人工智能原理及应用

课程实验四

姓名：王宇轩

学号：16B903007

专业：计算机科学与技术

《人工智能原理及应用》课程实验四

——学习算法

**16B903007**

**王宇轩**

**计算机科学与技术**

### 问题描述

#### 待解决问题的解释

选择的问题是**中文分词问题**。与英文句子中每个词都用空格隔开不同，中文句子中的词与词之间是没有标记的。而词又是许多自然语言处理任务中不可或缺的基本元素或特征，因此自动对中文句子进行分词就成了中文NLP领域最基础的问题。

#### 问题的形式化描述

解决中文分词问题有多种方法，本实验中采取序列标注的方法。即对中文句子中的每个字打标签，标签分为B、M和E共3种，其中B代表一个词的开始，M代表一个词的中间，E代表一个词的结尾。显然，如果我们能正确预测句子中每个词的标签，就能完成该句子的分词。

#### 学习方法介绍

**在实验中我选择了隐马尔科夫模型（HMM）。隐马尔可夫模型**是统计模型，它用来描述一个含有隐含未知参数的马尔可夫过程。其难点是从可观察的参数中确定该过程的隐含参数。然后利用这些参数来作进一步的分析。

隐马尔科夫模型能够对一个事件序列的概率进行估计，其中包括两种状态，一种是可观测到的状态，在分词问题中就是句子中的字，另一种是隐状态，在分词问题中就是每个字对应的标签。隐马尔科夫模型认为可观测状态是由隐状态产生的，因此模型中包括两种概率，一种是转移概率（transmission probability），即一个隐状态转移到下一个隐状态的概率。

另一种是发射概率（emission probability），即由隐状态产生可观测状态的概率。

隐马尔科夫模型一般有以下三个重要假设，它们是对现实情况的一种简化。

**假设一**：一阶马尔科夫假设，即每个隐层状态只受前一个隐层状态影响。

**假设二**：不动性假设，即状态与具体时间无关。

**假设三**：输出独立性假设，即输出只与当前隐层状态有关。

### 算法介绍

#### 算法的一般介绍

前面已经介绍了HMM的原理，这里具体介绍一下程序中的实现方法。对于一个字序列和一个标签序列（隐状态序列），HMM计算该标签出现概率为：

由于每个标签序列的分母都相同，只需要比较分子大小。对分子施加上述假设一和三。

为了计算方便，求其对数值，这样计算的时候只需要相加即可。

在训练过程中，在训练样本中统计每两个标签连续出现的次数bigram（例如）、每种标签出现次数unigram（例如）以及每个标签和每个词同时出现的次数（例如）。则转移概率为,发射概率为。当然在计算的时候就求出其log值，这样在预测时就能直接取出来相加了。

在预测过程中，先根据提供的词典算出每个句子的全切分（即所有可能的分词方式），然后计算出每种切分的分数，取其中分数最高的作为预测结果。这样做而不是直接在所有可能中搜索的原因是这样能利用外部词典大大提高分词准确率，此外也能大大加速预测。由于求全切分与该学习算法无关，这里就不具体介绍了，说明请见代码中注释。

#### 算法伪代码

输入：分好词的训练样本和未分词的测试样本

输出：测试样本的分词结果

算法描述：

1. 训练过程：
2. 对样本进行预处理，标出每个字的标签
3. 统计训练样本中unigram、bigram和标签与字共现的次数
4. 计算发射概率和转移概率的对数值并保存
5. 预测过程：
6. 读入外部词典
7. 根据词典生成每个句子的全切分
8. 用保存的两种概率计算每个切分的score
9. 选取其中score最高的作为预测的分词结果

### 算法实现

#### 实验环境和问题规模

1. 实验环境

OS: Ubuntu 16.04

编程语言: python 2.7.9

1. 问题规模

句子的长度、词典大小及句子的复杂程度都会影响问题规模。

#### 数据结构

unigram=Counter() #每种标签出现次数

bigram=Counter() #每两种标签共同出现次数

cooc=Counter() #标签和字共现次数

wordcount=Counter() #每个字出现次数

small\_dic=Counter() #每个词出现次数

emis=Counter() #发射概率

trans=Counter() #转移概率

single=Counter() #每种标签出现的概率

dic=Counter() #字典,格式为dic[c][len]={},c表示第一个字,len表示该词长度

graph=Counter() #表示全切分的有向图的邻接表

state=Counter() #保存对应位置是否被访问过

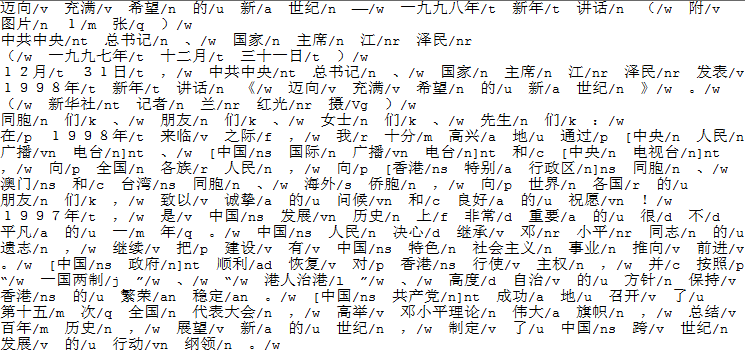
#### 实验结果

在包含 4379句的测试数据中，分词的精确率为89.16%，召回率为92.70%，F1 值为90.90%。

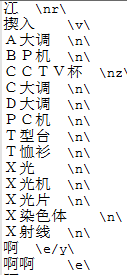
#### 系统最终及中间输出结果

由于本实验中数据是从文件读入，也是输出到文件，因此下面截取一些训练数据和部分预测结果文件。

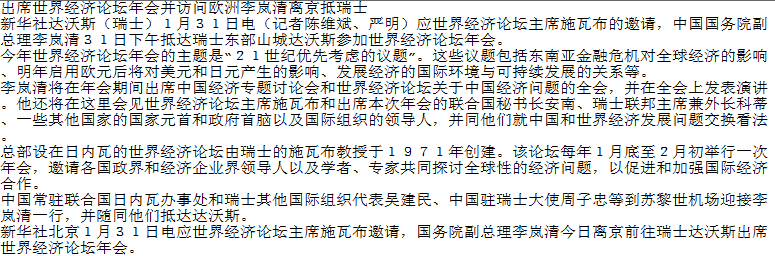
1. 训练样本 (199801.txt)



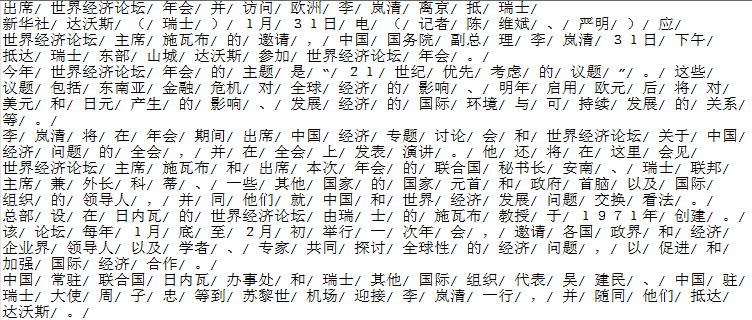
1. 词典 (199801dic.txt)



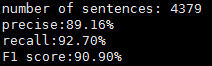
1. 测试样本 (199802\_devtxt.txt)



1. 测试样本的预测结果 (result.txt)



1. 用evaluate.py程序得到的评价结果



### 参考文献

机器学习中的隐马尔科夫模型（HMM）详解：

http://blog.csdn.net/baimafujinji/article/details/51285082

隐马尔科夫模型（HMM）攻略：

http://blog.csdn.net/likelet/article/details/7056068

### 附录-源代码及注释

这里给出用于解决中文分词问题的程序代码Segmentation.py以及用来评测的程序代码evaluate.py。整个项目包括实验中用到的数据和预测结果文件可以在我的github（<https://github.com/ProjectCodeZ/CNSegProg>）中找到。

**Segmentation.py代码如下：**

# -\*- coding: cp936 -\*-

import codecs

import math

import copy

import os

from collections import Counter

#将结果输出到文件

def writestr(filename,string):

output=codecs.open(filename,'a',encoding='gbk')

output.write(string)

output.close()

#若存在同名文件则删除

def init(filename):

if os.path.exists(filename):

os.remove(filename)

#读入字典

def get\_dic(filename):

input=codecs.open(filename,encoding='gbk')

#字典,格式为dic[c][len]={},c表示第一个字,len表示该词长度

dic=Counter()

for line in input:

tokens=line.strip().split()

small\_dic[tokens[0]]+=1

input.close()

for key in small\_dic:

c=key[0] #用每个词中第一个字作索引

if dic[c]==0:

dic[c]=Counter()

if dic[c][len(key)]==0:

dic[c][len(key)]=[]

dic[c][len(key)].append(key)

return dic

#计算切分好的句子的得分

def get\_score(words,labels):

score=0.0

i=0

while i<len(words):

length=0

if labels[i]=='B':

while labels[i+length]!='E':

length+=1

for j in range(length+1):

# 对过长的词进行惩罚,避免过长词出现

if length==1:

lamda=0.99

elif length==2:

lamda=0.9

elif length==3:

lamda=0.85

else:

lamda=0.85

score+=emis[words[i+j],labels[i+j]]\*lamda

if i>=1:

score+=trans[labels[i+j-1],labels[i+j]]

else:

score+=single[labels[i+j]]

i+=length

else:

score+=emis[words[i],labels[i]]

if i>=1:

score+=trans[labels[i-1],labels[i]]

else:

score+=single[labels[i]]

i+=1

return score

#初始化用于全切分的有向图的邻接表

def init\_graph(sentence):

length=len(sentence)

graph=Counter() #表示全切分的有向图的邻接表

state=Counter() #保存对应位置是否被访问过

for i in range(length):

graph[i]=[]

state[i]=[]

#graph[i].append(i+1)

state[i].append(0)

for i in range(length):

tmp\_count=1 #当前词的长度

ch=sentence[i]

sub\_dic=dic[ch]

if sub\_dic==0:

graph[i].append(i+1) #保存成单字

state[i].append(0)

continue

for j in range(i+1,length): #找到所有第i个字到j个字的词

ch+=sentence[j]

tmp\_count+=1 #词长+1

if sub\_dic[tmp\_count]==0: #词典中没有该长度的词

continue

if ch in sub\_dic[tmp\_count]: #词典中找到该词

graph[i].append(j+1) #保存在图中

state[i].append(0)

if not graph[i]:

graph[i].append(i+1) #保存成单字

state[i].append(0)

return (graph,state)

#计算所有切分路径

def get\_path(g,s):

all\_path=[]

single\_path=[]

length=len(g)

single\_path.append(0)

while not single\_path==[]:

top=single\_path[-1]

if single\_path[-1]==length:

ls=copy.deepcopy(single\_path)

all\_path.append(ls)

single\_path.pop()

continue

count=0

for i in range(len(g[top])): #对以第top个字开头的每个在字典中的词

if s[top][i]==0: #s用于保存对应位置是否被访问过

single\_path.append(g[top][i])

s[top][i]=1

break

else:

count+=1

if count == len(g[top]):

d=single\_path.pop()

for i in range(len(g[d])):

s[d][i]=0 #一个节点访问完栈顶节点出栈时对应访问状态清空

return all\_path

#根据划分的词进行BMES标注，保存词中每个字和对应标签

def labeling(word,words,labels):

length=len(word)

if length==1:

words.append(word[0])

labels.append('S')

elif length==2:

words.append(word[0])

labels.append('B')

words.append(word[1])

labels.append('E')

elif length>2:

words.append(word[0])

labels.append('B')

for i in range(length-2):

words.append(word[i+1])

labels.append('M')

words.append(word[length-1])

labels.append('E')

#计算转移概率和发射概率

def compute\_pro(filename):

input=codecs.open(filename,encoding='gbk')

unigram=Counter() #每种标签出现次数

bigram=Counter() #每两种标签共同出现次数

cooc=Counter() #标签和字共现次数

wordcount=Counter() #每个字出现次数

small\_dic=Counter() #每个词出现次数

for line in input:

words=[]

labels=[]

tokens=line.strip().split()

for sample in tokens:

wor=sample.split('/')

tmp=''

if wor[1]==u'w': #处理标点的情况

continue

else:

length=len(wor[0])

if wor[0][0]==u'[': #处理命名实体的情况

tmp=wor[0][1:length]

else:

tmp=wor[0]

#把这个词中每个字和对应标签加到words和labels后

labeling(tmp,words,labels)

small\_dic[tmp]+=1 #统计词频

for i in range(len(words)):

unigram[labels[i]]+=1

wordcount[words[i]]+=1

cooc[words[i],labels[i]]+=1 #统计标签和字共现次数

if i>1:

bigram[labels[i-1],labels[i]]+=1

lamda=1e-5

emis=Counter() #发射概率

trans=Counter() #转移概率

single=Counter() #每种标签出现的概率

total=0

for key in unigram:

total+=unigram[key]

for key in unigram:

single[key]=unigram[key]\*1.0/total

for key1 in unigram:

for key2 in unigram:

probability=(1.0-lamda)\*bigram[key1,key2]/unigram[key1]+lamda\*single[key1] #用回退进行平滑处理

#shoot[key1,key2]=math.log(probability)

trans[key1,key2]=math.log(probability)

for word in wordcount:

for label in unigram:

probability=(1.0-lamda)\*cooc[word,label]/unigram[label]+lamda\*single[label]

#trans[word,label]=math.log(probability)

emis[word,label]=math.log(probability)

return (trans,emis,single,small\_dic)

#返回输入词的BMES标注序列

def get\_tags(p):

tags=[]

for div in p:

tag=[]

for i in range(1,len(div)):

num=div[i]-div[i-1]

if num==1:

tag.append('S')

elif num==2:

tag.append('B')

tag.append('E')

else:

tag.append('B')

for i in range(num-2):

tag.append('M')

tag.append('E')

tags.append(tag)

return tags

#根据BMES标注返回词序列的输出格式

def get\_st(words,tags):

string=''

for i in range(len(words)):

if tags[i]=='S' or tags[i]=='E':

string+=words[i]+'/ '

else:

string+=words[i]

return string

#预测切分序列

def get\_res(filename):

input=codecs.open(test\_file,encoding='gbk')

for line in input:

if len(line)<2:

writestr(res\_file,name)

continue

seg=''

limit=len(line)-1

i=0

#预处理

while i<=limit:

if line[i] not in total\_pun:#当前字不在标点集中

seg+=line[i]

if i<limit and line[i+1] in total\_pun:#若下一个字在标点集中，用空格隔开

seg+=' '

else:#当前字在标点集中

seg+=line[i]

if i<limit and line[i+1] in special\_pun:#下一个字是时间词，直接读入并加空格隔开

seg+=line[i+1]+' '

i+=1

elif i<limit and line[i+1] not in total\_pun:#下一个字不在标点集中，用空格隔开

seg+=' '

i+=1

tokens=seg.strip().split()

string=''

#对预处理得到的每段进行处理

for st in tokens:

if st[0] in total\_pun:#当前词第一个字在标点集中

i=0

tags=''

while i<len(st):

if st[i] in single\_pun:#如果在真标点集中

if i<len(st)-1 and st[i]!=st[i+1]:#如果不是多字节符号则直接作为单字保存到输出序列

tags+=st[i]+'/ '

else:#如果是多字节符号则先不加分隔号

tags+=st[i]

elif st[i] in digit:#如果是数字

tags+=st[i]

if i<len(st)-1 and st[i+1] not in digit+special\_pun:#如果后面不是数字或时间词，加分隔号

tags+='/ '

elif st[i] in letter:#如果是字母，在后面加分隔号

tags+=st[i]+'/ '

if i<len(st)-1 and st[i+1] not in letter:

tags+='/ '

else:#如果不属于上述集合则直接保存

tags+=st[i]

i+=1

if tags[-1]!=' ':

tags+='/ '

string+=tags

else:

if len(st)<2:#如果是单字，在后面加分隔号

string+=st+'/ '

else:#如果不是单字

(graph,state)=init\_graph(st)#初始化有向图

path=get\_path(graph,state)#计算全切分路径

#print len(path)

labels=get\_tags(path)#计算BMES标注序列

label=[]

score=-1100000.0

for i in range(len(labels)):

tmp=get\_score(st,labels[i])#计算每条路径得分

if tmp>score:

score=tmp

label=labels[i]

string+=get\_st(st,label)#根据得到BMES标注返回输出格式

string+='\r\n'

writestr(res\_file,string)

input.close()

if \_\_name\_\_=="\_\_main\_\_":

training\_file='./199801.txt'

dic\_file='./199801dic.txt'

test\_file='./199802\_devtxt.txt'

res\_file='./result.txt'

(trans,emis,single,small\_dic)=compute\_pro(training\_file)

dic=get\_dic(dic\_file)

single\_pun=u'，。？！：；‘’“”、－』『《》（）％…×—'

digit=u'１２３４５６７８９０．'

letter=u'ＡＢＣＤＥＦＧＨＩＧＫＬＭＮＯＰＱＲＳＴＵＶＷＸＹＺ'

special\_pun=u'％年月日时分秒万亿'

total\_pun=single\_pun+digit+letter

init(res\_file)

get\_res(res\_file)

**evaluate.py代码如下：**

import codecs

pred=codecs.open('result.txt',encoding='gbk')

correct=codecs.open('199802\_dev.txt',encoding='gbk')

def getset(fileio):

set=[]

total=0

for line in fileio:

cnt=0

if line=='\r\n':

continue

tokens=line.strip().split()

total+=len(tokens)

tmp={}

for word in tokens:

if word[0]==u'[' and len(word)>1:

word=word.strip(u'[')

wor=word.split(u'/')

tmp[cnt]=wor[0]

cnt+=len(wor[0])

set.append(tmp)

return (set,total)

if \_\_name\_\_=="\_\_main\_\_":

(pre\_set,pre\_total)=getset(pred)

(cor\_set,cor\_total)=getset(correct)

print "number of sentences:",len(pre\_set)

correct\_cnt=0

for sent\_i in range(len(pre\_set)):

for word\_i in pre\_set[sent\_i]:

if word\_i in cor\_set[sent\_i] and cor\_set[sent\_i][word\_i]==pre\_set[sent\_i][word\_i]:

correct\_cnt+=1

p = float(correct\_cnt)/pre\_total

r = float(correct\_cnt)/cor\_total

print 'precision:{:.2f}%'.format(p\*100)

print 'recall:{:.2f}%'.format(r\*100)

print 'F1 score:{:.2f}%'.format(200\*p\*r/(p+r))