《机器学习》课程系列

HMM 模型*

武汉纺织大学数学与计算机学院

杜小勤

2020/05/09

Contents

*本系列文档属于讲义性质，仅用于学习目的。Last updated on: May 22, 2020。

1 基本概念

HMM 模型 (Hidden Markov Model) 是一种时序概率模型，用于建模由隐藏的 Markov 链随机生成观测序列的过程，它是一种生成式模型，广泛应用于语音识别、自然语言处理、生物信息、模式识别等领域。

首先，给出 HMM 模型的定义。

定义 1.1（*HMM* 模型）*HMM* 模型假设内部存在一个隐藏的 *Markov* 链和时序，在每一时序(时间步 t)，该 *Markov* 链依状态转移概率模型生成一个不可观测的随机状态，而每个状态又依观测概率模型生成一个随机观测。随着时序的演进，前者形成不可见的状态序列，后者形成可见的观测序列。

初始状态概率分布、状态转移概率分布与观测概率分布，一起共同确定了一个 HMM 模型。下面给出其形式化定义。

设 $\boldsymbol{Q}=\{q_1,q_2,\ldots,q_N\}$ 为所有可能的状态集合，$\boldsymbol{V}=\{v_1,v_2,\ldots,v_M\}$ 为所有可能的观测集合，其中 N 为状态个数，M 为观测个数。设 $\boldsymbol{I}=(I_1,I_2,\ldots,I_T)$ 为状态序列，其中 $I_t\in\boldsymbol{Q}(t=1,2,\ldots,T)$；$\boldsymbol{O}=(O_1,O_2,\ldots,O_T)$ 为观测序列，$O_t\in\boldsymbol{V}(t=1,2,\ldots,T)$，其中 T 为时序长度。

设初始概率向量为：

$$\boldsymbol{\pi}=(\pi(q_1),\pi(q_2),\ldots,\pi(q_N))^T=(\pi_1,\pi_2,\ldots,\pi_N)^T \tag{1}$$

其中 $\pi(q_j)=\pi_j=P(I_1=q_j)$ 为时刻 $t=1$ 处于状态 q_j 的概率。设 $\boldsymbol{A}$ 为 Markov 状态转移 (概率) 矩阵：

$$\boldsymbol{A}=[a_{q_i}(q_j)]_{N\times N}=[a_{ij}]_{N\times N} \tag{2}$$

其中：

$$a_{q_i}(q_j)=a_{ij}=P(I_{t+1}=q_j|I_t=q_i)\quad i=1,2,\ldots,N,\quad j=1,2,\ldots,N \tag{3}$$

表示时刻 t 处于状态 q_i 并在时刻 $t+1$ 转移到状态 q_j 的概率。设 $\boldsymbol{B}$ 为 Markov 观测 (概率) 矩阵：

$$\boldsymbol{B}=\left[b_{q_j}(v_k)\right]_{N\times M}=[b_{jk}]_{N\times M} \tag{4}$$

其中：

$$b_{q_j}(v_k)=b_{jk}=P(O_t=v_k|I_t=q_j)\quad j=1,2,\ldots,N,\quad k=1,2,\ldots,M \tag{5}$$

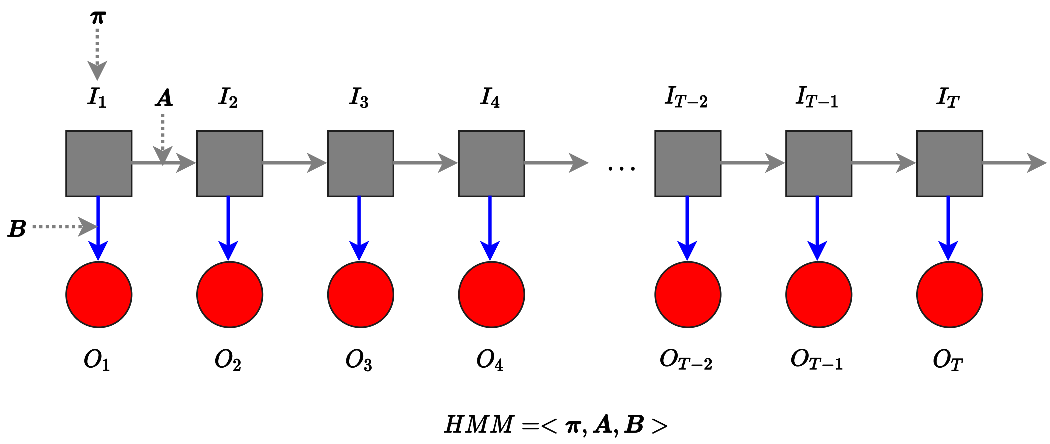

图 1-1: HMM 模型示意图

表示时刻 t 处于状态 q_j 观测到 v_k 的概率。

因此，HMM 模型由三元组 $\boldsymbol{\lambda} =< \boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B} >$ 组成，其中 $\boldsymbol{\pi}$ 和 $\boldsymbol{A}$ 生成状态序列，而 $\boldsymbol{B}$ 生成观测序列。如图1-1所示，展示了 HMM 模型示意图。图中，灰色方块构成隐状态序列，红色圆形构成观测序列，灰色实箭头表示隐状态之间的转移，蓝色实箭头表示某个状态处产生的观测。可以看出，HMM 模型是由三元组控制生成的。

实际上，在 HMM 模型的定义中，隐含了一个重要假设，即 Markov 性质，它包括状态与状态之间的转移、状态与观测之间的生成这 2 个方面的 Markov 性质——这也是 HMM 名称的由来：

- 齐次 Markov 假设。时刻 t 的状态，只依赖于时刻 $t-1$ 的状态，且与时刻 t 本身无关，也与其它所有状态以及观测无关，即假设 HMM 模型本身是由一个隐 Markov 链控制的：

$$P(I_t|I_{t-1}, O_{t-1}, \ldots, I_1, O_1) = P(I_t|I_{t-1}) \quad t = 2, 3 \ldots, T \tag{6}$$

- 观测独立性假设。时刻 t 的观测，只依赖于时刻 t 的状态，且与时刻 t 本身无关，也与其它所有状态以及观测无关：

$$P(O_t|I_t, O_{t-1}, I_{t-1} \ldots, I_1, O_1) = P(O_t|I_t) \quad t = 1, 2, \ldots, T \tag{7}$$

从图1-1中，也可以看出这 2 种 Markov 性质。

在 HMM 模型中，需要解决 3 个基本问题：

- 概率计算。

  在模型 $\boldsymbol{\lambda} =< \boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B} >$ 已知的情况下，计算观测序列 $\boldsymbol{O} = (O_1, O_2, \ldots, O_T)$ 的概率 $P(\boldsymbol{O}; \boldsymbol{\lambda})$；

- 预测或解码。

  在模型 $\boldsymbol{\lambda} =< \boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B} >$ 已知的情况下,求解与观测序列 $\boldsymbol{O} = (O_1, O_2, \ldots, O_T)$ 对应的最可能状态序列 $\boldsymbol{I} = (I_1, I_2, \ldots, I_T)$，即求解 $P(\boldsymbol{I}|\boldsymbol{O})$ 最大的状态序列；

- 学习。

  给定观测序列 $\boldsymbol{O} = (O_1, O_2, \ldots, O_T)$，估计与观测序列对应的最可能模型参数 $\boldsymbol{\lambda} =< \boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B} >$，即求解 $P(\boldsymbol{O}; \boldsymbol{\lambda})$ 最大的模型参数 $\boldsymbol{\lambda}$。

2 概率计算

本小节需要解决的问题是，在给定 HMM 模型 $\boldsymbol{\lambda} =< \boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B} >$ 的情况下，计算观测序列 $\boldsymbol{O}$ 的概率 $P(\boldsymbol{O}; \boldsymbol{\lambda})$。一种自然的方法是，利用如下公式，在所有可能的状态序列 $\boldsymbol{I}$ 上求和：

$$\begin{aligned} P(\boldsymbol{O}; \boldsymbol{\lambda}) &= \sum_{\boldsymbol{I}} P(\boldsymbol{O}, \boldsymbol{I}; \boldsymbol{\lambda}) = \sum_{\boldsymbol{I}} P(\boldsymbol{I}; \boldsymbol{\lambda}) P(\boldsymbol{O}|\boldsymbol{I}; \boldsymbol{\lambda}) \\ &= \sum_{I_1, I_2, \ldots, I_T} \pi(I_1) \cdot b_{I_1}(O_1) \cdot a_{I_1}(I_2) \cdot b_{I_2}(O_2) \ldots a_{I_{T-1}}(I_T) \cdot b_{I_T}(O_T) \end{aligned} \tag{8}$$

虽然上述方法简单直接，但是时间复杂度高——计算序列中包含大量的重复子序列。实际上，可以利用序列重复性特点，建立类似于动态规划那样的递推迭代公式[1]，即每次迭代可以直接引用前次的计算结果，从而避免了大量的重复计算。

2.1 前向算法

首先，介绍按时间步递增的计算算法：前向算法。

[1]但是，与动态规划不同的是，它没有最优化求解过程。

为了建立递推迭代公式计算 $P(\boldsymbol{O};\boldsymbol{\lambda})$(其中 $\boldsymbol{O}=(O_1,O_2,\ldots,O_T)$)，需要定义前向概率 $\alpha_t(q_i)$，它表示时刻 t 状态为 q_i 且已经生成的观测序列为 $(O_1,O_2,\ldots,O_t)$ 的概率：

$$\alpha_t(q_i)=P(O_1,O_2,\ldots,O_t,I_t=q_i;\boldsymbol{\lambda}) \tag{9}$$

在定义了前向概率后，第 1 步需要构建其初值，即 $t=1$ 时刻的值 $\alpha_1(q_i)$：

$$\alpha_1(q_i)=\pi(q_i)\cdot b_{q_i}(O_1)\quad i=1,2,\ldots,N \tag{10}$$

然后，构建其它时刻的前向概率，给出如下的递推迭代公式：

$$\alpha_t(q_i)=\left(\sum_{j=1}^{N}\alpha_{t-1}(q_j)\cdot a_{q_j}(q_i)\right)b_{q_i}(O_t)\quad \begin{matrix} i=1,2,\ldots,N \\ t=2,3,\ldots,T \end{matrix} \tag{11}$$

于是，可以得到：

$$P(\boldsymbol{O};\boldsymbol{\lambda})=\sum_{j=1}^{N}\alpha_T(q_j) \tag{12}$$

图2-2展示了前向概率计算的算法示意图。

从图2-2可以看出，前向算法之所以高效，原因在于，它充分利用了前次的计算结果，避免了直接计算算法的计算冗余性。

2.2 后向算法

既然可以依时间步递增的顺序构建递推迭代，那么也可以反过来，即依时间步递减的顺序建立递推迭代，这就是后向算法。

与前向算法一样，在后向算法中，为了建立递推迭代公式，需要先定义后向概率——在时刻 t，状态为 q_i，且从 $t+1$ 到 T 的观测序列为 $O_{t+1},O_{t+2},\ldots,O_T$ 的概率：

$$\beta_t(q_i)=P(O_{t+1},O_{t+2},\ldots,O_T|I_t=q_i;\boldsymbol{\lambda}) \tag{13}$$

在定义了后向概率后，第 1 步需要构建其初值，即 $t=T$ 时刻的值：

$$\beta_T(q_i)=1\quad i=1,2,\ldots,N \tag{14}$$

然后，构建其它时刻的后向概率，给出如下的递推迭代公式：

$$\beta_t(q_i)=\sum_{j=1}^{N}a_{q_i}(q_j)\cdot b_{q_j}(O_{t+1})\cdot\beta_{t+1}(q_j)\quad \begin{matrix} i=1,2,\ldots,N \\ t=T-1,T-2,\ldots,1 \end{matrix} \tag{15}$$

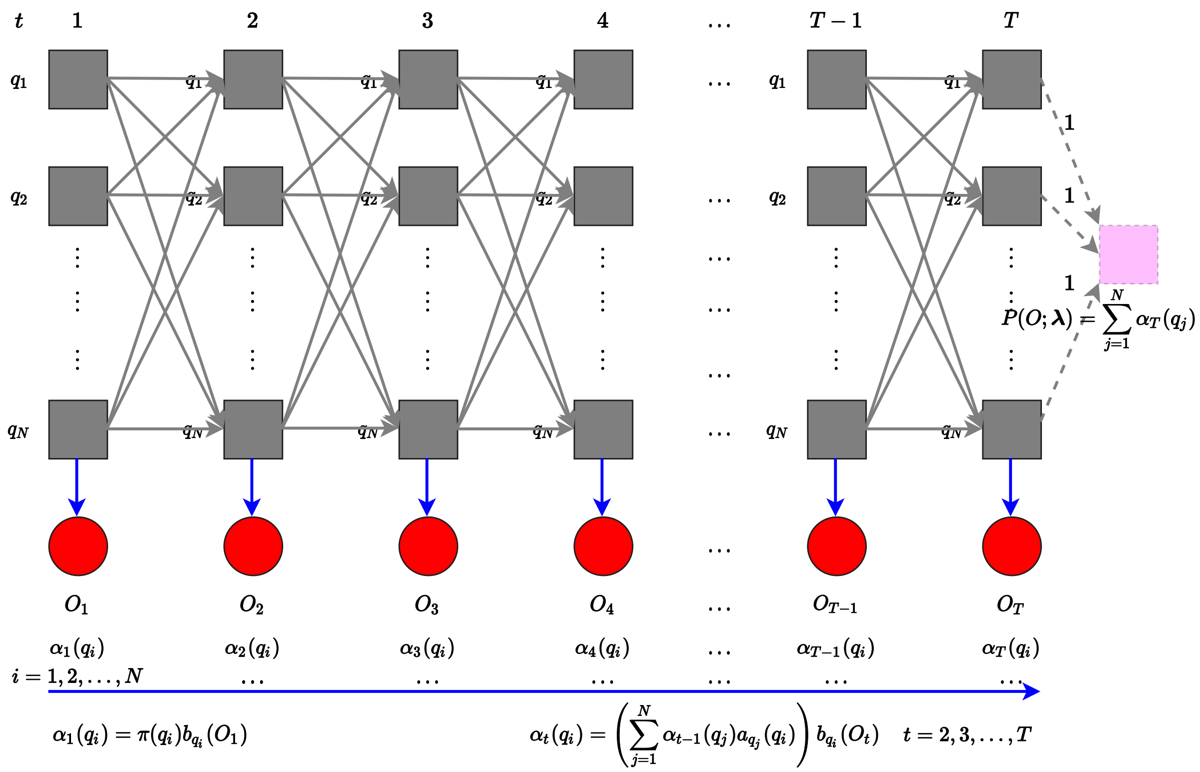

图 2-2: 前向概率计算示意图

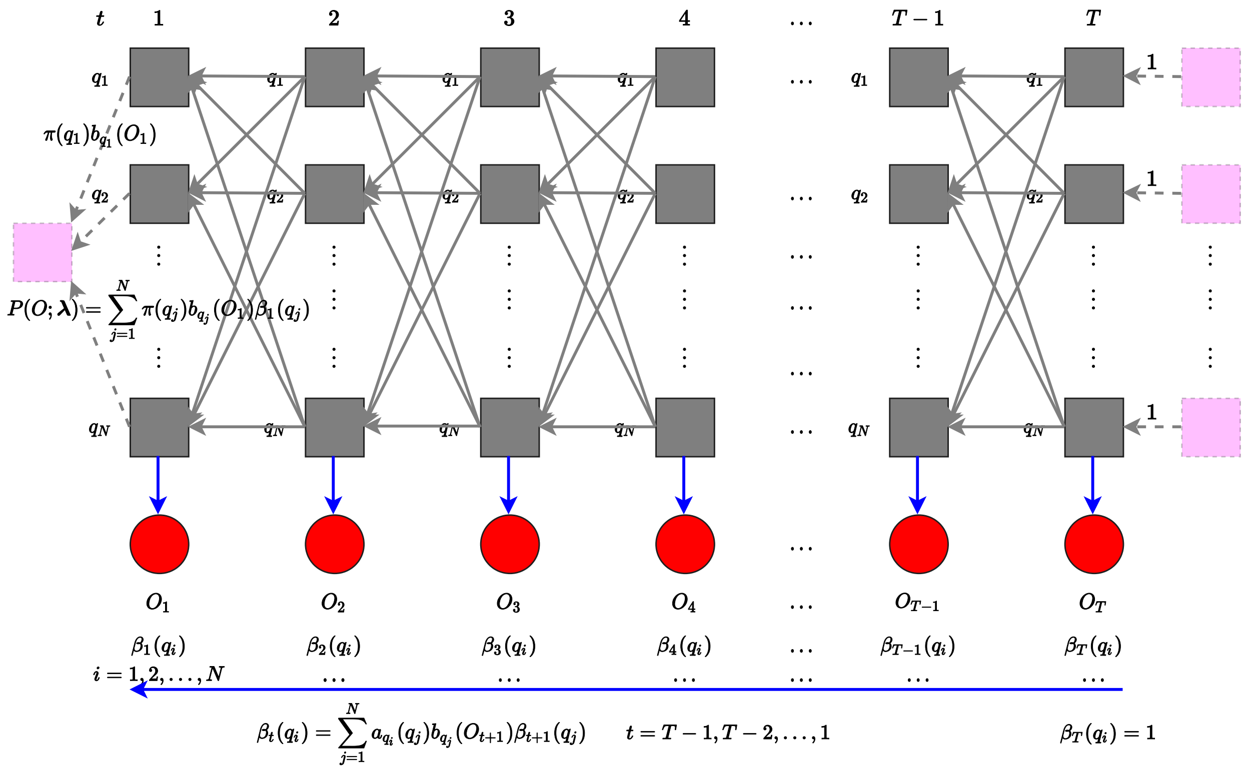

图 2-3: 后向概率计算示意图

于是，可以得到：

$$P(\boldsymbol{O};\boldsymbol{\lambda}) = \sum_{j=1}^{N} \pi(q_j) \cdot b_{q_j}(O_1) \cdot \beta_1(q_j) \tag{16}$$

图2-3展示了后向概率计算的算法示意图。

2.3 常见概率的计算

在定义了前向与后向概率之后，下面给出一些概率计算的实例：

- 观测序列的概率。

  除了公式 (12) 和公式 (16) 外，下面的公式，同样可以计算给定观测序列的概率——利用前向概率与后向概率的定义特点，将 2 者混合使用：

$$P(\boldsymbol{O};\boldsymbol{\lambda}) = \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t(q_i) \cdot a_{q_i}(q_j) \cdot b_{q_j}(O_{t+1}) \cdot \beta_{t+1}(q_j) \quad t = 1, 2, \dots, T-1 \tag{17}$$

- 单状态的概率。

  在给定模型参数 $\boldsymbol{\lambda}$ 和观测序列 $\boldsymbol{O}$ 的情况下，计算时刻 t 处于状态 q_i 的概率 $P(I_t = q_i|\boldsymbol{O};\boldsymbol{\lambda})$，并令 $\gamma_t(q_i) = P(I_t = q_i|\boldsymbol{O};\boldsymbol{\lambda})$：

$$\gamma_t(q_i) = P(I_t = q_i|\boldsymbol{O};\boldsymbol{\lambda}) = \frac{P(I_t = q_i,\boldsymbol{O};\boldsymbol{\lambda})}{P(\boldsymbol{O};\boldsymbol{\lambda})} = \frac{P(I_t = q_i,\boldsymbol{O};\boldsymbol{\lambda})}{\sum\limits_{j=1}^{N} P(I_t = q_j,\boldsymbol{O};\boldsymbol{\lambda})} \tag{18}$$

  再次利用前向概率与后向概率的定义特点，得到：

$$\begin{aligned} &P(I_t = q_i,\boldsymbol{O};\boldsymbol{\lambda}) = \alpha_t(q_i)\cdot\beta_t(q_i) \quad \Rightarrow \\ &\gamma_t(q_i) = P(I_t = q_i|\boldsymbol{O};\boldsymbol{\lambda}) = \frac{\alpha_t(q_i)\cdot\beta_t(q_i)}{\sum\limits_{j=1}^{N}\alpha_t(q_j)\cdot\beta_t(q_j)} \end{aligned} \tag{19}$$

  其中，$P(\boldsymbol{O};\boldsymbol{\lambda}) = \sum\limits_{j=1}^{N}\alpha_t(q_j)\cdot\beta_t(q_j)$，这是继公式 (12)、公式 (16) 与公式 (17) 之后的第 4 种形式。

- 连续双状态的概率。

  在给定模型参数 $\boldsymbol{\lambda}$ 和观测序列 $\boldsymbol{O}$ 的情况下，计算时刻 t 处于状态 q_i 且在时刻 $t+1$ 处于状态 q_j 的概率 $P(I_t = q_i, I_{t+1} = q_j|\boldsymbol{O};\boldsymbol{\lambda})$，并令 $\xi_t(q_i, q_j) = P(I_t = q_i, I_{t+1} = q_j|\boldsymbol{O};\boldsymbol{\lambda})$：

$$\xi_t(q_i,q_j) = P(I_t = q_i, I_{t+1} = q_j|\boldsymbol{O};\boldsymbol{\lambda}) = \frac{P(I_t = q_i, I_{t+1} = q_j,\boldsymbol{O};\boldsymbol{\lambda})}{\sum\limits_{k=1}^{N}\sum\limits_{l=1}^{N} P(I_t = q_k, I_{t+1} = q_l,\boldsymbol{O};\boldsymbol{\lambda})} \tag{20}$$

  再次利用前向概率与后向概率的定义特点，得到：

$$\begin{aligned} &P(I_t = q_i, I_{t+1} = q_j|\boldsymbol{O};\boldsymbol{\lambda}) = \alpha_t(q_i)\cdot a_{q_i}(q_j)\cdot b_{q_j}(O_{t+1})\cdot\beta_{t+1}(q_j) \quad \Rightarrow \\ &\xi_t(q_i,q_j) = P(I_t = q_i, I_{t+1} = q_j|\boldsymbol{O};\boldsymbol{\lambda}) = \frac{\alpha_t(q_i)\cdot a_{q_i}(q_j)\cdot b_{q_j}(O_{t+1})\cdot\beta_{t+1}(q_j)}{\sum\limits_{k=1}^{N}\sum\limits_{l=1}^{N}\alpha_t(q_k)\cdot a_{q_k}(q_l)\cdot b_{q_l}(O_{t+1})\cdot\beta_{t+1}(q_l)} \end{aligned} \tag{21}$$

  其中，$P(\boldsymbol{O};\boldsymbol{\lambda}) = \sum\limits_{k=1}^{N}\sum\limits_{l=1}^{N}\alpha_t(q_k)\cdot a_{q_k}(q_l)\cdot b_{q_l}(O_{t+1})\cdot\beta_{t+1}(q_l)$，这是本小节中第 5 种计算 $P(\boldsymbol{O};\boldsymbol{\lambda})$ 的方式。

在定义了 γ_t 和 ξ_t 这 2 个量之后，可以再次利用它们计算出一些有用的信息：

- 在给定模型参数 $\boldsymbol{\lambda}$ 和观测序列 $\boldsymbol{O}$ 的情况下，状态 q_i 出现的概率为：
$$\sum_{t=1}^{T} \gamma_t(q_i) \tag{22}$$
- 在给定模型参数 $\boldsymbol{\lambda}$ 和观测序列 $\boldsymbol{O}$ 的情况下，由状态 q_i 转出的概率为：
$$\sum_{t=1}^{T-1} \gamma_t(q_i) \tag{23}$$
- 在给定模型参数 $\boldsymbol{\lambda}$ 和观测序列 $\boldsymbol{O}$ 的情况下，由状态 q_i 转移到 q_j 的概率：
$$\sum_{t=1}^{T-1} \xi_t(q_i, q_j) \tag{24}$$

3 预测算法

预测算法的目标是，在给定模型参数 $\boldsymbol{\lambda}$ 和观测序列 $\boldsymbol{O}$ 的情况下，求解出与观测序列 $\boldsymbol{O}$ 匹配的最佳状态序列 $\boldsymbol{I}^*$，即：
$$\boldsymbol{I}^* = \arg\max_{\boldsymbol{I}} P(\boldsymbol{I}|\boldsymbol{O}) \tag{25}$$

3.1 贪心算法

贪心算法的基本思想是，在每个时刻 t，选取该时刻最可能出现的状态 I_t^*，这样就获得了一个“最优”状态序列 $\boldsymbol{I}^* = (I_1^*, I_2^*, \ldots, I_T^*)$。

如何选取时刻 t 最可能出现的状态呢？在前文中，我们定义了一个量 $\gamma_t(q_i)$，它表示时刻 t 状态 q_i 出现的概率。于是，简单地求解下式，可以获得时刻 t 的“最优”状态：
$$I_t^* = \arg\max_{1\leqslant i\leqslant N} \gamma_t(q_i) \quad t = 1, 2, \ldots, T \tag{26}$$
其中，$\gamma_t(q_i)$ 为：
$$\gamma_t(q_i) = P(I_t = q_i|\boldsymbol{O}; \boldsymbol{\lambda}) = \frac{\alpha_t(q_i)\cdot\beta_t(q_i)}{\sum\limits_{j=1}^{N} \alpha_t(q_j)\cdot\beta_t(q_j)} \tag{27}$$

虽然贪心算法非常简洁高效，但是它获得的解往往是局部最优解，只有在满足贪心选择性质时，整体最优解与局部最优解才是一致的[2]。另外，利用贪心算法获得的状态序列，可能存在实际情况中并不存在的相邻状态——例如，$I_{t-1} = q_i$ 和 $I_t = q_j$，而 $a_{q_i}(q_j) = 0$。

[2]参见《人工智能》课程系列之《Python 算法基础》，Chapter2-CN.pdf。

3.2 动态规划：Viterbi 算法

从前面的讨论可以看出，给定模型参数 $\boldsymbol{\lambda}$ 和观测序列 $\boldsymbol{O}$ 的情况下，求解与观测序列 $\boldsymbol{O}$ 匹配的最佳隐状态序列 $\boldsymbol{I}^*$，这是一个最优化问题 (公式 (25))。

如果将序列 $\boldsymbol{I}=(I_1,I_2,\ldots,I_T)$ 看作一条路径，那么问题将演变为求解 $P(\boldsymbol{I}|\boldsymbol{O})$ 值最大的那条路径。而路径求解问题，满足最优性原理或最优子结构性质——整体解与子问题的解满足一致性原理。于是，可以使用动态规划进行求解[3]。

因此，依据动态规划的算法流程，我们可以构造求解 (概率最大) 最优子序列的递推迭代公式，然后从初值 ($t=1$ 时刻) 开始，迭代应用递推式，直至 $t=T$ 时，算法停止。于是，获得了 (概率最大) 最优整体序列；最后，从目标节点进行回溯，便可以得到最佳路径，即最佳状态序列。

首先，定义时刻 t 到达状态 q_i 的最优子序列。它指的是，在给定模型参数 $\boldsymbol{\lambda}$ 和观测序列 $\boldsymbol{O}$ 的情况下，时刻 t 处于状态 q_i 时的最大条件概率：

$$P^*(I_t=q_i,I_{t-1},\ldots,I_2,I_1|O_t,O_{t-1},\ldots,O_2,O_1;\boldsymbol{\lambda}) \tag{28}$$

所对应的子序列 $I_1^*,I_2^*,\ldots,I_{t-1}^*$。于是：

$$\begin{aligned}
&P^*(I_t=q_i,I_{t-1},\ldots,I_2,I_1|O_t,O_{t-1},\ldots,O_2,O_1;\boldsymbol{\lambda})\\
&=\max_{I_1,I_2,\ldots,I_{t-1}} P(I_t=q_i,I_{t-1},\ldots,I_2,I_1|O_t,O_{t-1},\ldots,O_2,O_1;\boldsymbol{\lambda})\\
&=\max_{I_1,I_2,\ldots,I_{t-1}} \frac{P(I_t=q_i,I_{t-1},\ldots,I_2,I_1,O_t,O_{t-1},\ldots,O_2,O_1;\boldsymbol{\lambda})}{P(O_t,O_{t-1},\ldots,O_2,O_1;\boldsymbol{\lambda})}\\
&=\max_{I_1,I_2,\ldots,I_{t-1}} P(I_t=q_i,I_{t-1},\ldots,I_2,I_1,O_t,O_{t-1},\ldots,O_2,O_1;\boldsymbol{\lambda})
\end{aligned} \tag{29}$$

上式中最后一行的推导，利用了 $P(O_t,O_{t-1},\ldots,O_2,O_1;\boldsymbol{\lambda})$ 关于 $I_1,I_2,\ldots,I_{t-1}$ 的所有序列为常数的事实。为了推导方便，令：

$$\begin{aligned}
&\delta_t(q_i)\\
&=\max_{I_1,I_2,\ldots,I_{t-1}} P(I_t=q_i,I_{t-1},\ldots,I_2,I_1,O_t,O_{t-1},\ldots,O_2,O_1;\boldsymbol{\lambda})\\
&=\max_{I_1,I_2,\ldots,I_{t-1}} P(I_t=q_i,O_t,I_{t-1},O_{t-1},\ldots,I_1,O_1;\boldsymbol{\lambda}) \quad i=1,2,\ldots,N
\end{aligned} \tag{30}$$

[3] 参见《人工智能》课程系列之《Python 算法基础》，Chapter2-CN.pdf。

由此，建立动态规划的最优递推迭代公式为：

$$
\begin{aligned}
&\delta_{t+1}(q_i) \\
&= \max_{I_1,I_2,\ldots,I_t} P(I_{t+1}=q_i, O_{t+1}, I_t, O_t, \ldots, I_1, O_1; \boldsymbol{\lambda}) \\
&= \max_{I_1,I_2,\ldots,I_t=q_j,1\leqslant j\leqslant N} \delta_t(q_j)\cdot a_{q_j}(q_i)\cdot b_{q_i}(O_{t+1}) \quad i=1,2,\ldots,N \quad t=1,2,\ldots,T-1
\end{aligned}
\tag{31}
$$

根据定义，可以得到 δ 的初值：

$$
\delta_1(q_i)=\pi(q_i)\cdot b_{q_i}(O_1) \quad i=1,2,\ldots,N \tag{32}
$$

然后，定义时刻 $t+1$ 处于状态 q_i 时所对应的最优子序列 (或路径)[4]$I_1^*, I_2^*, \ldots, I_t^*$ 中的终端节点 (即 I_t^*) 为：

$$
\begin{aligned}
\psi_{t+1}(q_i) = I_t^* &= \arg\max_{q_j,1\leqslant j\leqslant N} \delta_t(q_j)\cdot a_{q_j}(q_i)\cdot b_{q_i}(O_{t+1}) \\
&= \arg\max_{q_j,1\leqslant j\leqslant N} \delta_t(q_j)\cdot a_{q_j}(q_i) \quad i=1,2,\ldots,N \quad t=1,2,\ldots,T-1
\end{aligned}
\tag{33}
$$

其初始值均设置为：

$$
\psi_1(q_i)=0 \quad i=1,2,\ldots,N \tag{34}
$$

原因是，路径起始点没有父节点。而在 $t=T$ 时刻：

$$
I_T^* = \arg\max_{q_i,1\leqslant i\leqslant N} \delta_T(q_i) \tag{35}
$$

I_T^* 即为整个最优序列或路径的终端节点 (目标)，从它出发，利用公式 (33) 进行回溯，即可得到最优路径。

下面给出 Viterbi 算法。

算法 3.1（*Viterbi* 算法）

[4]根据前面的定义，该最优子序列由 $\delta_{t+1}(q_i)$ 生成。

Input:

Model Parameters: $\boldsymbol{\lambda} =< \boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B} >$

Observation Sequence: $\boldsymbol{O} = (O_1, O_2, \ldots, O_T)$

Output:

Best Path: $\boldsymbol{I}^* = (I_1^*, I_2^*, \ldots, I_T^*)$

Algorithm:

$\delta_1(q_i) = \pi(q_i) \cdot b_{q_i}(O_1) \quad i = 1, 2, \ldots, N$

$\psi_1(q_i) = 0 \quad i = 1, 2, \ldots, N$

for $t = 1, 2 \cdots T-1$:

$$\delta_{t+1}(q_i) = \max_{I_1, I_2, \ldots, I_t = q_j, 1 \leqslant j \leqslant N} \delta_t(q_j) \cdot a_{q_j}(q_i) \cdot b_{q_i}(O_{t+1}) \quad i = 1, 2, \ldots, N$$
$$\psi_{t+1}(q_i) = \arg \max_{q_j, 1 \leqslant j \leqslant N} \delta_t(q_j) \cdot a_{q_j}(q_i) \quad i = 1, 2, \ldots, N$$

$I_T^* = \arg \max_{q_i, 1 \leqslant i \leqslant N} \delta_T(q_i)$

for $t = T-1 \cdots 1$:

$I_t^* = \psi_{t+1}(I_{t+1}^*)$

return $\boldsymbol{I}^* = (I_1^*, I_2^*, \ldots, I_T^*)$

4 学习算法

HMM 模型的学习分为 2 种：监督学习与无监督学习。前者的训练数据集，既包括观测序列，也包括对应的状态序列。而后者的训练数据集，只包括观测序列。

4.1 监督学习

设训练数据集 $\mathbf{T} = \{(\boldsymbol{O}_1, \boldsymbol{I}_1), (\boldsymbol{O}_2, \boldsymbol{I}_2) \ldots, (\boldsymbol{O}_S, \boldsymbol{I}_S)\}$，其中 S 表示训练数据集的样本个数，$\boldsymbol{O}_s$ 表示第 s 个样本的观测序列，$\boldsymbol{I}_s$ 表示第 s 个样本的状态序列，$s = 1, 2, \ldots, S$。

训练数据集 $\mathbf{T}$ 的似然函数为：

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{\lambda}) &= P\left(\boldsymbol{O},\boldsymbol{I};\boldsymbol{\lambda}\right) = \prod_{s=1}^{S} P(\boldsymbol{O}_s,\boldsymbol{I}_s;\boldsymbol{\lambda}) \\
&= \prod_{s=1}^{S} \pi(I_{s,1})\cdot b_{I_{s,1}}(O_{s,1})\cdot a_{I_{s,1}}(I_{s,2})\cdot b_{I_{s,2}}(O_{s,2})\ldots a_{I_{s,T-1}}(I_{s,T})\cdot b_{I_{s,T}}(O_{s,T}) \\
&= \prod_{s=1}^{S} \pi(I_{s,1})\cdot \prod_{s=1}^{S}\prod_{t=1}^{T-1} a_{I_{s,t}}(I_{s,t+1})\cdot \prod_{s=1}^{S}\prod_{t=1}^{T} b_{I_{s,t}}(O_{s,t})
\end{aligned}
\tag{36}
$$

两端取对数，得到对数似然函数：

$$
\log \mathcal{L}(\boldsymbol{\lambda}) = \sum_{s=1}^{S}\log\pi(I_{s,1}) + \sum_{s=1}^{S}\sum_{t=1}^{T-1}\log a_{I_{s,t}}(I_{s,t+1}) + \sum_{s=1}^{S}\sum_{t=1}^{T}\log b_{I_{s,t}}(O_{s,t}) \tag{37}
$$

观察到模型的三个参数相互独立，可以独立求解。首先，求解 $\pi(I_1=q_i)=\pi(q_i)$(其中 $i=1,2,\ldots,N$)，它需要满足约束条件 $\sum\limits_{i=1}^{N}\pi(I_1=q_i)=1$，列出拉格朗日优化函数如下[5]：

$$
L(\boldsymbol{\pi}) = -\sum_{s=1}^{S}\log\pi(I_{s,1}) + \alpha\left(\sum_{i=1}^{N}\pi(I_1=q_i)-1\right) \tag{38}
$$

其中 α 为拉格朗日乘数。求解偏导数 $\frac{\partial L(\boldsymbol{\pi})}{\partial\pi(I_1=q_i)}$，并令其等于 0：

$$
\begin{aligned}
&\frac{\partial L(\boldsymbol{\pi})}{\partial\pi(I_1=q_i)} = -\sum_{s=1}^{S}\frac{I\{I_{s,1}=q_i\}}{\pi(I_1=q_i)} + \alpha = -\frac{\sum\limits_{s=1}^{S} I\{I_{s,1}=q_i\}}{\pi(I_1=q_i)} + \alpha = 0 \quad \Rightarrow \\
&\pi(I_1=q_i)\alpha = \sum_{s=1}^{S} I\{I_{s,1}=q_i\}
\end{aligned}
\tag{39}
$$

其中 $I\{I_{s,1}=q_i\}$ 为指示函数，当花括号中所列条件满足时，其值为 1，否则为 0。利用公式 $\sum\limits_{i=1}^{N}\pi(I_1=q_i)=1$，对上式两端按 i 求和，得到：

$$
\alpha = \sum_{i=1}^{N}\sum_{s=1}^{S} I\{I_{s,1}=q_i\} = \sum_{s=1}^{S}\sum_{i=1}^{N} I\{I_{s,1}=q_i\} = S \tag{40}
$$

于是，得到：

$$
\pi(I_1=q_i) = \pi(q_i) = \frac{\sum\limits_{s=1}^{S} I\{I_{s,1}=q_i\}}{\alpha} = \frac{\sum\limits_{s=1}^{S} I\{I_{s,1}=q_i\}}{S} \tag{41}
$$

[5]按照惯例，将最大化问题转换为等价的最小化问题。另外，$\pi(I_1=q_i)$ 也需要满足不等式约束 $\pi(I_1=q_i)\geqslant 0$，为求解方便，没有列出。只需在求解完毕之后，验证此约束条件是否满足即可。

上式意义非常明确。容易验证，$\pi(I_1 = q_i) \geqslant 0$ 成立。

下面,求解 $a_{I_t=q_i}(I_{t+1} = q_j) = a_{q_i}(q_j)$(其中 $i, j = 1, 2, \ldots, N$,$t = 1, 2, \ldots, T-1$)，它需要满足约束条件 $\sum\limits_{j=1}^{N} a_{q_i}(q_j) = 1$，列出拉格朗日优化函数如下[6]：

$$L(\boldsymbol{A}) = -\sum_{s=1}^{S}\sum_{t=1}^{T-1} \log a_{I_{s,t}}(I_{s,t+1}) + \alpha \left(\sum_{j=1}^{N} a_{q_i}(q_j) - 1\right) \tag{42}$$

其中 α 为拉格朗日乘数。求解偏导数 $\frac{\partial L(\boldsymbol{A})}{\partial a_{q_i}(q_j)}$，并令其等于 0：

$$\begin{aligned}
\frac{\partial L(\boldsymbol{A})}{\partial a_{q_i}(q_j)} &= -\sum_{s=1}^{S}\sum_{t=1}^{T-1} \frac{I\{I_{s,t} = q_i, I_{s,t+1} = q_j\}}{a_{q_i}(q_j)} + \alpha \\
&= -\frac{\sum\limits_{s=1}^{S}\sum\limits_{t=1}^{T-1} I\{I_{s,t} = q_i, I_{s,t+1} = q_j\}}{a_{q_i}(q_j)} + \alpha = 0 \quad \Rightarrow \\
a_{q_i}(q_j)\alpha &= \sum_{s=1}^{S}\sum_{t=1}^{T-1} I\{I_{s,t} = q_i, I_{s,t+1} = q_j\}
\end{aligned} \tag{43}$$

利用公式 $\sum\limits_{j=1}^{N} a_{q_i}(q_j) = 1$，对上式两端按 j 求和，得到：

$$\begin{aligned}
\alpha &= \sum_{j=1}^{N}\sum_{s=1}^{S}\sum_{t=1}^{T-1} I\{I_{s,t} = q_i, I_{s,t+1} = q_j\} \\
&= \sum_{s=1}^{S}\sum_{t=1}^{T-1}\sum_{j=1}^{N} I\{I_{s,t} = q_i, I_{s,t+1} = q_j\} \\
&= \sum_{s=1}^{S}\sum_{t=1}^{T-1} I\{I_{s,t} = q_i\}
\end{aligned} \tag{44}$$

于是，得到：

$$a_{q_i}(q_j) = \frac{\sum\limits_{s=1}^{S}\sum\limits_{t=1}^{T-1} I\{I_{s,t} = q_i, I_{s,t+1} = q_j\}}{\alpha} = \frac{\sum\limits_{s=1}^{S}\sum\limits_{t=1}^{T-1} I\{I_{s,t} = q_i, I_{s,t+1} = q_j\}}{\sum\limits_{s=1}^{S}\sum\limits_{t=1}^{T-1} I\{I_{s,t} = q_i\}} \tag{45}$$

上式意义非常明确。容易验证，$a_{q_i}(q_j) \geqslant 0$ 成立。

最后，求解 $b_{I_t=q_j}(O_t = v_k) = b_{q_j}(v_k)$(其中 $j = 1, 2, \ldots, N$，$k = 1, 2, \ldots, M$，$t = 1, 2, \ldots, T$)，它需要满足约束条件 $\sum\limits_{k=1}^{M} b_{q_j}(v_k) = 1$，列出拉格朗日优化函数如

[6]基于同样的理由，不列出不等式约束，求解完毕后再验证。

下[7]：

$$L(\boldsymbol{B}) = -\sum_{s=1}^{S}\sum_{t=1}^{T}\log b_{I_{s,t}}(O_{s,t}) + \alpha\left(\sum_{k=1}^{M} b_{q_j}(v_k) - 1\right) \tag{46}$$

其中 α 为拉格朗日乘数。求解偏导数 $\frac{\partial L(\boldsymbol{B})}{\partial b_{q_j}(v_k)}$，并令其等于 0：

$$\begin{aligned}
\frac{\partial L(\boldsymbol{B})}{\partial b_{q_j}(v_k)} &= -\sum_{s=1}^{S}\sum_{t=1}^{T}\frac{I\{I_{s,t}=q_j, O_{s,t}=v_k\}}{b_{q_j}(v_k)} + \alpha \\
&= -\frac{\sum\limits_{s=1}^{S}\sum\limits_{t=1}^{T} I\{I_{s,t}=q_j, O_{s,t}=v_k\}}{b_{q_j}(v_k)} + \alpha = 0 \quad \Rightarrow \\
b_{q_j}(v_k)\alpha &= \sum_{s=1}^{S}\sum_{t=1}^{T} I\{I_{s,t}=q_j, O_{s,t}=v_k\}
\end{aligned} \tag{47}$$

利用公式 $\sum\limits_{k=1}^{M} b_{q_j}(v_k) = 1$，对上式两端按 k 求和，得到：

$$\begin{aligned}
\alpha &= \sum_{k=1}^{M}\sum_{s=1}^{S}\sum_{t=1}^{T} I\{I_{s,t}=q_j, O_{s,t}=v_k\} \\
&= \sum_{s=1}^{S}\sum_{t=1}^{T}\sum_{k=1}^{M} I\{I_{s,t}=q_j, O_{s,t}=v_k\} \\
&= \sum_{s=1}^{S}\sum_{t=1}^{T} I\{I_{s,t}=q_j\}
\end{aligned} \tag{48}$$

于是，得到：

$$b_{q_j}(v_k) = \frac{\sum\limits_{s=1}^{S}\sum\limits_{t=1}^{T} I\{I_{s,t}=q_j, O_{s,t}=v_k\}}{\alpha} = \frac{\sum\limits_{s=1}^{S}\sum\limits_{t=1}^{T} I\{I_{s,t}=q_j, O_{s,t}=v_k\}}{\sum\limits_{s=1}^{S}\sum\limits_{t=1}^{T} I\{I_{s,t}=q_j\}} \tag{49}$$

上式意义非常明确。容易验证，$b_{q_j}(v_k) \geqslant 0$ 成立。

4.2 Baum-Welch 算法

在一些情况下，如果缺少标注数据，那么训练数据集 $\mathbf{T}$ 中就没有与观测序列对应的状态序列，即只有 $\mathbf{T} = \{\boldsymbol{O}_1, \boldsymbol{O}_2, \ldots, \boldsymbol{O}_S\}$(其中 S 表示样本个数，$\boldsymbol{O}_s$ 表示第 s 个样本的观测序列，$s = 1, 2, \ldots, S$)。在这种情况下，如何估计模型参数 $\boldsymbol{\lambda} =< \boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B} >$？

[7]基于同样的理由，不列出不等式约束，求解完毕后再验证。

此时，可以为每个观测序列 $\boldsymbol{O}_s$ 引入相应的隐状态序列 $\boldsymbol{I}$。于是，HMM 模型成为一个含隐变量的概率模型：

$$\mathcal{L}(\boldsymbol{\lambda}) = P(\boldsymbol{O};\boldsymbol{\lambda}) = \prod_{s=1}^{S} P(\boldsymbol{O}_s;\boldsymbol{\lambda}) = \prod_{s=1}^{S}\sum_{\boldsymbol{I}} P(\boldsymbol{O}_s,\boldsymbol{I};\boldsymbol{\lambda}) \tag{50}$$

对照 EM 算法，直接写出其 Q 函数：

$$Q(\boldsymbol{\lambda}^{t+1},\boldsymbol{\lambda}^{t}) = \sum_{s=1}^{S}\sum_{\boldsymbol{I}} P(\boldsymbol{I}|\boldsymbol{O}_s;\boldsymbol{\lambda}^{t})\log P(\boldsymbol{O}_s,\boldsymbol{I};\boldsymbol{\lambda}^{t+1}) \tag{51}$$

其中，$\boldsymbol{\lambda}^{t+1}$ 是 Q 函数的优化目标；$P(\boldsymbol{O}_s,\boldsymbol{I};\boldsymbol{\lambda}^{t+1})$ 为：

$$P(\boldsymbol{O}_s,\boldsymbol{I};\boldsymbol{\lambda}^{t+1}) = \pi^{t+1}(I_1)\cdot b_{I_1}^{t+1}(O_{s,1})\cdot a_{I_1}^{t+1}(I_2)\cdot b_{I_2}^{t+1}(O_{s,2})\ldots a_{I_{T-1}}^{t+1}(I_T)\cdot b_{I_T}^{t+1}(O_{s,T}) \tag{52}$$

$\log P(\boldsymbol{O}_s,\boldsymbol{I};\boldsymbol{\lambda}^{t+1})$ 为：

$$\log P(\boldsymbol{O}_s,\boldsymbol{I};\boldsymbol{\lambda}^{t+1}) = \log\pi^{t+1}(I_1) + \sum_{\tau=1}^{T-1}\log a_{I_\tau}^{t+1}(I_{\tau+1}) + \sum_{\tau=1}^{T}\log b_{I_\tau}^{t+1}(O_{s,\tau}) \tag{53}$$

于是，Q 函数可写为：

$$\begin{aligned}
Q(\boldsymbol{\lambda}^{t+1},\boldsymbol{\lambda}^{t}) &= \sum_{s=1}^{S}\sum_{\boldsymbol{I}} P(\boldsymbol{I}|\boldsymbol{O}_s;\boldsymbol{\lambda}^{t})\log P(\boldsymbol{O}_s,\boldsymbol{I};\boldsymbol{\lambda}^{t+1})\\
&= \sum_{s=1}^{S}\sum_{\boldsymbol{I}} P(\boldsymbol{I}|\boldsymbol{O}_s;\boldsymbol{\lambda}^{t})\left(\log\pi^{t+1}(I_1) + \sum_{\tau=1}^{T-1}\log a_{I_\tau}^{t+1}(I_{\tau+1}) + \sum_{\tau=1}^{T}\log b_{I_\tau}^{t+1}(O_{s,\tau})\right)\\
&= \sum_{s=1}^{S}\sum_{\boldsymbol{I}} P(\boldsymbol{I}|\boldsymbol{O}_s;\boldsymbol{\lambda}^{t})\log\pi^{t+1}(I_1) +\\
&\quad \sum_{s=1}^{S}\sum_{\boldsymbol{I}} P(\boldsymbol{I}|\boldsymbol{O}_s;\boldsymbol{\lambda}^{t})\sum_{\tau=1}^{T-1}\log a_{I_\tau}^{t+1}(I_{\tau+1}) +\\
&\quad \sum_{s=1}^{S}\sum_{\boldsymbol{I}} P(\boldsymbol{I}|\boldsymbol{O}_s;\boldsymbol{\lambda}^{t})\sum_{\tau=1}^{T}\log b_{I_\tau}^{t+1}(O_{s,\tau})
\end{aligned} \tag{54}$$

与监督情形一样，三个参数的求解，相互独立，可以分别求解。首先，求解 $\pi^{t+1}(I_1=q_i)=\pi^{t+1}(q_i)$(其中 $i=1,2,\ldots,N$)，它需要满足约束条件 $\sum_{i=1}^{N}\pi^{t+1}(q_i)=1$，列出拉格朗日优化函数如下：

$$L(\boldsymbol{\pi}) = -\sum_{s=1}^{S}\sum_{\boldsymbol{I}} P(\boldsymbol{I}|\boldsymbol{O}_s;\boldsymbol{\lambda}^{t})\log\pi^{t+1}(I_1) + \alpha\left(\sum_{i=1}^{N}\pi^{t+1}(q_i)-1\right) \tag{55}$$

其中 α 为拉格朗日乘数。求解偏导数 $\frac{\partial L(\boldsymbol{\pi})}{\partial \pi^{t+1}(q_i)}$，并令其等于 0：

$$
\begin{aligned}
\frac{\partial L(\boldsymbol{\pi})}{\partial \pi^{t+1}(q_i)} &= -\sum_{s=1}^{S}\frac{P(I_1=q_i|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)}{\pi^{t+1}(q_i)}+\alpha \\
&= -\frac{\sum\limits_{s=1}^{S}P(I_1=q_i|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)}{\pi^{t+1}(q_i)}+\alpha=0 \quad \Rightarrow \\
\pi^{t+1}(q_i)\alpha &= \sum_{s=1}^{S}P(I_1=q_i|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)
\end{aligned} \tag{56}
$$

利用公式 $\sum\limits_{i=1}^{N}\pi^{t+1}(q_i)=1$，对上式两端按 i 求和，得到：

$$
\alpha=\sum_{i=1}^{N}\sum_{s=1}^{S}P(I_1=q_i|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)=\sum_{s=1}^{S}\sum_{i=1}^{N}P(I_1=q_i|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)=S \tag{57}
$$

于是，得到：

$$
\pi^{t+1}(q_i)=\frac{\sum\limits_{s=1}^{S}P(I_1=q_i|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)}{\alpha}=\frac{\sum\limits_{s=1}^{S}P(I_1=q_i|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)}{S}=\frac{\sum\limits_{s=1}^{S}\gamma_{s,1}^{t}(q_i)}{S} \tag{58}
$$

其中，$\gamma_{s,1}^{t}(q_i)$ 表示，第 t 次迭代中第 s 个样本，在时刻 $\tau=1$ 时，位于隐状态 q_i 的概率[8]。容易验证，$\pi^{t+1}(q_i)\geqslant 0$ 条件成立。

然后，求解 $a_{I_\tau=q_i}^{t+1}(I_{\tau+1}=q_j)=a_{q_i}^{t+1}(q_j)$(其中 $i,j=1,2,\ldots,N$，$\tau=1,2,\ldots,T-1$)，它需要满足约束条件 $\sum\limits_{j=1}^{N}a_{q_i}^{t+1}(q_j)=1$，列出拉格朗日优化函数如下：

$$
L(\boldsymbol{A})=-\sum_{s=1}^{S}\sum_{\boldsymbol{I}}P(\boldsymbol{I}|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)\sum_{\tau=1}^{T-1}\log a_{I_\tau}^{t+1}(I_{\tau+1})+\alpha\left(\sum_{j=1}^{N}a_{q_i}^{t+1}(q_j)-1\right) \tag{59}
$$

其中 α 为拉格朗日乘数。求解偏导数 $\frac{\partial L(\boldsymbol{A})}{\partial a_{q_i}^{t+1}(q_j)}$，并令其等于 0：

$$
\begin{aligned}
\frac{\partial L(\boldsymbol{A})}{\partial a_{q_i}^{t+1}(q_j)} &= -\sum_{s=1}^{S}\frac{\sum\limits_{\tau=1}^{T-1}P(I_\tau=q_i,I_{\tau+1}=q_j|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)}{a_{q_i}^{t+1}(q_j)}+\alpha \\
&= -\frac{\sum\limits_{s=1}^{S}\sum\limits_{\tau=1}^{T-1}P(I_\tau=q_i,I_{\tau+1}=q_j|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)}{a_{q_i}^{t+1}(q_j)}+\alpha=0 \quad \Rightarrow \\
a_{q_i}^{t+1}(q_j)\alpha &= \sum_{s=1}^{S}\sum_{\tau=1}^{T-1}P(I_\tau=q_i,I_{\tau+1}=q_j|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)
\end{aligned} \tag{60}
$$

[8]注意，在本小节中，t 表示迭代步，τ 表示观测序列和隐状态序列的时间步。

利用公式 $\sum\limits_{j=1}^{N} a_{q_i}^{t+1}(q_j)=1$，对上式两端按 j 求和，得到：

$$\begin{aligned}\alpha &= \sum_{j=1}^{N}\sum_{s=1}^{S}\sum_{\tau=1}^{T-1} P(I_\tau=q_i, I_{\tau+1}=q_j|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)\\ &= \sum_{s=1}^{S}\sum_{\tau=1}^{T-1}\sum_{j=1}^{N} P(I_\tau=q_i, I_{\tau+1}=q_j|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)\\ &= \sum_{s=1}^{S}\sum_{\tau=1}^{T-1} P(I_\tau=q_i|\boldsymbol{O}_s;\boldsymbol{\lambda}^t) = \sum_{s=1}^{S}\sum_{\tau=1}^{T-1}\gamma_{s,\tau}^t(q_i)\end{aligned} \tag{61}$$

于是，得到：

$$a_{q_i}^{t+1}(q_j) = \frac{\sum\limits_{s=1}^{S}\sum\limits_{\tau=1}^{T-1} P(I_\tau=q_i, I_{\tau+1}=q_j|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)}{\alpha} = \frac{\sum\limits_{s=1}^{S}\sum\limits_{\tau=1}^{T-1}\xi_{s,\tau}^t(q_i,q_j)}{\sum\limits_{s=1}^{S}\sum\limits_{\tau=1}^{T-1}\gamma_{s,\tau}^t(q_i)} \tag{62}$$

其中，$\xi_{s,\tau}^t(q_i,q_j)$ 表示，第 t 次迭代中第 s 个样本，在时刻 τ，从状态 q_i 转移到状态 q_j(时刻 $\tau+1$) 的概率；$\gamma_{s,\tau}^t(q_i)$ 表示，第 t 次迭代中第 s 个样本，在时刻 τ 时，位于隐状态 q_i 的概率。容易验证，$a_{q_i}^{t+1}(q_j)\geqslant 0$ 条件成立。

最后，求解 $b_{I_\tau=q_j}^{t+1}(O_\tau=v_k)=b_{q_j}^{t+1}(v_k)$(其中 $j=1,2,\dots,N$，$k=1,2,\dots,M$，$\tau=1,2,\dots,T$)，它需要满足约束条件 $\sum\limits_{k=1}^{M} b_{q_j}^{t+1}(v_k)=1$，列出拉格朗日优化函数如下：

$$L(\boldsymbol{B}) = -\sum_{s=1}^{S}\sum_{\boldsymbol{I}} P(\boldsymbol{I}|\boldsymbol{O}_s;\boldsymbol{\lambda}^t)\sum_{\tau=1}^{T}\log b_{I_\tau}^{t+1}(O_{s,\tau}) + \alpha\left(\sum_{k=1}^{M} b_{q_j}^{t+1}(v_k)-1\right) \tag{63}$$

其中 α 为拉格朗日乘数。求解偏导数 $\frac{\partial L(\boldsymbol{B})}{\partial b_{q_j}^{t+1}(v_k)}$，并令其等于 0：

$$\begin{aligned}\frac{\partial L(\boldsymbol{B})}{\partial b_{q_j}^{t+1}(v_k)} &= -\sum_{s=1}^{S}\frac{\sum\limits_{\tau=1}^{T} P(I_\tau=q_j|O_{s,\tau}=v_k;\boldsymbol{\lambda}^t)}{b_{q_j}^{t+1}(v_k)}+\alpha\\ &= -\frac{\sum\limits_{s=1}^{S}\sum\limits_{\tau=1}^{T} P(I_\tau=q_j|O_{s,\tau}=v_k;\boldsymbol{\lambda}^t)}{b_{q_j}^{t+1}(v_k)}+\alpha=0 \quad\Rightarrow\\ b_{q_j}^{t+1}(v_k)\alpha &= \sum_{s=1}^{S}\sum_{\tau=1}^{T} P(I_\tau=q_j|O_{s,\tau}=v_k;\boldsymbol{\lambda}^t)\end{aligned} \tag{64}$$

利用公式 $\sum\limits_{k=1}^{M} b_{q_j}^{t+1}(v_k) = 1$，对上式两端按 k 求和，得到：

$$
\begin{aligned}
\alpha &= \sum_{k=1}^{M}\sum_{s=1}^{S}\sum_{\tau=1}^{T} P(I_\tau = q_j | O_{s,\tau} = v_k; \boldsymbol{\lambda}^t) \\
&= \sum_{s=1}^{S}\sum_{\tau=1}^{T}\sum_{k=1}^{M} P(I_\tau = q_j | O_{s,\tau} = v_k; \boldsymbol{\lambda}^t) \\
&= \sum_{s=1}^{S}\sum_{\tau=1}^{T} P(I_\tau = q_j | \boldsymbol{O}_s; \boldsymbol{\lambda}^t) = \sum_{s=1}^{S}\sum_{\tau=1}^{T} \gamma_{s,\tau}^t(q_j)
\end{aligned}
\tag{65}
$$

于是，得到：

$$
b_{q_j}^{t+1}(v_k) = \frac{\sum\limits_{s=1}^{S}\sum\limits_{\tau=1}^{T} P(I_\tau = q_j | O_{s,\tau} = v_k; \boldsymbol{\lambda}^t)}{\alpha} = \frac{\sum\limits_{s=1}^{S}\sum\limits_{\tau=1}^{T} I\{O_{s,\tau} = v_k\}\gamma_{s,\tau}^t(q_j)}{\sum\limits_{s=1}^{S}\sum\limits_{\tau=1}^{T} \gamma_{s,\tau}^t(q_j)} \tag{66}
$$

容易验证，$b_{q_j}^{t+1}(v_k) \geqslant 0$ 条件成立。

将监督学习的模型参数求解公式 (41)、公式 (45)、公式 (49)，分别与无监督学习的模型参数求解公式 (58)、公式 (62)、公式 (66) 进行对比，可以发现，在数据集提供状态序列的情况下：

- 令 $\gamma_{s,1}^t(q_i) = 1$，如果 q_i 是状态序列的第 1 个状态，否则，令 $\gamma_{s,1}^t(q_i) = 0$，那么公式 (58) 将还原为公式 (41)；
- 令 $\gamma_{s,\tau}^t(q_i) = 1$，如果 q_i 是状态序列的第 τ 个状态，否则，令 $\gamma_{s,\tau}^t(q_i) = 0$；令 $\xi_{s,\tau}^t(q_i, q_j) = 1$，如果时刻 τ 的状态为 q_i，而时刻 $\tau+1$ 的状态为 q_j，否则，令 $\xi_{s,\tau}^t(q_i, q_j) = 0$，那么公式 (62) 将还原为公式 (45)；
- 公式 (62) 的分析，同样适用于公式 (66)，此时，公式 (66) 将还原为公式 (49)；

下面，给出 Baum-Welch 算法。

算法 **4.1**（*Baum-Welch* 算法）

Input:

Dataset: $\boldsymbol{O}=\{\boldsymbol{O}_1,\boldsymbol{O}_2,\ldots,\boldsymbol{O}_S\}$, $\boldsymbol{O}_s$: observation sequence.

Parameters: $\boldsymbol{\lambda}=<\boldsymbol{\pi},\boldsymbol{A},\boldsymbol{B}>$ (Initialization:$\boldsymbol{\lambda}^0$)

max_steps: $\mathcal{T}$

Output:

Parameters: $\boldsymbol{\lambda}^{\mathcal{T}}$

Algorithm:

for $t=0\cdots\mathcal{T}-1$:

E-step: $s=1,2,\ldots,S$, $\tau=1,2,\ldots,T$, $i,j=1,2,\ldots,N$

Calculate forward probability α_s^t for each $\boldsymbol{O}_s$

Calculate backward probability β_s^t for each $\boldsymbol{O}_s$

Calculate $\gamma_{s,\tau}^t(q_i)$ by α_s^t and β_s^t, using Equation(19)

Calculate $\xi_{s,\tau}^t(q_i,q_j)$ by α_s^t and β_s^t, using Equation(21)

M-step: $\pi^{t+1}(q_i)=\frac{\sum\limits_{s=1}^{S}\gamma_{s,1}^t(q_i)}{S}$

$$a_{q_i}^{t+1}(q_j)=\frac{\sum\limits_{s=1}^{S}\sum\limits_{\tau=1}^{T-1}\xi_{s,\tau}^t(q_i,q_j)}{\sum\limits_{s=1}^{S}\sum\limits_{\tau=1}^{T-1}\gamma_{s,\tau}^t(q_i)}$$

$$b_{q_j}^{t+1}(v_k)=\frac{\sum\limits_{s=1}^{S}\sum\limits_{\tau=1}^{T}I\{O_{s,\tau}=v_k\}\gamma_{s,\tau}^t(q_j)}{\sum\limits_{s=1}^{S}\sum\limits_{\tau=1}^{T}\gamma_{s,\tau}^t(q_j)},\quad k=1,2,\ldots,M$$

5 HMM 模型实验

HMM 模型的实现

```
In [1]: import numpy as np
        import copy
```

```
In [2]: class HiddenMarkovModel:
            #Q: 状态集合, V: 观测集合
            def __init__(self, Q, V, threshold = 1e-6):
                self.Q, self.V, self.threshold = Q, V, threshold

            #O: 观测序列, A: 状态转移概率, B: 观测概率分布, pi: 初始概率分布
            def forward(self, O, A, B, pi):
                N, T = len(self.Q), len(O)
                self.alphas = np.zeros((T, N)) # 保存各阶段的 alpha 值
                self.alphas[0] = pi * B[:, self.V.index(O[0])] # 初始化 alpha[0]
                for t in range(1, T): # 递推计算各阶段 alpha 值
                    self.alphas[t] = np.dot(self.alphas[t-1], A) * B[:,
                     self.V.index(O[t])]
                return np.sum(self.alphas[T-1]) # 返回观测序列 O 的概率

            #O: 观测序列, A: 状态转移概率, B: 观测概率分布, pi: 初始概率分布
            def backward(self, O, A, B, pi):
                N, T = len(self.Q), len(O)
                self.betas = np.zeros((T, N)) # 保存各阶段的 beta 值
                self.betas[T-1] = np.ones(N) # 初始化 beta[T-1]
                for t in range(T-2, -1, -1): # 反向递推计算各阶段 beta 值
                    self.betas[t] = np.dot(self.betas[t+1] * B[:,
                     self.V.index(O[t+1])], A.T)
                # 返回观测序列 O 的概率
                return np.dot(pi * B[:, self.V.index(O[0])], self.betas[0])

            #O: 观测序列, A: 状态转移概率, B: 观测概率分布, pi: 初始概率分布
            def viterbi(self, O, A, B, pi):
                N, T = len(self.Q), len(O)
                self.deltas = np.zeros((T, N)) # 保存各阶段的 delta 值
                # 保存各阶段的 psi 值
                self.psis = np.zeros((T, N), dtype = np.int16)
                self.deltas[0] = pi * B[:, self.V.index(O[0])] # 初始化 delta[0]
                for t in range(1, T): # 递推计算各阶段 delta 值
                    last = self.deltas[t-1].reshape(-1, 1) * A
                    self.deltas[t] = np.max(last, axis = 0) * B[:,
                     self.V.index(O[t])]
                    self.psis[t] = np.argmax(last, axis = 0)
                pstar = np.max(self.deltas[T-1])
                path = [np.argmax(self.deltas[T-1])]
                for t in range(T-1, 0, -1):
                    path.insert(0, self.psis[t, path[0]])
                # 返回与观测序列 O 对应的最优概率与最优状态
```

```
        return pstar, [index+1 for index in path]

    # 使用 Baum-Welch 算法训练 HMM 模型
    #O: 观测序列矩阵（多个样本），A、B、pi: 初始概率
    def fit(self, O, A, B, pi, n_iter = 100):
        N, M, D, T = len(self.Q), len(self.V), len(O), len(O[0])
        self.A, self.B, self.pi = copy.deepcopy(A), copy.deepcopy(B),
         copy.deepcopy(pi)
        # 迭代所需参数初始化
        gamma, xi, mask = np.zeros((D, T, N)),
         np.zeros((D, T, N, N)), np.zeros((D, T, N, M))
        for step in range(n_iter):
            # 执行 E-step
            for d, o in enumerate(O): # 遍历每个观测序列
                p = self.forward(o, self.A, self.B, self.pi)
                p = self.backward(o, self.A, self.B, self.pi)
                gamma[d] = self.alphas * self.betas / p
                for t in range(T-1):
                    xi[d, t] = self.alphas[t].reshape(-1, 1) * self.A *
                     self.B[:, self.V.index(o[t+1])].reshape(1, -1) *
                     self.betas[t+1] / p
                    mask[d, t, :, self.V.index(o[t])] = np.ones(N)
                mask[d, T-1, :, self.V.index(o[T-1])] = np.ones(N)
            # 执行 M-step
            old_A, old_B, old_pi = copy.deepcopy(self.A),
             copy.deepcopy(self.B), copy.deepcopy(self.pi)
            self.A = np.sum(np.sum(xi[:, :-1, :, :], axis = 1), axis = 0) /
             np.sum(np.sum(gamma[:, :-1, :], axis = 1), axis = 0).
             reshape(-1, 1)
            self.B = np.sum(np.sum(gamma.reshape(D, T, N, 1)*mask, axis = 1),
             axis = 0) / np.sum(np.sum(gamma, axis = 1), axis = 0).
             reshape(-1, 1)
            self.pi = np.sum(gamma[:, 0, :], axis = 0) / D
            if max(np.abs(self.A - old_A).max(),
             np.abs(self.B - old_B).max(), np.abs(self.pi - old_pi).max()) <
             self.threshold:
                break
        print(self.A)
        print(self.B)
        print(self.pi)
```

测试：使用前向算法与后向算法计算给定观测序列的概率；使用 Viterbi 算法求解最优概率与最优状态序列

验证例 10.2 与习题 10.1

设置模型参数

```
In [3]: Q = ['1', '2', '3'] # 状态集合
        V = ['红', '白'] # 观测集合
```

```
# 模型参数
A = [[0.5, 0.2, 0.3],
     [0.3, 0.5, 0.2],
     [0.2, 0.3, 0.5]]
B = [[0.5, 0.5],
     [0.4, 0.6],
     [0.7, 0.3]]
pi = [0.2, 0.4, 0.4]
```

测试自定义的 HMM 模型

```
In [4]: myhmm = HiddenMarkovModel(Q, V)
        O = ['红', '白', '红']
        print(myhmm.forward(O, np.array(A), np.array(B), np.array(pi)))
        print(myhmm.backward(O, np.array(A), np.array(B), np.array(pi)))
        print(myhmm.viterbi(O, np.array(A), np.array(B), np.array(pi)))
```

```
0.130218
0.130218
(0.014699999999999998, [3, 3, 3])
```

使用 hmmlearn 库

```
In [5]: # 在 anaconda3 下安装: pip install hmmlearn
        from hmmlearn import hmm
```

```
In [6]: model = hmm.MultinomialHMM(n_components = len(Q))
        model.startprob_ = np.array(pi)
        model.transmat_ = np.array(A)
        model.emissionprob_ = np.array(B)
        O = np.array([0, 1, 0]).reshape(-1, 1) #0: 红, 1: 白
        logprob = model.score(O) # 观测序列的 log 概率
        print(np.exp(logprob))
        # 观测序列对应的最佳隐状态
        logprob, states = model.decode(O, algorithm = 'viterbi')
        print(np.exp(logprob), [state+1 for state in states])
```

```
0.130218
0.0147 [3, 3, 3]
```

```
In [7]: # 自定义 HMM
        O = ['红', '白', '红', '白']
        print(myhmm.forward(O, np.array(A), np.array(B), np.array(pi)))
        print(myhmm.backward(O, np.array(A), np.array(B), np.array(pi)))
        print(myhmm.viterbi(O, np.array(A), np.array(B), np.array(pi)))
        # 库 HMM
```

```
O = np.array([0, 1, 0, 1]).reshape(-1, 1) #0: 红, 1: 白
print(np.exp(model.score(O)))
logprob, states = model.decode(O, algorithm = 'viterbi')
print(np.exp(logprob), [state+1 for state in states])
```

```
0.0600908
0.0600908
(0.0030239999999999993, [3, 2, 2, 2])
0.0600908
0.003024 [3, 2, 2, 2]
```

习题 10.2

```
In [8]: Q = ['1', '2', '3'] # 状态集合
        V = ['红', '白'] # 观测集合

        # 模型参数
        A = [[0.5, 0.1, 0.4],
             [0.3, 0.5, 0.2],
             [0.2, 0.2, 0.6]]
        B = [[0.5, 0.5],
             [0.4, 0.6],
             [0.7, 0.3]]
        pi = [0.2, 0.3, 0.5]
```

```
In [9]: # 自定义 HMM
        myhmm = HiddenMarkovModel(Q, V)
        O = ['红', '白', '红', '红', '白', '红', '白', '白']
        print(myhmm.forward(O, np.array(A), np.array(B), np.array(pi)))
        print(myhmm.backward(O, np.array(A), np.array(B), np.array(pi)))
        print(myhmm.viterbi(O, np.array(A), np.array(B), np.array(pi)))
        # 库 HMM
        model = hmm.MultinomialHMM(n_components = len(Q))
        O = np.array([0, 1, 0, 0, 1, 0, 1, 1]).reshape(-1, 1) #0: 红, 1: 白
        model.startprob_ = np.array(pi)
        model.transmat_ = np.array(A)
        model.emissionprob_ = np.array(B)
        print(np.exp(model.score(O)))
        logprob, states = model.decode(O, algorithm = 'viterbi')
        print(np.exp(logprob), [state+1 for state in states])
```

```
0.00347670944928
0.00347670944928
(3.0245685119999991e-05, [3, 3, 3, 3, 3, 3, 2, 2])
0.00347670944928
3.024568512e-05 [3, 3, 3, 3, 3, 3, 2, 2]
```

模型参数的学习：Baum-Welch 算法的验证
从文件导入测试数据

In [10]:
```
import pandas as pd
data = pd.read_csv('data_python.csv')

O = data['Visible'].values
```

In [11]:
```
# 状态与观测集合
Q = ['A', 'B'] # 状态集合
V = [0, 1, 2] # 观测集合
N, M = len(Q), len(V)
# 初始化 A、B、pi: 使用固定参数
A = np.ones((N, N))
A = A / np.sum(A, axis = 1).reshape(-1, 1)
B = np.array(((1, 2, 3), (4, 5, 6)))
B = B / np.sum(B, axis = 1).reshape(-1, 1)
pi = np.array((0.5, 0.5))
# 自定义 HMM
myhmm = HiddenMarkovModel(Q, V)
myhmm.fit([O], A, B, pi)
print('*'*40)
# 库 HMM
model = hmm.MultinomialHMM(n_components = N, n_iter = 100, tol = 1e-6)
model.startprob = pi
model.transmat = A
model.emissionprob_ = B
model.init_params = 'st'
model.fit(O.reshape(-1, 1))
print(model.transmat_)
print(model.emissionprob_)
print(model.startprob_)
```

```
[[ 0.77384134  0.22615866]
 [ 0.2278316   0.7721684 ]]
[[ 0.07801696  0.17193173  0.7500513 ]
 [ 0.33355442  0.36773984  0.29870574]]
[  9.39942699e-51   1.00000000e+00]
****************************************
[[ 0.77384134  0.22615866]
 [ 0.2278316   0.7721684 ]]
[[ 0.07801696  0.17193173  0.7500513 ]
 [ 0.33355442  0.36773984  0.29870574]]
[  9.39942699e-51   1.00000000e+00]
```

In [12]:
```
# 初始化 A、B、pi: B 和 pi 使用随机参数
A = np.ones((N, N))
A = A / np.sum(A, axis = 1).reshape(-1, 1)
```

```
        B = np.random.rand(N, M)
        B = B / np.sum(B, axis = 1).reshape(-1, 1)
        pi = np.random.rand(N)
        pi = pi / np.sum(pi)
        # 自定义 HMM
        myhmm = HiddenMarkovModel(Q, V)
        myhmm.fit([O], A, B, pi)
        print('*'*40)
        # 库 HMM
        model = hmm.MultinomialHMM(n_components = N, n_iter = 100, tol = 1e-6)
        model.startprob = pi
        model.transmat = A
        model.emissionprob_ = B
        model.init_params = 'st'
        model.fit(O.reshape(-1, 1))
        print(model.transmat_)
        print(model.emissionprob_)
        print(model.startprob_)
```

```
[[ 0.60933483  0.39066517]
 [ 0.18197208  0.81802792]]
[[ 0.41127129  0.41308927  0.17563944]
 [ 0.10919346  0.20251869  0.68828785]]
[  1.00000000e+00   8.09753825e-79]
****************************************
[[ 0.6092341   0.3907659 ]
 [ 0.18199494  0.81800506]]
[[ 0.41132036  0.41310339  0.17557625]
 [ 0.10918331  0.20252108  0.68829562]]
[  1.00000000e+00   7.38479742e-79]
```

```
In [13]: # 初始化 A、B、pi: 全部使用随机参数
        A = np.random.rand(N, N)
        A = A / np.sum(A, axis = 1).reshape(-1, 1)
        B = np.random.rand(N, M)
        B = B / np.sum(B, axis = 1).reshape(-1, 1)
        pi = np.random.rand(N)
        pi = pi / np.sum(pi)
        # 自定义 HMM
        myhmm = HiddenMarkovModel(Q, V)
        myhmm.fit([O], A, B, pi)
        print('*'*40)
        # 库 HMM
        model = hmm.MultinomialHMM(n_components = N, n_iter = 100, tol = 1e-6)
        model.startprob = pi
        model.transmat = A
        model.emissionprob_ = B
```

```
        model.init_params = 'st'
        model.fit(O.reshape(-1, 1))
        print(model.transmat_)
        print(model.emissionprob_)
        print(model.startprob_)
```

```
[[ 0.90391605  0.09608395]
 [ 0.13815637  0.86184363]]
[[ 0.2884067   0.34442701  0.36716629]
 [ 0.08546423  0.16113607  0.75339969]]
[  1.00000000e+00   8.58605653e-67]
****************************************
[[ 0.67514367  0.32485633]
 [ 0.33398649  0.66601351]]
[[ 0.38345488  0.35659649  0.25994862]
 [ 0.02227811  0.18034526  0.79737663]]
[  1.00000000e+000   8.08048669e-140]
```

说明：Baum-Welch 算法的训练结果与提供的样本数量、算法的迭代次数与阈值、算法的终止策略等有很大的关系。

6 参考文献

1. Trevor Hastie, Robert Tibshirani, Jerome Friedman. The Elements of Statistical Learning, Data Mining, Inference, and Prediction. Second Edition. Springer Series in Statistics.

2. 周志华。《机器学习》，清华大学出版社，2016 年 1 月第 1 版。

3. Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.

4. Kevin P. Murphy. Machine Learning: A Probabilistic Perspective, The MIT Press, 2012.

5. 李航。《统计学习方法》，清华大学出版社，2019 年 5 月第 2 版。

6. https://github.com/wzyonggege/statistical-learning-method.

7. http://www.cnblogs.com/pinard/p/6972299.html.

8. https://www.cnblogs.com/pinard/p/7001397.html.

9. https://xbuba.com/questions/38628872.

10. http://www.adeveloperdiary.com/data-science/machine-learning/derivation-and-implementation-of-baum-welch-algorithm-for-hidden-markov-model/.