基于HMM的拼音输入法

一.文件说明

```
.
  hmm.py          //HMM类以及viterbi算法实现部分
  main.py         //输入法运行程序
  pysplit.py      //处理拼音序列切分
  实验报告.pdf
├─corpus
  │  corpus_pre.txt //预处理后的语料文本
  └─wiki_zh   // 语料
      ├─AA
      ├─AB
      ├─AC
      ├─AD
      ├─AE
      ├─AF
      ├─AG
      ├─AH
      ├─AI
      ├─AJ
      ├─AK
      ├─AL
      └─AM
├─data
      emiss_prob.json         //发散概率矩阵
      pinyin.txt              //合法的拼音序列
      pinyin_states.json      //同音字记录
      start_prob.json         //初始概率矩阵
      trans_prob.json         //转移概率矩阵

└─train
       dataprocess.py       . //语料预处理文件
       train.py               //HMM模型训练
```

二.原理

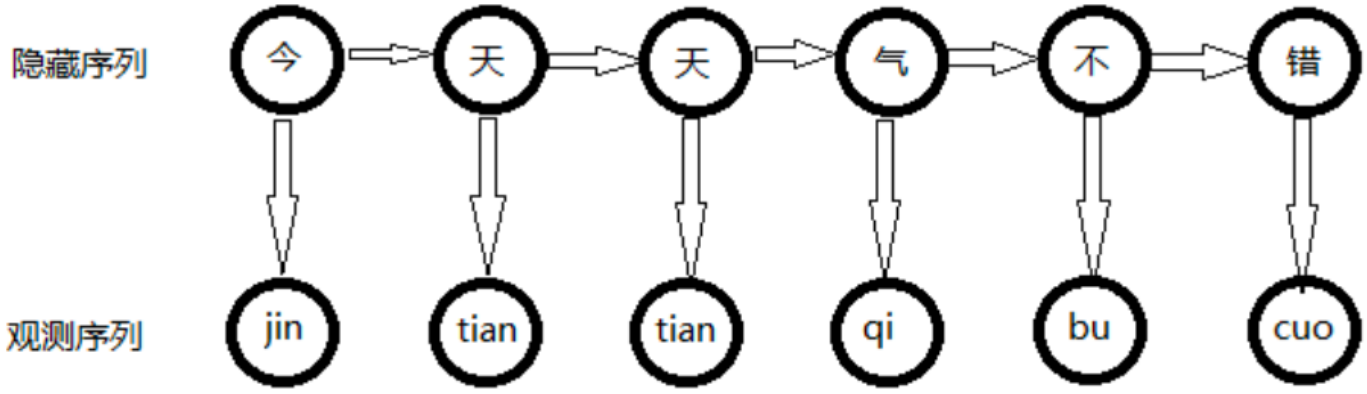
对于一个给定的拼音串，比如：nan jing da xue。其中每个拼音都对应多个汉字，整体上构成一个复杂的网络，对应着多种组合。而结合语境来看，每个字的出现是受其他字影响的。在本次实验中，我们假设每个字出现仅与其上一个字有关，即最简单的二元模型。
那么我们要想得到最有可能出现的字符串，就转变成了在该网络中找到一条概率最大的路径。这恰好对应隐马尔可夫模型(Hidden Markov Model，HMM)。我们把拼音作为观察值，把汉字当作状态，那么可以用HMM来建模拼音输入。

三.HMM模型以及viterbi算法

3.1 HMM介绍

隐马尔可夫模型 (Hidden Markov Model, HMM) 是一种统计模型，用来描述一个含有隐含未知参数的马尔可夫过程。

隐马尔可夫模型有两个关键的概念：状态 (state) 和 观测 (observation)。隐马尔可夫链随机生成的状态的序列，称为状态序列；每个状态生成一个观测，由此产生的随机的观测的序列，称为观测序列。序列的每一个位置又可以看作一个时刻。



隐马尔可夫模型由初始状态概率π、状态转移矩阵A 、以及观测概率矩阵B决定。一个隐马尔可夫模型可用三元符号表示：$\lambda = (\pi, A, B)$。
初始状态概率π和状态转移矩阵A确定了隐藏的马尔可夫链，生成了不可观测的状态序列。
观测概率矩阵B确定了如何从状态生成观测值，与状态序列一起确定了如何产生观测序列。

3.2 训练HMM

所以我们首先根据语料库生成对应的马尔卡夫模型，采用频率代替概率的方法来计算(π, A, B)。具体计算方法可见 `train/train.py` 。

```
def init_trans(seqs):
    """
    @function:找出字典中每个汉字后面出现的汉字集合，并统计概率。
    """

    trans_prob = {}
    for seq in seqs:

        if len(seq) == 0:
            continue

        seq = [_ for _ in seq]


        seq.insert(0, 'BOS')
        seq.append('EOS')

        # init the occurence of "[pre][post]"
        for index, post in enumerate(seq):
            if index:
                pre = seq[index - 1]
                if post not in trans_prob.keys():
                    trans_prob[post] = {}
                if pre not in trans_prob[post].keys():
                    trans_prob[post][pre]=1
                else:
                    trans_prob[post][pre]+=1

    for key in trans_prob.keys():
        tot = sum(trans_prob.get(key).values())
        for pre in trans_prob.get(key).keys():
            trans_prob[key][pre] = math.log(trans_prob[key].get(pre) / tot)

    save('trans_prob', trans_prob)
```

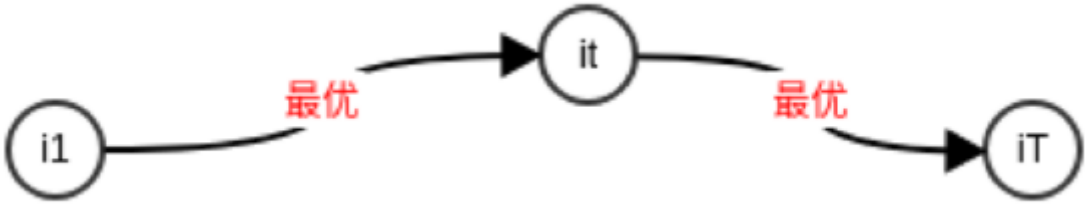
生成的start_prob.json部分如下：


```
{
  "\u6570": -6.966439166100104,
  "\u7ed3": -6.582637439430219,
  "\u53d8": -8.072827543718121,
  "\u4ece": -5.520751177004685,
  "\u7531": -4.89711838032943,
  "\u8ba1": -7.581392454184374,
  "\u91cf": -8.682874185280436,
  "\u4e3a": -5.075808042790791,
  "\u57fa": -6.621372782268083,
  "\u5bf9": -5.501771845443446,
  "\u65e9": -7.243079914416236,
  "\u7f8e": -5.693628420956052,
  "\u800c": -4.14814496309403,
  "\u4e16": -6.1502726616844186,
  "\u56e0": -4.709415527983289,
  "\u81f4": -8.42250129219787,
  "\u76f4": -6.1295974632686745,
  "\u4eca": -6.122244488963416,
```


3.3 预测以及viterbi算法

隐马尔可夫模型的预测问题，也称为解码 (decoding) 问题，就是在已知隐马尔可夫模型$\lambda = (\pi, A, B)$ 和观测序列$O = (o_1, o_2, \cdots, o_T)$ 的情况下，求使得观测序列条件概率 [公式] 最大的状态

序列$I=(i_1,i_2,\cdots,i_T)$. 即给定观测序列，求最有可能的状态序列。
对于HMM的预测问题，我们通常采用viterbi算法
根据动态规划原理，最优路径具有这样的特性：如果最优路径在时刻t通过结点i_t^*，则这一路径从结点i_t^*到终点i_T^** 的部分路径，对于从i_t^*到i_T^*的所有可能路径来说，也必须是最优的。



只需要从时刻t = 1 开始，递推地计算从时刻1 到时刻t且时刻t状态为(q_i) 的各条部分路径的最大概率（以及取最大概率的状态）。于是在时刻t=T的最大概率即为最优路径的概率P^* ，最优路径的终结点i_T^*也同时得到。之后为了找出最优路径的各个结点，从终结点i_T^*开始，由后向前逐步求得结点$i_{T-1}^*,\cdots,i_1^*$。得到最优路径。

1. 初始化：

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

$$\Phi_1(i)=0$$

2. 迭代求解：

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

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

3. 终止：

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

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

4. 最优路径（隐状态序列）回溯：

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

代码实现

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

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)
```

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 inself.pinyin_states.get(seq[n][i])])
```

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。

四.结果展示

采用命令行操作的方式

```
智能拼音输入法
操作介绍:
1.输入exit退出.
2. w,s上下页查找
请输入拼音或退出:nanjingdaxve
0.南京大学
1.南京大穴
2.南京大雪
3.南京大削
4.南京达血
0
结果为:  南京大学
```

五.不足

在本次实验中，基于二元假设，即认为当前汉字出现的概率仅与前一个汉字有关，这种假设在某些情况下是不准确的，在查询资料时，发现我们可以考虑改进二元字模型至三元字模型，这样会提高对一些三元词组识别的准确率，比如：青花瓷，晶状体等。

参考文献

1. 基于 HMM 隐马尔可夫模型的智能拼音输入法 (带 Web 前端)
2. 基于Bigram+HMM的拼音汉字转换
3. Python与HMM实现简单拼音输入法