拼音输入法实验报告

计 64 陶东来 学号:2016011322

April 8, 2017

1 算法基本思路

显然，在二元文法的前提下，当前这个拼音 p_i 所对应的汉字 x_i 只需考虑其前一个字 x_{i-1} 及其对应的拼音 p_{i-1} 即可。首先不考虑拼音，我们有：

$$P(\Pi_{j=1}^{i} x_j) = P(\Pi_{j=1}^{i-1} x_j) P(x_i|x_{i-1})$$

接下来考虑拼音的影响。在二元文法的假设下，有：

$$P(\Pi_{j=1}^{i} x_j|\Pi_{j=1}^{i} w_j) = P(\Pi_{j=1}^{i-1} x_j|\Pi_{j-1}^{i-1} w_j) P(x_i|x_{i-1}w_i w_{i-1})$$

因此现在的关键就变成了计算 $P(x_i|x_{i-1}w_i w_{i-1})$。只要有了这个概率，我们就可以使用 Viterbi algorithm 求出最后的结果了。

1.1 Naive version

我们先“假装” $P(x_i|x_{i-1}w_i w_{i-1}) = P(x_i|x_{i-1})$，由此实现程序的第一个版本，之后所有的修改都会基于这一版本。

在这种情况下，我们只需简单地统计所有的字符二元组即可。大约数分钟即可完成数据的收集，并且可以应付很多情况。如”zhong hua ren min gong he guo”，已经可以得出“中华人民共和国”这样我们想要的结果了。

但是有一个非常严峻的问题，就是先前我们的假设是显然有问题的，最鲜明的例子就是”mo fa”，这个版本的程序会给出“无法”。这是因为“无”是一个多音字，而实际上“无法”这个词真正应该归类在“wu fa”而不是“mo fa”。这就体现出了先前我们假想的问题，即“多音字分流”。

1.2 Improved version

我们考虑对多音字分流这个问题进行修正。事实上，只要我们将多音字的不同读音视作不同的字，那么先前的假设就是正确的了。因此，我们考虑对原文进行注音，那么之后的部分就和 Naive version 类似了。

这是一个非常美好的设想。但是由于手头没有词语 -拼音的词典，我无法自己进行注音，只能通过注音库来完成这一工作。在这里我使用的是 pypinyin 这个包。它的工作原理是先使用分词库对原文进行分词，然后再对照词典进行注音。这样带来的结果是我单单想要对那个最小的文件（20M）进行注音就要花费无法承受的时间。并且，我无法对这个库本身进行任何优化。

1.3 Final version

因此我选择退而求其次，对所有包含多音字的语段，取出 1% 进行注音。这样，由于语料库本身足够大，这样抽取出的信息也足够获得 $P(x_i|x_{i-1}w_iw_{i-1})$。尽管如此，在我使用了多线程并行的前提下，还是花费了 1.5 小时才处理完了所有的语料。

在加上线性平滑和对同学们建立的测试样例集的支持，现在的版本已经可以达到很好地效果了。

当然仍有不足。比如“qin dian ta dang xia yi ren”，我的输入法会给出“钦点他当下一人”，其实我们想要的是“钦点他当下一任”。只不过这个问题恐怕就算是基于字的三元模型也不见得能解决，要基于词的模型才能解决。但是对于所有文本的分词由之前看来，时间无法承受；并且我们也没有词典，想要获得词我们只能通过分词实现。而且考虑到注音的过程十分耗时，如果使用三元文法我会不得不提高注音的采样比例，这样不仅会导致模型膨胀，而且会导致训练时间进一步增长。这两点使得我暂时放弃了在模型层面的改进。

而对于平滑取的系数来说，我第一次取就取到了不错的效果。之后并没有多做改动。

1.4 收获

本次实验让我熟悉了 python 和多线程通信编程技巧，了解并实现了 Viterbi algorithm 和其他一些概率模型。这个实验可以说是人工智能中非常简单的一部分，以致现在普通人并不会将它算作人工智能。尽管如此，整个过程依然非常精彩有趣。

希望之后会有其他有意思的实验:-)。