拼音输入法-实验报告

于越洋

1. 实验环境

本次实验在Macbook Pro M4 24GB上使用 `python` 语言完成，运行环境为 `python3.12.9`，需要安装 `tqdm` 库，运行方式参见代码包中的 `README.md`。

2. 语料库和数据预处理方法

本次实验仅使用实验要求中提供的**新浪新闻2016年的新闻语料库**进行训练。
数据预处理由 `preprocess.py` 完成，分为四步：

1. **获得可用与不可用字符集**。读取一二级汉字表，获得可用汉字集合 `allowed_characters_table`，配合自定义的常见标点符号、数字、字母集合，用于后续筛选可用词语。
2. **提取语料库语句**。读取存有语料库的输入文件，提取每条语料中 `title` 字段和 `html` 字段中的语句。注意到 `html` 字段中的语句大多以 `"原标题"` 开头，为避免偏向，此处将语句起始处的 `"原标题"` 去除。所有提取出的语句加入 `sentences` 列表中返回。
3. **对语句进行分词**。使用列表模拟队列，控制队列长度为所需要的词组长度，这样每个新的字加入队尾、旧的字弹出队首时就得到了一个新的n元词，每当遇到不可用字符时清空队列，以保证预处理后的数据中不会出现不可用字符，且不会出现本应被不可用符号分割的分词。
4. **统计词频并存储**。使用 `Counter` 统计分词列表中所有分词的词频，按照分词的字元数分别将统计结果以 `json` 形式输出至不同的文件中。
   数据处理由 `process.py` 完成，分为两步：
5. **读取词频统计结果**。按顺序读取输入文件夹中所有n元词的词频，进行累加后返回。
6. **计算概率并存储**。对于一元词，即单个汉字，计算其在所有汉字中的出现概率；对于n元词，计算前缀n-1元词出现时该词出现的概率。所有概率处理完成后按字元数存储至不同的 `json` 文件中。

3. 二元模型分析

3.1 基本思路

分析语料库，得到每个字的出现概率和每个二元词出现的条件概率，即对于每个汉字w，可以得到其在所有汉字中的出现概率$P(w)$，对于二元词w_1w_2，可以得到在其第一个字出现的情况下出现这个二元词的概率$P(w_1w_2|w_1)$。一个完整的句子的出现概率近似为

$$
\begin{aligned}
P(w_1w_2\dots w_n) &\approx P(w_{n-1}w_n|w_{n-1})P(w_{n-2}w_{n-1}|w_{n-2})\dots P(w_1w_2|w_1)P(w_1) \\
&= P(w_1)\prod_{i=2}^{n}P(w_{i-1}w_i|w_{i-1})
\end{aligned}
$$

为方便计算，两边取负对数，则上式变为

$$
-\log P(w_1w_2\dots w_n) \approx -\log P(w_1) - \sum_{i=2}^{n}\log P(w_{i-1}w_i|w_{i-1})
$$

取

$$
\begin{aligned}
&weight_{sentence} = -\log P(w_1w_2\dots w_n) \\
&weight_{w_1} = -\log P(w_1) \\
&weight_{w_i|w_{i-1}} = -\log P(w_{i-1}w_i|w_{i-1}),\ i = 2,3,\dots,n
\end{aligned}
$$

则

$$
weight_{sentence} \approx weight_{w_1} + \sum_{i=2}^{n} weight_{w_i|w_{i-1}}
$$

要求在给定拼音串下最可能出现的句子，即需要找到使得$P(w_1w_2\dots w_n)$最大的汉字串$w_1w_2\dots w_n$，也就是找到使得$weight_{sentence}$最小的汉字串$w_1w_2\dots w_n$，该问题可以使用动态规划算法求解。

3.2 算法原理

给定一个拼音串$p_1p_2\dots p_n$时，选取拼音汉字表中拼音p_i对应的所有汉字$w_{i,j}$，加入候选字集合`candidates`，并以如下方式进行处理权重：

- 对于p_1，将候选字的权重初始化为一元权重$weight_{w_{1i}}$；
- 对于$p_2 \sim p_n$，将候选字的权重初始化为`float('inf')`。
  从第二个拼音开始，遍历该拼音对应的所有候选字$w_{i,j}$，与前一个拼音所有的候选字$w_{i-1,k}$两两组合，计算组合权重$weight_{w_{i-1,k}w_{i,j}} = weight_{w_{i-1,k}} + weight_{w_{i,j}|w_{i-1,k}}$，然后选取$weight_{w_{i,j}} = min\{weight_{w_{i-1,k}w_{i,j}}|k = 1,2,\dots,m_{i-1}\}$作为该候选字的权重，并将该候选字的前继记为$w_{i-1,k}$，式中$m_{i-1}$表示前一个拼音的候选词数量。
  当最后一个拼音p_n的候选字遍历完成后，从p_n的候选字中选取权重最小者，沿记录的前继回溯至首个候选字，此时这n个候选字组成的句子就是在这个拼音串下出现概率近似最大的句子。

3.3 实验效果

实现二元模型生成算法的代码为`generate.py`，该代码在给定测试集合下生成500个句子，句准确率为38.7%，字准确率为85.0%，训练时间约为17秒，生成500个句子的总时间约为1.6秒。

3.3.1 好的例子

- `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
- `qing shan lv shui jiu shi jin shan yin shan`
  青山绿水就是金山银山
- `shi san jie quan guo ren da yi ci hui yi`
  十三届全国人大一次会议
- `zhong hua min zu de wei da fu xing`
  中华民族的伟大复兴

3.3.2 坏的例子

- `bei jing shi shou ge ju ban guo xia ao hui yu dong ao hui de cheng shi`
  北京**市**首个举办过夏奥会**于**冬奥会的城市
- `wo men yao ai xi shi jian jiu xiang ai xi zi ji de sheng ming yi yang`
  我们要爱惜时间就像爱惜自己的**声明**一样
- `qing cang da shuai mai`
  青**藏**大甩卖
- `de guo zong li mo ke er ri qian fa biao yan shuo`
  的国总理默克尔日前发表演说
- `yi cheng shi de ke xue tai du`
  一城市的科学态度

3.3.3 分析

从以上的例子中可以看出几个问题：

1. **语料库选择**。由于选择的是新浪新闻语料库，所以新闻中常用词汇的出现频率偏高，使得模型更倾向于输出这类词汇，导致偏向新闻性质的语句输出效果良好，好的例子大多属于这一类。其他类型的语句输出效果略差，比如坏的例子中的前两句，由于新闻语料中经常出现“北京市”“xx会于xx召开”“发表声明”等语句，导致模型偏向于这类输出。
2. **多音字处理**。模型并未考虑多音字问题，使得存在多音字时容易选择读音不合适但出现概率较高的候选字，比如坏的例子中的第三句，“青藏”是新闻语料中出现频率较高的组合，但其读音并不符合该句的拼音。
3. **首字选择错误率高**。由于首字的权重使用单字权重而非二元词的条件权重，如“的”“一”这种出现次数有绝对优势的单字，会拥有较大幅度小于其他单字的权重，其读音作为首个拼音出现时，模型更容易选取这些常见单字作为首字，导致首字错误，坏的例子中的第四、五两句就反映了这一问题。

4. **考虑范围过小**。仅使用二元模型，每个字的生成仅考虑前一个字，难以形成完整且连续的句子。

4. 思考题

4.a

`gbk`是中国国家标准，主要含有简体和繁体中文字符，使用双字节固定长度编码，主要用于中文系统；而`utf-8`是Unicode联盟开发的编码方式，几乎覆盖全球所有语言的字符，使用可变长度编码，广泛用于各种系统。
存在其他编码方式，比如`ASCII`编码，用于纯英文文本和早期计算机系统；`GB2312`编码，用于早期中文系统和简体中文书籍排版；`utf-16`编码，用于Java字符串、Windows内部字符处理；`utf-32`编码，用于需要随机访问字符的应用等。

4.b

假设读音数量为C，则平均每个读音对应的字数约为$\frac{V}{C}$，那么`viterbi`算法存储所有的候选节点需要$O\left(n\frac{V}{C}\right)=O(nV)$的空间。算法初始化第一列的候选字时间复杂度为$O\left(\frac{V}{C}\right)=O(V)$，从第二列开始，获取该列候选字的时间复杂度为$O\left(\frac{V}{C}\right)=O(V)$，每个候选字都需要和前一列的所有候选字两两组合，有$(\frac{V}{C})^2$种组合结果，那么第二列到最后一列总共需要遍历$(\frac{V}{C})^2(n-1)$种可能的组合，因此总的时间复杂度为$O\left(nV+\left(\frac{V}{C}\right)^2(n-1)\right)=O(nV+nV^2)$，即$O(nV^2)$。

4.c

区别：代码A在读取输入时立即打印输出值，每次循环都会立即输出输入的值；而代码B先将所有输入存在一个列表中，在循环结束后按顺序打印，会在最后一次性输出所有输入的值。
正误：代码A是即时处理输入的方式，符合“原封不动输出”的要求，是正确的；代码B是收集所有输入后一次性输出的方式，也符合“原封不动输出"的要求，也是正确的。

5. 多元模型分析

5.1 基本思路

引入三元模型和四元模型以增加模型生成时考虑的范围，对句子出现概率的计算更改为

$$P(w_1w_2\dots w_n)\approx P(w_1)P(w_1w_2|w_1)P(w_1w_2w_3|w_1w_2)\prod_{i=4}^{n}P(w_{i-3}w_{i-2}w_{i-1}w_i|w_{i-3}w_{i-2}w_{i-1})\ (*)$$

取权重为

$$
\begin{aligned}
&weight_{sentence} = -\log P(w_1 w_2 \dots w_n) \\
&weight_{w_i} = -\log P(w_i),\ i = 1, 2, \dots, n \\
&weight_{w_{i-1}w_i|w_{i-1}} = -\log P(w_{i-1}w_i|w_{i-1}),\ i = 2, 3, \dots, n \\
&weight_{w_{i-2}w_{i-1}w_i|w_{i-2}w_{i-1}} = -\log P(w_{i-2}w_{i-1}w_i|w_{i-2}w_{i-1}),\ i = 3, 4, \dots, n \\
&weight_{w_{i-3}w_{i-2}w_{i-1}w_i|w_{i-3}w_{i-2}w_{i-1}} = -\log P(w_{i-3}w_{i-2}w_{i-1}w_i|w_{i-3}w_{i-2}w_{i-1}),\ i = 4, 5, \dots, n
\end{aligned}
$$

则对$(*)$式两边取负对数可得

$$
weight_{sentence} \approx weight_{w_1} + weight_{w_1w_2|w_1} + weight_{w_1w_2w_3|w_1w_2} + \sum_{i=4}^{n} weight_{w_{i-3}w_{i-2}w_{i-1}w_i|w_{i-3}w_{i-2}w_{i-1}}
$$

仍使用动态规划算法求解，寻找使得$weight_{sentence}$最小的路径。

5.2 算法实现

5.2.1 预处理过程

添加对三元词中前两个字出现后第三个字出现的概率$P(w_{i-2}w_{i-1}w_i|w_{i-2}w_{i-1})$和四元词中前三个字出现后第四个字出现的概率$P(w_{i-3}w_{i-2}w_{i-1}w_i|w_{i-3}w_{i-2}w_{i-1})$的计算。

5.2.2 生成过程

保留使用二元模型时的权重初始化方式。
对于第二个拼音，仍使用二元模型进行处理。
对于第三个拼音，遍历该拼音对应的列中的候选字$w_{i,j}$和前一列的候选字$w_{i-1,k}$，两两组合，计算组合权重$weight_{w_{i-2,l}w_{i-1,k}w_{i,j}} = weight_{w_{i-1,k}} + weight_{w_{i-2,l}w_{i-1,k}w_{i,j}|w_{i-2,l}w_{i-1,k}}$，式中$w_{i-2,l}$取为$w_{i-1,k}$记录的前继节点，选取$weight_{w_{i,j}} = min\{weight_{w_{i-2,l}w_{i-1,k}w_{i,j}}|k = 1, 2, \dots, m_{i-1}\}$，并将$w_{i,j}$的前继记为$w_{i-1,k}$，式中$m_{i-1}$表示前一列的候选字数量。
从第四个拼音开始，仍遍历该拼音对应的列中的候选字$w_{i,j}$和前一列的候选字$w_{i-1,k}$，两两组合，计算组合权重$weight_{w_{i-3,q}w_{i-2,l}w_{i-1,k}w_{i,j}} = weight_{w_{i-1,k}} + weight_{w_{i-3,q}w_{i-2,l}w_{i-1,k}w_{i,j}|w_{i-3,q}w_{i-2,l}w_{i-1,k}}$，式中$w_{i-2,l}$取为$w_{i-1,k}$记录的前继节点，$w_{i-3,q}$取为$w_{i-2,l}$的前继节点，选取$weight_{w_{i,j}} = min\{weight_{w_{i-3,q}w_{i-2,l}w_{i-1,k}w_{i,j}}|k = 1, 2, \dots, m_{i-1}\}$，并将$w_{i,j}$的前继记为$w_{i-1,k}$，式中$m_{i-1}$表示前一列的候选字数量。
当最后一个拼音p_n的候选字遍历完成后，从p_n的候选字中选取权重最小者，沿记录的前继回溯至首个候选字，此时这n个候选字组成的句子就是在这个拼音串下出现概率近似最大的句子。

5.2.3 特殊处理

- 多元模型的生成过程中可能遇到多元词不在权重列表中的情况，此时采用退化处理，即当四元词不存在时，退化为使用三元词的条件权重处理；当三元词不存在时，退化为使用二元词的条件权重处理；当二元词不存在时，使用单字权重处理。
- 考虑到输出准确率与模型元数存在一定的正相关关系，因此在计算时为n元模型的条件权重乘以权重系数$weight_n$，如二元模型的权重计算$weight_{w_{i-1,k}w_{i,j}} = weight_{w_{i-1,k}} + weight_{w_{i,j}|w_{i-1,k}}$调

整为$weight_{w_{i-1,k}w_{i,j}} = weight_{w_{i-1,k}} + weight_{w_{i,j}|w_{i-1,k}} \times weight_2$，模型元数越多权重系数越小，使其能够在汉字的选择中发挥更大的作用。

5.3 效果分析

5.3.1 好的例子

- `wo men yao ai xi shi jian jiu xiang ai xi zi ji de sheng ming yi yang`
  我们要爱惜时间就像爱惜自己的生命一样
- `qing cang da shuai mai`
  清仓大甩卖
- `ta han you de dan bai zhi bi niu rou he yi ban yu lei hai yao gao`
  它含有的蛋白质比牛肉和一般鱼类还要高

5.3.2 坏的例子

- `bei jing shi shou ge ju ban guo xia ao hui yu dong ao hui de cheng shi`
  北京**市**首个举办过夏奥会与冬奥会的城市
- `de guo zong li mo ke er ri qian fa biao yan shuo`
  的国总理默克尔日前发表演说
- `zi qiang bu xi hou de zai wu`
  自强不息**后的再无**

5.3.3 分析

从以上例子中可以看出，在引入了三元模型和四元模型后，模型考虑的范围更远，输出的语句连贯性更强，正确率有所改善，这在好的例子中的第一、二句和坏的例子中的第一句中都有所体现。
但是，多元模型也难以解决部分词语出现概率大幅占优的问题，比如坏的例子中的第一句仍输出为“北京市”而不是“北京是”；且由于首字仍采用一元模型进行输出，在处理”的“这种出现频率极高的单字时仍难以避免与其组合的词汇权重大幅小于其他组合的问题，犯错率仍然较高，比如坏的例子中的第二句仍存在这一问题；此外，由于只采用了不超过四元的模型，当面对较长词汇时就难以和前面的内容产生联系，导致部分词汇被曲解而整体出错，比如坏的例子中的第三句。

5.4 不同模型的对比

模型	二元模型	多元模型
句准确率	38.7%	52.3%
字准确率	85.0%	88.5%
训练时间	$16.77s$	$74.29s$
生成时间	$1.59s$	$2.85s$

由上表可以看出，相比二元模型，多元模型训练和生成所花的时间更长，达到了二元模型的几倍，这是因为三元词和四元词的数量远多于二元词，因此需要更多的时间来处理。多元模型的生成准确率较二元模型也有明显的增长，尤其是句准确率，比二元模型高出16%，这是因为多元模型能更好地“理解”一个句子中前后词汇间的相关性，从而更容易生成出正确的句子。

6. 调参尝试

6.1 二元模型调参

主要针对二元模型中退化情况（二元条件权重不存在，使用一元权重）的惩罚值进行调参，得到如下表所示的结果

惩罚值	0	2	4	5	10	15
句准确率	32.5%	38.7%	38.3%	37.5%	36.3%	36.3%
字准确率	82.6%	85.0%	85.0%	84.6%	84.3%	84.3%

可见，当惩罚值不断增大时，生成准确率先升后降，最后趋于稳定。分析原因为：惩罚值较小时，退化情况下一元权重有可能起到比其他二元权重更大的作用，导致模型更倾向于走向一元模型的选择，降低准确率；而当惩罚值较大时，一元权重与惩罚值之和远大于二元权重，使一元模型无法发挥作用，生成准确率趋向于二元模型本身的准确率。因此选定合适的惩罚值，有利于提高模型的生成效果。

6.2 多元模型调参

主要针对多元模型中的束宽度（每个拼音保留的最优候选字数量）进行调参，得到如下表所示的结果

束宽度	4	6	8	16	32	64
句准确率	41.5%	45.9%	49.7%	51.9%	52.1%	52.3%
字准确率	84.3%	86.4%	87.5%	88.3%	88.5%	88.5%
生成时间	$0.50s$	$0.81s$	$0.96s$	$1.50s$	$2.26s$	$2.85s$

从表中可以看出，当束宽度增大时，生成准确率不断提高，最后趋于稳定，生成时间不断增长。分析原因为：束宽度越大，意味着viterbi算法中需要枚举的候选字组合越多，生成时间自然会更长；更大的束宽度也使得每列中都能保留更多的次优解，能够更好地应对局部次优解在后续过程中变为最优解的情况（否则该最优解有可能在中途就被舍去），因此生成准确率有所提高。当束宽度增加到超过大部分拼音的候选字数量时，再增大束宽度不会对算法过程产生明显影响，于是生成准确率趋于稳定。总之，在计算资源足够的情况下，可以考虑选择更大的束宽度进行生成。

7. 纠错

经测试，实验文档中给出的评测脚本中命令 `cp -r sina_news_gbk/ corpus/` 会将 `sina_news_gbk` 中的所有文件粘贴至 `corpus/` 目录下，而 `cp -r sina_news_gbk corpus/` 才是将 `sina_news_gbk` 文件夹粘贴至 `corpus/` 目录下。

8. 后记

完成本实验的代码部分总共花费约20小时，报告部分花费约5小时。完成基本的算法并未花费很长时间，更多的时间花在调整参数、尝试不同的策略以提高生成准确率。目前所做的尝试有：平滑处理概率、加入惩罚机制、为不同模型采用不同的概率权重、改进viterbi算法中权重计算方式、采用不同的候选字组合方式、采用束搜索等，最终保留了这一版使用模型概率权重和束搜索（但宽度值较大并未发挥作用）的代码，使得句准确率提高到52.3%，个人感觉较为满意。通过不同的尝试，我逐渐理解了不同的处理方式在模型效果中发挥的作用，收获颇丰。