• 问题描述

假设有一中文句子，而该中文句子的每个字在该句中都有BEMS共四种状态，其中B代表该字是某词语中的起始字，M代表是某词语中的中间字，E代表是某词语中的结束字，S则代表是单字成词。中文分词问题是根据某中文句子得到其每个字的状态。比如：

“我在家里吃饭”——》（分词）“我|在|家里|吃中饭”——》（状态）“SSBEBME”

• 问题分析

已知一系列的中文句子（序列）X以及句中每个字的状态（隐状态）Z.

我们将中文的每个字表示为字典里的序号，而每个字在序列中都有4个状态，由此将中文分词问题转换为离散HMM模型。

但是相较于问题1，问题2中并不是单个长序列，而一系列短序列，为了训练一系列短序列，我们需要对原训练方法进行改进。

最简单的思路，是将所有的短序列合并成一个长序列，但是如此以来无法训练出初始状态参数。

因此我们建立一个批量的HMM训练方法，此时我们需要重新改写参数优化的最大似然式：

假设有多个序列样本：$\{X_1, X_2, \cdots, X_M\}$

$$Q(\theta,\theta') = \sum_{m}^{M} \ln P(X_m, Z_m|\theta) = \sum_{m}^{M} \left\{ \sum_{k=1}^{K} \gamma(z_{1k}) \ln \pi_k + \sum_{n=2}^{N} \sum_{j=1}^{K} \sum_{k=1}^{K} \vartheta(z_{n-1,j}, z_{n,k}) \ln \Lambda_{jk} + \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \ln N(x_n|\mu_k, \Sigma_k) \right\}$$

可以看出上式同单序列的训练方法非常一致，只是需要将多个序列的进行叠加起来，不过要注意的是不同序列在叠加之前，需要进行归一化处理！相关

```
def train_batch(self, X, Z_seq=list()):
        # 针对于多个序列的训练问题，其实最简单的方法是将多个序列合并成一个序列，而唯一需要调整的是初始状态概率
        # 输入X类型：list(array)，数组链表的形式
        # 输入Z类型：list(array)，数组链表的形式，默认为空列表（即未知隐状态情况）
        self.trained = True
        X_num = len(X) # 序列个数
        self._init(self.expand_list(X)) # 发射概率的初始化

        # 状态序列预处理，将单个状态转换为1-to-k的形式
        # 判断是否已知隐藏状态
        if Z_seq!=list():
            Z = []
            for n in range(X_num):
                Z.append(np.zeros((len(X[n]),self.n_state)))
                for i in range(len(Z[n])):
                    Z[n][i][int(Z_seq[n][i])] = 1
        else:
            Z = []  # 初始化状态序列list
            for n in range(X_num):
                Z.append(list(np.ones((len(X[n]), self.n_state))))

        for e in range(self.n_iter):  # EM步骤迭代
            # 更新初始概率过程
            #  E步骤
            print "iter: ", e
            b_post_state = []  # 批量累积：状态的后验概率，类型list(array)
            b_post_adj_state = np.zeros((self.n_state, self.n_state)) # 批量累积：相邻状态的联合后验概率，数组
            b_start_prob = np.zeros(self.n_state) # 批量累积初始概率
            for n in range(X_num): # 对于每个序列的处理
                X_length = len(X[n])
                alpha, c = self.forward(X[n], Z[n])  # P(x,z)
```

```
                beta = self.backward(X[n], Z[n], c)  # P(x|z)

                post_state = alpha * beta / np.sum(alpha * beta) # 归一化!
                b_post_state.append(post_state)
                post_adj_state = np.zeros((self.n_state, self.n_state))  # 相邻状态的联合后验概率
                for i in range(X_length):
                    if i == 0: continue
                    if c[i]==0: continue
                    post_adj_state += (1 / c[i]) * np.outer(alpha[i - 1],
                                                           beta[i] * self.emit_prob(X[n][i])) * self.transm

                if np.sum(post_adj_state)!=0:
                    post_adj_state = post_adj_state/np.sum(post_adj_state)  # 归一化!
                b_post_adj_state += post_adj_state  # 批量累积:状态的后验概率
                b_start_prob += b_post_state[n][0] # 批量累积初始概率

            # M步骤,估计参数,最好不要让初始概率都为0出现,这会导致alpha也为0
            b_start_prob += 0.00001*np.ones(self.n_state)
            self.start_prob = b_start_prob / np.sum(b_start_prob)
            b_post_adj_state += 0.001
            for k in range(self.n_state):
                if np.sum(b_post_adj_state[k])==0: continue
                self.transmat_prob[k] = b_post_adj_state[k] / np.sum(b_post_adj_state[k])

            self.emit_prob_updated(self.expand_list(X), self.expand_list(b_post_state))
```

实验结果:

```
startprob_prior:  [  5.81866978e-01   3.40459414e-08   3.40459414e-08   4.18132954e-01]
transmit:  [[  1.05236493e-08   9.55715970e-02   9.04428382e-01   1.05236493e-08]
 [  7.69852048e-08   3.00850841e-01   6.99149005e-01   7.69852048e-08]
 [  4.10833036e-01   1.14029208e-08   1.07175551e-08   5.89166942e-01]
 [  3.75113756e-01   1.09568388e-08   1.06352559e-08   6.24886222e-01]]
我要回家吃饭:  [ 0.  2.  0.  2.  3.  3.]
中国人民从此站起来了:  [ 0.  2.  0.  2.  0.  2.  3.  0.  2.  3.]
经党中央研究决定:  [ 0.  2.  0.  2.  0.  2.  0.  2.]
江主席发表重要讲话:  [ 0.  1.  2.  0.  2.  0.  2.  0.  2.]
```


这里的0 、1、 2、 3表示状态B、M、E、S

比如"我要回家吃饭"的分词结果为"我要|回家|吃|饭"

比如"江主席发表重要讲话"的分词结果为"江主席|发表|重要|讲话"

这个程序运行有点慢，实际不需要使用完整的HMM模型也能快速完成中文分词工作，具体可以参考这个代码https://github.com/fxsjy/finalseg，其代码也非常简单，利用了一个简化的HMM思路，完成了一个快速中文分词。