1 基本信息

题名：**GodTian Pinyin: An awesome Chinese Input Method**
作者：胡求 (MG1633031)，李崇杰 (MG1633040)，尹良良 (MG1633094)
邮箱：huqiu00@163.com
2016.11.15

Abstract

经过自然语言处理课程的一段时间的学习，我们在所学的基础上充分利用现有资料，学习相关理论知识，合三人之力研制了一款简易的中文拼音输入法–GodTian Pinyin。这款输入法利用了马尔科夫链，隐马尔科夫模型，维特比算法等理论作为指导，并经过严格，精准，高效的核心算法实现和简洁的用户交互界面实现，最终取得了良好的输入效果。

2 任务描述

实现一个中文拼音输入法。

经过分析，我们决定分以下几步来对输入法软件进行实现：

- 核心功能算法实现与测试
- 核心功能算法的接口实现
- 核心算法与用户交互界面（UI）的对接

3 技术路线

在中文拼音输入法中，我们需要完成拼音序列到汉字序列的转换，比如输入 “nihao”，输入法会给出我们想输入的字 “你好”，到这里我们就可以问出几个问题：

- 如何切分拼音？如：用户输入 “xiana”，输入法应该判断用户想输入 “xian a”（闲啊）还是 “xia na”（夏娜）还是 “xi an a”（西安啊）?
- 如何实时给用户以反馈？
- 对于切分好的拼音，怎样找出用户最想输入的一串中文显示给用户？
- 用户输入的拼音是错的的情况下，如何容忍这种错误？该如何显示？

也许我们还能问出更多的问题，中文拼音输入法就是这样，总有可以继续抠下去的细节。那么我们如何解决上面的问题？我们的方案如下：

- 如何切分拼音？这里我们暂时采用最长匹配的方式，也就是说，如果用户输入的首个串是拼音或者是某个合法拼音的前缀，那么我们会继续向后发现，等待用户输入，直到用户输完后发现这个字符（假设是第 n 个）与原来 n-1 个不是合法的拼音也不是合法的拼音的前缀，那么此时将前面 n-1 串切分成拼音，这就完成了一个拼音的发现，比如说输入 “xiant”（想输 xiantian），则我们会扫描这个串，一直到 “xian”，到 “xiant” 的时候发现既不是合法拼音的前缀也不是合法拼音，那么从 t 前面划分开，得到 “xian’t”，同样的道理发现后续的拼音。在实时任务中，用户即使没有输完我们仍应该显示东西，那么我们先切分拼音，最多只会有最后一个是不完整的拼音前缀，那么我们将完整的和不完整的分开处理。假设是 “xian’t” 的情况，我们将 “xian” 放入 viterbi 算法中，通过

HMM 得出概率最大的一个输出串，然后将最后的 “t” 在训练过的 Trie 树中搜索出所有以 “t” 为前缀的字，以及他们出现的频率，取频率最高的若干个，然后得到他们的拼音，与前面 n-1 个拼音组合起来跑 Viterbi 算法，得到最可能的一个中文串。具体 Trie 树会在后面讲解。

- 如何实时给用户以反馈？

上面其实已经初步解释了如何实时反馈，实时反馈我们要做的就是用户每输一个字母，我们就能够显示出用户可能想要打的字，那么，以一个字母开头的拼音有很多，每个拼音对应的字也可能有很多，也即结果有很多，但是我们又不能漏掉，所以只能考虑所有的字，比较选出概率最大的若干个字，这时候我们可以采用 Trie 树来解决。Trie 树就是前缀树，说白了就是将拼音的字母按顺序顺着根插入到树中，每个叶子节点就是一个拼音，这个拼音就是顺着根一路走下来取的字母的顺序组合，这样我们就可以找出以任意字符串为前缀的所有拼音，方法就是 dfs 遍历每一个以其为前缀的子树的叶子节点，这时候我们叶子节点存的其实是一个字典，key 为这个拼音对应的可能的字，value 为这个字出现的频率，以作为比较。

- 对于切分好的拼音，怎样找出用户最想输入的一串中文显示给用户？

这里我们使用隐马尔可夫模型，将用户想输入的中文字作为隐状态，用户输入的拼音为显状态，通过最大似然估计即频率估计出 HMM 的三个矩阵的值，最后通过 viterbi 算法找出概率最大的若干个中文字串显示出来。

- 用户输入的拼音是错的的情况下，如何容忍这种错误？该如何显示？

由于考虑到实现高度容错的复杂性，我们假设用户会输入正确的拼音，在想分割的时候会自行添加分隔符 “”’，由于大部分输入法用户绝大部分时间都会输入正确的拼音，所以，这样一个假设既简化了实现的过程，又没有损失太大的用户体验。

4 用到的数据

由于训练 HMM 模型的需要，我们从搜狗实验室找到了 SogouQ 用户查询数据集，预处理成合法的句子之后大约有 360M，且为了避免查询句太短，我们也增加了将近 30M 的搜狐新闻数据作为训练语料，这里面包含了很多的长句子。通过这两个语料的训练，我们得到了长句和短句皆可表现较好效果的 HMM 模型。并且我们还可以继续拓展语料，以增加我们 HMM 模型的准确性，这是后话，不提。

5 遇到的问题及解决方案

1. UI 界面的问题，由于 UI 设计的复杂性与不同系统的考虑，出现了许多莫名其妙的 BUG，这使得我们花了许多时间。

2. viterbi 算法的效率问题，由于以某个字母开头的拼音对应的字有很多个，假设我们取最优的 K 个，我们需要将这 K 个与前面已有的拼音组合，然后跑一遍 Viterbi 算法，由于 Viterbi 算法从一个状态转移到另一个状态的计算量很大，我们使用了记忆（cache）的方法来加速，具体方法就是记录下某一个完整拼音串所对应的 viterbi 算法的最后一个状态的相关情况，这样如果我们再次遇到这个拼音串（A）加上另一个拼音（B）跑 viterbi 的情况，我们就不需要从这个组合串的开头开始跑 viterbi 算法了，而是直接从 A 串跑完 viterbi 的最后一个状态（从记忆单元读取）开始，向 B 进行转移。这个记忆单元会随着程序而一直存在，并且我们对这个对象做了持久化，在输入法启动时我们会

读取这个文件（记忆单元），这也就意味着，如果我们曾经输入过某个拼音串，那么我们以后再输入同样的拼音串的时候，不再需要跑核心算法，而是直接显示结果，这样在速度上就取得了显著的提高，就会出现，输入法越用越好用，越用越快的好处，当然这牺牲了一些存储空间，但是如今我们都不缺存储空间。

3. 重复计算的问题，比如在用户觉得打错了的时候，往后退格，这时就会退到某一个前缀，但是其实这个前缀我们是算过了的，也显示过了的，就是说我们退回到我们以前显示过的内容的时候，如果不加优化，那么又会重新跑一遍核心的 viterbi 算法，这样就会很慢，那么我们还是利用 cache 思想，将输入的拼音串以及对应的显示结果相对应并且存起来，这样我们就做到了飞速的退格操作。

4. Python 语言固有的性能问题，解决这个问题只有更换语言，事实上用 C++ 语言实现的话我相信会快很多，这在后面可以考虑用 C++ 实现，这也是完全可行的。

6 性能评价

我们实现了秒级的未输入过的长句的秒级输入，以及输入过的句子的毫秒级别的输入，以及毫秒级别的退格操作。

7 程序的运行环境

1. Python 2.7

2. 需要安装的 Python 包：Tkinter, cPickle, pypinyin 等模块

8 执行方法及参数

在项目 Project 目录下，运行

```
$ python gui.py
```

即可。
（EXE 版本还有一点编译问题，明天上课演示）