1]+'/M'+i[3*(k+1)+1:]
                    str=str+'M'
                i=i.replace(i[len(i)-1],i[len(i)-1]+'/E')
                str=str+'E'
            dst.write(i+'\t')
        dst.write('\n')
    else:
        break
```

处理结果截图：



π 的求解：

π 是初始状态概率分布，就是说，对于语料库每个词，初始状态的发生概率（实际上词中词尾的初始概率就是　0），代码如下：

```
import math
pai=[]                #初始状态概率
A=[]                  #状态转移矩阵
B=[]                  #状态为某一值的时候出现观察值的概率
state=['S','B','M','E']    #状态集
src=open("test.txt",'r')
line=src.readline()      #读入字符串
src.close()
src=open("state.txt",'r')
str=line
lenth=len(line)


#开始求初始矩阵　π
key=str[0]
Allnum=0
Snum=0
Bnum=0
print(key)
while True:
    l=src.readline()
    if l:
        Rst=l.split()
        for i in Rst:
            if i[0]==key:
                if i[2]=='S':
                    Snum=Snum+1
```

```
                        elif i[2]=='B':
                            Bnum=Bnum+1
                        Allnum=Allnum+1
        else:
            break
pai.append(math.log(Snum+2,10))  #+2 是为了防止出现次数为  0 或者 1
pai.append(math.log(Bnum+2,10))
pai.append(math.log(2,10))
pai.append(math.log(2,10))
```

P.S对于本工程来讲，求一个概率往往非常小，所以我们都用在语料库中出现次数的对数，或者直接用出现次数代替。

A 的求解：

A 矩阵就是对语料库求一个状态转移概率，因为此时语料库的每个字都做了状态的标注，所以我们可以这么求 A 矩阵的值：将语料库中所有的表示状态的英文字母按顺序提出，存放到一个字符串中，然后进行逐字扫描，观察每个英文字母后面跟着的字母是什么，并分别记录出现个数，作为状态转移的概率，实现主要代码如下：

```
#其实一共就八种转换
Allnum=0
SS=0
SB=0
BM=0
BE=0
MM=0
ME=0
ES=0
EB=0
for i in range(0,len(str)-1):
    if str[i]=='S':
        if str[i+1]=='S':
            SS=SS+1
        if str[i+1]=='B':
            SB=SB+1
    if str[i]=='B':
        if str[i+1]=='M':
            BM=BM+1
        if str[i+1]=='E':
            BE=BE+1
if str[i]=='M':
    if str[i+1]=='M':
        MM=MM+1
    if str[i+1]=='E':
        ME=ME+1
```

```
        if str[i]=='E':
            if str[i+1]=='S':
                ES=ES+1
            if str[i+1]=='B':
                EB=EB+1
```

B 矩阵的求解：

B 也是分析语料库求解，对特定分词实例，逐字读入，并分别对其扫描语料库，求解对于每个该字， 后面跟的词状态的类别， 同时统计个数作为 B 阵的值，关键代码如下：

```
#初始化 B 矩阵
for    i in range(0,4):
        B.append([])
        for j in range(0,lenth):
            B[i].append(2)
src=open("state.txt",'r')
strSrc=""    #strSrc 保存所有词
while True:
        line=src.readline()
        if line:
            Rst=line.split()
            for i in Rst:
                strSrc=strSrc+i
        else:
            break
for i in range(0,len(str)):
        for j in range(0,len(strSrc)):
            if strSrc[j]==str[i]:
                if strSrc[j+2]=='S':
                    B[0][i]=B[0][i]+1
                if strSrc[j+2]=='B':
                    B[1][i]=B[1][i]+1
                if strSrc[j+2]=='M':
                    B[2][i]=B[2][i]+1
                if strSrc[j+2]=='E':
                    B[3][i]=B[3][i]+1
```

维特比算法求解最大概率输出序列：

这个算法在前一小节详细叙述过， 通过使用两个数组不断保存概率的最大值和前项记录，从而缩减了计算时间，具体实现如下

```
#分词过程   T1 T2 都是  4Xlen(str)
T1=[]
T2=[]
```

```
for i in range(0,4):
        T1.append([])
        T2.append([])
        for j in range(0,len(str)):
                T1[i].append(0)
                T2[i].append(-1)
for i in range(0,4):
        T1[i][0]=pai[i]*B[i][0]
        T2[i][0]=0
finalState=0     #保存最终的产生节点
finaltemp=0
for i in range(1,len(str)):
        for j in range(0,4):
                temp=0
                for k in range(0,4):
                        if T1[k][i-1]*A[k][j]*B[j][i]>temp:
                                temp=T1[k][i-1]*A[k][j]*B[j][i]
                                T2[j][i]=k
                                finaltemp=j
                if i==len(str)-1:
                        finalState=finaltemp
                T1[j][i]=temp
print(T2)
print(finalState)
StateL=[]      #保存最终序列
temp=T2[finalState][len(str)-1]
for i in range(0,len(str)):
        StateL.insert(0,temp)
        temp=T2[temp][len(str)-i-2]
StateL.pop(0)
StateL.append(3)    #因为是保存前一个状态，所以要进行去头补尾操作
print(StateL)
strNew=""
for i in range(0,len(str)):
        if StateL[i]==0:
                strNew=strNew+str[i]+'/'
        elif StateL[i]==1 or StateL[i]==2:
                strNew=strNew+str[i]
        else:
                strNew=strNew+str[i]+'/'
```

以上代码中，同时实现了最大序列的逆推输出和分词实现。

模型评估：

在工程的开展和实现中发现，　虽然隐马尔可夫模型的切分效果较一般机械分词方法更好，但仔细观察其五元组，　其中的 A 矩阵的求解不关联现有输入分词实例，而仅仅是对语料库的分析，　这可以说是隐马尔可夫模型的一种先天性不足。　再次，根据中文语法的特殊性，　语素后置的特点，　词语之间的联系往往是后置的中心词来决定，这也是这个算法的不足之处。

四、　　系统演示与分析

对系统演示结果进行说明，　测试一些课程中演示的样例，　根据结果说明为什么对或者为什么错。最后对系统提出改进方案

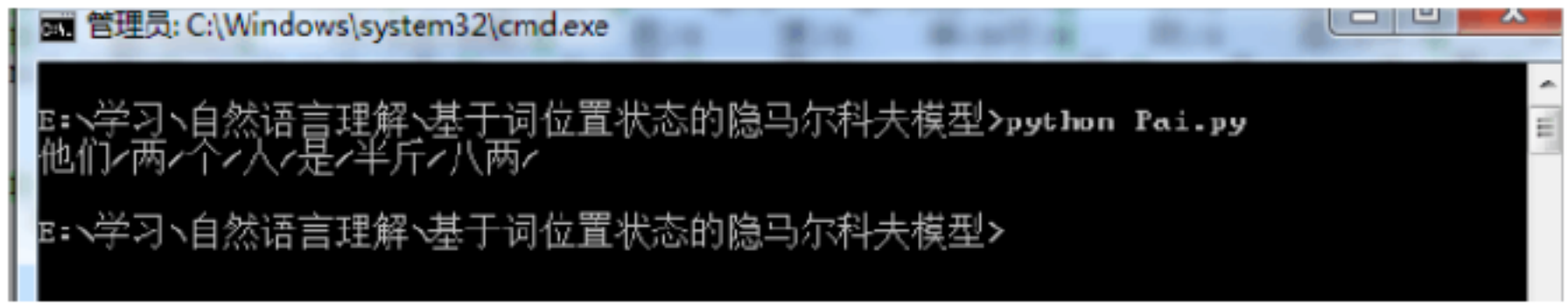
对于课件《隐马尔可夫的模型及其应用》的几个例子的演示：

1、欢迎新老师生前来就餐

准确切分应为欢迎　/新老 /师生 /前来 /就餐

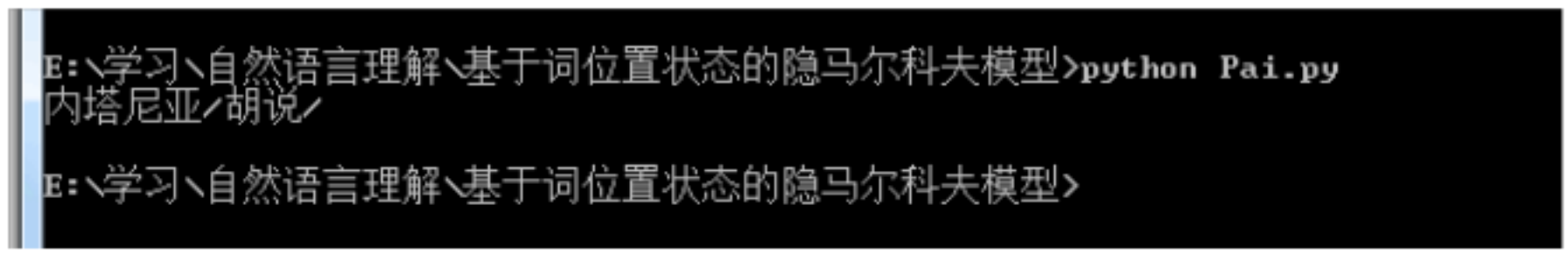
新老切开是因为词库中不存在“ 新老 ”的词，生前和前来切错是因为该模型考虑的是上一个词的状态（向前看一位）　。

2、他们两个是半斤八两



半斤八两切开是因为训练词库中没有半斤八两

3、内塔尼亚胡说



胡说被切出来是因为说作为单字的概率明显低于作为词尾

模型的改进：

之前看到过一篇论文，根据中文词词素靠后的特点，设计出了一种逆向的隐马尔科夫模型，就是从后往前推，这可以是一种不错的改进。

五，对本门课的感想、意见和建议

微软的创始人盖茨说过：　IT　界的下一个大事件是计算机视觉，以及与深度学习的结合。而这门自然语言的课对我来说，　就是深度学习的一个入门。　深度学习给人的感觉就是让机器模拟人的大脑干一些原本只有人类才能干的活　------- 这实在是太迷人了。本次工程所处理的切分字串就是理解词串的第一步，　当机器面对输入的任意一句话，　不仅要通过切分好的字串来找到字串间的联系，还要理解这句话的逻辑意思，然后做出相应的动作，迎合人们的需求。

这正是计算机改变世界的一个真真切切的例子。
当然，通过这门课，我也学习到了许多算法：前向后向算法，维特比算法　　…..初步将隐马尔科夫模型应用到了一个实例中，让我感受到了算法的魅力与模型解决问题的精彩之处。