拼音输入法实验

计01　容逸朗　2020010869

1　实验环境

- CPU: 四核 i7-4770 2.2GHz
- RAM: 16 GB 1600 MHz DDR3
- SSD: APPLE SSD SM0256F
- Python 3.9.5

2　模型使用方法

2.1　文件结构

文件结构如下，具体文件的作用请参考第 3 节的内容：

```
├┬ readme.md
│ ├ data
│ │  ├ input.txt
│ │  ├ std_output.txt
│ │  └ output.txt
│ ├ raw_data
│ │  ├ hanzilist.txt（一二级汉字表）
│ │  ├ pinyinlist.txt（拼音-汉字对应表）
│ │  └ corpus（存放语料库数据）
│ │    ├ sina_news_gbk（新浪新闻语料库）
│ │    └ wiki_zh（维基百科语料库）
│ └ src
│   ├ compare.py
│   ├ gen_chi_map.py
│   ├ gen_pinyin_list.py
│   ├ get_freq_sina.py
│   ├ get_freq_wikipedia.py
│   ├ main.py
│   ├ normal_2.py, normal_23.py, normal_3.py（无平滑模型）
│   ├ smooth_2.py, smooth_23.py（平滑模型）
│   └ pretrained_data, pretrained_data_with_wikipedia
│     ├ frequency1.txt
│     ├ frequency2.txt
│     ├ frequency3.txt
│     ├ frequencyF.txt
```

```
        ├─ hanzidecode.txt
        ├─ hanzimap.txt
        └─ pinyinlist.txt
```

2.2 使用方法

首先进入到文件夹的根目录内，这时你应该可以看见：`readme.md` 以及文件夹 `data`，`src`，`raw_data`，`pretrained_data`，`pretrained_data_with_wikipedia` 。

请确保你是在文件根目录下开启命令行，然后在命令行输入：

```
cd src
python3 main.py -h
```

这时你应该看到：

命令	参数	用途
-gc	无	拼音、一二级汉字预处理
-gf	无	统计新浪语料库对应汉字频率
-gfb	无	统计新浪与维基语料库对应汉字频率
-r	对应模型文件	运行模型并对比结果
-d	选用的统计文件，默认为"pretrained_data"	选定语料库（请先运行 -gf 或 -gfb）
-c	无	对比输出文件
-if	文件名，默认为"../data/input.txt"	指定输入文件的位置
-of	文件名，默认为"../data/output.txt"	指定输出文件的位置
-sf	文件名，默认为"../data/std_output.txt"	指定对比文件的位置

其中：

1. `-r` 指令的参数可以选择 normal_2.py, normal_3.py, normal_23.py, smooth_2.py, smooth_3.py
2. `-d` 指令的参数可以选择 pretrained_data 和 pretrained_data_with_wikipedia
3. `-if`，`-of`，`-sf` 是以当前目录（src）下的对应位置，例如："../data/input.txt"

请注意：由于文件大小限制，可以先在 https://github.com/brightmart/nlp_chinese_corpus 下载维基百科语料库，解压缩后放入 `raw_data/corpus/wiki_zh` 内。（否则执行命令 `-gfb` 会出错）

接着可以运行：

```
python3 main.py -gc -gf
```

若你已下载维基语料库且放入指定位置后，也可以运行：

```
python3 main.py -gfb
```

生成基础语料的统计文件。

然后可以尝试下面的代码开始预测拼音：

```
python3 main.py -r normal_2.py
python3 main.py -d pretrained_data_with_wikipedia -r normal_23.py
python3 main.py -d pretrained_data_with_wikipedia -r smooth_23.py
```

其中：

1. 第一行是使用新浪语料库的字二元模型；
2. 第二行是使用新浪语料库与维基语料库的字二、三元模型。
3. 第三行是使用新浪语料库与维基语料库的字二、三元平滑模型。

除此之外也可以使用其他命令组合测试喔！

3 预处理

这里我选用了 `sina_news_gbk` 中的所有新闻内容和维基百科 `nlp_chinese_corpus/wiki2019zh` 的内容作为统计用语料。

统计好的数据存放在 `pretrained_data` （新浪新闻）或 `pretrained_data_with_wikipedia` （新浪新闻与维基百科）文件夹下。

3.1 汉字编号

生成数据程序： `src/gen_chi_map.py` ；
结果存放位置： `pretrained_data/hanzilist.txt` ；

这里汉字编号的主要思想是根据 `raw_data/hanzimap.txt` 的顺序来对每一个汉字由 1 开始进行编号并使用 `dict` 存储汉字到数字和数字到汉字的映射关系，操作完成后，利用 `pickle` 库将生成的字典存放于预定的目录中。

3.2 拼音-汉字表

生成数据程序： `src/gen_pinyin_list.py` ；
结果存放位置： `pretrained_data/pinyinlist.txt` ；

根据 `raw_data/pinyinlist.txt` ，我们可以得到每一个拼音可能对应的汉字表，为此我们可以将其储存为一个字典，此字典的键值为拼音，内容为一个包含所有可能汉字的列表。

3.3 频率表

生成数据程序： `src/get_frequency.py` ；
结果存放位置： `pretrained_data/frequency1.txt` 等文件；

首先遍历语料库，将里面的句子按标点符号和换行符分开，然后分别统计单字出现的次数、单字在句首出现的次数，两个字（或更多）连续出现的次数，最后将这些数据以字典的形式分别存放到文件中。

4 模型

4.1 思想

不妨假设每一个字只与前面生成的字有关，与后面的字无关，这时可以通过条件概率公式来计算被预测句子的可能性：

$$P(x_1, x_2, x_3, \ldots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)\ldots P(x_n|x_1, x_2, \ldots, x_{n-1}) \tag{1}$$

这里 x_i 指拼音 p_i 预测出的文字。

4.2 字的二元模型

考虑到 $P(x_i|x_1x_2\ldots x_{i-1})$ 的预处理难度较大，为此我们可以假设每一个字只与前面的一个字有关，此时 (1) 式变为：

$$P(x_1, x_2, x_3, \ldots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_2)\ldots P(x_n|x_{n-1}) \tag{2}$$

根据概率公式，我们有：

$$P(x_i|x_{i-1}) = \frac{P(x_i, x_{i-1})}{P(x_{i-1})} \approx \frac{C(x_i, x_{i-1})}{C(x_{i-1})} \tag{3}$$

为了计算上的简便，这里将概率运算转化为汉字在语料库出现的次数。上式中 $C(x_i, x_{i-1})$ 指 x_{i-1} 和 x_i 在语料库中连续出现的次数，$P(x_{i-1})$ 指 x_{i-1} 在语料库中出现的次数。

除此之外，我们还需要找出一种方法计算 $P(x_1)$ ，一种简单的方法是根据 x_{i-1} 出现的总数与拼音对应汉字的出现总数的比来计算，公式如下：

$$P(x_1) = \frac{C(x_1)}{\sum_{x \in PY_1} C(x)} \tag{4}$$

这里 PY_i 指的是句子中第 i 个拼音对应的汉字表。

4.3 字的 n 元模型

n 元模型是指，每次生成一个字时只考虑前面 $n-1$ 个字的模型，对 (1) 式作微调得到：

$$\begin{aligned} P(x_1, x_2, x_3, \ldots, x_n) &= \prod_{i=1}^{n} P(x_i|x_{i-n+1}, x_{i-n+2}, \ldots, x_{i-1}) \\ &= \prod_{i=1}^{n} \frac{P(x_{i-n+1}, x_{i-n+2}, \ldots, x_i)}{P(x_{i-n+1}, x_{i-n+2}, \ldots, x_{i-1})} \end{aligned} \tag{5}$$

从中可以知道 n 元模型需要 $n-1$ 元模型作基础。

4.4 词的二元模型

思路与字的二元模型类似，在这里约定 $w_i = x_{i1}x_{i2}\ldots x_{ik}$ 为一个单词，那么 (2) 式可被表示为：

$$P(w_1, w_2, \ldots, w_m) = P(w_1)P(x_2|w_1)\ldots P(w_m|w_{m-1}) \tag{6}$$

想要实现词二元模型需要有一个好的词典，同时配合动态规划做句子划分，复杂度较大。

5 算法

5.1 搜索方式

5.1.1 字 n 元模型（暴搜）

字 n 元模型相当于对每一个子句尝试所有的可能值，因此算法退化为暴搜，此时复杂度达到 $O(|PY_i|^n)$ 是不可接受的。

5.1.2 字二元模型

由于二元模型只考虑前一个字，因此只需要考虑所有以 PY_{i-1} 中汉字为结尾且长度为 $i-1$ 的最优句便可。利用这种思路可以得到长度为 i 且结尾为 $w \in PY_i$ 的概率的转移方程：

$$\begin{aligned} P(W_{i,w}) &= \max_{w_1 \in PY_{i-1}} \left(P(W_{i-1,w_1}) \cdot P(w|W_{i-1,w_1}) \right) \\ &= \max_{w_1 \in PY_{i-1}} \left(P(W_{i-1,w_1}) \cdot P(w|w_1) \right) \end{aligned} \tag{7}$$

上式 $W_{i,w}$ 表示长度为 i 且结尾为 w 的句子。

5.1.3 字三元模型

对 (7) 式做推广，可以得到简单的三元字模型递推式：

$$\begin{aligned} P(W_{i,w}) &= max_{w_1,w_2}(P(W_{i-2,w_2}) \cdot P(w_1|W_{i-2,w_2}) \cdot P(w|W_{i-1,w_1})) \\ &= max_{w_1,w_2}(P(W_{i-2,w_2}) \cdot P(w_1|w_2) \cdot P(w|w_1,w_2)) \end{aligned} \tag{8}$$

这里为符合三元模型的定义，对于 $P(w_1|W_{i-2,w_2})$ 我们只考虑 w_1 的前一个字 w_2 而不考虑位置更前的汉字。也就是说在此式中需要同时计算二元模型和三元模型。

5.2 首字优化

倘若使用整个语料库来计算句字首字符，会导致一些出现次数较大的字占据了句子的首部，因此可以采用下面的方法来做一个平衡：

$$P(x_1) = \frac{C_F(x_1) + C(F_1) \cdot \frac{\sum_{x_0 \in PY_1} C_F(x_0)}{\sum_{x_0 \in PY_1} C(x_0)}}{\sum_{x_0 \in PY_1} C(x_0)} \tag{9}$$

其中 $C_F(x_1)$ 代表 x_1 在句首出现的总数。

5.3 平滑化

5.3.1 字二元平滑模型

倘若两个字未在语料库中同时出现过，那么假设他们同时出现的概率为 0 是不合理的，为此可以引入平滑化模型，即：

$$P(x_i|x_{i-1}) \approx \alpha \frac{C(x_i, x_{i-1})}{C(x_{i-1})} + (1-\alpha)P(x_{i-1}), \quad \alpha \in [0,1] \tag{10}$$

5.3.2 字三元平滑模型

由 (5) 和 (10) 式推广可得，字三元的平滑模型应如下式所示：

$$\begin{aligned} P(x_i|x_{i-1},x_{i-2}) &= \beta\frac{C(x_i,x_{i-1},x_{i-2})}{C(x_{i-1},x_{i-2})} + (1-\beta)\frac{P(x_i,x_{i-1})}{P(x_{i-1})}, \quad \alpha,\beta\in[0,1] \\ &= \beta\frac{C(x_i,x_{i-1},x_{i-2})}{C(x_{i-1},x_{i-2})} + (1-\beta)\left[\alpha\frac{C(x_i,x_{i-1})}{C(x_{i-1})} + (1-\alpha)P(x_{i-1})\right] \end{aligned} \tag{11}$$

6 效果

6.1 测试准确率

下面的测试结果是基于给定的测试样例给出的，其中平滑化参数的意义请见 (6), (7) 两式，字二、三元模型指两个模型同时计算得出结果的情况。

6.1.1 字准确率

模型	首字优化	平滑化参数	新浪新闻（基础）	新浪新闻＋维基百科
字二元模型	无	无	85.13	84.44
字二元模型	有	无	85.19	84.33
字二元模型	无	$\alpha = 1-10^{-6}$	85.13	83.95
字二元模型	有	$\alpha = 1-10^{-6}$	85.20	84.37
字二元模型	无	$\alpha = 1-10^{-7}$	85.15	84.30
字二元模型	有	$\alpha = 1-10^{-7}$	85.20	84.37
字二元模型	无	$\alpha = 1-10^{-8}$	85.15	84.45
字二元模型	有	$\alpha = 1-10^{-8}$	85.19	84.33
字三元模型	有	无	88.47	89.84
字二、三元模型	有	无	90.75	91.03
字二、三元模型	有	$\alpha = 1-10^{-7}, \beta = 1-10^{-5}$	90.80	**91.17**
字二、三元模型	有	$\alpha = 1-10^{-7}, \beta = 1-10^{-6}$	90.75	91.10
字二、三元模型	有	$\alpha = 1-10^{-7}, \beta = 1-10^{-7}$	90.77	91.11
字二、三元模型	有	$\alpha = 1-10^{-7}, \beta = 1-10^{-8}$	90.77	91.08

从上表可见，字二元模型的字准确率可以达到 85% 左右，字二、三元混合模型的字准确率则可以达到 90% 上下。对于一些简单的优化（如首字优化和平滑处理）则可以提升小量的效能。采用更多的数据集（新浪新闻、维基百科），在字二元模型的效果会降低，但在三元模型的表现却更好，这可能是因为维基百科的资料不太符合测试集的语境导致的。

6.1.2 句准确率

模型	首字优化	平滑化参数	新浪新闻（基础）	新浪新闻＋维基百科
字二元模型	无	无	39.40	36.60
字二元模型	有	无	39.80	36.80
字二元模型	无	$\alpha = 1 - 10^{-6}$	39.40	35.80
字二元模型	有	$\alpha = 1 - 10^{-6}$	40.00	36.80
字二元模型	无	$\alpha = 1 - 10^{-7}$	39.60	36.40
字二元模型	有	$\alpha = 1 - 10^{-7}$	40.00	36.80
字二元模型	无	$\alpha = 1 - 10^{-8}$	39.60	36.60
字二元模型	有	$\alpha = 1 - 10^{-8}$	39.80	36.80
字三元模型	有	无	51.20	53.80
字二、三元模型	有	无	58.00	57.80
字二、三元模型	有	$\alpha = 1 - 10^{-7}, \beta = 1 - 10^{-5}$	**58.20**	57.80
字二、三元模型	有	$\alpha = 1 - 10^{-7}, \beta = 1 - 10^{-6}$	**58.20**	57.80
字二、三元模型	有	$\alpha = 1 - 10^{-7}, \beta = 1 - 10^{-7}$	**58.20**	57.80
字二、三元模型	有	$\alpha = 1 - 10^{-7}, \beta = 1 - 10^{-8}$	**58.20**	57.80

结论和字二元模型类似。只使用新浪新闻的二元模型句准确率在 40% 左右，而采用新浪新闻＋维基百科的句准确率则只有 36% 左右。而字二、三元模型的表现在两个数据集上相差无几，大约是 58% 左右，这是由于几个字连续出现的组合情况较为单一，因此在 500MB 左右的数据集已经复盖了大部分的常用词，故二、三元模型在两个数据集上的表现相近。

6.2 例子

下面展示的是以字二、三元混合模型，含有首字优化，平滑化（参数为 $\alpha = 1 - 10^{-7}, \beta = 1 - 10^{-5}$）的例子。

6.2.1 好例子

拼音	预测句
zhi dao bo shi bi ye mi mang de shi hou	直到博士毕业迷茫的时候
ta yang le yi zhi qing wa dang chong wu	他养了一只青蛙当宠物
dou po cang qiong	斗破苍穹
wo cong wei jian guo you ru ci hou yan wu chi zhi ren	我从未见过有如此厚颜无耻之人
er ling yi jiu nian wei wu si yun dong yi bai zhou nian	二零一九年为五四运动一百周年

6.2.2 坏例子

预测句	原句
是你**再补资金**中陶醉在自然的怀抱	**使**你**在不自禁**中陶醉在自然的怀抱
祝你早安**武**安和**万**安	祝你早安**午**安和**晚**安
化化运动员**与**生结弦	**花滑**运动员**羽**生结弦
数据结构**优**点**一四**	数据结构**有**点**意思**
本次普查活动有助于帮助同学们走出心**里雾**区	本次普查活动有助于帮助同学们走出心**理误**区
毕竟老**佛爷**不是什么恶魔	毕竟老**夫也**不是什么恶魔
有些侦察**院**已经**触发**了	有些侦察**员**已经**出发**了
对不起我**市**警察	对不起我**是**警察
北京**市**首个举办过夏奥会于冬奥会的城市	北京**是**首个举办过夏奥会于冬奥会的城市

在第一句中，不自禁被翻译为补资金，其原因在于新闻中**补资金**在**补资**的基础上出现的次数比（538/575）较**不自禁**之于**不自**的比（415/2077）为大，因此在三元模型的限制下产生了错误的结果。同时首字的 **是** （2827538）/ **时**（338347）也是出于同样的原因才会预测错误。

由于在新闻报导中 **我市** 的出现概率显著比 **我是** 来得更大，因此在大部分情况下句子中的 **是** 会被识别为 **市** （如第七、八句），这与人们平时的习惯有所不同，因此会预测错误。

上面的句子都反映了三元模型缺乏对上下文判断的有效机制，以及语料库的偏好与日常句子之间的差距会导致翻译不准确的问题，因此，一种合理的解决方法是利用更好的技术（深度学习）或是更好的模型（词二元模型）来降低预测错误的机率。