实验报告

姓名: 方科晨
学号: 2021013400

实验环境

本实验在 `Linux` 操作系统， `Python 3.10.5` 版本环境下实现，使用到的包有 `os, json, tqdm, math, sys`

语料库及预处理方法

在 `process.py` 中对数据进行了预处理。代码输入为 `一二级汉字表.txt` `拼音汉字表.txt` 和 `sina_news_gbk` 目录下的语料。输出为 `1_word.txt`， `2_word.txt` 和 `3_word.txt` 。

代码首先读取 `一二级汉字表.txt` 获得所有需要处理的汉字，然后读取 `拼音汉字表.txt` 获得每个汉字可能的所有读音。

随后，代码读取 `sina_news_gbk` 目录下的所有语料文件，然后用字典记录每个读音对应的所有字的出现次数，以及两（三）个连续读音对应的所有两（三）个字对的出现次数。其中每个字会在它的所有读音中都被记录一次。

最后将这两个字典分别输出到文件 `1_word.txt`， `2_word.txt` 和 `3_word.txt` 中。

同时，在最后三元模型的使用中，为了加快数据读取的速度，使用 `simplify.py` 来提炼数据，对出现次数小于等于 10 的三元字组进行剔除，然后输出到 `3_simple_word.txt`

模型

基本思路、公式推导和实现过程

模型使用基于字的二元模型。令 O 为拼音串， S 为字串，则有 $P(S|O)$ 为在给定拼音串下字串为 S 的概率，且由概率论知识可得 $P(S|O)=\frac{P(S)P(O|S)}{P(O)}$ ，其中 $P(O)$ 固定，且在给定句子下，读音一般是唯一的，也即 $P(O|S)\approx 1$ 。而 $P(S)=\prod_{i=1}^{n}P(w_i|w_1\cdots w_{i-1})$ ，当我们使用二元模型，也就是每个字的概率只与相邻的字相关，与其他的字独立时，就有 $P(S)=\prod_{i=1}^{n}P(w_i|w_{i-1})$ 。最终，我们要求使 $P(S|O)$ 的 S ，也就变成了求

$\arg\max_S \prod_{i=1}^{n} P(w_i|w_{i-1}) = \arg\min_S -\sum_{i=1}^{n} \log(P(w_i|w_{i-1}))$

那么做法就非常显然了，直接使用动态规划，令 $f[i][j]$ 为当前到第 i 个字符，且最后一个字符为 j 时的最小 $-\log P$ 值的和，且用 $g[i][j]$ 记录下取到极值时第 $i-1$ 个字符的编号。那么转移式就为：

$$f[i][j] = \min_k(f[i-1][k] - \log P(c_j|c_k)) = \min_k(f[i-1][k] - \log \frac{P(c_j, c_k)}{P(c_k)})$$

其中 c_j 为编号为 j 的字符，根据预处理的文件 `1_word.txt` 和 `2_word.txt`， $P(c_j, c_k)$ 和 $P(c_j)$ 的比值通过查词典可以求得，即 $\frac{P(c_j,c_k)}{P(c_k)} \sim \frac{cnt(c_jc_k)}{cnt(c_k)}$，其中 $cnt(S)$ 表示串 S 在语料库中出现的次数。时间复杂度为 $O(nm^2 \log \Sigma)$，其中 n 为拼音串总长度， m 为一个拼音同音字的最大个数， Σ 为词典大小。

除此之外，为了避免某两个拼音同时出现的次数在语料中为 0，在式子中对条件概率进行平滑处理，即 $P(w_i|w_{i-1}) = \lambda P(w_i|w_{i-1}) + (1-\lambda)P(w_i)$，在代码中，我们取 $\lambda = 0.999999999$

实验效果

二元模型的代码实现在 `pinyin.py` 中。在给定的输入输出中，通过 `compare.py` 比较得到了句正确率为 84.0%，单字正确率为 39.7%

```
Number of lines that are the same: 199/501, 39.72055888223553%
Number of characters that are the same: 4400/5235, 84.04966571155683%
```

例子分析

Good Case

> 人民群众喜闻乐见
>
> 青山绿水就是金山银山
>
> 世界一流大学
>
> 消除恐惧的最好办法就是面对恐惧
>
> 清华大学是世界一流大学

这几个例子模型都能够正确识别，可以发现这些例子里基本上都是由两个字的组合组成的，且这

些两个字的组合一般很少有歧义，因此在二元模型下表现较好

Bad Case

> 北京*市*首个举办过夏奥会与冬奥会的城市
>
> 每*个*四年一次的冬奥会在今年召开了
>
> 学会处理人*机*关系
>
> 李老师喜欢海阔天空*的*畅谈
>
> 编写*着*中文章的人往往华中区重
>
> 该账号开通*津津*四十八小时细分二十九晚
>
> *故事后果*年假驾护和瓜红灯笼

其中前四个例子可以发现在错字连带着附近的一两个字看起来是挺正确的，但是从整句来看却是明显错误的，这是因为只用了二元模型，考虑字的选择时只估计了附近字的影响。

后三个错误则明显整句话都不是很有逻辑

对比参数

模型中有参数 λ，即 `lamb`，以下是选取不同的参数后得到的句正确率和单字正确率：

λ	句正确率	字正确率
0.9	0.8%	51.0%
0.999	3.0%	59.4%
0.99999	35.1%	82.8%
0.9999999	39.5%	84.0%
0.999999999	39.7%	84.0%

可以发现，在一定范围内，随着 λ 的增大，句正确率和字正确率都有所上升，看出当模型倾向于两个字的条件概率而非单字概率时正确率会增加。

时空复杂度分析

时间复杂度为 $O(nm^2 \log \Sigma)$，空间复杂度为 $O(\Sigma + nm)$，其中 n 为拼音串总长度， m 为

一个拼音同音字的最大个数， Σ 为词典大小。

计算次数约为 $2 * 5235 * 109 * \log_2 3578449 = 24845607.61683302$ ，实际运行时间为 $11.20s$

选做

三元模型

类似于之前提到的字的二元模型，我们同样可以设计关于字的三元模型，即两个距离小于等于二的字之间是相关的，则我们有如下的式子：
$P(S) = \prod_{i=1}^{n} P(w_i|w_1 \cdots w_{i-1}) = \prod_{i=1}^{n} P(w_i|w_{i-2}w_{i-1})$ 。最终我们要求的便是 $\arg\max_S \prod_{i=1}^{n} P(w_i|w_{i-2}w_{i-1}) = \arg\min_S -\sum_{i=1}^{n} \log(P(w_i|w_{i-2}w_{i-1}))$ 。相应的，我们同样可以设计一个动态规划算法：

$$f[i][k][l] = \min_j(f[i-1][j][k] - \log P(c_l|c_jc_k)) = \min_j(f[i-1][j][k] - \log \frac{P(c_j, c_k, c_l)}{P(c_j, c_k)})$$

同理的，我们为了避免概率退化为零，使用平滑的方法，将概率 $P(c_l|c_jc_k) = (1-\lambda-\gamma)P(c_l) + \lambda P(c_l|c_k) + \gamma P(c_l|c_jc_k)$ 。除此之外，为了求 $P(c_j, c_k, c_l)$ ，我们在数据预处理的时候也需要将字三元组的出现次数统计一下，得到 `3_word.txt` 。为了提高数据加载速度，我们将语料库中出现次数小于等于 10 的三元字组剔除，原本 $1.3G$ 的数据提炼到 $436M$ ，得到 `3_simple_word.txt` 。我们最终对使用两个预处理数据的结果都分别进行了统计分析。

最终，我们就有了时间复杂度为 $O(nm^3 \log \Sigma)$ ，空间复杂度为 $O(\Sigma + nm^2)$ ，其中 n 为拼音串总长度， m 为一个拼音同音字的最大个数， Σ 为词典大小。三元模型的代码位于 `pinyin3.py` 中。

通过 `compare.py` ，我们得到了如下的结果（其中参数选择为 $\gamma = 0.999999999, \lambda = 0.000000000999999$ ）

`3_simple_word.txt`

```
Number of lines that are the same: 296/501, 59.08183632734531%
Number of characters that are the same: 4702/5235, 89.81852913085004%
```

`3_word.txt`

```
Number of lines that are the same: 326/501, 65.06986027944112%
Number of characters that are the same: 4784/5235, 91.38490926456542%
```

二元模型与三元模型之间的结果比较如下表：

模型	句正确率	字正确率
二元模型 $\lambda = 0.999999999$	39.7%	84.0%
三元模型 $\gamma = 0.999999999, \lambda = 0.000000000999999$（使用提炼后的三元组信息）	59.0%	89.8%
三元模型 $\gamma = 0.999999999, \lambda = 0.000000000999999$（使用完整的三元组信息）	65.0%	91.3%

发现三元模型的效率较二元模型得到了较明显的改善。同时，我们发现提炼的三元组信息的确导致了准确率的下降。

另外的评价指标

为了评估二元模型的准确率，还可以使用任意相邻字对的准确率，即统计相邻字对都正确的字对数除以所有相邻字对的字对数；除此之外，如果能够借助分词工具，还可以按照词语的正确率来进行评估。