拼音输入法实验报告

2020210931 汪绪恒

概况

这个文件夹包含了我实现的拼音输入法作业，其中包含五个目录。

- bin/ 目录下包含了项目用到的可执行文件，主要为 `pinyin` 和 `evaluate`，分别为项目的主程序和评估程序；
- data/ 目录下包含了项目中用到的输入输出文件；
- model/ 目录下包含了项目预先训练好的模型文件；
- src/ 目录包含了项目的源代码，其中 `main.cpp` 为主程序文件，`model.cpp` 为模型建立文件，`predict.cpp` 为预测文件；
- test/ 目录下是一些开发过程中用到的测试文件。

为了运行本项目，您需要使用 Linux 系统，并且在项目的主文件夹下（README.md 所在的文件夹），运行以下命令

```
bin/pinyin data/test_input.txt data/test_output.txt
```

该命令会默认使用**三元模型**作为拼音输入法的模型，对于一千条测试语句，该命令大约需要执行两分钟。同时，本项目也支持使用二元模型，您可以通过显式指定

```
bin/pinyin -n2 data/test_input.txt data/test_output.txt
```

来完成这一点。显然，也可以使用 `-n3` 来显式指定使用三元模型。

运行完成后，转化的结果被存储在了 `data/test_output.txt` 文件中。本项目还提供了 `evaluate` 命令，来评估转化结果的好坏

```
bin/evaluate data/test_output.txt data/test_ans.output
```

该命令会打印转化的字准确率与行准确率。

二元模型的实现

拼音输入法的基本原理是，给定输入法拼音序列 O，计算最有可能的汉字序列，即计算序列 S，使得

$$P(S|O) = \frac{P(S)P(O|S)}{P(O)}$$

最大。对于给定的拼音序列，$P(O)$ 乃是一个常量，而 $P(O|S)$ 用识别信度代替，可以用常数 1 来近似它。因此，问题转化成了求这样一个汉字序列 S，使得 $P(S)$ 最大。不妨将 S 写作 $S = w_1w_2...w_n$，其中 w_i 表示序列中第 i 个汉字，这样

$$P(S) = P(w_1w_2...w_n) = P(w_1)P(w_2|w_1)...P(w_n|w_1w_2...w_{n-1})$$

在二元模型中，认为每个汉字的出现仅与它前面一个汉字有关，因此上式可以简化为

$$P(S) = P(w_1)P(w_2|w_1)P(w_3|w_2)...P(w_n|w_{n-1})$$

为了求上式的最大值，等价于求做了负对数以后的最小值，即

$$min\{-\Sigma_{i=1}^{n} log(P(w_i|w_{i-1}))\}$$

因此，可以预先对语料库进行处理，统计每个汉字单独出现的频率，以及两个汉字同时出现的频率。这样，就可以计算出二元后验概率

$$P(w_i|w_{i-1}) = \frac{P(w_{i-1}w_i)}{P(w_{i-1})}$$

随后，就可以通过 Viterbi 算法，在多项式时间内求解上述最值了。然而需要注意的是，在对新的拼音进行预测时，有可能 w_{i-1} 与 w_i 没有同时出现过，即 $P(w_i|w_{i-1}) = 0$。为了解决这个问题，可以对该后验概率进行平滑，例如引入系数 λ，用

$$\lambda P(w_i|w_{i-1}) + (1-\lambda)P(w_i)$$

来代替 $P(w_i|w_{i-1})$。整个项目的整体实现步骤如下：

- 利用 `src/model.cpp` 生成模型文件。`src/model.cpp` 首先读入拼音汉字表，建立拼音到汉字的映射；之后读入了九个语料文件，分别计算每个词出现的词频与两个连续汉字出现的词频，并将结果写入了 `model/prior.txt` 与 `model/posterior.txt` 文件中；
- 利用 `src/predict.cpp` 对输入的拼音序列进行预测。在 `src/predict.cpp` 中首先读入两个模型文件 `model/prior.txt` 与 `model/posterior.txt`，建立单个汉字以及两个汉字到词频的映射，并且实现了用于二元模型的 Viterbi 算法，输出最有可能与给定拼音序列对应的汉字序列。

参数选择

在二元模型中引入了参数 λ，用于对后验概率 $P(w_i|w_{i-1})$ 进行平滑，该参数反映了后验概率与先验概率之间权重的比例。我对 λ 的值进行调整，进行了一系列实验，不同的 λ 对应的识别准确率如下表所示：

λ	word acc	line acc
1.0	54.21%	1.74%
0.999	86.58%	43.00%
0.99	86.80%	43.47%
0.95	86.89%	43.67%
0.90	86.94%	43.53%
0.85	86.75%	42.60%

从中可以看出，当 $\lambda = 0.90$ 时，模型具有最高的词识别率；而当 $\lambda = 0.95$ 时，模型具有最高的句识别率，此时词识别率也并未有较为显著的衰减。因此最终是选择了 $\lambda = 0.95$ 作为二元模型中用到的参数。

实验结果分析

我从网络上获得了一些语句作为测试集。以下首先展示一些好的转化结果与差的转化结果，随后对产生该结果的原因进行分析。

Good Case

- zhong guo you zhe hen feng fu de zi ran zi yuan

  中国有着很丰富的自然资源

- liang hao de xue xi huan jing shi xue sheng men qu de hao cheng ji de bao zheng

  良好的学习环境是学生们取得好成绩的保证

- jin tian bang wan you wai bin che dui cong zhe tiao lu tong guo

  今天傍晚有外宾车队从这条路通过

- wo men ying gai xue hui yun yong xian jin ji shu dui chuan tong zhi zao ye jin xing gai zao

  我们应该学会运用先进技术对传统制造业进行改造

Bad Case

- wo xin li you zhong shuo bu chu lai de wei dao

  我心理由中说不出来的味道（✕）

  我心里有种说不出来的味道（✓）

- zhong guo de shao shu min zu dou shi neng ge shan wu de min zu

  中国的少数民族都是能歌善恶的民族（✕）

  中国的少数民族都是能歌善舞的民族（✓）

- wo men xue xiao shi xing xue fen zhi

  我们学校实行学分之（✕）

  我们学校实行学分制（✓）

- ni ying gai ba zhei xie gong zuo xiang mi shu jiao dai qing chu

  你应该把这些工作项秘书角大清楚（✕）

  你应该把这些工作向秘书交代清楚（✓）

结果分析

从四个好的例子可以看出，本项目对于较为正式、偏书面语的句子具有良好的效果，这可能是由于项目使用的语料库都是从新闻中获得，其中的用词比较正式和规范。这显然会导致一些潜在的问题，当被转换的句子偏向于非正式的场合时，转换的效果会比较差。差例一就反映了这种情况——相对于「心里有」这样较为口语的短语，二元模型更倾向于选择更为正式的组合「心理」、「理由」。同时，该例也展示了二元模型的一个固有问题——目光短浅，只能看到前一个汉字——如果可以同时看到前两个汉字，「心里有」这样的组合显然是优于「心理由」的，可能就能纠正这一差错。

第二、三个差例也同样展示了「目光短浅」的问题。「能歌善舞」作为一个成语，其出现频次是高于「能歌善恶」的，然而二元模型只能看到前一个汉字，因此相对于「善舞」，本模型更倾向于二元频次更高（也更为正式的）「善恶」。

差例二与差例四同时展示了「多音字」的问题。在差例二中，「善恶」这个搭配里「恶」是读「è」而非「wù」，但本模型对于多音字的不同读音只是简单地将词频进行累加，而没有加以区分。差例四也是同样的问题，「大」显然读「dài」的频次是明显低于「dà」的，但是两者的词频是累加的，因此模型更倾向于选择词频更高的「大」字。

二元模型改进

针对上面提出的问题，本项目从模型层面提出了两种解决方案，即分别是考虑多音字与使用三元模型，将在下面进行详细阐述。

考虑多音字

在原始的二元模型中，是直接使用常数 1 来代替识别信度 $P(O|S)$，正是这样的近似导致了上述分析的多音字的问题。对于任意一个多音字而言，不妨设其为 w，它对应的拼音为 $o_1, o_2, ..., o_m$，即具有 m 个不同的读音。则每一个读音的识别信度

$$P(o_i|w) = \frac{I(w, o_i)}{\Sigma_{j=1}^{m} I(w, o_j)}$$

其中 $I(w, o_i)$ 表示多音字 w 读 o_i 出现的频次。多音字的改进版本就将考虑上述识别信度，而不是简单地用常数 1 来代替。

为了获得语料库的拼音信息，我是使用了 python 的 pypinyin 库对原料库进行注音。而为了提高注音的准确率，首先是使用了另一个 python 库 jieba 分词，对语料库的句子进行划分。注音完毕之后只需要统计各个汉字读每个读音的频次就可以了。这部分工作在 `src/polyphone.py` 中实现，它读取了全部的九个语料库库，并将最终的多音字识别信度信息被存储在了 `model/poly_mapping.txt` 当中，想要运行 `src/polyphone.py`，只需要在根目录下执行

```
python src/polyphone.py
```

即可，但是我没有将语料文件放在项目文件夹中，您可以自行将语料库放置在 `data/train` 目录下。python2 版本用户需要显式指定使用 `python3`。**注意，该命令在无其他进程时，需要在单个 Intel I7 处理器上运行大约三个小时。**这是由于分词和注音需要较大的计算开销。

使用三元模型

为了在一定程度上缓解二元模型「目光短浅」的问题，考虑使用三元模型对其进行改进。使用三元模型的基本思路是简单的，无非在此前的基础上，再增加对连续出现的三元组频次进行统计。然而在理论上，对于大约七千个汉字而言，大约存在 $7000^3 = 343 \times 10^9$ 个三元组组合，如果每个三元组使用四个字节存储其频次信息，则大约需要 $1000G$ 的内存空间，这显然是不现实的。然而在实际应用中，上述的三元组是极其稀疏的，存在大量从未出现的三元组。因此，在本项目中是使用了 `c++ str` 中的 `unordered_map` 作为三元组的底层数据结构，`map` 的 `key` 是三元组，而 `map` 的 `value` 是该三元组出现的频次。`unordered_map` 底层使用 `hash table` 实现，因此具有较高的存取效率。

构建三元组模型的工作在 `src/model.cpp` 中实现，可以直接在根目录下执行

```
bin/model
```

运行可执行文件，需要注意的是我没有把语料文件放在项目文件夹中，因为实在是太大了，您可以自行将语料文件放置在 `data/train` 目录下。该命令大约需要执行二十分钟，构建的三元模型

被存放在 `model/posterior3.txt` 当中，该文件大约有 300MB 大小，可见相对于此前分析的 1000G 要小了不少。由于该文件还是太大了，本项目中没有包含该文件。

获得了三元模型之后，还需要对预测期间的 Viterbi 算法进行修改。在二元模型的 Viterbi 算法中，每一个汉字只需要考虑前一个层次的所有汉字，因此其时间复杂度为 $O(KL^2)$，其中 K 表示输入拼音序列的长度，L 表示每一个层次的节点数量。在三元模型的 Viterbi 算法中，对于当前的节点，是需要计算从此前两个层次到它的最优路径，故时间复杂度为 $O(KL^3)$，这也使得三元模型在预测阶段的效率更低。

与二元模型类似，在三元模型中仍然需要对三元后验概率进行平滑，这里采用的策略也是类似的——引入参数 w，作为三元概率与二元概率之间的权重系数，即使用

$$wP(w_i|w_{i-1}w_{i-2}) + (1-w)P(w_i|w_{i-1})$$

来代替原始的 $P(w_i|w_{i-1}w_{i-2})$，这里 $P(w_i|w_{i-1})$ 仍然用 $\lambda P(w_i|w_{i-1}) + (1-\lambda)P(w_i)$ 来代替。因此最终用到的三元概率为

$$wP(w_i|w_{i-1}w_{i-2}) + (1-w)\lambda P(w_i|w_{i-1}) + (1-w)(1-\lambda)P(w_i)$$

我固定 $\lambda = 0.95$，通过调整 w 对其进行参数选择，其结果如下表所示：

w	word acc	line acc
0.90	91.13%	57.5%
0.95	91.34%	58.7%
0.99	91.82%	59.9%
0.999	91.75%	59.4%

可见，在 $w = 0.99$ 时词准确率与句准确率都达到了最优值，因此选择 $w = 0.99$。

尽管得到的三元模型大小相对于理论分析的 1000G 已经小了不少，但是对于程序在运行期间读入一个 300MB 的文件仍然是一个不小的开销。因此考虑对三元模型进行压缩，将其中出现频次最低的一些项移除掉。经过实验分析，该压缩操作对三元模型的预测准确率并未产生显著影响，当压缩量适当时反而还会提高预测准确率，这部分工作在 模型评估 部分进行了详细的阐述。

模型评估

至此，我已经获得了若干不同的模型，以下对它们的性能进行评估，评估的结果见下表

model	word acc	line acc
二元模型	86.89%	43.67%
二元模型 + 多音字	87.78%	44.70%
三元模型 + 多音字	89.71%	50.80%
zip1 + 多音字	90.71%	55.70%
zip2 + 多音字	91.82%	59.90%
zip5 + 多音字	90.79%	56.70%

其中，`zip1` 表示移除出现频次为 1 的三元组后得到的压缩的三元模型，`zip2` 与 `zip5` 同理。从上表中可以看出，添加了多音字支持后二元模型的预测准确率获得了一个明显的提升，而三元模

型相对于二元模型预测准确率也提升显著。这种性能提升是意料之中的，说明此前对于二元模型性能瓶颈的分析是正确的。

意料之外的是，使用了压缩的三元模型后，模型的预测准确率反而有了较大的飞跃，尤其是句准确率，有了接近五个百分点的提升。对该现象的解释是，在原始的三元模型中，出现频率较低的三元组之间并不具有显著的关联关系，只是所谓的噪声，将这部分噪声过滤掉后，模型反而可以更好地利用三元信息了。

容易注意到，当压缩量太大时，例如 `zip5`，三元模型的性能又有所下降，这是意料之中的，因为此时将有用的三元信息也过滤掉了。因此，最终的三元模型是使用了 `zip2` 的版本，它将所有出现频次不超过 2 的三元组过滤掉，最终的模型文件存储在 `model/posterior3zip2.txt` 中，其大小仅为 114MB，相对于原始的三元模型又小了不少。

总结与展望

本项目首先是实现了基于二元模型的拼音输入法，在分析其缺陷的基础上对其作出了改进，主要包括引入了多音字的识别和使用三元模型。最终针对三元模型模型文件较大，预测阶段时间与空间消耗严重的问题，提出对三元模型进行压缩，竟然意外地获得了准确率上的提升，本项目也对该现象的原因也进行了分析。

通过本项目，我的收获如下：

- 拼音输入法对于模型的复杂度较为敏感，采用更加复杂的模型可以显著获得更好的预测准确率，然而如何在准确率与时空开销之间进行权衡是一个关键的问题；
- 完整地走完了项目开发与优化的全过程，对于实际项目开发的流程有了更加深入的理解；
- 在实现过程也遇到了很多实际问题，例如 pypinyin 库进行了注音的字却不是标准汉字；三元模型内存占用量过大等，在解决这些实际问题的过程磨练了自己的编程能力。

为了进一步对模型进行优化，也可以考虑

- 使用更加复杂的模型，例如四元模型。但如何处理此时的时空开销也是一个非常重要的问题；
- 对句子的开头和结尾进行单独识别。某些汉字用在句首句尾的频率明显高于其他汉字，基于这一特点，可以考虑在句首句尾添加一个占位符，用于标记各个汉字出现在句首句尾的频率；
- 对时空性能进行优化。例如改进 Viterbi 算法，每次更新时只是选择前 N 个较好的路径，从而避免三元模型中 $O(KL^3)$ 的复杂度；
- 使用更加全面的语料库。在本项目中只是使用了新浪新闻的语料库，此前也提到，该语料库用于较为正式，对于非正式的语句预测效果较差。因此可以考虑使用更加口语化的语料库，从而使得模型在多个场景下具有更好的鲁棒性。