基于二、三元语言模型的拼音输入法编程作业实验报告

张钰晖 2015011372 yuhui-zh15@mails.tsinghua.edu.cn 计算机系计55班

如果要使用程序，请直接参考本文第六章：文件组织

一、算法框架

基于字的二元和三元模型，实现一个拼音到汉字的转换程序。主要采用了HMM模型和Viterbi算法，下面会详细进行介绍。

隐马尔科夫（HMM）模型

隐马尔可夫模型 (Hidden Markov Model) 是一种统计模型，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。拼音输入法中可观察的参数就是拼音，隐含的参数就是对应的汉字。

维特比（Viterbi）算法

Vieterbi算法是一种动态规划算法。它用于寻找最有可能产生观测事件序列的维特比路径——隐含状态序列，特别是在马尔可夫信息源上下文和隐马尔可夫模型中。

由于这是人工智能和自然语言处理的基本模型，不再赘述，详见维基百科隐马尔科夫模型和维特比算法。

二、实现方法

①统计频数

注意：这部分功能由`/src/process_bigram.py`（统计二元模型）和`/src/process_trigram.py`完成（统计三元模型）

对提供的sina_news文件夹下文件进行逐文件统计，用python建立两个dict数据结构，分别为start_probability和transition_probability，二元模型前者统计单字出现的频数，后者统计前一个字与后一个字出现的频数，三元模型前者统计两个相邻字出现的频数，后者统计前两个字与后一个字出现的频数。

二元模型下，考虑以下一个样例：`北京的简称是京，我很爱北京，特别是北京天安门。`那么有`start_probability['京']=4`，`transition_probability['北']['京']=3`。为了节约储存空间，对于未出现的边不出现在字典中，在程序运行过程中会判断。

最后将start_probability和transition_probability两个dict使用json.dumps()方法保存为json格式文件，二元模型分别为start_probability.json和transition_probability.json，三元模型分别为start_probability_trigram.json和transition_probability_trigram.json，便于python读取。

②在HMM模型中进行Viterbi过程

考虑长度为n的由汉字$w_1,w_2,...,w_n$组成的句子S，输入法汉字识别过程中等价于使概率$P(S)=\prod_{i=1}^{n}P(w_i|w_0...w_{i-1})$最大，其中$w_0$为起始符，用于处理基础情况。

二元模型下，上述公式降解为$P(S)=\prod_{i=1}^{n}P(w_i|w_{i-1})$，三元模型下，上述公式降解为$P(S)=\prod_{i=1}^{n}P(w_i|w_{i-1}w_{i-2})$，这是HMM模型。

由条件概率公式和大数定律$P(w_i|w_{i-1})=\frac{P(w_iw_{i-1})}{P(w_{i-1})}\approx\frac{F(w_iw_{i-1})}{F(w_{i-1})}=\frac{N(w_iw_{i-1})}{N(w_{i-1})}$，$P(w_i|w_{i-1}w_{i-2})=\frac{P(w_iw_{i-1w_{i-2}})}{P(w_{i-1}w_{i-2})}\approx\frac{F(w_iw_{i-1}w_{i-2})}{F(w_{i-1w_{i-2}})}=\frac{N(w_iw_{i-1}w_{i-2})}{N(w_{i-1}w_{i-2})}$，其中$F(w_iw_{i-1})$为$w_i$和$w_{i-1}$相邻出现的频率，$F(w_{i-1})$为$w_{i-1}$单独出现的频率，$N(w_iw_{i-1})$为$w_i$和$w_{i-1}$相邻出现的频数，$N(w_{i-1})$为$w_{i-1}$单独出现的频数，故可以通过频数去估计出概率$P(w_i|w_{i-1})$，三元情况下同理。

为了使$P(S)$最大，采用基于动态规划思想的Vieterbi算法。

二元模型下该算法需要四个参数，分别为pinyin, start_probability, transition_probability, emission_probability，四个参数均为dict()数据结构。

pinyin储存了对于输入的拼音可能对应的汉字，例如`pinyin['wo']=[我,窝,卧...]`。

start_probability储存了单字在语料库出现的频数，例如`start_probability['我']=10000`。

transition_probability储存了上一个汉字与下一个汉字在语料库共同出现的频数，例如`transition_probability['我']['是']=2000`。

emission_probability储存了每个字为某个拼音的概率（处理多音字用，该库为网上寻找处理后得到），例如`emission_probability['我']['wo']=1, emission_probability['得']['de']=0.7, emission_probability['得']['dei']=0.2`。

Vieterbi算法的基本思想即动态规划，在给定上述四个参数后，在第$t-1$步由字符$y0$转移到第t步字符y的概率为`V[t][y] = max(V[t - 1][y0] * transition_probability[y0][y] * emission_probability[y][obs[t]] / start_probability[y0] for y0 in obs[t - 1])`（需处理一些边界情况），不断进行下去，每一步在path中记录路径，最终取出第n步概率最大的串即为转换结果。详见`/bin/bigram.py`中`viterbi()`函数。

三元模型下，该算法需要六个参数，分别为pinyin, start_probability_bigram, transition_probability_bigram, start_probability_trigram, transition_probability_trigram, emission_probability，六个参数均为dict()数据结构，存储内容同上，转移方程也类似，只不过需要从二维动态规划拓展到三维动态规划（非边界情况每次取出3个字符），详见代码。详见`/bin/trigram.py`中`viterbi()`函数。

三、实验效果

训练集采用网络学堂的提供的新浪新闻语料库（201601-201611），测试集采用修正部分错误的网络学堂同学共同发布的数据（共769句话），但由于该数据噪音高、句子生僻、问题较多、甚至存在方言俚语，故实际准确率应比测试结果高较多百分点，但是极端数据更具有说服力，能考察输入法的极限情况，故仍以该测试集作为结果指标量化。统计程序为`/src/statistic.py`。

测试环境：

操作系统 macOS Sierra 10.12.4

CPU 2.7GHz Intel Core i5

内存 8GB 1867 MHz DDR3

①二元模型（Bigram）：

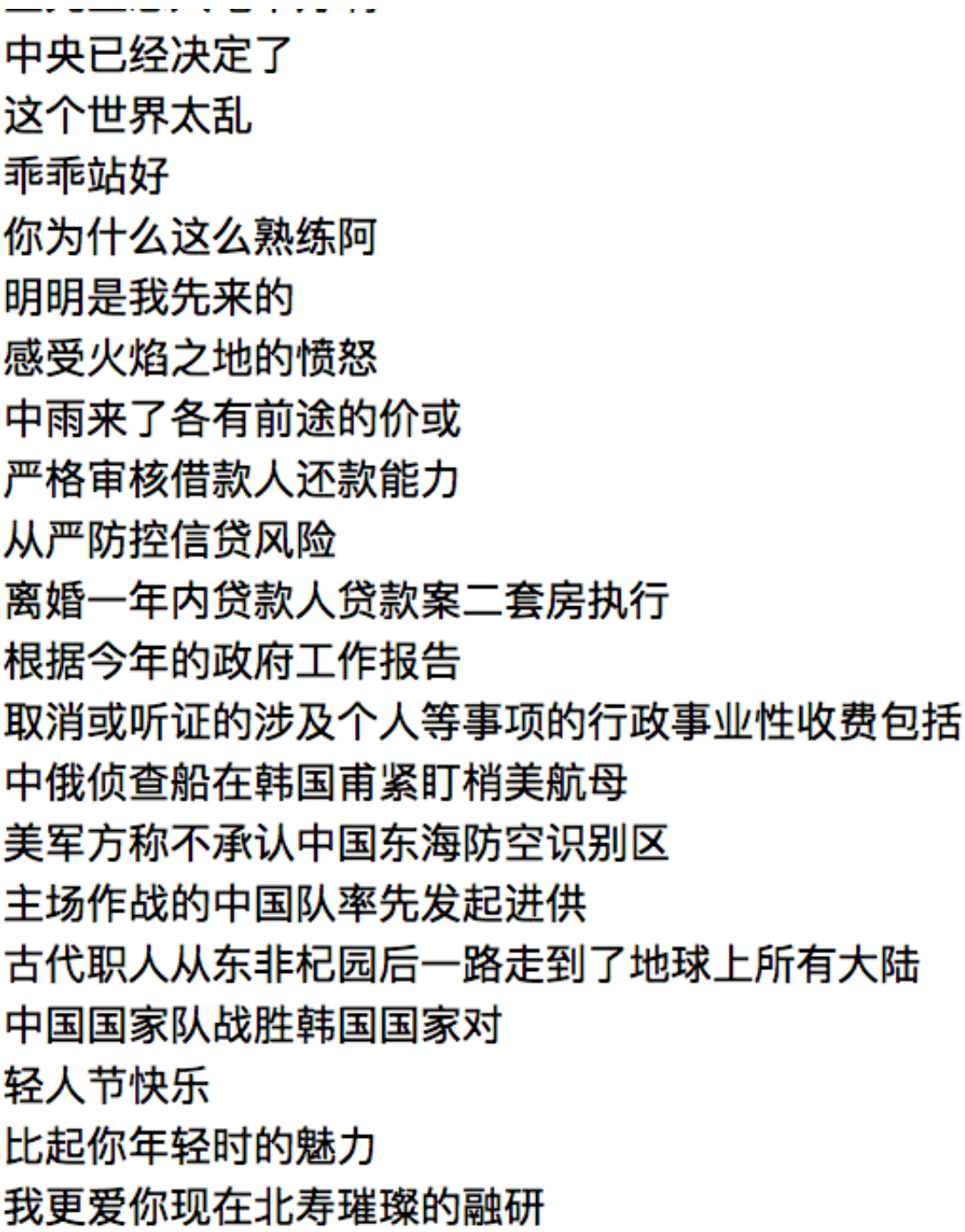
内存：约400MB（含虚拟内存）

时间：约5s

句准确率：29.13%

字准确率：77.43%

部分结果展示：



由上面结果可看出，由于语料库是新浪新闻，故新闻类转换效果较好，甚至较难的'从严防控信贷风险','美军方不承认中国东海防空识别区'这种本机输入法都不一定能转换出来的句子也可以转换出来，但是生活类和文学类句子表现明显不尽人意，比如'情人节快乐'这种基本短语竟会转换出现问题，诗句基本全军覆没，这也可以理解，毕竟一是诗句古文在新闻中出现较少，二是诗句大部分语句都是上下文有关的，很难仅仅用二元模型得出满意结果。

②三元模型（Trigram）：

内存：约6000MB（含虚拟内存）

时间：约90s

参数lamda = 0.9时：（下有详细解释）

句准确率：49.16%

字准确率：85.58%

部分结果展示：

比起你年轻时的美丽
我更爱你现在备受摧残的容颜
多年以后
当她面对行刑队的时候
依然会想起那个
父亲带他见识冰块的遥远的下午
他是罪恶之光
余年制或
一起在荒岛上等待星辰
黄昏与月光一同升起来
午后太阳下的阴影
美女与野兽
清明节快来了
好像放假啊
恰似那朵莲花不胜凉风的娇羞
你存在在我的存在
你挥霍了我的崇拜
山有木兮木有枝
我答答的马蹄声是个美丽的错误
铁马冰河入梦来

由于语料库是新浪新闻，三元模型中新闻类转换效果依然非常良好，甚至有些好的让人惊讶，与二元模型类似，不再展示。与二元模型相比，明显的改善是一些文学类句子，由于三元模型的每一个字考虑范围都比二元多了一个字，这实际上多了很多很多信息，致使很多诗句得以正确表达，比如'铁马冰河入梦来'，'山有木兮木有枝'，'我更爱你备受摧残的容颜'等等，在二元模型中被转换成了'铁马病和如梦来'，'山游目西姆有制'，'我更爱你现在北寿璀璨的融研'，这也很容易理解，比如上述两句话三个字读和两个字读确实完全不同。故最终三元模型在这个极端测试集上的句准确率取得了飞跃。

虽然准确率取得了良好的提升，但是由于三元模型需要存储很多的信息，处理过程也多了一层循环，故与二元模型相比，内存消耗明显变大，速度也慢了很多。

四、参数选择

①平滑加权参数lamda

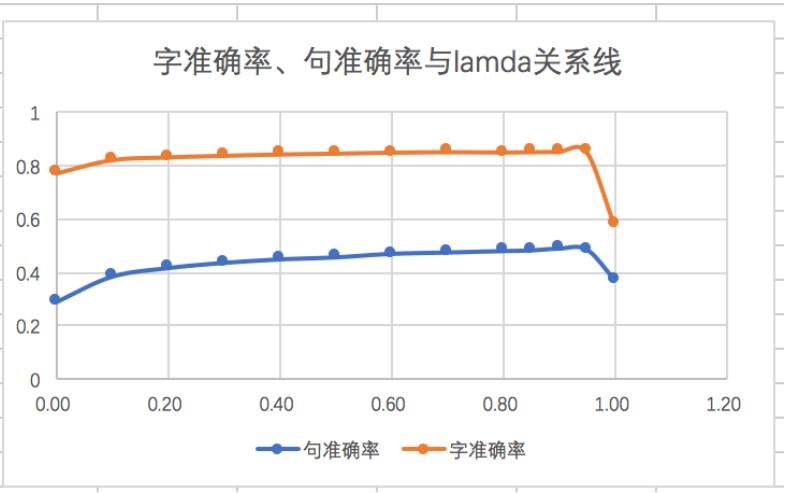
机器学习中，理论上高元模型一定比低元模型效果好。但越是高元模型，对数据的需求量越大，越容易因为数据量不足产生High-Bias的情况，故需要用低元模型去平滑，例如二元模型可以用一元模型去平滑，三元模型可以用二元模型去平滑。lamda表示权重，如lamda=0.6表示高元模型权重为0.6，低元模型权重为0.4。

在二元模型中，参数lamda=1.0时效果最好，因为二元模型需要数据量较小，已经充分训练。故只用二元模型准确率最高。

在三元模型中，参数lamda=0.9时效果最好，因为三元模型需要数据量较大，给的训练集不足以训练充分，故需要低位模型起到平滑作用，即0.9的权重由三元模型决定，0.1的权重由二元模型决定。

三元模型字准确率、句准确率与lamda的关系见下图：

lamda	句准确率	字准确率
0.00	0.2913	0.7743
0.10	0.3862	0.8244
0.20	0.4187	0.8352
0.30	0.4382	0.8413
0.40	0.4512	0.8462
0.50	0.4590	0.8495
0.60	0.4720	0.8530
0.70	0.4772	0.8549
0.80	0.4824	0.8540
0.85	0.4850	0.8550
0.90	0.4915	0.8558
0.95	0.4889	0.8560
1.00	0.3771	0.5817



②训练集规模size

如前所述，越是高元模型，对数据的需求量越大，越容易因为数据量不足产生High-Bias的情况。因此，扩大数据量能有效降低High-Bias的情况。

在网络学堂上提供了201601-201611共11个月的新浪新闻数据集，笔者在此基础上又从实验室服务器上搜集到了201501-201512共12个月的新浪新闻数据集进行共同训练。

二元模型中，增加训练集规模没有显著提高，提高率不足1个百分点，说明二元模型已经充分训练，印证了上述论断，即lamda=1.0时二元模型效果最好，因为训练已经非常充分。

三元模型中，由于笔者电脑内存不足，统计11个月的文件即消耗将近6GB内存，故无法进行测试，理论上效果应该会有提高，因为在上述论断中lamda=0.9时三元模型效果最好，说明训练并不充分（但也已经比较充分）。

③发射矩阵emission_probability

这是HMM模型中的一个基本矩阵。该矩阵是从网上获取的统计资料经过处理得到，用于处理多音字，比如'的'有两个发音，分别为'de'和'di'，前者概率明显高于后者，虽然'的'单字频率极高，但用户在输入'di'时，不应优先打出该字，乘上该概率emission_probability['的']['di']后就能较好的解决上述问题。通过引入该矩阵使得字准确率可提高大约7%-10%的准确率。

五、总结收获

①收获

这次的编程作业非常有前沿性，选择了当今热门研究方向自然语言处理作为作业，但也非常具有挑战性。在认真完成这个作业的过程中，我理解了语言模型、统计语言处理等基础知识，学会了如何处理“大”数据（之前作业从来没有处理过这么多数据），如何基于统计结果结合概率论与数理统计的知识表征结果，如何从二元模型自然地拓展到三元模型，如何遇到问题自己去调试，如何解决中文编码问题，如何利用Python自带数据结构和模块简化编程等等，使我受益匪浅。

中文输入法，是当今时代人人都离不开的工具，记得小学Windows XP时代，智能ABC、全拼输入法这种伪智能输入法给人们汉字输入带来了极大的困扰，直到搜狗拼音输入法问世，输入法才赢得了革命性的进步，在电脑上打字成为了一种享受。但万万没想到，曾经觉得颠覆时代的智能输入法核心代码竟然仅仅不足百行代码就可以实现。当然实际肯定有很多很多细节，比如拼音断字、用户行为记录、模糊音、词库、缩小资源占用、提高速度等等，我只是实现了核心中的一部分，但是效果已经令我足够满意，远远超过智能ABC、全拼输入法这种XP时代的输入法。

可以说，从单纯的模仿人类到机器学习是人工智能革命性的进步，在如今的大数据时代，这种思想愈发重要，基于统计学习而处理自然语言也向着巅峰时刻迈进。《数学之美》读后让我震撼，几十年建立的基于人类认知的翻译系统几月内便被基于机器学习的翻译系统超越，有时候转换一种思想，前路便会豁然开朗，这也启示着我在今后的学习中多转换思维，多进行尝试。

这次作业我认为非常好，使我收获颇丰，更激起了对科研浓厚的兴趣，感谢老师的悉心讲授与准备，希望今后能完成更多这种有趣的作业。

②改进方案

正如前文所说，囿于电脑性能限制，不得统计更多文件来减轻三元的High-Bias问题，故后期可尝试在服务器上统计更多的数据，检查效果；

可以尝试更换更多的语料库，新浪新闻偏重于新闻类资讯，生活类短句出现较少，后期可以尝试在一些别的语料库上进行统计，比如诗词字典，或许可以使诗句准确率有较大幅度改进。

可以尝试分词，用THULAC分词后制作基于词的二元模型或许效果也会很好。

可以用C++实现，比Python性能和效率会好很多。

通过阅读宗成庆先生的《统计自然语言处理》，我又明白了可以统计的时候可以在句前句末插入起止符，这样就避免了复杂的特殊情况处理，比如二元语法处理句首第一个汉字，三元语法处理句首前两个汉字。

可以通过平滑方法对概率进行平滑，比如二元transition_probability没有统计到的转移边，将其初始化为k，其余统计到的边均加k，其中$0 < k < 1$，避免一句话中因为两个字没有在语料库出现而导致整句话概率为0。(Johnson和Jeffreys加法平滑方法)

我猜测或许可以反向统计，进行逆向vieterbi动归过程，因为汉字中谓语更具有确定性，比如'我吃饭'，如果正向统计，'我吃'后面可以接很多东西，比如'我吃饭','我吃瓜',...，而逆向进行，'饭'前面基本只能接'吃'字，这样确定性就会提高。当然这是我的一个猜测，至于具体效果如何，还有待实验考证。

由于时间短暂，我只能进行这么多探索，虽然有很多不足，但基本搭起了baseline。后期完全可以尝试上述改进方案，更加深入的尝试科研与探索的过程。

六、文件组织

①bin文件夹下包含bigram.py文件、trigram.py文件和lib文件夹

- bigram.py文件是基于二元模型实现的拼音输入法，准确率略低，但占用资源小，加载速度快。运行方式：`python bigram.py inputfile outputfile`，会从inputfile读取拼音，转换后输出到output文件中。
- trigram.py文件是基于三元模型实现的拼音输入法，准确率较高，但占用资源大（加载完成在macOS下需要占用较大内存），加载时间较长（大概需要60s读入统计数据）。运行方式：`python trigram.py inputfile outputfile`，会从inputfile读取拼音，转换后输出到output文件中。
- lib文件夹包含了bigram.py和trigram.py需要的数据文件，均以json格式储存，便于python直接读入。pinyin.json存储了拼音到汉字的对应关系，start_probability.json存储了每个单字出现的频数，transition_probability.json存储了上一个字连接下一个字出现的频数，start_probability_trigram.json存储了两个字相邻出现的频数，transition_probability.json存储了上两个字连接下一个字出现的频数，emission_probability.json存储了每个字发某个音的概率（处理多音字）。（另有sinanews_201501_201611.zip存储了新浪新闻201501~201611的统计数据，sinanews_201601_201611.zip存储了新浪新闻201601~201611的统计数据）

②data文件夹下包含input.txt文件、output_standard.txt文件、output_bigram.txt文件、output_trigram.txt文件、sampleinput.txt文件、sampleoutput.txt文件和trigram_with_lamda文件夹

- input.txt文件存储了修正部分错误的同学共同汇集的测试集的拼音
- output_standard.txt存储了修正部分错误的同学共同汇集的测试集的汉字
- output_bigram.txt文件存储了二元模型转换input.txt的结果
- output_trigram.txt文件存储了三元模型转换input.txt的结果
- sampleinput.txt文件存储了提供的四句样例输入的拼音
- sampleoutput.txt文件存储了提供的四句样例输入的汉字
- trigram_with_lamda文件夹下有13个文件，记录了如上所述不同lamda值下，三元模型对input.txt的转换结果，如lamda10.txt代表lamda=0.10时的转换结果

③src文件夹下包含hanzibiaoutf8.txt文件、pinyinutf8.txt文件、pinyinjson.py文件、process_bigram.py文件、process_trigram.py文件和statistic.py文件

- hanzibiaoutf8.txt文件存储了所有汉字的utf8编码
- pinyinutf8.txt文件存储了所有汉字的拼音

- pinyinjson.py文件用于将pinyinutf8.txt处理为pinyin.json文件，便于python直接读取
- process_bigram.py文件用于统计二元模型所需要的start_probability.json和transition_probability.json，需要将待统计的sinanews文件夹放在src目录下
- process_trigram.py文件用于统计三元模型所需要的start_probability_trigram.json和transition_probability_trigram.json，需要将待统计的sinanews文件夹放在src目录下
- statistic.py文件用于统计字准确率和句准确率，运行方式`python statistic.py file1 file2`，file1是标准文件，file2是待对比文件
- 另有一些细碎的简单的处理文件没有放入

④README.pdf

即本文档