隐马尔可夫模型之Baum-Welch算法详解

隐马尔可夫模型之Baum-Welch算法详解

前言

在上篇博文中，我们学习了隐马尔可夫模型的概率计算问题和预测问题，但正当要准备理解学习问题时，发现学习问题中需要EM**算法**的相关知识，因此，上一周转而学习了EM算法和极大似然估计，对隐藏变量的求解有了一些自己的理解，现在我们继续回过头来学习隐马尔可夫模型的学习问题。EM算法的相关介绍可参照博文 EM算法及其推广学习笔记。如果对隐马尔可夫模型还不胜了解的话，可参看博文隐马尔可夫学习笔记（一）。

学习问题

隐马尔可夫模型的学习，根据训练数据是包括观测序列和对应的状态序列还是只有观测序列，可以分别由监督学习与非监督学习实现。本节首先介绍监督学习算法，而后介绍非监督学习算法——Baum-Welch算法（也就是EM算法）。

监督学习问题

假设已给训练数据包含S个长度相同的观测序列和对应的状态序列$\{(O_1, I_1), (O_2, I_2), \ldots, (O_S, I_S)\}$ ，那么可以利用极大似然估计方法来估计隐马尔可夫模型的参数，具体方法如下。

1.转移概率a_{ij} 的估计

设样本中时刻t处于状态i时刻t+1转移到j的频数为A_{ij} ，那么状态转移概率为a_{ij} 的估计是

$$\hat{a}_{ij} = \frac{A_{ij}}{\sum_{j=1}^{N} A_{ij}}, i = 1, 2, \ldots, N, j = 1, 2, \ldots, N$$

直接根据给定的O和I进行频数统计，在海藻模型中，我们可以统计100天前的天气转移次数，如在100天内，统计从sunny -> sunny 的次数，sunny -> cloudy 的次数，sunny - > rainy的次数，分别记作a_1, a_2, a_3，那么$a_{\text{sunny}->\text{any state}} = [\frac{a_1}{a_1+a_2+a_3}, \frac{a_2}{a_1+a_2+a_3}, \frac{a_3}{a_1+a_2+a_3}]$。因此，状态转移矩阵可以根据给定的隐藏序列I 计算得出。

2.观测概率$b_j(k)$的估计

设样本中状态为j并观测为k的频数是B_{jk} ，那么状态为j观测为k的概率$b_j(k)$的估计是

$$\hat{b}_j(k) = \frac{B_{jk}}{\sum_{k=1}^{M} B_{jk}}, j = 1, 2, \ldots, N; k = 1, 2, \ldots, M$$

根据公式，我们可以知道B_{jk}跟观测序列和隐藏状态均有关系，所以给定一组观测序列和对应的隐藏状态如：

O = " dry "," damp "," soggy ", I = " sunny "," cloudy "," rainy ",当然这里的数据还不够多，假设我们有足够多的数据，那么统计sunny -> dry的次数，sunny -> dryish的次数，sunny -> damp的次数，sunny - > rainy的次数，分别记作b_1, b_2, b_3, b_4，那么

$b_{\text{sunny}->\text{any observation}} = [\frac{b_1}{sum}, \frac{b_2}{sum}, \frac{b_3}{sum}, \frac{b_4}{sum}], sum = b_1 + b_2 + b_3 + b_4$ 由此可以根据O和I 算出B_{ij}。

3.初始状态概率π_i的估计$\hat{\pi}_i$为S个样本中初始状态为q_i的频率

由于监督学习需要使用训练数据，而人工标注训练数据往往代价很高，有时就会利用非监督学习的方法。

非监督学习问题

上述监督学习问题给定了大量一一对应的观察序列和隐藏序列，用最简单的概率统计方法就能求得转移矩阵，观测概率矩阵的频数，注意这里是频数而非概率。这部分的内容相对简单，但针对非监督学习问题时，由于多了隐藏变量，而系统的各种参数均未知，因此求解非监督学习问题时，就存在一定难度，本文用到的知识有极大似然估计，EM算法，基础概率论，如果对这些知识还不够熟悉的话，建议回到前言提到的链接，看完链接内容后，对理解Baum-Welch算法将大有帮助。

Baum-Welch算法

刚才提到了，非监督学习问题是为了计算模型参数λ，使得在该参数下$P(O|\lambda)$的概率最大。这个问题便是我们的极大似然估计，但$P(O|\lambda)$并非孤立的存在，其背后与隐含状态相联系。这句话应该怎么理解呢，在海藻模型中，如我们观测到某一海藻序列$O = \{" dry ", " damp ", " soggy "\}$，但是什么决定了海藻的湿度情况呢，很明显天气的因素占有很大的一部分，因此盲人在对海藻模型进行建模时，就把隐含的天气转移状态给考虑进去了。正如双硬币模型中，由于实习生b的失误，每组数据我们并不清楚是A掷的还是B掷的，遇到信息缺失的情况，就导致了用单纯的极大似然估计求导法是无法得到解析解的。

假设给定训练数据中包含S个长度为T的观测序列$\{O_1, O_2, \ldots, O_S\}$而没有对应的状态序列，目标是学习隐马尔可夫模型$\lambda = (A, B, \pi)$的参数。我们将观测序列数据看作观测数据O，状态序列数据看作不可观测的隐数据I，那么隐马尔可夫模型事实上是一个含有隐变量的概率模型

$$P(O|\lambda) = \sum_I P(O|I, \lambda)P(I|\lambda)$$

它的参数学习可以由EM算法实现。

1.确定完全数据的对数似然函数

所有观测数据写成$O = (o_1, o_2, \ldots, o_T)$，所有隐数据写成$I = (i_1, i_2, \ldots, i_T)$，完全数据是$(O, I) = (o_1, o_2, \ldots, o_T, i_1, i_2, \ldots, i_T)$。完全数据的对数似然函数是$\log P(O, I|\lambda)$。

2.EM算法的E步：求Q函数$Q(\lambda, \hat{\lambda})$

$$Q(\lambda, \hat{\lambda}) = \sum_I \log P(O, I|\lambda)P(O, I|\hat{\lambda})$$

其中，$\hat{\lambda}$是隐马尔可夫模型参数的当前估计值，λ是要极大化的隐马尔可夫模型参数。上式公式需要注意两点，第一，仅仅取$P(O, I|\lambda)$的对数，$P(O, I|\hat{\lambda})$是在对数的外面；第二，$P(O, I|\hat{\lambda})$是确定的值，即它可能为[0,1]中的任何值，根据$\hat{\lambda}$算出。如果仔细观察式子的话，该式就是对随机变量I求期望。即$E(f(I)), f(I) = \log P(O, I|\lambda)$。又

$$P(O, I|\lambda) = \pi_{i_1} b_1(o_1) a_{i_1 i_2} b_2(o_2) \cdots a_{i_{T-1} i_T} b_T(o_T)$$

于是函数$Q(\lambda, \hat{\lambda})$可以写成：

$$Q(\lambda, \hat{\lambda}) = \sum_I \log \pi_{i_1} P(O, I|\hat{\lambda}) + \sum_I \left(\sum_{t=1}^{T-1} \log a_{i_t i_{t+1}}\right) P(O, I|\hat{\lambda}) + \sum_I \left(\sum_{t=1}^{T} \log b_{i_t}(o_t)\right) P(O, I|\hat{\lambda})$$

式中求和都是对所有训练数的序列总长度T进行的。

3.EM算法的M步：极大化Q函数$Q(\lambda, \hat{\lambda})$求模型参数$A, B, \pi$

由于要极大化的参数在上式中单独地出现在3个项中，所以只需要对各项分别极大化。

（1）上式中的第一项可以写成：

$$\sum_{I} \log \pi_{i_1} P(O, I|\hat{\lambda}) = \sum_{i=1}^{N} \log \pi_i P(O, i_1 = i|\hat{\lambda})$$

注意到π_i满足约束条件$\sum_{i=1}^{N} \pi_i = 1$，利用拉格朗日乘子法，写出拉格朗日函数：

$$\sum_{i=1}^{N} \log \pi_i P(O, i_1 = i|\hat{\lambda}) + \gamma(\sum_{i=1}^{N} \pi_i - 1)$$

对其求偏导数并令结果为0

$$\frac{\partial}{\partial \pi_i}[\sum_{i=1}^{N} \log \pi_i P(O, i_1 = i|\hat{\lambda}) + \gamma(\sum_{i=1}^{N} \pi_i - 1)] = 0$$

得

$$P(O, i_1 = i|\hat{\lambda}) + \gamma \pi_i = 0$$

对i求和得到γ

$$\gamma = -P(O|\hat{\lambda})$$

于是得

$$\pi_i = \frac{P(O, i_1 = i|\hat{\lambda})}{P(O|\hat{\lambda})}$$

（2）上式中的第二项可以写成

$$\sum_{I} (\sum_{t=1}^{T-1} \log a_{i_t i_{t+1}}) P(O, I|\hat{\lambda}) = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{t=1}^{T-1} \log a_{ij} P(O, i_t = i, i_{t+1} = j|\hat{\lambda})$$

类似第一项，应用具有约束条件$\sum_{j=1}^{N} = 1$的拉格朗日乘子法可以求出

$$a_{ij} = \frac{\sum_{t=1}^{T-1} P(O, i_t = i, i_{t+1} = j|\hat{\lambda})}{\sum_{t=1}^{T-1} P(O, i_t = i|\hat{\lambda})}$$

（3）上式中的第三项可以写成

$$\sum_{I} (\sum_{t=1}^{T} \log b_{i_t}(o_t)) P(O, I|\hat{\lambda}) = \sum_{j=1}^{N} \sum_{t=1}^{T} \log b_j(o_t) P(O, i_t = j|\hat{\lambda})$$

同样用拉格朗日乘子法，约束条件是$\sum_{k=1}^{M} b_j(k) = 1$。注意，只有在$o_t = v_k$时$b_j(o_t)$对$b_j(k)$的偏导数才不为0，以$I(o_t = v_k)$表示，求得

$$b_j(k) = \frac{\sum_{t=1}^{T} P(O, i_t = j|\hat{\lambda}) I(o_t = v_k)}{\sum_{t=1}^{T} P(O, i_t = j|\hat{\lambda})}$$

正因为给出了Q函数，所以进行M步时，我们可以通过求导的方式来求得所有参数的值。但虽然知道了公式的推导过程，实际该如何操作却还是很含糊。不急，接下来我们就开始尝试把这些公式映射到物理空间中去，一步步分析它们的实际含义。

算法实际物理含义

EM算法中M步的各公式的难点在于如何求得这些概率，如a_{ij}该公式分子分母上的联合概率如何计算。其实在我看来，对隐马尔可夫模型中的各种概率计算最后均是映射到节点上去做计算。当然，我们先来观察由EM算法推导出的参数计算公式。

观察式子a_{ij}和$b_j(k)$，你会发现不管是分子，还是分母，它们都是概率计算，只不过对应的一些状态不一样。具体以a_{ij}举例，如在a_{ij}的分母上计算式子$P(O, i_t = i, i_{t+1} = j|\hat{\lambda})$，仔细想想，我们在计算什么的时候，有遇到过类似的式子？其实在阐述隐马尔可夫模型的第一个概率计算问题时，我们就做过类似的求解。概率计算是为了计算$P(O|\lambda)$的概率，但我们是把式子扩展为$\sum_{I} P(O, I|\lambda)$进行计算的。即我们需要在任何隐藏状态序列下求出$P(O|\lambda)$的概率。由此我们用前向算法和后向算法来求解该问题，很好的把概率计算问题，映射到了物理节点上去做计算，并且借助物理节点存储中间变量的特性大大的简化了算法的复杂度。

同样地，$P(O, i_t = i, i_{t+1} = j|\hat{\lambda})$不就可以看成是对

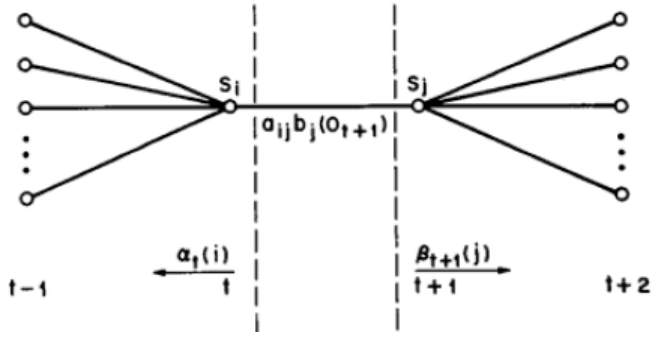
$$P(O, i_t = i, i_{t+1} = j|\hat{\lambda}) = \sum_{\text{除了}t,t+1\text{时刻外所有隐含序列}I} P(O, I|\hat{\lambda})$$

在海藻模型中，针对观测序列长度为3的情况，也就是说我们有了I的所有可能组合。如$I_1 = (\text{" sunny "}, \text{" sunny "}, \text{" sunny "}), I_2 = (\text{" sunny "}, \text{" sunny "}, \text{" rainy "}), \ldots, I_27 = (\text{" rainy "}, \text{" rainy "}, \text{" rainy "})$，我们需要求解t=2和t=3的时刻，那么我们只需要累加除了时刻2和3的其他所有节点。

于是我们就有了前向后向算法中$\xi_t(i, j)$的定义了，它的定义式为：

$$\xi_t(i, j) = P(i_t = q_i, i_{t+1} = q_j|O, \lambda)$$

该定义式是不是和我们的a_{ij}分子上式子很像？其实它们本质上就是一个东西，只是这定义在分子的基础上除了$P(O|\lambda)$的概率罢了。我们再来看图



这就是$\xi_t(i, j)$实际的物理含义了，图中所有节点的和就为，在给定模型λ和观测O，在时刻t处于状态q_i且在时刻t+1处于状态q_j的概率。a_{ij}分子上的式子我们给出了具体的求解公式，这不就是对应了上图嘛，除了t和t+1时刻，累加其他所有时刻的隐含状态（节点），主要原因在于对节点图的计算就是对$\sum_{I} P(O, I|\hat{\lambda})$的计算过程。

还记得前向算法和后向算法是如何定义中间节点的嘛，为了计算$P(O|\lambda)$，我们给每一个t时刻的隐含状态节点定义了实际的物理含义，即把它们命名为$\alpha_t(i)$和$\beta_t(i)$，两个中间变量分别从两边进行有向边加权和有向边汇聚，形成一种递归结构，并且由此不断传播至两端，对任意t=1时刻，和t=T时刻，分别进行累加就能求得$P(O|\lambda)$，我们还举出了一个小例子，来论证前向算法和后向算法只要满足有向边加权和有向边汇聚就能得到算法的一致性，今天我们根据前向后向算法做进一步的例子推广，从而真正理解$\xi_t(i,j)$的物理含义。

书中利用前向概率和后向概率的定义可以将观测序列概率$P(O|\lambda)$统一写成

$$P(O|\lambda)=\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j),t=1,2,\ldots,T-1$$

此时分别当t=1和t=T-1时，前向算法和后向算法分别成立。什么意思呢，也就是根据上式，我们从t=1时刻不断向前递推，将得到前向算法的计算公式，从t=T-1时刻不断向后递推，将得到后向算法的计算公式。这不是废话嘛，没错，但我们实际的来操作一把，注意递推的中间过程，能够帮助我们论证节点图的另外一个重要的性质，也就是节点图的推广性质。

假设我们从t =T-1时刻开始递推，那么上述式子，把t=T-1代入得

$$P(O|\lambda)=\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_{T-1}(i)a_{ij}b_j(o_T)\beta_T(j)$$

由于α是对i的累加，跟j无关，所以可以把它提到前面去，得

$$P(O|\lambda)=\sum_{i=1}^{N}\alpha_{T-1}(i)\sum_{j=1}^{N}a_{ij}b_j(o_T)\beta_T(j)$$

又因为$\beta_t(i)$的递推公式

$$\beta_t(i)=\sum_{j=1}^{N}a_{ij}b_j(o_{t+1})\beta_{t+1}(j)$$

于是得

$$P(O|\lambda)=\sum_{i=1}^{N}\alpha_{T-1}(i)\beta_{T-1}(i)$$

由$\alpha_{T-1}(i)$的递推公式得

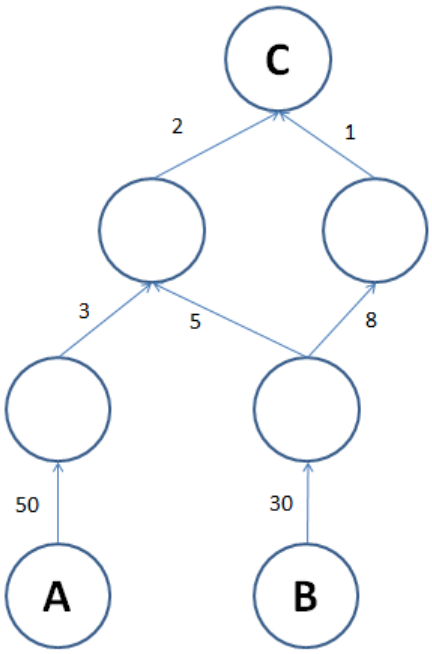
$$\begin{aligned}P(O|\lambda)&=\sum_{i=1}^{N}\beta_{T-1}(i)\sum_{j=1}^{N}\alpha_{T-2}(j)a_{ji}b_i(o_{T-1})\\&=\sum_{j=1}^{N}\sum_{i=1}^{N}\alpha_{T-2}(j)a_{ji}b_i(o_{T-1})\beta_{T-1}(i)\\&=\sum_{j=1}^{N}\alpha_{T-2}(j)\beta_{T-2}(j)\end{aligned}$$

所以说，不断的根据α,β两个递推式展开，合并，展开，合并我们能够得到

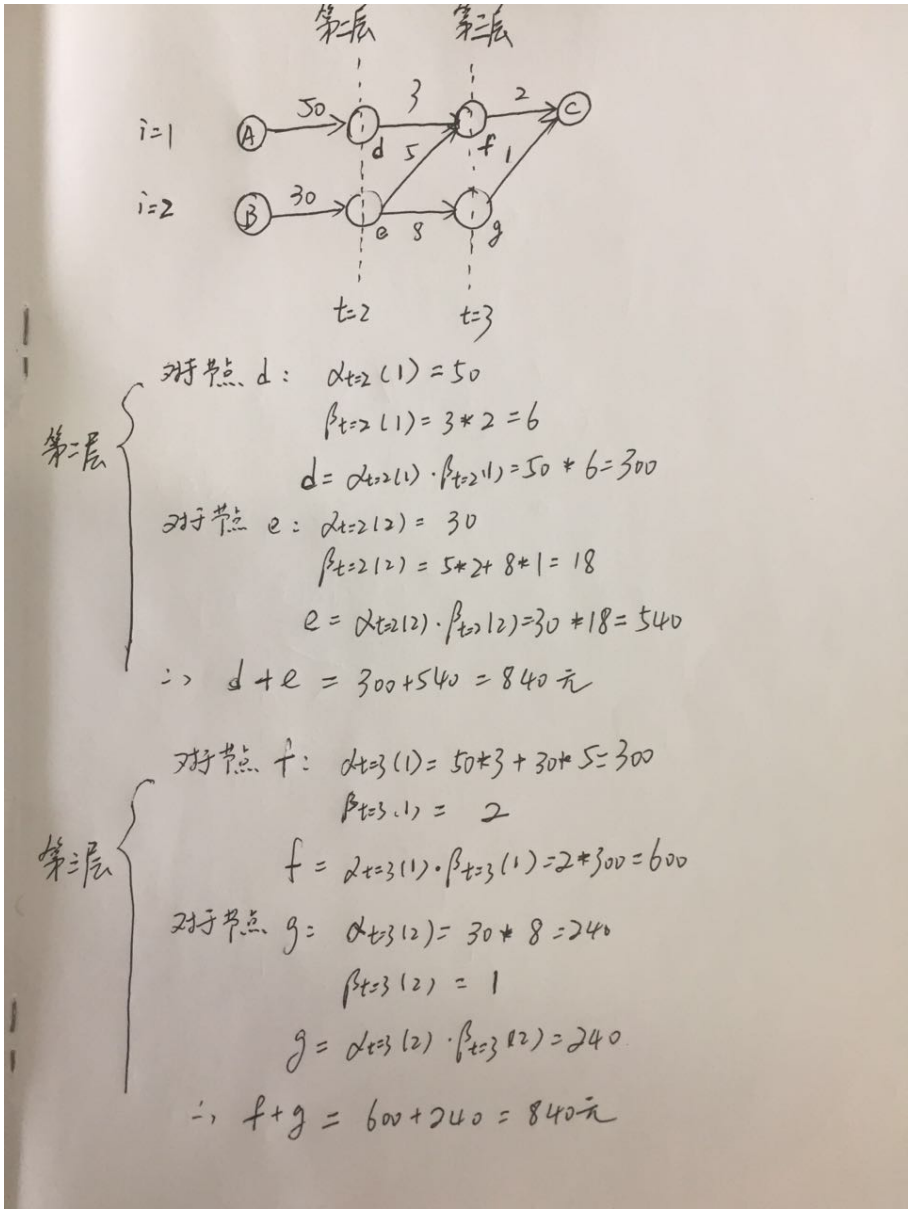
$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_{T-1}(i)\beta_{T-1}(i) = \sum_{j=1}^{N} \alpha_{T-2}(j)\beta_{T-2}(j) = \cdots = \sum_{i=1}^{N} \alpha_1(i)\beta_1(i)$$

这是根据前向算法和后向算法的递推公式推导出来的，而前向算法和后向算法的递推式则是由节点图的性质而来，两者之间是等价的，所以说上述公式能很好的论证节点的某些性质，还记得在提出后向算法时举的例子，商户卖货问题。

我们把问题用节点图表示



从A=1,B=1，向上传播得C=840元，从C=1，向下传播得A+B=840元。而$P(O|\lambda)$这个式子告诉我们，从中间某一层节点，从C出发到达该层和从A,B出发到达该层，等到的两个中间变量进行相乘，且对该层所有节点进行累加完毕后，获得的资源总数不变，即840元。不信，我们分别对第二层和第三层的节点计算一遍，如下图



由此，我们明确了$\xi_t(i,j)$的物理含义，它只是从t时刻出发，由前向算法导出的中间节点S_i和从t+1时刻出发，由后向导出的中间节点S_j，且节点S_i和S_j中间还有一条加权有向边的关系$a_{ij}\,b_j(o_{t+1})$，所以我们得

$$P(i_t=q_i,i_{t+1}=q_j,O|\lambda)=\alpha_t(i)a_{ij}\,b_j(o_{t+1})\beta_{t+1}(j)$$

$$\xi_t(i,j)=\frac{P(i_t=q_i,i_{t+1}=q_j,O|\lambda)}{P(O|\lambda)}=\frac{\alpha_t(i)a_{ij}\,b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}\,b_j(o_{t+1})\beta_{t+1}(j)}$$

也就是说，我们对a_{ij}分子的计算，我们完全可以有前向算法和后向算法中定义的各个中间变量来求出来。同理，《统计学习方法》中还定义了另外以变量

$$\gamma_t(i)=P(i_t=q_i|O,\lambda)=\frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}\,b_j(o_{t+1})\beta_{t+1}(j)}$$

于是我们有了真正的对λ计算的公式

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$b_j(k) = \frac{\sum_{t=1,o_t=v_k}^{T-1} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

$$\pi_i = \gamma_1(i)$$

算法（Baum-Welch算法）

输入：观测数据$O = (o_1, o_2, \ldots, o_T)$
输出：隐马尔可夫模型参数
（1）初始化
对$n = 0$，选取$a_{ij}^{(0)}, b_j(k)^{(0)}, \pi_i^{(0)}$，得到模型$\lambda^{(0)} = (A^{(0)}, B^{(0)}, \pi^{(0)})$
（2）递推，对$n = 1, 2, \ldots$

$$a_{ij}^{(n+1)} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$b_j^{(n+1)}(k) = \frac{\sum_{t=1,o_t=v_k}^{T-1} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

$$\pi_i^{(n+1)} = \gamma_1(i)$$

（3）终止，得到模型参数$\lambda^{(n+1)} = (A^{(n+1)}, B^{(n+1)}, \pi^{(n+1)})$

理论分析总算完毕了，简单总结一下前向后向算法，首先隐马尔可夫模型参数的估计问题是一个隐藏变量的极大似然估计，因此我们用到了EM算法来解决上述参数估计问题，从EM算法中，求得Q函数，从而能够对Q函数进行求偏导，得到极大似然函数的极值，求偏导算出了参数估计的公式，与先前$\hat{\lambda}$参数产生了关系，并进一步需要计算大量联合概率，而联合概率的计算巧妙的使用了节点图的各种性质，用中间变量降低了节点计算的复杂度，导出了对计算有帮助的定义，方便参数求解。

Code Time

Baum-Welch算法的Python实现

接着前文hmm.py和test.py文件继续添加。

1.在hmm.py中添加baum_welch_train算法

```
def baum_welch_train(self, observations, criterion=0.05):
    n_states = self.A.shape[0]
    # 观察序列的长度T
    n_samples = len(observations)

    done = False
    while not done:
        # alpha_t(i) = P(o_1,o_2,...,o_t,q_t = s_i | hmm)
        # Initialize alpha
        # 获得所有前向传播节点值 alpha_t(i)
```

```
        alpha = self._forward(observations)

        # beta_t(i) = P(o_t+1,o_t+2,...,o_T | q_t = s_i , hmm)
        # Initialize beta
        # 获得所有后向传播节点值 beta_t(i)
        beta = self._backward(observations)

        # 计算 xi_t(i,j) -> xi(i,j,t)
        xi = np.zeros((n_states, n_states, n_samples - 1))
        # 在每个时刻
        for t in range(n_samples - 1):
            # 计算P(O | hmm)
            denom = sum(alpha[:, -1])
            for i in range(n_states):
                # numer[1,:] = 行向量，alpha[i,t]=实数，slef.A[i,:] = 行向量
                # self.B[:,observations[t+1]].T = 行向量,beta[:,t+1].T = 行向量
                numer = alpha[i, t] * self.A[i, :] * self.B[:, observations[t + 1]].T * beta[:, t + 1].T
                xi[i, :, t] = numer / denom

            # 计算gamma_t(i) 就是对j进行求和
            gamma = np.sum(xi, axis=1)

            # need final gamma elements for new B
            prod = (alpha[:, n_samples - 1] * beta[:, n_samples - 1]).reshape((-1, 1))
            # 合并T时刻的节点
            gamma = np.hstack((gamma, prod / np.sum(prod)))
            # 列向量
            newpi = gamma[:, 0]
            newA = np.sum(xi, 2) / np.sum(gamma[:, :-1], axis=1).reshape((-1, 1))
            newB = np.copy(self.B)

            # 观测状态数
            num_levels = self.B.shape[1]
            sumgamma = np.sum(gamma, axis=1)
            for lev in range(num_levels):
                mask = observations == lev
                newB[:, lev] = np.sum(gamma[:, mask], axis=1) / sumgamma

            if np.max(abs(self.pi - newpi)) < criterion and \
                            np.max(abs(self.A - newA)) < criterion and \
                            np.max(abs(self.B - newB)) < criterion:
                done = 1
            self.A[:], self.B[:], self.pi[:] = newA, newB, newpi
```

2.在hmm.py中添加模拟序列生成函数

```
def simulate(self, T):
    def draw_from(probs):
        # np.random.multinomial 为多项式分布，1为实验次数，类似于投掷一枚骰子，丢出去是几，probs每个点数的概率，均为1/6
        # 给定行向量的概率，投掷次数为1次，寻找投掷的点数
        return np.where(np.random.multinomial(1, probs) == 1)[0][0]

    observations = np.zeros(T, dtype=int)
    states = np.zeros(T, dtype=int)
    states[0] = draw_from(self.pi)
    observations[0] = draw_from(self.B[states[0], :])
    for t in range(1, T):
        states[t] = draw_from(self.A[states[t - 1], :])
        observations[t] = draw_from(self.B[states[t], :])
    return observations, states
```

回到海藻模型，我们可以用这样一串代码完成Baum-Welch算法的训练，并且评估其准确率。

3.在test.py中添加测试代码

```
# 参数估计
observations_data, states_data = h.simulate(100)
guess = hmm.HMM(array([[0.33, 0.33, 0.34],
```

```
                        [0.33, 0.33, 0.34],
                        [0.33, 0.33, 0.34]]),
                array([[0.25, 0.25, 0.25, 0.25],
                        [0.25, 0.25, 0.25, 0.25],
                        [0.25, 0.25, 0.25, 0.25]]),
                array([0.7, 0.15, 0.15])
                )
guess.baum_welch_train(observations_data)
# 预测问题
states_out = guess.state_path(observations_data)[1]
p = 0.0
for s in states_data:
    if next(states_out) == s: p += 1

print(p / len(states_data))
```

经过多次测试，本算法的预测准确率在0.3~0.5。可见隐马尔可夫模型的参数估计的准确率还没有到令人满意的程度。

参考文献

1. EM算法及其推广学习笔记
2. 隐马尔可夫学习笔记（一）
3. 李航. 统计学习方法[M]. 北京：清华大学出版社，2012

- 上一篇 隐马尔可夫学习笔记（一）
- 下一篇 EM算法及其推广学习笔记