## 机器学习大作业——使用HMM实现中文分词算法

1. 问题分析

中文分词问题属于自然语言处理（NLP）中的一类问题，其与机器学习有关的三要素包括：主要模型是隐马尔可夫模型模型（HMM），解决策略是使用动态规划思想，而优化算法则是使用维特比算法。

HMM具有五个要素：状态值集合、观察值集合、转移概率矩阵、发射概率矩阵以及初始状态分布。在中文分词问题中，这五个要素具体分别是：

1. 状态值集合是{B,M,E,S}。其中，B代表begin，意思是当前汉字处于词语中的起始处；M代表middle，意思是当前汉字处于词语的中间；E代表end，意思是当前汉字处于词语的末尾；S代表single，意思是当前汉字独立成词。
2. 观察值集合：所有汉字甚至包括标点符号（但是不包括空格、特殊字符以及换行符）所组成的集合。在HMM模型中文分词中，输入时一个句子（也就是观察值序列），输出是这个句子中每个汉字的状态值。一般来说，训练时，输入的训练数据集越大，包含的汉字以及词语的种类越多（即多样性强），数据集本身的分词科学，则输出的结果更为准确，贴合语义。
3. 初始状态概率分布矩阵：一般用Pi矩阵表示。Pi矩阵包含了每句话（或每一行）第一个字符属于{B,M,E,S}这四种状态的概率。未训练时，初始值都是0，需要经过训练。
4. 状态转移概率矩阵：一般用A矩阵表示。A矩阵包含了由上一个状态到该状态的转移概率。转移概率是马尔可夫链；根据有限历史性假设，目前的状态只与上一个状态有关。因此，在中文分词问题中，A矩阵就是一个4×4的矩阵（代码中将使用字典来存储），矩阵的横坐标和纵坐标分别是{B,M,E,S}。
5. 发射概率矩阵：一般用B矩阵表示。B矩阵一般用字典存储。下式子中，P(Observed[i]|Status[j])就是从B矩阵获得的。



HMM模型主要有三个基本问题：

1. 概率计算问题：给定模型λ=(A,B,Pi)和观测序列O=(o1,o2,...,oT)，计算在模型λ下观测序列O出现的概率P(O|λ)。
2. 学习问题：已知观测序列O=(o1,o2,...,oT)，估计模型λ=(A,B,Pi)的参数A、B、Pi，使得在该模型下观测序列概率P(O|λ)最大。
3. 预测问题：已知模型λ=(A,B,Pi)和观测序列O=(o1,o2,...,oT)，求对给定观测序列条件概率P(T|O)最大的状态序列I=(i1,i2,...,iT)，即给定观测序列，求最有可能的对应的状态序列。

可以看出，中文分词问题主要与后两个问题有关，具体问题描述为：

1. 学习问题：已知训练数据集（已经分好词的中文文段，一般先将其转化为每个汉字的状态（{B,M,E,S}）序列（例如“所以 我 常常 要 到 那 园子 里 去”转化为['B','E','S','B','E','S','S','S','B','E','S','S']）），估计A、B、Pi矩阵的值使得它们最适合给定的训练数据集；
2. 预测问题：已知在上一个问题学到的HMM参数（即A、B、Pi矩阵），以及观测序列——也就是之前从未见过完全相同的一句话，求最有可能的给定的状态序列，也就是这句话当中每个汉字对应的状态，并最终将其转换为分词结果（例如说，将状态序列['B','E','S','B','E','S','S','S','B','E','S','S']转化为“所以 我 常常 要 到 那 园子 里 去”）。

在预测问题中，一般使用的算法是维特比算法，其基本流程如下：

1. 先从前向后推出一步步路径的最大可能，最终会得到一个从起点连接每一个终点的m条路径（假设有m个终点）。

（2）确定终点之后在反过来选择前面的路径。

（3）确定最优路径。

二、数据准备

训练使用的数据集来自于pku\_training.utf8，这是一个专门用于分词训练的数据集，其中包含由65536个中国汉字组成的中文文档，所有文档已经被人工分好词。

测试使用的数据集节选自史铁生的小说《我与地坛》，其中一些比较生僻的汉字和词语被删去，以防止在测试时程序发生错误。

1. 实现思路

（1）首先，建立一个HMM模型，其包含的参数包括可能出现的状态的集合[‘B’,’M’,’E’,’S’]、状态转移概率矩阵A、发射概率矩阵（或观测概率矩阵）B、初始状态概率矩阵Pi。

（2）读入数据集pku\_training.utf8，使用utf-8-sig编码打开文件。

（3）对A、B、Pi进行学习。具体的做法是：一行一行地读入文本，将文本的特殊字符（包括换行符）去掉，转换成每个汉字以及它们对应的状态值，统计每个状态出现在初始位置（也就是每一行的开头）的概率，记入Pi矩阵；统计上一个状态转移到当前状态的概率，记入A矩阵；统计当前汉字处于当前状态的概率，记入B矩阵。

（4）学习完成后，逐行读入测试数据集，对每一行（也就是观测值序列），求出最优路径——也就是最优的状态序列，即最匹配学到的模型的每个汉字的状态，再根据这些状态来完成分词。

四、主要代码注释：

*# mark函数：为每个词（即变量si）标注状态*def mark(si):  
 mark\_=[]  
 *# 若该词的长度为1，判断其为独立成词，状态为Single('S')* if(len(si)==1):  
 mark\_.append(**'S'**)  
 *# 否则，将该词的状态设定为['B','M',...,'M','E']('B':Begin;'M':Middle,'E':'End')* else:  
 mark\_+=[**'B'**]+[**'M'**]\*(len(si)-2)+[**'E'**]  
 return mark\_  
*# get\_data\_lines函数：从位于data\_path目录的文件中提取每一行并封装为列表*def get\_data\_lines(data\_path):  
 *# 为了消除文本开头的特殊字符\ufeff，设置encoding为utf-8-sig* fp = open(data\_path, **'r'**, encoding=**'utf-8-sig'**)  
 return fp.readlines()  
*# HMM模型*class HMM(object):  
 *# 类初始化函数：初始化HMM模型的参数* def \_\_init\_\_(self):  
 *# 文段里每个字有四种状态：  
 # B:Begin（在词开头）  
 # M：Middle（在词中间）  
 # E：End（在词结尾）  
 # S：Single（作为单字词）* self.status=[**'B'**,**'M'**,**'E'**,**'S'**]  
 *# 状态转移概率矩阵，即A矩阵，用字典表示，其key为[前一个状态][当前状态]。* self.A\_dict={}  
 *# 观测概率矩阵，即B矩阵，用字典表示，其key为[当前状态][当前状态对应的字]* self.B\_dict={}  
 *# 初始概率矩阵，即π矩阵，用字典表示，其key为[当前状态]* self.Pi\_dict={}  
 *# HMM.HMMtrain函数：从已经分好词的文段中学习A、B、Pi矩阵。* def HMMtrain(self,train\_path):  
 *# 计算每个状态出现的次数，字典形式，其key为状态'B''M''E''S'* status\_count={}  
 *# A、Pi、B的学习初值* for st in self.status:  
 self.A\_dict[st]={s:0.0 for s in self.status}  
 self.Pi\_dict[st]=0.0  
 self.B\_dict[st]={}  
 status\_count[st]=0  
 *# 获得训练数据集，以一行为元素的列表* train\_data=get\_data\_lines(train\_path)  
 *# 在处理第lis行文本* lis=0  
 *# 处理每一行文本* for line in train\_data:  
 *# 正在处理的行数+1，这里是从第1行开始算，以便于下面将初始状态转化为概率* lis+=1  
 *# 去掉文本前后的特殊字符，例如换行符* line=line.strip()  
 *# 如果这一行为空* if not line:  
 continue  
 *# ji\_list：这一行含有的字表  
 # 例如说，如果这一行是“所以 我 常常 要 到 那 园子 里 去”  
 # 则得到的字表为['所','以','我','常','常','要','到','那','园','子','里','去']* ji\_list=[]  
 for w in line:  
 *# ji\_list不包含用于分词的空格，但是包含标点符号* if w!=**' '**:  
 ji\_list.append(w)  
 *# si\_list：这一行含有的词表  
 # 仍然以“所以 我 常常 要 到 那 园子 里 去”为例子  
 # 得到的词表是['所以','我','常常','要','到','那','园子','里','去']* si\_list=line.split()  
 *# ji\_status：每个字的状态（'B/M/S/E'之一种）  
 # 将每个词传入mark函数，得到的是该词含有的每个字的状态的列表  
 # 例如说：mark(‘如果’）=['BE'];mark('旅游业')=['BME']；  
 # mark('现代控制原理基础')=['BMMMMMME'];mark('的')=['S']  
 # 对于一行，以“所以 我 常常 要 到 那 园子 里 去”为例  
 # 得到的每个字状态为['B','E','S','B','E','S','S','S','B','E','S','S']* ji\_status=[]  
 for si in si\_list:  
 *# 由于mark(tshi)返回值也是列表，因此使用extend函数将列表融合（attend会将列表当成一个新的元素）* ji\_status.extend(mark(si))  
 *# enumerate返回一个列表，其元素是二元组(i,ji\_status[i])* for index,ji\_st in enumerate(ji\_status):  
 *# 统计对应的状态数+1* status\_count[ji\_st]+=1  
 if index==0:  
 *# 统计初始状态(即index等于0，每行首个字符），对应的状态数+1* self.Pi\_dict[ji\_st]+=1  
 else:  
 *# 统计状态转移次数，A的key为[前一个状态][当前状态]，从前一个状态到当前状态的转移+1* self.A\_dict[ji\_status[index-1]][ji\_st]+=1  
 *# 统计状态观测次数，B的key为[当前状态][当前状态对应的字]，当前字观测到当前状态的次数+1  
 # 等号右侧，查询B\_dict[ji\_st][ji\_list[index]]，如果不存在这个二元key（因为在这个train函数中，  
 # 上面初始化A的时候同时分配了两个key，但初始化B的时候只分配了前一个key，第二个key由于取自文段中的字，  
 # 会在计算B矩阵的时候动态分配，因此初始化的时候没有顾及），则get函数返回其第二个参数  
 # 也就是0，等号右侧为1；如果存在，则等号右侧相当于等号左侧自+1.* self.B\_dict[ji\_st][ji\_list[index]]=self.B\_dict[ji\_st].get(ji\_list[index],0)+1  
 *# 将初始状态矩阵转化为概率，将每个状态在lis行累积出现的次数除以lis* for key,value in self.Pi\_dict.items():  
 self.Pi\_dict[key]=value/lis  
 *# 将状态转移次数转化为概率，除以（转移前的）状态总数* for key,value in self.A\_dict.items():  
 for key1,value1 in value.items():  
 self.A\_dict[key][key1]=value1/status\_count[key]  
 *# 将状态观测次数转化为概率，除以key对应的状态总数* for key,value in self.B\_dict.items():  
 for key1,value1 in value.items():  
 *# 加1平滑，否则可能会在viterbi函数中报错* self.B\_dict[key][key1]=(value1+1)/status\_count[key]  
 *# viterbi函数求解HMM最佳路径  
 # 注意：对一行文本* def viterbi(self,text):  
 *# delta见统计学习方法209页的变量δ* delta=[{}]  
 *# 最佳路径* path={}  
 *# 初始化变量delta和path* for st in self.status:  
 *# 若B\_dict[st][text[0]]存在，则返回此值，否则返回0（表示之前训练的时候没有见过这个字）* delta[0][st]=self.Pi\_dict[st]\*self.B\_dict[st].get(text[0],0)  
 path[st]=st  
 *#从第2个字开始（即除去初始状态以外）* for i in range(1,len(text)):  
 delta.append({})  
 new\_path={}  
 *# 检查这个字是不是不在B矩阵的第二个key里，也就是这个字是不是在之前训练的时候没有见过* not\_in\_B\_dict=text[i] not in self.B\_dict[**'B'**].keys() and text[i] not in self.B\_dict[**'M'**].keys() \  
 and text[i] not in self.B\_dict[**'S'**].keys() and text[i] not in self.B\_dict[**'E'**].keys()  
 *# 遍历每一个状态* for st in self.status:  
 if not not\_in\_B\_dict:  
 *# 如果之前训练的时候见过这个字，则Pro\_B等于b\_i(o\_i)* Pro\_B=self.B\_dict[st].get(text[i],0)  
 else:  
 *# 如果见过，则等于0* Pro\_B=0  
 *# 见统计学习方法209页：  
 # 以下t为第t个状态，等效于本代码中的第i个字符  
 # delta\_t\_i=max{1<=j<=N}[delta\_t-1(j)\*a[j][i]]b\_i(o\_t)  
 # phi=argmax{1<=j<=N}[delta\_t-1(j)\*a[j][i]]* (pro,phi)=max([(delta[i-1][st\_]\*self.A\_dict[st\_].get(st,0)\*Pro\_B,st\_) for st\_ in self.status if delta[i-1][st\_]>0])  
 delta[i][st]=pro  
 *# 更新路径，添加当前的状态* new\_path[st]=path[phi]+st  
 path=new\_path  
 *# 更新最后一个P\*和phi* (pro, phi) = max([(delta[len(text) - 1][st\_], st\_) for st\_ in self.status])  
 *# 返回最优路径的最大概率和最优路径* return pro,path[phi]  
 *# 分词函数* def divide(self,text):  
 *# 从测试文本中获得最大概率和最优路径* prob,path=self.viterbi(text)  
 *# 这个分词开始的地方和下一个分词开始的地方* begin, next = 0, 0  
 *# 遍历这一文本（行）的每个字* for i, char in enumerate(text):  
 pos = path[i]  
 *# 如果这个字被判断为状态B，将它指定为分词开始* if pos == **"B"**:  
 begin = i  
 *# 如果这个字被判断为状态E，将它指定为分词结束，获取begin到该字符的子串作为分词之一  
 # 将该字符的下一个位置作为下一次分词的开始* elif pos == **"E"**:  
 yield text[begin:i + 1]  
 next = i + 1  
 *# 如果这个字被判断为状态S，将它指定为单字成词，获取该字作为分词之一* elif pos == **'S'**:  
 yield char  
 next = i + 1  
 *# 将该行剩余的字符作为一个分词* if next < len(text):  
 yield text[next:]  
*# main是测试函数*def main():  
 *# HMM模型* H=HMM()  
 *# 训练的数据集是pku\_training.utf8。获得A、B、Pi矩阵。* H.HMMtrain(**'pku\_training.utf8'**)  
 *# 测试数据集是novel.txt，打开编码同样是utf-8-sig，避免\ufeff等特殊字符的产生* test\_f = open(**"novel.txt"**, **"r"**, encoding=**"utf-8-sig"**)  
 *# 读取，获得以每一行内容为元素的列表* test\_text = test\_f.readlines()  
 *# 去除每一行的换行符* for i in range(len(test\_text)):  
 test\_text[i]=test\_text[i].strip()  
 *# 显示原文* print(**"原文："**)  
 for i in range(len(test\_text)):  
 print(test\_text[i])  
 print(**'----------------------------------------------------'**)  
 *# 显示分词结果* print(**"分词结果："**)  
 for i in range(len(test\_text)):  
 *# H.divide(test\_text[i])获得分词结果（yield关键字获得一个generator对象，其可迭代，内容是分词结果）* for j in H.divide(test\_text[i]):  
 print(j,end=**'/'**)  
 print(**'**\n**'**,end=**''**)  
main()

一种分词结果如下：

我/在/好/几/篇/小说/中/都/提到/过/一/座/废弃/的/古园/，/实际/就/是/地坛/。/许多/年前/旅游业/还/没有/开展/，/园子/荒芜/冷落/得/如同/一片/野地/，/很少/被/人/记起/。/

地坛/离/我家/很/近/。/或者/说/我家/离/地坛/很/近/。/总之/，/只好/认为/这/是/缘分/。/地坛/在/我出/生前/四百/多年/就/座落/在/那儿/了/，/而自/从/我/的/祖母/年/轻时/带着/我/父亲/来/到/

北京/，/就/一/直住/在/离/它/不远/的/地方/一五十多/年间/搬过/几/次/家/，/可/搬来/搬去/总是/在/它/周围/，/而且/是/越/搬离/它/越近/了/。/我常/觉得/这/中间/有/着/宿命/的/味道/：/仿佛/

这/古园/就/是/为/了/等/我/，/而/历尽/沧桑/在/那儿/等待/了/四百/多年/。/

它/等待/我/出生/，/然后/又/等待/我活/到/最狂/妄/的/年龄/上/忽地/残废/了/双腿/。/四百/多/年里/，/它/一面/剥蚀/了/古殿/檐头/浮夸/的/琉璃/，/淡/褪/了/门壁/上/炫耀/的/朱/红/，/

坍了/一段/段/高墙/又/散落/了/玉砌/雕栏/，/祭坛/四/周/的/老柏/树/愈见/苍幽/，/到/处/的/野草/荒藤/也/都/茂盛得/自在/坦荡/。/

这/时候/想/必我/是/该来/了/。/