基于HMM的拼音输入法

一、文件说明

```
HMM_pinyin

└── corpus // 语料

└── params // 模型参数

        └── init_prob.json // 初始概率矩阵

        └── emiss_prob.json // 发射概率矩阵

        └── trans_prob.json // 转移概率矩阵

        └── pinyin_states.json // 同音字记录

└──  preprocessing.py // 预处理语料的脚本，用于过滤杂乱字符

└── count_for_hmm.py // 计算HMM模型所需要的参数，即params下的文件

└── pysplit.py // 用于处理拼音序列切分的模块

└── hmm.py // HMM类，包含viterbi算法实现部分

└── input_method.ui // 输入法ui界面

└── input_method.py // 根据ui界面对应生成的python代码

└── input_method_slots.py // 定义输入法的接口和响应动作

└── main.py // 程序的入口

└── readme.md

└── readme.pdf
```

二、功能简介

实现一个简单的基于HMM的拼音输入法

三、原理

汉语有很多字拼音相同。可以根据上下文来挑选概率大的字。如“wo”可对应汉字“我、窝、握、卧”等，“de”也可以对于“的、得、地、德”等。但是“wode”对应的概率比较大的应该属于“我的”。

我们把拼音作为观察值，把汉字当作状态，那么可以用HMM来建模拼音输入。转移概率可从语料库训练而得，生成概率则需考虑多音字（在不知道多音字概率的情况下可假设等概率）。

四、HMM模型&viterbi算法

4.1、HMM模型

1. 马尔可夫性可描述为

$$\forall h, s \leq t, \mathrm{P}(X(t+h)=y|X(s)=x(s))=\mathrm{P}(X(t+h)=y|X(t)=x(t))$$

2. 三个重要参数
   - 初始概率矩阵π：$\pi_i=\mathrm{P}(q_1=i)$
   - 转移概率矩阵a：$a_{ij}=\mathrm{P}(q_{t+1}=j|q_t=i)$
   - 发射概率矩阵b：$b_j(k)=\mathrm{P}(o_t=k|q_t=j)$
3. 本次实验求解问题为HMM的预测问题，即已知模型参数和观测序列，求解对应的状态序列。

4.2、viterbi算法

1. 初始化：

$$\delta_1(i)=\pi_i b_i(o_1), 1 \leq i \leq N$$

$$\Phi_1(i)=0$$

2. 迭代求解：

$$\delta_t(j)=\max_{1\leq i\leq N}\delta_{t-1}(i)a_{ij}b_j(o_t)$$

$$\Phi_t(j)=\operatorname{argmax}_{1\leq i\leq N}\delta_{t-1}(i)a_{ij}b_j(o_t)$$

3. 终止：

$$\mathrm{P}^*=\max_{1\leq i\leq N}\delta_T(i)$$

$$q_T^*=\operatorname{argmax}_{1\leq i\leq N}\delta_T(i)$$

4. 最优路径（隐状态序列）回溯：

$$q_t^*=\Phi_{t+1}(q_{t+1}^*), t=T-1, T-2, \ldots, 2, 1$$

五、实验过程

5.1、语料获取与处理

由于原语料文件`corpus/pinyin_train.txt`中语料有限，且拼音标注文件`pinyin.txt`中拼音标注有限，因此我从网上找了一份人民日报2014年的新闻语料`corpus/2014_corpus.txt`作为补充，同时利用`pypinyin`对语料进行拼音标注来生成自己的拼音标注库。

由于原始语料中存在许多不需要的符号和字符，所以编写了`preprocessing.py`来对原始语料进行过滤提取，提取使用正则表达式。

```
chinese = re.compile(r'[\u4E00-\u9FA5`~!@#$%^&*()_\-+=<>?:"{}|,.;·~！@#￥%……&* （）——\-+={}|《》？：“”【】、；‘，。、|\n]{1,}')
```

最后生成预处理语料`corpus/2014_corpus_pre.txt`。

5.2、模型参数计算

在HMM类中使用了以下5项数据实现：

1. init_prob：存储汉字的初始概率矩阵。

```
{
    "汉字1": 概率1,
    "汉字2": 概率2,
    // ...
    "汉字n": 概率n
}
```

2. trans_prob：存储汉字之间的转移概率矩阵（采用二元模型）。

```
{
    "汉字1": {"前置字1": 概率11, "前置字2": 概率12, /*...*/, "前置字n": 概率1n},
    "汉字2": {"前置字2": 概率21, "前置字2": 概率22, /*...*/, "前置字m": 概率2m},
    // ...
    "汉字w": {"前置字1": 概率w1, "前置字2": 概率w2, /*...*/, "前置字t": 概率wt}
 }
```

3. emiss_prob：存储汉字到拼音的转移概率矩阵。

```
{
    "汉字1": {"拼音1": 概率11, "拼音2": 概率12, /*...*/, "拼音n": 概率1n},
    "汉字2": {"拼音2": 概率21, "拼音2": 概率22, /*...*/, "拼音m": 概率2m},
    // ...
    "汉字w": {"拼音1": 概率w1, "拼音2": 概率w2, /*...*/, "拼音t": 概率wt}
 }
```

4. pinyin_states：存储同一个拼音的多个汉字，供计算时遍历使用。

```
{
    "拼音1": ["汉字1", "汉字2", /*...*/, "汉字n"],
    "拼音2": ["汉字2", "汉字2", /*...*/, "汉字m"],
    // ...
    "拼音w": ["汉字1", "汉字2", /*...*/, "汉字t"]
 }
```

5. pyList：存储拼音规则表。

```
["合法的拼音1", "合法的拼音2", ..., "合法的拼音n"]
```

在计算时，首先从语料中提取汉字句段，通过正则表达式实现。

```
chinese = re.compile(r'[\u4e00-\u9fa5]{2,}')
```

然后，在每个句段的前后加上`BOS`和`EOS`分别作为句首和句尾的标识符，方便进行处理和矩阵的统一化存储。

在计算好1-4矩阵的概率后，以上述1-4的格式作为json文件进行存储（因为会产生高维稀疏矩阵，所以不采用传统的矩阵方式便于压缩空间）[1]。

最后，通过以下拼音规则求出pyList[2]：

1. 根据拼音规则得出以下三个列表

```
smList = ['b', 'p', 'm', 'f', 'd', 't', 'n', 'l', 'g', 'k', 'h', 'j', 'q',
'x', 'z', 'c', 's', 'r', 'zh', 'ch', 'sh', 'y', 'w'] // 声母表
ymList = ['a', 'o', 'e', 'i', 'u', 'v', 'ai', 'ei', 'ui', 'ao', 'ou', 'iu',
'ie', 've', 'er', 'an', 'en', 'in', 'un', 'ang', 'eng', 'ing', 'ong', 'uai',
'ia' ,'uan' ,'uang', 'uo', 'ua'] // 韵母表，为了简化规则，表中除了韵母外也增加了另一
些拼音组合
ztrdList = ['a', 'o', 'e', 'ai', 'ei', 'ao', 'ou', 'er', 'an', 'en', 'ang',
'zi', 'ci', 'si', 'zhi', 'chi', 'shi', 'ri', 'yi', 'wu', 'yu', 'yin',
'ying', 'yun', 'ye', 'yue', 'yuan'] // 整体认读音节表
```

2. 然后将smList中的字符串和ymList中的字符串进行组合可构成合法的拼音（即声母和韵母组合），将ztrList中的字符串直接作为合法的拼音（整体认读音节本身就是合法的拼音）。

5.3、拼音解码为汉字实现

*viterbi*算法实现时为了防止出现精度爆炸现象，所以对每个概率都做了*log*平滑，且将乘法转换为加法防止数位溢出

```
self.min_f = -3.14e+100 # 用于log平滑时所取的最小值，用于代替0
```

1. 将输入的拼音字符串根据pyList表和设定规则切分为不同的拼音组合，通过pysplit文件中的函数实现（采用递归方式即深度优先搜索找出所有合法拼音切分）。
2. 生成数组viterbi用于记录每个位置上各状态的概率，格式如下：

```
viterbi[pos][word] = (probability, pre_word)
# pos是目前节点的位置，word为当前汉字即当前状态，probability为从pre_word上一汉字即上一
状态转移到目前状态的概率
```

viterbi算法实现部分

3. 针对每个拼音切分，首先根据第一个拼音，从pinyin_states中找出所有可能的汉字s，然后通过init_prob得出初始概率，通过emiss_prob得出发射概率，从而算出viterbi[0][s]。

```
viterbi[0][s] = (self.init_prob.get(s, self.min_f) + self.emiss_prob.get(s,
{}).get(seq[n][0], self.min_f), -1) # seq[n][0]为第一个拼音
```

4. 同样遍历pinyin_states，找出所有可能与当前拼音相符的汉字s，利用动态规划算法从前往后，推出每个拼音汉字状态的概率viterbi[i+1][s]。

```
viterbi[i + 1][s] = max([(viterbi[i][pre][0] + self.emiss_prob.get(s,
{}).get(seq[n][i + 1], self.min_f) + self.trans_prob.get(s, {}).get(pre,
self.min_f), pre) for pre in self.pinyin_states.get(seq[n][i])])
```

5. 最后取概率最大的串（可从大到小取多个串），即概率最大的viterbi[n][s]（s为末尾的汉字），然后对串进行回溯即可得对应拼音的汉字。

5.4、GUI界面编写

采用PyQt5进行编写[3]，可通过记录击键次数来判断模型的好坏。

参考

1. iseesaw/Pinyin2ChineseChars: 实现基于Bigram+HMM的拼音汉字转换系统（输入法Demo） (github.com)
2. 再谈Python之拼音拆分 - 简书 (jianshu.com)
3. Lancer-He/pinyin_IME_HMM: 基于pyqt和hmm的拼音输入法 (github.com)