拼音输入法实验报告

计 81 严韫洲 2018011299

这份实验报告中我们将主要讨论拼音输入法的实现细节，其中会包括具体的使用方式，特殊的考虑，测试结果以及不同模型的比较等等。

github 地址：https://github.com/Billyyanyz/Pinyin

1 使用方式

我们提供了五种训练模型的模式和两种运行模式，用于在不同的场合中的不同功能。

1.1 训练模式

整个模型训练过程包括四个步骤：确定汉字范围，预处理原始语料，统计处理后语料的词频信息和翻译统计数据为概率矩阵。我们将它们包装成为了若干训练模式来适应不同的需求。

执行命令 `python -m train` 即可了解全部五种用法。他们是：

- `run_all`：完全重新训练整个模型
- `process`：保持汉字库不变，重新训练模型
- `add_new`：完全保持原有模型的情况下，加入新的原始语料训练模型（注意需要更改 `mat_address_list.txt`：为新的原始语料地址，否则原有的语料会重新统计一次）
- `reanalyze`：重新统计已处理过的语料（用于更改 `accept_triple_character_bound` 常数后）
- `translate`：重新将统计数据翻译为概率转移矩阵（用于更改 `zero_possibility_bound` 常数后）

1.2 运行模式

我们提供了两种使用模型的方式：

- 运行 `pinyin.py` 可以接收 `input.txt` 中的拼音输入，在 `output.txt` 中输出对应汉字
- 运行 `tester.py` 可以接收 `test.txt` 中的拼音-句子对组成的测试集，在 `standard.txt` 和 `answer.txt` 中分别输出标准答案和实际答案，然后在 `result.txt` 中输出准确率数据。

2 实现细节

在这一节中我们将具体阐释实现过程中的若干细节。

2.1 语料选取

由于我们习得的模型中的数据完全由我们选取的语料决定，语料的选取也因此至关重要。

我没有选择编写爬虫软件来从网络上爬取语料。我认为利用这种方式获取多个领域的广泛的数据需要巨量的数据，而涉猎广泛这一要求对于爬虫也并不是一件容易的事情。

考虑到输入法的首要考虑应该是使用方便，我们编写的输入法应该首先符合我们自身的输入习惯。我从这一点出发，最后找到了既易于获取又效果较好的语料集：**QQ 聊天记录**。

我的语料库最终是由题目提供的九个新浪新闻文本文件和包括我在内一共四位 QQ 好友的聊天记录组成的。（感谢小火，洛凌波和 billfan 三位同学慷慨地分享出自己的聊天记录，出于隐私和文件大小考虑我们的仓库中不会包括全部语料，只选取一则新浪新闻作为示例）

2.2 语料预处理

我们在语料预处理环节进行了多方面的考虑。

首先，我们获得的原始语料都是比较繁杂的，其中不乏有网页上或聊天记录中格式化的内容。例如，新浪新闻中大量重复出现“原标题”字样，或是聊天记录中由于作为文字素材导出而无法显示的“表情”，“图片”字样。因此，我们除了实现默认版的语料处理之外，还为两种类型的语料分别编写了独特的**筛选系统**，来排除反复出现又非正常语段的内容：

具体实现中，我们将所有需要处理的语料的地址存储在 `mat_address_list.txt` 中，其中地址的格式形如：`/0.txt|1|GBK`，第一项表示 `original_material` 目录下的文件名，第二项表示语料类型（0 表示默认，1 表示新浪新闻，2 表示聊天记录，我们建议针对每种语料都编写独特的筛选程序段来得到最佳效果），第三项表示文件编码格式。

其次，考虑到多音字对结果的影响，我们调用 pypinyin 库对筛选后的语料均进行**注音**。注意到该库在实际应用中有相当多的错误，我们为了增强程序的**鲁棒性**，我们进行额外的处理，将错误的注音再修改为词库内的默认注音。通过这一系列操作我们已经可以忽略多音字问题，因为一个字的不同读音已经被作为不同的汉字进行考虑了。（除非 pypinyin 库注音错误的同时我们按后到原则选取的默认注音也是错误的，针对这一问题只有在实际遇到时特判改错予以解决）

最后，我们充分考虑句首和句尾这两个因素，在处理语料时进行分句，在句首和句尾分别添加`#`和`$`作为**标识符**。我们在此处做了更为精细的处理：句中经常会出现数字，字母等，他们很可能是单词的一部分（例如 A 股、4 月 1 日、13 个、等等），因此我们不能一遇到非汉字就认定为句末。我们在实现时，判断如果当前字符**非字母非数字**，才在缓冲区中加上`$`输出为句子，然后加上`#`标识为句首。

2.3 字频统计

字频统计时我们先进行单字和双字的统计，其中为了节省空间，在进行双字的统计时，我们没有预先将所有字都加入字典中，而是利用 `Defaultdict` 和 `Counter` 进行**即时**的词条添加。

经测算，未使用这两者时，二元字频文件大小达到了 500M 左右，使用这项技术后大小减小到了 45M 左右。

而在三字统计时，我们没有将所有的三字全部纳入其中。我们在进行双字统计时，设置了接受三元字频统计的**阈值** accept_triple_character_bound，只有当二元字频达到了这一阈值后，我们才将它纳入三元字频的词条内。

我们设置阈值为 10，最后得到的三元字频文件大小控制在了 420M 左右，得到的效果也令人满意。

2.4 概率计算

在将统计数据转化为频率时，我们继续忽略未出现过的二元（三元）字，这一问题会在实际计算时进行解决。不过在单字上，我们是将所有字全部设置为词条的，因此需要在此处就予以解决。不过解决方式是统一的。

我们频率的计算方式和 ppt 上描述的方式是一致的，三元模型完全类似，这里不作赘述。而为了计算的便利性我们对概率均取自然对数进行存储和操作，这会导致部分概率为 0 的字频产生数学错误，我们的解决方式是设置 zero_possibility_bound，用较大的负数代替负无穷。这一设置不仅解决了零概率的计算问题，实质上也对结果进行了初步的平滑操作。我们在实际计算时会和 ppt 上的描述一致，进行第二次平滑，得到更好的结果。

我们设置的这一零概率值为-60，事实上最开始时我们尝试过-1000，-100 等值，但由于大多数非零概率值大约在-10 左右，而句子长度又有限，它们实际上并无本质差别。

2.5 隐马尔可夫模型

我们在实际运用上述概率时，再外加一层平滑，即如 ppt 上所述。最终的计算公式为：

$$P(p_i|p_{i-1},p_{i-2}) = triple_{character_{weight}} \times P[p_{i-2}p_{i-1},p_i] + \left(1 - triple_{character_{weight}}\right) \times double_{character_{weight}} \times P[p_{i-1},p_i] + (1 - triple_{character_{weight}}) \times (1 - double_{character_{weight}}) \times P[p_i]$$

另外如果我们所需要的转移概率在上述数据中无法得到，我们就用 zero_possibility_bound 代替。

3 测试结果

3.1 模型比较

我们一共测试了两大类 HMM 模型，两类即为上述描述的一般的二元和三元字频模型。我在测试过程中曾经构想了第三类模型，这是我在实现二元模型中临时考虑的一个全新模型，但在得到三元字频模型的结果后最终放弃了测试，因为三元字频模型的结果下这一考虑已经无法有效地进行改进，不过我仍将这一考虑记录在下方，权当启发。

这一模型是基于二元字频模型进行改进的。考虑到汉语的句中是以词作为表意单元的，我希望可以用一种简单的方式来体现出这种以词为单位的考虑方式（从而可以无需重新对语料进行分词处理统计词频）。我得到的做法是：利用 jieba 库中的 dict.txt 文件，收集所有的

词组。然后在转移过程中，额外加一个二进制维度，表示上一个单字是否已经和前字组成词组。如若已经组词，那么我们在这次转移中，将可以和该字组词的单字的转移概率调低一个值 beta（因为一个字不应同时处于两个词中），如若未组词，则不作调整（因为单字作为单元出现概率也是较大的）。

其余两个模型各自调参的结果如下：

二元模型 d 值	字准确率	句准确率
0.8	2852/3410	131/341
0.85	2864/3410	133/341
0.9	2869/3410	135/341
0.92	2873/3410	135/341
0.94	2873/3410	134/341
0.96	2872/3410	133/341

二元模型的 d 值对结果影响相对比较明显，我们取到最优值为 0.92

三元模型 t 值（d=0.92）	字准确率	句准确率
0.85	3161/3410	243/341
0.88	3161/3410	343/341
0.9	3162/3410	244/341
0.92	3160/3410	242/341

三元模型的 t 值对结果的影响就比较小了，由于加权关系 d 值对结果的影响已经微乎其微，我们就不再进行调整。

我们分别调整参数，比较两个模型取得的最好结果如下：

模型	参数	字准确率	句准确率
二元模型	d=0.92 (t=0)	2873/3410（84.3%）	135/341（39.6%）
三元模型	d=0.92, t=0.9	3162/3410（92.7%）	244/341（71.5%）

可见三元模型确实较二元模型有着极为明显的优势。

为了进一步具体地描述这种优势，我们挑选出具有代表性的若干样例来进行解释。（下列样例中，第一行为拼音，第二行为二元模型结果，第三行为三元模型结果）

bei jing shi yi ge mei li de cheng shi
北京**市**一个美丽的城市
北京**是**一个美丽的城市

在统计数据中**京-市**二元组有约 45000 次出现，而**京-是**二元组仅有约 600 次。不过显然我们并非希望得到“北京市”的结果，而希望得到“是一个”。这一结果可以通过**是一-个**三元组的高转移概率得到。

ai chi xiao xiong bing gan
爱吃小熊**丙肝**
爱吃小熊**饼干**

类似地我们可以在**小熊**和**饼干**之间利用三元组建立联系，防止由于新闻中多次提到丙肝而导致的偏差。

mei guo de hua lai shi bi ni men bu zhi dao gao dao na li qu le
美国的**话来是**比你们不知道高到哪里去了
美国的**华莱士**比你们不知道高到哪里去了

而当遇到三字词的时候，二元模型在许多场合都显得力不从心，而三元模型则得心应手。

xian di chuang ye wei ban er Zhong dao beng cu
现地创业**委办二**中**到**崩殂

先帝创业**未半而**中**道**崩殂

遇到连续的几乎专用的特殊用法（例如古诗文）时三元模型的优势会更加明显。

3.2 模型优势

我们继续进行横向比较，通过测试集内外的精心设计的样例，阐释本输入法的横向比较优势。（这些样例在询问到的其它同学的输入法中均无法得到准确的答案）

首先最令我惊喜的是，通过加入日常沟通的语料，我们的输入法竟然学到了不少古文。（猜测是因为确有群友说过这些话语，因为尚有不少的古文未能成功识别）

大珠小珠落玉盘
此诚危急存亡之秋也
停车坐爱枫林晚
寻寻觅觅冷冷清清凄凄惨惨戚戚

其次我们的输入法也学到了许多网络流行语。由于 QQ 聊天记录的时效性即使是最近发生的事件或是最近风靡的流行短语我们也能轻松地学会。

某科学的超电磁炮
新型冠状病毒
十七张牌你能秒我
千层饼

最后让我非常满意的是，我和我的同学所在的一些小众亚文化圈中的专有名词也能够正确地输出，这无疑在实际应用上是及其便利的，也符合我利用 QQ 聊天记录作为语料的初衷。

青蛙的**体术**是真的**草**
阿空的**弹幕非常硬**（东方非想天则）
我**起手七对一向听**爽了（日麻）
我的**主牌**里**满编**了**泰菲力**
洁斯凯控打**红烧**是优势对局（万智牌）
这次**同图挑战**是**砍二**和**黑叔叔**
印加**丘陵羊**（文明 5）

除了一般语料之外，收集个人所在群体的聊天素材可以得到为个人量身定制的使用舒适的输入法，我认为这对于输入法来说是非常重要的。在使用时如果没有上述识别，很可能需要逐字地进行选择，这和一般的输入错误又有所不同，会更耗费时间，体验更差。

3.3 模型劣势和改进空间

当然，这个三元模型远未达到完美。（下列样例中，第一行为拼音，第二行为三元模型结果，第三行为标准结果）

yi zhi ke ai de da huang gou
一只可爱的大**荒沟**
一只可爱的大**黄狗**

大黄狗这一词在二元模型中成功地输出了，但在三元模型中输出反而失败了。

ta shi wo de mu qin
他是我的母亲

她是我的母亲

性别问题我们没有能够解决。这一问题在有些句子中没有出现，但在另一些句子中出现了。在三元模型下一般情况下这个代词是完全没有办法和母亲产生任何联系的，这会需要更高阶的模型或是单独的称谓处理来解决。

liang zi jiu chan zuo wei liang zi tong xun de zhong yao zi yuan
量子**就产**作为量子通讯的重要资源
量子**纠缠**作为量子通讯的重要资源

语料库中的词组还不够完善，导致仍有一些词组未能得到识别。

bai yu tiao zhu luan ru chuan
百余条主乱入船
白雨跳珠乱入船

古文的识别也未能做到高准确率，这很大程度上由语料中古文出现频率决定。

为了解决这些问题，我们可以采用以下的方法加以改进：

- 导入更多的一般语料和个人语料，丰富词库
- 导入专用古诗文语料，并保证它们能成功纳入到三元字频中（例如反复导入十次）
- 扩展模型到更高阶的情况，或利用 jieba 分词实现词模型的转移（需要相当完备的词组库）