拼音输入法实验报告

刘雅迪
计 26
学号：2021010521

一、数据集与实验环境：

数据集为给定的新浪新闻语料库，实验环境为 Windows 10。
使用的库主要有 json, re, pypinyin

二、文件结构说明

```
.
├── README.md
├── data
│   ├── std_input.txt
│   └── std_output.txt
├── src
│   ├── preprocess.py
│   ├── pinyin.py
│   ├── pinyin_triple.py
│   └── test.py
├── training_data
│   ├── 拼音汉字表.txt
│   └── 一二级汉字表.txt
└── 语料库
    └── sina_news_gbk
        ├── 2016-04.txt
        ├── 2016-05.txt
        ├── 2016-06.txt
        ├── 2016-07.txt
        ├── 2016-08.txt
        ├── 2016-09.txt
        ├── 2016-10.txt
        └── 2016-11.txt
```

请按照上述文件结构放置训练材料和语料库材料。

./data/std_input.txt 和./data/std_output.txt 为标准输入输出。

./src/preprocess.py 为数据预处理文件，./src/pinyin.py 和./src/pinyin_triple.py 分别为基于字的二元和三元模型的拼音输入法，./src/test.py 为测试输出的文件。

三、数据预处理：

具体代码为 python ./src/preprocess.py

通过处理训练材料和语料库，得到并储存了以下文件：

训练材料	得到的文件	格式	作用
拼音汉字表.txt	./src/pinyin2char.json	{'pinyin': ['汉字 1', '汉字 2', ...], ...}	建立拼音和汉字之间的对应

			关系
sina_news_gbk	./src/char_frequency.json	{'pinyin': {'汉字 1': 频率 1, '汉字 2': 频率 2, ...}, ...}	按拼音得到单字的频率
sina_news_gbk	./src/char_possibility.json	{'pinyin': {'汉字 1': 概率 1, '汉字 2': 概率 2, ...}, ...}	按拼音得到单字的概率
sina_news_gbk	./src/word_frequency.json	{'pinyin1_pinyin2': {'词语 1': 频率 1, '词语 2': 频率 2, ...}, ...}	按拼音得到双字的频率
sina_news_gbk	./src/word_possibility.json	{'pinyin1_pinyin2': {'词语 1': 概率 1, '词语 2': 概率 2, ...}, ...}	按拼音得到双字的概率
sina_news_gbk	./src/trip_frequency.json	{'pinyin1_pinyin2_pinyin3': {'词语 1': 频率 1, '词语 2': 频率 2, ...}, ...}	按拼音得到三字的频率
sina_news_gbk	./src/trip_possibility.json	{'pinyin1_pinyin2_pinyin3': {'词语 1': 概率 1, '词语 2': 概率 2, ...}, ...}	按拼音得到三字的概率

首先使用 re.compile 定义符号模式，去除语料库中的标点符号和数字。由于在训练过程中发现这种方式清洗不彻底，会有一些网络符号残留，故又通过正则表达式匹配汉字的方式进一步保留语料库中的汉字。

然后遍历清洗后的语料库中的文字，得到单字的频率 char_frequency 和双字的频率 word_frequency，再将其转化成概率，保存为 json 格式的文件。

此外，在双字概率的处理中，由于会有多音字的影响，在整段文字的语境下，通过 pypinyin 库中的 lazy_pinyin 函数将汉字转化为拼音。而在三字概率的处理中，由于三字已经可以形成一个小语境，故只对这个三个字用 lazy_pinyin 函数。

四、训练与评估

基于字的二元模型，具体代码为 python ./src/pinyin.py <input_file> <output_file>

如 python ./src/pinyin.py ./data/std_input.txt ./data/output.txt

（一）算法描述

将输入的拼音以句为单位，转换为拼音列表。通过维特比算法处理拼音列表，完成拼音到文字的转换。

在二元模型中，$P(w_i|w_{i-1}) = \frac{P(w_i w_{i-1})}{P(w_{i-1})}$，故整句的概率为 $P(S) = \prod_{i=1}^{n} P(w_i|w_1 \dots w_{i-1})$。

在本实验中，概率 p 全部用-log(p)代替，乘法由加法代替，求最大值转为求最小值。由于 $\log(\prod_{i=1}^{n} p_i) = \sum_{i=1}^{n} \log(p_i)$，故令$p' = -log(p)$，有$\sum_{i=1}^{n} p_i'$最小时句子的概率最大。

使用维特比算法进行动态规划，初始化取句子的第一个拼音的所有可能单字的概率，随后前 i 位的概率等于前 i-1 位的概率加上连接第 i 位单字的概率，通过条件概率的公式可知连接的概率即为第 i-1 位第 i 位拼音对应的双字的概率减去第 i-1 位拼音对应单字的概率。同时在动态规划的过程中就进行判断与更新，保证当前路径保存的始终是扫描过的拼音串的最佳路径。

考虑到搜索单字和双字时可能遇到搜索不到的情况，融入二元的平滑模型。但与平滑模型不同的是，在某个单字或二元组不存在时，并不设置 p'的概率为正无穷。因为观察到生成的 char_possibility 中绝大部分单字的概率在 10 以下，极少数达到 16 多一点，而生成的

word_possibility 中大部分二元组的概率在 10 以下，极少数达到 13 多一点，故将未出现的二元组的 p’设置为 17，未出现的单字的 p’设置为 10。

算法最后会返回一个键为汉语句子，值为概率的字典，将其排序后选择 p’最小的输出，即为这句拼音对应的最优的汉语句子。

（二）实验效果

具体代码为 python ./src/test.py

在给定的测试样例上：

测试句子总数				测试总字数	
501				5235	
正确句子数量	句子准确率	句内错别字≥4	占比	正确字数	字准确率
207	41.32%	76	15.17%	4461	85.21%

训练时间：

在数据预处理步骤上花费时间较长，由于只是希望得到处理好的文件，故没有考虑专门用算法优化时间效率。

处理拼音汉字表耗时	0.00899958610534668 s
处理一二级汉字表耗时	0.0010006427764892578 s
处理语料库得到单字的概率耗时	6224.297502040863 s
处理语料库得到双字的概率耗时	578.2185373306274 s
生成一句话的平均时间	0.006553708436246404 s
生成所有给定测试样例的总时间	3.3950066566467285 s

（三）效果展示

1. 完全正确的语句：

- ji qi xue xi shi dang xia fei chang huo re de ji shu
  机器学习是当下非常火热的技术
- ren gong zhi neng ji shu fa zhan xun meng
  人工智能技术发展迅猛
- jin nian qing kuang bu tai hao
  今年情况不太好
- ji dong che jia shi yuan pei xun shou ce
  机动车驾驶员培训手册
- ta de ren sheng gui ji jiu shi yi ge tan xin suan fa
  他的人生轨迹就是一个贪心算法
- zui hou cheng wei le yi ge zai lian shu xie dai ma de pu tong ren
  最后成为了一个在脸书写代码的普通人
- ren min qun zhong xi wen le jian
  人民群众喜闻乐见
- qing shan lv shui jiu shi jin shan yin shan
  青山绿水就是金山银山
- yi xi jin ping tong zhi wei zong shu ji de dang zhong yang
  以习近平同志为总书记的党中央
- zou zhong guo te se she hui zhu yi dao lu
  走中国特色社会主义道路

- hu lian wang chan ye you ju da qian li
  互联网产业有巨大潜力

2. 部分错误语句

- bei jing shi shou ge ju ban guo xia ao hui yu dong ao hui de cheng shi
  北京市首个举办国夏奥会与冬奥会的城市
  北京是首个举办国夏奥会与冬奥会的城市
- zhong guo chen wen ying dui mao yi mo ca
  中国陈雯颖对贸易摩擦
  中国沉稳应对贸易摩擦
- yi ban hao rang ren min man yi de jiao yu wei zong zhi
  一般好让人民满意的教育为宗旨
  以办好让人民满意的教育为宗旨
- yi qing mian qian zhong sai liang guo zai ci xie shou gong ke shi jian
  疫情面前中塞两国再次携手工课时间
  疫情面前中塞两国再次携手共克时艰
- you jue dui de yan lun zi you ma
  有绝对的言论自由马
  有绝对的言论自由吗
- qing zang da shuai mai
  青藏大甩卖
  清仓大甩卖

3. 具体分析

可以看到，以正确语句为例，对于一些长难句、成语俗语、网络用语等，输入法都能给出准确的回答。但是在错误语句样例中，输入法在一些简单的句子上的翻译也会出错，甚至给出一些不太合理的回答。如“中国陈雯颖对贸易摩擦”，单看这句输出，除了句子不太完整外没有明显的语法错误，符合动态规划的规则，但是与应有的输出相比显得不太合理，可能给的拼音串再长一点，输出的准确率会提高。

此外，拼音法对于一些虚词、语气词的处理不太完美，而更偏向于输出名词，如将“是”输出“市”，将“吗”输出“马”。而对于一些专有名词，如“清仓”输出“青藏”，可能是由于语料库中有对于“青藏”的描写。

总之，拼音输入法错误案例的输出既有维特比算法本身的缺陷，也有语料库自身局限性的原因。

（四）性能分析

适当加入平滑，对于某个位置上的字的概率，考虑单字概率的部分占比，设置比例数 lbd，对在测试样例上生成的句子进行同样的分析，结果如下：

	正确句子数量	句子准确率	句内错别字≥4	正确字数	字准确率
lbd = 0.6	166	33.13%	104	4272	81.60%
lbd = 0.5	175	34.93%	102	4324	82.60%
lbd = 0.4	185	36.93%	93	4374	83.55%
lbd = 0.3	192	38.32%	87	4403	84.11%
lbd = 0.2	200	39.92%	85	4423	84.49%
lbd = 0.12	205	40.92%	79	4457	85.14%
lbd = 0.11	207	41.32%	78	4463	**85.25%**

lbd = 0.1	207	41.32%	78	4462	**85.23%**
lbd = 0.09	207	41.32%	79	4461	**85.21%**
lbd = 0.08	205	40.92%	79	4458	85.16%
lbd = 0.0	207	41.32%	76	4461	85.21%

从上述表格中可以看出，计算某个字的概率，还是应该主要以二元模型为主，减少单字概率的占比，但是也可以适当加点占比，可能效果会更好，如表格中当 lbd = 0.11 的时候。

（五）时间与空间复杂度分析

1. 时间复杂度

对输入的拼音列表进行迭代，这一步的时间复杂度为 O(n)，其中 n 是拼音列表的长度。

在每次迭代中，对当前拼音所对应的所有汉字进行遍历，时间复杂度取决于当前拼音对应的汉字数量，假设为 m。

在内部循环中，需要检查当前拼音和前一个拼音所对应的汉字组合是否在预先生成的单字或双字概率文件中。因为这些预先计算好的可能性是存储在字典中，所以可以通过哈希表等方法在 O(1)时间内访问。

对每个组合计算概率，并选择概率最高的组合，时间复杂度为 O(m)。

故算法的整体时间复杂度为 $O(nm^2)$。

2. 空间复杂度

设拼音列表有 n 个拼音，平均每个拼音对应 m 个汉字，则储存数据的空间复杂度近似 $O(n + nm + nm^2)$。而维特比算法使用的递归深度为 O(n)，故整体空间复杂度为 $O(nm^2)$。

五、算法改进

（一）基于字的三元模型

数据预处理也在./src/preprocess.py 中

训练代码：python ./src/pinyin_triple.py <input_file> <output_file>，

如 python ./src/pinyin_triple.py ./data/std_input.txt ./data/output_triple.txt

处理语料库得到三字的概率耗时	2799.219786167145 s
生成一句话的平均时间	0.018263945322550698 s
生成所有给定测试样例的总时间	9.26927924156189 s

1. 算法描述

与基于字的二元模型算法类似，用-log(p)代替概率 p，转化为求加法的最小值。$P(w_i|w_{i-1}w_{i-2}) = \frac{P(w_iw_{i-1}w_{i-2})}{P(w_{i-1})P(w_{i-2})}$。对于维特比算法，初始化取句子的第一个拼音的所有可能单字的概率，随后前 i 位的概率等于前 i-1 位的概率加上连接第 i 位单字的概率，通过条件概率的公式可知连接的概率即为第 i-2 位第 i-1 位第 i 位拼音对应的三字的概率减去第 i-1 位第 i 位拼音对应双字的概率。同时在动态规划的过程中就进行判断与更新，保证当前路径保存的始终是扫描过的拼音串的最佳路径。

同样的，融入三元的平滑模型，且不将 p’的值设置为正无穷。如果三元模型不存在于数据库中，将三元模型的 p’设置为 17，若三元模型存在而前两位拼音的二元模型不存在，则将二元模型的 p’设置为 10。

2. 实验效果：

测试句子总数	测试总字数

501				5235	
正确句子数量	句子准确率	句内错别字≥4	占比	正确字数	字准确率
282	56.29%	94	18.76%	4452	85.04%

3. 与字的二元模型的比较

	字的二元模型	字的三元模型
正确句子数量	207	282
句子准确率	41.32%	56.29%
句内错别字≥4 的句子数	76	94
占比	15.17%	18.76%
正确字数	4461	4452
字准确率	85.21%	85.04%
生成一句话的平均时间	0.006553708436246404 s	0.018263945322550698 s
生成所有给定测试样例的总时间	3.3950066566467285 s	9.26927924156189 s

由上表可以看出，基于字的三元模型的拼音输入法的句准确率增大近 15%，而字准确率几乎保持不变。虽然花费更多的时间生成结果，但是实验效果还是十分可观的。不过若是更加追求字准确率的话，建议还是使用字的二元模型，花费的时间更少，效率更高。

4. 效果展示

主要看了一下二元模型挑选出来的例句。

（1）正确的句子：

- yi ban hao rang ren min man yi de jiao yu wei zong zhi
  以办好让人民满意的教育为宗旨
- qing zang da shuai mai
  清仓大甩卖
- you jue dui de yan lun zi you ma
  有绝对的言论自由吗

（2）错误的句子

- bei jing shi shou ge ju ban guo xia ao hui yu dong ao hui de cheng shi
  北京市首个举办国夏奥会与冬奥会的城市
  北京是首个举办国夏奥会与冬奥会的城市
- zhong guo chen wen ying dui mao yi mo ca
  中国陈文英对毛衣瘼拆
  中国沉稳应对贸易摩擦
- yi qing mian qian zhong sai liang guo zai ci xie shou gong ke shi jian
  疫情面前中塞两国再次携手攻克时艰
  疫情面前中塞两国再次携手共克时艰

（3）具体分析

可以看到基于字的三元模型的拼音输入法克服了部分二元模型的缺点，首尾的虚词与介词大部分能正确处理，且对专有词汇的处理更准确，如“清仓”等。

然而，基于字的三元模型的拼音输入法仍然极大地依赖语料库，比如“北京是”依然会输出“北京市”，说明算法的提升有一定的瓶颈，最后还得通过扩大语料库的方式提高输入法的准确性。

5. 性能分析

在基于字的三元模型的基础上考虑字的二元模型的占比，设置比例数 lbd，对在测试样例上生成的句子进行同样的分析，结果如下：

	正确句子数量	句子准确率	句内错别字≥4	正确字数	字准确率
lbd = 0	282	56.29%	94	4452	85.04%
lbd = 0.1	283	56.49%	94	4464	85.27%
lbd = 0.2	283	56.49%	94	4462	85.23%
lbd = 0.25	283	56.49%	93	4465	85.29%
lbd = 0.3	286	57.09%	93	4471	**85.41%**
lbd = 0.35	285	56.89%	92	4472	**85.43%**
lbd = 0.4	285	56.89%	91	4469	85.37%
lbd = 0.5	284	56.69%	89	4470	85.39%

可以看到，适当融入二元模型后，句准确率和字准确率都略有提高，在测试的参数中，当 lbd = 0.35 时准确率提高最多。

6. 时间与空间复杂度分析

与基于字的二元模型分析类似，设拼音列表有 n 个拼音，平均每个拼音对应 m 个汉字。则时间复杂度仍为 $O(nm^2)$，但空间复杂度为 $O(nm^3)$。

（二）对多音字的处理

由于不确定多音字对输入法效果的影响有多大，分别测试了考虑多音字和不考虑多音字的双字模型的字句准确率。

如果考虑多音字的话，在数据预处理阶段就利用 pypinyin 库中的 lazy_pinyin 函数处理语句，在计算字的频率时同一个字的不同发音对应的字的数量是不同的。而不考虑多音字的话计算字的频率时同一个字的不同发音对应的字的数量是相同的。

结果如下：

	句准确率	字准确率
考虑多音字	41.32%	85.21%
不考虑多音字	41.32%	85.27%

几乎没有差别，究其原因可能是因为数据预处理生成单字的概率时使用了 pinyin2char.json 文件，而这里面仍然会有多音字，只不过上面不考虑多音字的做法只是将这些字的数量设置为一样。

而在数据预处理方面，可能由于遍历的原因，不考虑多音字生成对应文件所花的时间多于考虑多音字（8059s vs 6224s），故采用考虑多音字的处理方式。

六、实验感受

一开始十分担忧自己能否顺利完成，因为不擅长写算法与 oj 题，写出能运行的维特比算法就花了很久时间，由于没有给 1_word.txt 和 2_word.txt 文件，要测试算法的正确性只能在本地构建这些文件。好不容易 oj 上能有 27.2 分了，以为会很容易就改到 35+，然而在这里卡了好几天，不管怎么修改算法，分数只降不升。我几乎都快放弃了，好不容易说服自己大不了不要这部分的分数，冷静下来先写实验二的部分。没想到在调试实验二时修改了一下算法，并且在这个基础上将其跑通了，再放到 oj 上运行能有 37 分了。说真的，那一刻整个人都激动的发抖，唯一的想法是这次大作业我可以完成了，并且可能可以完成的很好。

实验二的部分麻烦在于有很多参数可以调，使用的算法与文件也具有多样性，此外根据要求书写实验报告也十分耗费精力与时间，不过写完后还是成就感满满。

通过本次实验，我成功实现了拼音转文字的任务，更加深入地理解了维特比算法与动态规划，学会了字的二元三元模型，并能够融会贯通通过调整参数以及结合不同模型，提升模型的句准确率和字准确率，同时了解到了语料库对于模型建立的重要性。