拼音输入法 实验报告

计**81** 肖光烜 **2018011271**

设计说明

文件结构

```
.
├── data
|   ├── input.txt
|   ├── output_bi.txt
|   ├── output_quad.txt
|   └── output_tri.txt
├── log
|   ├── eval_bi.log
|   ├── eval_bi_01.log
|   ├── eval_bi_10.log
|   ├── eval_bi_37.log
|   ├── eval_bi_55.log
|   ├── eval_quad.log
|   ├── eval_tri.log
|   ├── eval_tri_001.log
|   ├── eval_tri_01.log
|   ├── eval_tri_111.log
|   ├── eval_tri_123.log
|   └── eval_tri_199.log
├── model
|   ├── bigram.pkl
|   └── trigram.pkl
├── src
|   ├── data_process
|   |   ├── data_process.ipynb
|   |   ├── data_process.py
|   |   ├── datasplit.py
|   |   ├── prepare_input.py
|   |   └── raw2train.py
|   ├── eval.py
|   ├── eval_bi.sh
|   ├── eval_bi_param.sh
|   ├── eval_quad.sh
|   ├── eval_tri.sh
|   ├── eval_tri_param.sh
|   ├── hmm.py
|   ├── interact_bi.sh
|   ├── interact_quad.sh
|   ├── interact_tri.sh
|   ├── main.py
|   ├── pinyin_hanzi.py
|   ├── pinyin_im.py
|   ├── test_bi.sh
|   ├── test_quad.sh
```

```
|   ├── test_tri.sh
|   ├── train_bi.sh
|   ├── train_quad.sh
|   └── train_tri.sh
└── table
    ├── README.md
    ├── hanzi2id.txt
    ├── hanzi2pinyin_id.npy
    ├── pinyin2hanzi_id.pkl
    └── pinyin2id.txt
```

`src/` 文件夹放置所有源代码，其中`/src/data_process/`中为数据预处理过程的一些脚本，与运行模型无关。本拼音输入法采用OOP风格模块化设计：`src/hmm.py`为一阶、二阶、三阶隐马尔科夫模型，均继承了一个通用隐马尔科夫模型抽象类；`src/pinyin_hanzi.py`为处理拼音汉字相互转化和编码解码的模块；`src/pinyin_im.py`为拼音输入法模块，封装了一个隐马尔可夫模型和编码器；`src/main.py`为运行拼音输入法的主程序；`src/eval.py`为模型测试所用程序。其中还附送了一些bash脚本文件，其中`interact_.*`脚本用于加载训练好的模型进行命令行交互输入，`test_.*`的脚本可以加载`data/input.txt`文件进行文件输入运行。

`data/` 文件夹放置文件输入输出的`input.txt`、`output.txt`文件。

`table/`文件夹保存了必要的拼音表和汉字表，这是模型运行所必需的文件。

`model/`下附了已经训练好的二元、三元模型。

`log/`下附了训练和测试过程的日志文件，可以通过这些文件看实验结果。

使用方式

在`src/`文件夹下有现成的bash脚本负责各项任务，并且`model/`附了二元模型和三元模型已经训练好的pickle文件，可以直接加载进行测试。其中`interact_.*`脚本用于加载训练好的模型进行命令行交互测试，`test_.*`的脚本可以读取`data/input.txt`文件进行文件输入运行，这两个都可以在目录中直接运行。其中`train_.*`和`eval_.*`脚本由于没有附带训练数据，不能直接运行。

同时此模型也支持命令行参数指定运行模式，命令行参数使用如下：

```
usage: main.py [-h] [--train TRAIN] [--load LOAD] [--save SAVE]
               [--input INPUT] [--output OUTPUT] [--table_dir TABLE_DIR]
               [--mode {test,interact,eval}]
               [--model {bigram,trigram,quadgram}] [--lamb LAMB [LAMB ...]]

optional arguments:
  -h, --help            show this help message and exit
  --train TRAIN         Training data directory.
  --load LOAD           Load trained model file.
  --save SAVE           Output trained model file.
  --input INPUT         Input pinyin text file.
  --output OUTPUT       Output hanzi text file.
  --table_dir TABLE_DIR
                        Hanzi & Pinyin tables directory.
  --mode {test,interact,eval}
                        Run mode.
  --model {bigram,trigram,quadgram}
                        HMM model to use.
  --lamb LAMB [LAMB ...]
```

```
List of HMM linear smoothing parameters.
```

问题描述

给定一定长度的拼音串$O = \{o_1, o_2, \ldots, o_n\}$， 要求给出其最有可能表示的汉字串$X = \{x_1, x_2, \ldots, x_n\}$. 这类问题可以用隐马尔科夫模型（HMM）来解决。

算法思路

如前所述，考虑用隐马尔可夫模型解决此问题，即将拼音串O当做可见的状态串（Observation），将其背后的汉字串视为导出观测串的隐状态串（Hidden States），每一个汉字x_i可以根据一定的分布$p(o|x_i)$导出对应的拼音（对应于多音字）。根据不同的马尔科夫假设，可以分别得到一阶、二阶、三阶隐马尔可夫模型，分别对应于自然语言处理中的Bigram、Trigram、Quadgram语言模型。在对应的语言模型上进行参数估计，随后使用经典的Viterbi动态规划算法求解最大后验概率的隐状态序列。

Bigram 模型

Bigram模型即二元模型、一阶隐马尔可夫模型，此模型假设下一个汉字只与上一个汉字有关，而与隐状态串中之前其他所有状态无关：

$$p(x_n|(x_1, x_2, \ldots, x_n - 1)) = p(x_n|x_{n-1})$$

参数估计

需要估计三种参数：

- 起始概率：汉字出现的概率分布。

$$\hat{p}(x_i) = \frac{count\{x_i\}}{count\{*\}}$$

- 转移概率：汉字转移到下一个汉字的概率分布。

$$\hat{p}(x_j|x_i) = \frac{count\{(x_i, x_j)\}}{count\{(x_i, *)\}}$$

- 发射概率：汉字对应到拼音上的概率分布。

$$\hat{p}(o_j|x_i) = \frac{count\{x_i -> o_j)\}}{count\{(x_i -> *)\}}$$

Viterbi 算法

如果通过暴力穷举所有可能的来求得最大后验概率隐状态串，其时间复杂度至少是$\Omega(W^n)$级别的，其中W为隐状态总状态数（汉字总数），n为串长度。这种复杂度是难以接受的，而且浪费了一阶马尔科夫假设。

根据一阶马尔科夫假设，每一个隐状态仅与上一个隐状态相关，因此可以维护一个层次表记录每一层每一个可能状态相应的后验概率。在层与层之间转移时采用全连接的方式根据贝叶斯公式

$$p((x_1, \ldots, x_n)|(o_1, \ldots, o_n)) \propto p((x_1, \ldots, x_{n-1})|(o_1, \ldots, o_{n-1})) \times p(x_n|x_{n-1}) \times p(o_i|x_i)$$

求得下一层每个状态的后验概率，由此自底向上递推即可获得最优后验概率隐状态串，并且将复杂度降低到$O(nW^2)$. 这个算法就是著名的Viterbi解码算法。

平滑化

由于语料有限，Bigram模型的转移矩阵（$W \times W \approx 10^8$）中许多元素都是0，即相应的模式并未在语料中出现过。如果Viterbi算法中某两层之间的所有连接都从未在语料中出现过，下一层估计的所有概率就归0了。这显然是不合适的，因为之前的信息全部浪费掉了。一种可能做法是通过线性插值对转移概率进行平滑化：

$$p_{intercepted}(x_n|x_{n-1}) = \lambda_0 p(x_n) + \lambda_1 p(x_n|x_{n-1}),\ \Sigma_{i=0}^{1}\lambda_i = 1$$

Trigram 模型

Trigram模型即三元模型、二阶隐马尔可夫模型，此模型假设下一个汉字只与上两个汉字有关，而与隐状态串中之前其他所有状态无关：

$$p(x_n|(x_1, x_2, \ldots, x_n - 1)) = p(x_n|(x_{n-2}, x_{n-1}))$$

参数估计

同样需要估计三种参数：

- 起始概率：汉字出现的概率分布。

$$\hat{p}(x_i) = \frac{count\{x_i\}}{count\{*\}}$$

- 转移概率：二元汉字组转移到下一个汉字的概率分布。

$$\hat{p}(x_k|(x_i, x_j)) = \frac{count\{(x_i, x_j, x_k)\}}{count\{(x_i, x_j, *)\}}$$

- 发射概率：汉字对应到拼音上的概率分布。

$$\hat{p}(o_j|x_i) = \frac{count\{x_i -> o_j)\}}{count\{(x_i -> *)\}}$$

Viterbi 算法

根据二阶马尔科夫假设，每一个隐状态仅与上两个隐状态相关，因此可以维护一个层次表记录每两层每一个可能二元对相应的后验概率。在层与层之间转移时采用全连接的方式根据贝叶斯公式求得下一层每个状态的后验概率，由此自底向上递推即可获得最优后验概率隐状态串，并且将复杂度降低到$O(nW^3)$.

平滑化

同样可以通过线性插值对转移概率进行平滑化：

$$p_{intercepted}(x_n|(x_{n-1}, x_{n-2})) = \lambda_0 p(x_n) + \lambda_1 p(x_n|x_{n-1}) + \lambda_2 p(x_n|(x_{n-1}, x_{n-2}))\ \Sigma_{i=0}^{2}\lambda_i = 1$$

Quadgram 模型

参数估计

同样需要估计三种参数：

- 起始概率：汉字出现的概率分布。

$$\hat{p}(x_i) = \frac{count\{x_i\}}{count\{*\}}$$

- 转移概率：二元汉字组转移到下一个汉字的概率分布。

$$\hat{p}(x_l|(x_i, x_j, x_k)) = \frac{count\{(x_i, x_j, x_k, x_l)\}}{count\{(x_i, x_j, x_l, *)\}}$$

- 发射概率：汉字对应到拼音上的概率分布。

$$\hat{p}(o_j|x_i) = \frac{count\{x_i->o_j)\}}{count\{(x_i->*)\}}\hat{p}(o_j|x_i) = \frac{count\{x_i->o_j)\}}{count\{(x_i->*)\}}$$

Viterbi 算法

根据三阶马尔科夫假设，每一个隐状态仅与上三个隐状态相关，因此可以维护一个层次表记录每两层每一个可能三元对相应的后验概率。在层与层之间转移时采用全连接的方式根据贝叶斯公式求得下一层每个状态的后验概率，由此自底向上递推即可获得最优后验概率隐状态串，并且将复杂度降低到$O(nW^4)$.

平滑化

同样可以通过线性插值对转移概率进行平滑化：

$$p_{intercepted}(x_n|(x_{n-1},x_{n-2},x_{n-3})) = \lambda_0 p(x_n) + \lambda_1 p(x_n|x_{n-1})$$
$$+ \lambda_2 p(x_n|(x_{n-1},x_{n-2})) + \lambda_3 p(x_n|(x_{n-1},x_{n-2},x_{n-3})) \quad \Sigma_{i=0}^{3}\lambda_i = 1$$

N-gram 模型

根据以上推到，可归纳得到N-gram模型使用Viterbi算法的时间复杂度为$O((n-N+1)W^N)$。这是合理的，当N接近字符串长度n时，一层Viterbi算法的估计就相当于穷举了所有种隐状态串的组合形式，与暴力枚举无异。

由此可以从另一个角度思考Viterbi算法，Viterbi算法是基于N阶马尔科夫模型的最大后验估计算法，（N+1）即为至少所需暴力枚举的长度。因此将n层暴力枚举长度限定到（N+1）层正是N阶马尔科夫假设带来的优化所在。

实现细节

数据处理

我将语料中的所有句子用非汉字字符划分为句子，再将这些句子中长度小于2的剔除，因为根据这种划分方法划出的单字符句子多数情况下属于噪声，且去掉不会给整体分布带来本质影响。由此得到31709393个句子。

多音字处理

注意到语料仅提供了汉字序列，并无对应的拼音序列。因此如果仅使用汉字预料，对发射概率仅能进行等概率假设的估计，这在处理多音字问题上将会出现令人啼笑皆非的结果，例如：

> 未进行多音字矫正：kai ju 开车
>
> 进行多音字矫正：kai ju 开局

原因在于，每个汉字的每个读音使用的频率是有很大差异的，如果认为每个读音都等概率的话。有些**常见词语的罕见读音形式**就会被过分估计，例如“kai ju”会被错误的估计为“开车”而非“开局”。

为了解决这一情况，首先我将提供的汉字表和拼音表中的所有可能的汉字-拼音对(x_i,o_j)进行重新编码，得到新的无歧汉字集W'。比如“车”拥有两个汉字-拼音对（车,che）、（车,ju），分别给予不同编码。随后，我利用pypinyin库将训练语料进行注音，再重编码为训练语料。重编码后的“汉字”就仅拥有一个读音，因此就不必估计发射概率，多音字问题也得到了解决。

$$\hat{p}(o_j|x_i) = \begin{cases} 1 & (x_i,o_j) \in W' \\ 0 & (x_i,o_j) \notin W' \end{cases}$$

数据增强

由于提供的新浪新闻语料年代过于久远，不能提供当下新兴的一些词汇，我将近期的清华新闻也引入了训练，但是效果并不显著。原因在于两个语料都属于新闻，缺乏生活用语、百科知识。如果要进一步增强此输入法的能力，可以采用更为多元的语料进行训练。

实验结果

我原定将sina新闻中的31709393个句子按照8:2的比例划分为训练集和测试集进行测试，但是随后发现测试过程耗时过长，于是我改用其中80%的句子进行训练，再剩下的20%句子中随机选取2000个进行测试。这种方法得到的测试结果是可以接受的，其95%置信区间的宽度最大不超过3%，是较为准确的准确率测试范围。

在评价标准上，我选取了逐字准确率和逐句准确率两种指标。由于测试集中的句子并非实际生活中最常使用的句子，存在噪声，因此逐句准确率只能部分反映模型的建模能力。逐字准确率更能体现模型的性能，因为在输入法使用过程中，实际上如果长句子中只有少部分字，人们往往倾向于去单独修改这些字而不是重新一个一个打一遍句子，所以逐字准确率越高，用户体验越好的假设是可以成立的。

模型对比

模型（参数选择）	逐字准确率(%)	逐句准确率(%)	10字句测试时间(s)
Bigram (0.01,0.99)	88.06	51.10	**0.025**
Trigram (1e-4, 1e-2, 1)	**95.92**	**80.50**	0.25
Quadgram (1e-6, 1e-4, 1e-2, 1)	95.92	80.50	5.0

由此可见，的确考虑之前的状态越多，模型的表现越好，但是由此带来的是测试时间和占用空间的迅速提高。也可看出，当模型从二元进化到三元时的提升十分显著，尤其是在逐句准确率上，这是由于模型能够更好综合先前的信息进行整体推断。但是当模型从三元进化到四元时，从这2000个测试样例来看没有任何提升。这是由于汉语中本身绝大多数的词语的长度都小于6，因此三元模型是足够的。四元模型仅仅在长度大于等于6且某字前第三个字对此字生成有影响时才会起到效果，这种模式从实验来看是很少的，因此三元模型对于汉语建模是足够的了，其余不能解决的应是HMM或数据本身的问题。

参数选择

线性插值法是当更高阶的转移没有出现过，从而采用低阶转移概率近似的一种“权宜之计”。因此作为权宜之计的低阶转移概率不应拥有过高的权重，否则将”喧宾夺主“，掩盖高阶转移概率的宝贵性。根据这一原则，可以推导合适的参数选择范围。

高阶转移概率的权重与低阶转移概率的权重的比值$r_i = \frac{\lambda_{i+1}}{\lambda i}$拥有以下直观含义：

> 高阶转移概率每提高Δp与低阶转移概率提高$r_i \Delta p$起到的效果等同。

由此也可以看出，权重的绝对数值并无意义，真正具备意义的是权重之间的比值，因此参数选择上并不一定遵循总和为1的约束。

以下是参数选择的实验效果，由于一次实验时间较长，仅用二元、三元模型选取了具有代表性的参数进行测试。

Bigram

λ_0, λ_1	逐字准确率(%)	逐句准确率(%)	描述
1, 0	49.81	2.08	退化为单字独立分布模型
0.5, 0.5	86.40	46.12	字频与转移概率同权重
0.3, 0.7	87.49	49.15	三七开
0.01, 0.99	**88.06**	**51.10**	一比一百
0, 1	87.95	50.98	不进行平滑化

Trigram

$\lambda_0, \lambda_1, \lambda_2$	逐字准确率(%)	逐句准确率(%)	描述
1, 1, 1	95.21	76.66	三种概率等权重
1, 2, 3	95.66	78.61	等差数列
1, 9, 9	95.60	78.49	一阶二阶等权重，零阶小权重
1e-4, 1e-2, 1	**95.92**	**80.50**	等比数列
0, 0, 1	3.73	0.00	不进行平滑化

由此可见不论是不进行平滑化还是平均配比都不能得到较好的效果，最好的效果在高低阶权重比稳定在较高水平时（本实验中采用100）达到。

案例分析

Bigram

优秀案例

- 两会在北京召开
- 西红柿炒鸡蛋
- 清华大学计算机系
- 全面从严治党

错误案例

正确	预测
通过有交通信号灯的控制路口时	通过有交通信号灯的空之路口市
你的理解是正确的	你的历届时正确的
中纪宏观经济学	中级宏观经济学
顺职员违纪	吮指原味鸡
他养了一只青蛙作为宠物	他养了一致青瓦作为宠物
寻寻觅觅冷冷清清凄凄惨惨戚戚	寻寻觅觅冷冷清清凄凄惨惨凄凄

从优秀案例和错误案例均可窥见二元模型的特点：对于环环相扣的常用词汇建模能力较强，例如“西红柿炒鸡蛋”、“全面从严治党”；对于需要长依赖，不常见的组合建模能力较弱，例如“中级宏观经济学”。

Trigram

优秀案例

- 通过有交通信号灯的控制路口时
- 你的理解是正确的
- 中级宏观经济学
- 吮指原味鸡
- 他养了一只青蛙作为宠物
- 寻寻觅觅冷冷清清凄凄惨惨戚戚
- 猪软骨春笋拉面

错误案例

正确	预测
煤炭从储存车间进入机器	煤炭从出村车间进入机器
是韩国近代医疗的发源地	是韩国晋代医疗的发源地
长江中下游及其以北地区多在昨天气温跌入低谷	长江中下游及其以北地区多在做天气温跌入低谷
就这样在零下的气温	就这样在岭下的气温
见此骗局有利可图	键词骗局有利可图

首先，三元模型完美解决了以上二元模型列举的所有错误案例，可见其对低频搭配和长依赖解决能力较好。然而根据错误案例也可看出此模型的问题："韩国晋代"问题在于此模型仅仅基于字频，没有引入外部知识；"岭下的气温"说明此模型仍然依赖较短，而且基于正向传播，并没有从后向前推断的能力，然而这对于整句理解十分有必要。如果能够先看到“气温”再看到“零下”，此例不难推出。

Quadgram

Quadgram模型相较于Trigram模型没有观测到任何提升，而且发现了一个Trigram能做对但是Quadgram做不对的例子，十分迷惑。

错误案例

正确	预测
猪软骨春笋拉面	猪软骨春损辣面

展望与收获

从数据上，此模型可以采用更加多元的数据进行训练，从而能够解决百科知识、生活用语稀缺的问题。

从方法上，此模型可以引入双向马尔科夫链，利用两个方向的信息进行预测，解决后向依赖问题。采用深度学习方法可以引入跳跃连接，更好加强模型对句子整体的认识。同时也可以加入词级别的隐马尔可夫问题，但是需要良好的分词工具和完备的汉语词表做支撑。

从评价上，可以引入更加丰富的模型表现评价指标，例如汉明距离、Precision、Recall、F1 score等等。

此次实验中，我通过自学和动手实现深入理解了隐马尔可夫模型及其预测算法，同时不断调试和优化程序过程也巩固了我的编程水平和算法功底。同时，我在面对诸多问题（多音字消歧、空间爆炸、参数选择）时独立思考、解决问题的过程有别于传统的作业，我体会到更像科学研究一样的独立探索过程。尽管这次作业做得很辛苦漫长，但我依然享受这一过程，它让我对相关知识拥有了极其深刻的印象，同时提供给我短小精悍的独立科研体验，我收获良多。