HMM 学习笔记

Xiaolin Hu

March 12, 2022

Abstract

参考知乎上的一篇文章学习了一下，或者说是温习了一下 HMM 算法。然后在这边写一下学习笔记。里面有我根据伪代码写出来的 HMM 的 python 代码 (渣代码)。

1 隐马尔可夫模型用于处理的问题

隐马尔科夫模型用于求解包含如下条件的问题:
1）HMM 模型一般用于处理序列问题，诸如时间序列，状态序列。
2）问题中的序列包含两种数据，一种是显式表示的观测序列；另一类是隐式表示在观测序列节点后的隐藏状态序列。

1.1 实例

举个例子来说有三个箱子，我们成为 1 号，2 号，3 号。然后每个箱子里面都有红球和蓝球。闭着眼我们**有放回**的抽三次（注意我们不知道每个球是在哪个箱子中抽的）。最后得到了 [蓝球，蓝球，红球] 这样一组结果, 这也就是我们的观测序列。而每个球来自哪个箱子这个我们是不知道的但从客观上来说确实存在一个箱子抽取顺序序列比如 [1 号，1 号，3 号]，这也就我们的隐藏状态。

2 模型定义

在 HMM 模型中，我们假设 V 为所有的观察状态集合（visible），Q 为所有的隐藏状态序列。在上面的例子中，V 为 [红球，蓝球]，隐藏状态 Q 为 [1 号，2 号，3 号]。其中 N, M 分别为隐藏状态长度，观察状态长度。

$$Q = \{q_1, q_2, \ldots, q_N\}, V = \{v_1, v_2, \ldots, v_M\} \tag{1}$$

HMM 模型基于两个重要的假设前提
1）一阶马尔可夫性假设：即任意时刻的隐藏状态只依赖于它前一个隐藏状态。比如说 t_3 时的状态只会依赖 t_2 的状态，而与 t_1 的状态没有关系。2）观测独立性假设：即任意时刻的观察状态只仅仅依赖于当前时刻的隐藏状态。比如说当前时刻我从箱子中摸出红球的概率只会受这个箱子红

球的摸出的概率影响。而与之前的箱子是几号，概率怎么样都没有关系。
对于一个长度为 T 的序列，我们假设 O 为对应的观察序列，I 为对应的隐藏状态序列。

$$\begin{aligned} O &= \{o_1, o_2, \ldots, o_T\}, o_t \in V \\ I &= \{i_1, i_2, \ldots, i_T\}, i_t \in Q \end{aligned} \tag{2}$$

我们假设 t 时刻的隐藏状态为 $i_t = q_i$，在 $t+1$ 时刻的隐藏状态为 $i_{t+1} = q_j$，则从状态 q_i 到 q_j 的隐藏状态转移概率为 a_{ij}。a_{ij} 组成马尔科夫链的状态转移矩阵 A。ps: 注意状态转移矩阵不随时间变化而变化是一个 time free 量。

$$\begin{aligned} a_{ij} &= P(i_{t+1} = q_j | i_t = q_i) \\ A &= \left[a_{ij}\right]_{N \times N} \end{aligned} \tag{3}$$

假设 t 时刻的隐藏状态为 $i_t = q_i$，在 t 时刻的观察状态为 $o_t = v_k$)，则在 t 时刻的观察状态生成概率为 b_{ik}。b_{ik} 组成观察状态生成矩阵 B。

$$\begin{aligned} b_{ik} &= P(o_t = v_k | i_t = q_i) \\ B &= \left[b_{ik}\right]_{N \times M} \end{aligned} \tag{4}$$

除此之外，我们还需要在 $t = 1$ 的时候设立隐藏状态的初始概率分布 Π:

$$\Pi = \left[\pi(i)\right]_N, \pi(i) = P(i_1 = q_i) \tag{5}$$

一个 HMM 模型通过初始概率分布 Π，状态转移概率矩阵 A 和观测状态概率矩阵 B。Π, A 决定状态序列，B 决定观测序列。我们可以用三元组 $\lambda = (A, B, \Pi)$。

2.1 序列生成过程

序列生成可以按下属代码方式生成

```
import random


def hmm(A, B, PI, T):
    """
    :param A: 马尔科夫链的状态转移矩阵 A
    :param B: 观察状态生成矩阵 B
    :param PI: 初始概率分布 PI
    :param T: 目标生成序列长度 T
    :return O: 观察序列 O={o1, o2,...,oT} I: 隐藏状态序列
    """
    # 1) 根据初始状态概率分布 PI 生成隐藏状态
    PI = PI
    I = []
    O = []
    # 2) 生成观察状态序列长度 N 和隐藏状态序列长度 M
    N = A.shape[0]
```

```
    M = B.shape[1]
    pi = random.uniform(0, 1)
    tmp = 0
    for i in range(N):
        tmp = tmp + PI[i]
        if pi < tmp:
            i1 = i
            break
    i_now = i1
    I.append(i_now)
    tmp = 0
    for i in range(T):
        cmp = random.uniform(0, 1)
        for k in range(len(B[i_now])):
            tmp = tmp + B[i_now][k]
            if cmp < tmp:
                o_now = k
                break
        O.append(o_now)
        if len(I) < T:
            cmp = random.uniform(0, 1)
            for j in range(len(A[i_now])):
                tmp = tmp + A[i_now][j]
                if cmp < tmp:
                    i_now = j
                    break
        else:
            break
    return O, I
```

2.2 HMM 求解的三类基本问题

HMM 模型一共有三类经典的问题需要解决：

1）评估观察序列概率。即给定模型 $\lambda = (A, B, \Pi)$ 和观测序列 $O = \{o_1, o_2, \ldots, o_T\}$ 计算在模型参数 λ 下观测到观测序列 O 出现的概率 $P(O|\lambda)$。求解这类问题可以用到前向后向算法，这类问题是三类中最简单的。

2）预测问题，也叫解码问题。即给定模型 $\lambda = (A, B, \Pi)$ 和观测序列 $O = \{o_1, o_2, \ldots, o_T\}$ 求出给定观测序列最可能对应的隐藏状态序列。求解这类问题我们用到基于动态规划的维特比算法，这类问题是三类中复杂度居中的。

3）模型参数学习问题。即给定观测序列 $O = \{o_1, o_2, \ldots, o_T\}$，估计模型参数 $\lambda = (A, B, \Pi)$ 使该模型下观测序列的条件概率 $P(O|\lambda)$ 最大。求解这个问题需要用到 EM 算法的鲍姆-韦尔奇（Baum-Welch）算法。是三类问题中最为复杂的。

3 前向后向算法评估观察序列概率

3.1 观测序列的一般求解法

首先我们已知模型参数 $\lambda=(A,B,\Pi)$，同时得到了观测序列 $O=\{o_1,o_2,\ldots,o_T\}$。那么我们就可以直接暴力求解出所要求得的条件概率 $P(O|\lambda)$。上述的这几类方法也是我们常说的 Model-based 方法。Model-based 的核心就是模型参数已知。我们需要去做的只是在这个模型中找到想要的解。暴力求解如下：

首先我们定义一个任意隐藏状态序列 $I=\{i_1,i_2,\ldots,i_T\}$ 的出现概率为

$$\begin{aligned} &P(I|\lambda)=\pi_{i_1}a_{i_1i_2}a_{i_2i_3}\ldots a_{i_{T-1}i_T} \\ &\pi_{i_1}\in\Pi, a_{i_{T-1}i_T}\in A \end{aligned} \tag{6}$$

对于**固定**的状态序列 $I=\{i_1,i_2,\ldots,i_T\}$，我们要求的观测序列 $O=\{o_1,o_2,\ldots,o_T\}$ 出现的概率为：

$$\begin{aligned} &P(O|I,\lambda)=b_{i_1o_1}bi_2o_2\ldots b_{i_T,o_T} \\ &b_{i_T,o_T}\in B \end{aligned} \tag{7}$$

则在给定模型参数 λ 的情况下，O,I 联合出现的概率为（全概率公式）：

$$\begin{aligned} P(O,I|\lambda)&=P(I|\lambda)P(O|I,\lambda) \\ &=\pi_{i_1}b_{i_1o_1}a_{i_1i_2}b_{i_2o_2}\ldots a_{i_{T-1}i_T}b_{i_T,o_T} \end{aligned} \tag{8}$$

全概率公式为：

$$P(B|A)=\frac{P(A,B)}{P(A)} \tag{9}$$

然后通过边缘概率分布，即可得到观测序列 O 在模型 λ 下出现的条件概率 $P(O|\lambda)$:

$$\begin{aligned} P(O|\lambda)&=\sum_I P(O,I|\lambda) \\ &=\sum_I P(I|\lambda)P(O|I,\lambda) \\ &=\sum_{i_1,i_2,\ldots,i_T}\pi_{i_1}b_{i_1o_1}a_{i_1i_2}b_{i_2o_2}\ldots a_{i_{T-1}i_T}b_{i_T,o_T} \end{aligned} \tag{10}$$

边缘概率分布为：

$$P(x)=\sum_y P(x,y)=\sum_y P(x|y)P(y) \tag{11}$$

上述算法的缺点在于需要罗列出所有状态 O 的概率可能。对于隐藏状态为 N，序列长度为 T 的情况下算法复杂度为 $O(TN^T)$。算法太过耗时，这就需要基于动态规划的前向后向算法。

3.2 前向算法求解 HMM 观测序列

前向算法本质上属于动态规划的算法，也就是我们要通过找到局部状态递推的公式，这样一步步的从子问题的最优解拓展到整个问题的最优解。在前向算法中，通过定义“前向概率”来定义动态规划的这个局部状态。

假设在 t 时刻隐藏状态为 q_i，当前的观测序列为 $\{o_1,o_2,\ldots,o_t\}$ 设当前观测序列的概率为前向概率。记为：

$$\alpha_t(i)=P(o_1,o_2,\ldots,o_t,i_t=q_i|\lambda) \tag{12}$$

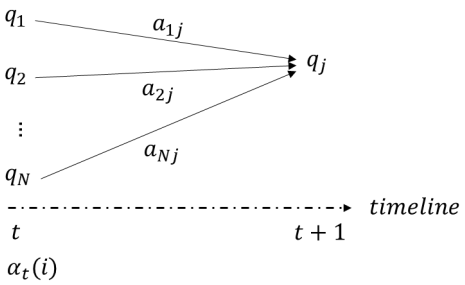

Figure 1: Forward Algorithm

因为是动态规划，所以我们希望能递推出到达 $t+1$ 时刻的隐藏状态 q_j 的概率。如图所示，应该是 t 时刻所有前向概率与所在状态转移到隐藏状态 q_j 的乘积和。即 $\sum_{j=1}^{N} \alpha_t(i)a_ij$。而前向概率是一个包含观测变量 o_t 的量，所以我们要在前面的基础上乘上在 $t+1$ 时刻上观测到 o_{t+1} 的观测状态概率。即：

$$\alpha_{t+1}(j) = \big[\sum_{j=1}^{N} \alpha_t(i)a_{ij}\big] b_{io_{t+1}} \tag{13}$$

想要求得观测序列 O 在模型 λ 下出现的条件概率只需按时轴，将所有隐藏状态序列对应的前向概率相加即可。$\alpha_T(i)$ 表示观测序列在 T 时刻为序列 $\{o_1, o_2, \ldots, o_T\}$，并且 T 时刻的隐藏状态为 q_i 的概率。注意：这里在 T 时刻时观测状态已经确定，为 o_T 所以不需要再乘上观测状态概率。得到条件概率 $P(O|\lambda)$ 为：

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{14}$$

注意这里的 T 是指 T 时刻生成的观测序列 $O = \{o_1, o_2, \ldots, o_T\}$ 是一个时间累积量。从递推公式可以求出前向算法的时间复杂度为 $O(TN^2)$。相较于暴力解法数量级得到了明显的下降。

3.3 前向算法求解 HMM 代码

$P(O|\lambda)$ 观测序列概率算法输入为 $\lambda = (A, B, \Pi)$ 和观测序列 $O = \{o_1, o_2, \ldots, o_T\}$。代码如下：

```
def forward_algorithm(A, B, PI, O):
    """
    :param A: 马尔科夫链的状态转移矩阵 A
    :param B: 观察状态生成矩阵 B
    :param PI: 初始概率分布 PI
    :param O: 观察序列 O={o1, o2,...,oT}
    :return p_O: 观测序列在模型下的概率
    """
    N = A.shape[0]
    M = B.shape[1]
    alpha = []
    alpha_1 = []
    for i in range(PI):
```

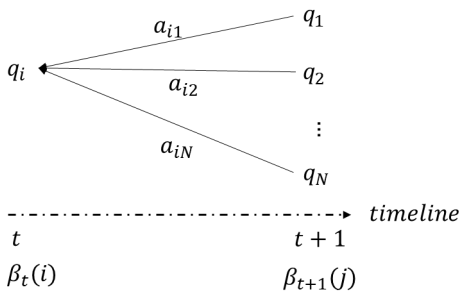

Figure 2: Backward Algorithm

```
    alpha_1.append(PI[i]*B[i][O[0]])
alpha.append(alpha_1)
alpha_t_ = alpha_1
for t in range(1, len(O)):
    alpha_t = []
    for i in range(N):
        alpha_t_i = []
        for j in range(N):
            alpha_t_i.append(alpha_t_*A[i][j])
        alpha_t_i = sum(alpha_t_i)*B[i][O[t+1]]
        alpha_t.append(alpha_t_i)
    alpha.append(alpha_t)
alpha_T = alpha[-1]
p_0 = sum(alpha_T)
return p_0
```

3.4 后向算法求解 HMM 观测序列

后向算法和前向算法非常类似，都是用的动态规划，唯一的区别是选择的局部状态不同，后向算法用的是“后向概率”。假设在 t 时刻隐藏状态为 q_i，当前的观测序列为 $\{o_1, o_2, \ldots, o_t\}$ 设当前观测序列的概率为后向概率。记为：

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots, o_T | i_t = q_i, \lambda) \tag{15}$$

后向算法的递推思想是建立在，假设我们已经找到了时刻 $t+1$ 时各个隐藏状态的后向概率 $\beta_{t+1}(j)$。我们希望得到 t 时刻各个隐藏状态的后向概率 $\beta_t(i)$，如图所示。t 时刻隐藏状态为 q_i，$t+1$ 时刻隐藏状态为 q_j 的概率为 $a_{ij}\beta_{t+1}(j)$。由观测状态可知，在 t 时刻观测序列为 $\{o_{t+1}, o_{t+2}, \ldots, o_T\}$。$t$ 时隐藏状态为 q_i 的概率也是这个时刻的后向概率为：

$$\beta_t(i) = \sum_{j=1} N a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \tag{16}$$

对于初始化的 T 时刻，各个隐藏状态的后向概率为：

$$\beta_T(i) = 1, i = 1, 2, \ldots, N \tag{17}$$

关于这点，我们可以这么理解。其实后向概率描述了当前所有可能状态从先前的某一状态演变而来的概率。举个例子来说，$\beta_t(i)$ 描述了 t 时刻我们的状态为 q_i，在 $t+1$ 时刻我们来到了 q_1 表现出 o_{T+1} 的可能性或者来到 q_2 或者 q_3 一直到 q_N 这所有可能性表现出 o_{T+1} 的总可能性。

3.5 后向算法求解 HMM 代码

后向算法 python 代码如下：

```
def backward_algorithm(A, B, PI, O):
    """
    :param A: 马尔科夫链的状态转移矩阵 A
    :param B: 观察状态生成矩阵 B
    :param PI: 初始概率分布 PI
    :param O: 观察序列 O={o1, o2,...,oT}
    :return p_O: 观测序列在模型下的概率
    """
    N = A.shape[0]
    M = B.shape[1]
    beta = []
    beta_t_ = [1] * N
    beta.append(beta_t_)
    for t in range(1, len(O)):
        beta_t = []
        for i in range(N):
            beta_ti = []
            for j in range(N):
                beta_ti.append(A[i][j]*B[j][O[len(O)-t]]*beta_t_[j])
            beta_ti = sum(beta_ti)
            beta_t.append(beta_ti)
        beta.append(beta_t)
    beta_1 = beta[-1]
    p_O = 0
    for i in range(beta_1):
        p_O = p_O + PI[i]*B[i][O[0]]*beta_1[i]

    return p_O
```

3.6 HMM 常用的概率和定义

利用前向概率和后向概率我们可以计算出 HMM 中单个状态和两个状态的概率公式以及一些特定状态下的期望。

1）对于给定的模型 λ 和观测序列 O，我们设在时刻 t 处于隐藏状态 q_i 的概率记为：

$$\gamma_t(i) = P(i_t = q_i|O,\lambda) = \frac{P(i_t = q_i, O|\lambda)}{P(O|\lambda)} \tag{18}$$

利用前向概率和后向概率可以定义出：

$$P(i_t = q_i, O|\lambda) = \alpha_t(i)\beta_t(i) \tag{19}$$

为什么在给定模型 λ 的情况下可以这样定义 $P(i_t = q_i, O|\lambda)$。我们可以从前向概率和后向概率的定义着手。前向概率表示的是 $t-1$ 时刻所有可能状态抵达当前时刻 t 隐藏状态 q_t 的概率和。而后向概率表示的是 $t+1$ 时刻所有可能状态是由 t 时刻当前隐藏状态出发的概率和。可以说前向概率接通了过去的所有可能性，后向概率接通了未来的所有可能性。因为是一阶马尔科夫链，过去和未来被中间的现在所间隔，互相独立。所以当同时满足过去和未来的时候，过去的可能和未来的可能相乘。也就是前向概率和后向概率相乘。也就是在给定模型的，观测序列未定的情况下，t 时刻隐藏状态为 q_i 的可能性。而在除以观测序列在给定模型的概率后 $P(O|\lambda)$ 后我们将这一 t 时刻隐藏状态为 q_i 的可能性限制在了固定的观测序列下。也就得到了 $\gamma_t(i)$。重写表达式得到：

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \tag{20}$$

2) 对于给定模型 λ 和观测序列 O，在时刻 t 处于状态 q_i，且时刻 $t+1$ 处于状态 q_j 的概率记为：

$$\xi_t(i,j) = P(i_t = q_i, i_{t+1} = q_j|O,\lambda) = \frac{P(i_t = q_i, i_{t+1} = q_j, O|\lambda)}{P(O|\lambda)} \tag{21}$$

而 $P(i_t = q_i, i_{t+1} = q_j, O|\lambda)$ 可以通过前后向概率来表示为:

$$P(i_t = q_i, i_{t+1} = q_j, O|\lambda) = \alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j) \tag{22}$$

这个理解方法和前面在 t 时刻的十分相似。主要的变化是从过去连接到 t 而从未来连接到 $t+1$ 并且中间多了一个隐藏状态转移和 $t+1$ 时刻的观测概率。理解这些之后甚至可以轻松构造三状态甚至更多状态的概率。最后重写 $\xi_t(i,j)$ 得到如下表达式：

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{m=1}^{N}\sum_{n=1}^{N} \alpha_t(m)a_{mn}b_n(o_{t+1})\beta_{t+1}(n)} \tag{23}$$

3) 将 $\gamma_t(i)$ 和 $\xi_t(i,j)$ 在各个时刻 t 求和, 可以得到：

在观测序列 O 下状态 i 出现的期望值 $\sum_{t=1}^{T} \gamma_t(i)$

在观测序列 O 下由状态 i 转移的期望值 $\sum_{t=1}^{T-1} \gamma_t(i)$

在观测序列 O 下由状态 i 转移的期望值 $\sum_{t=1}^{T-1} \xi_t(i,j)$

上面这些常用的概率值在求解模型参数学习问题中将会有广泛的应用。

4 维特比算法解码隐藏状态序列

在预测和解码问题中，我们通过给定的模型和观测序列，求解给定观测序列条件下，最可能出现的对应的隐藏状态序列。HMM 模型中做解码问题最常用的算法是维特比算法，当然还有其它的求解方法。维特比算法是一个通用的求序列最短路径的动态规划算法，利用这一特性，我们可以用维特比算法来做分词，也就是求解 HMM 最可能的隐藏序列。

4.1 关于 HMM 最可能隐藏状态序列求解的相关概述

在 HMM 模型的解码问题中，给定模型 $\lambda = (A, B, \Pi)$ 和观测序列 $O = \{o_1, o_2, \ldots, o_T\}$，求给定观测序列 O 的条件下。最有可能出现对应的状态序列 $I^* = \{i_1^*, i_2^*, \ldots, i_T^*\}$，即 $P(I^*|O)$ 要最大化。

一种可能的近似解法是求出观测序列 O 在每个时刻 t 最可能的隐藏状态 i_t^*，然后得到一个近似的隐藏状态序列 $I^* = i_1^*, i_2^*, \ldots, i_T^*$。在给定模型 λ 和观测序列 O 时，在时刻 t 处于状态 q_i 的概率为我们先前得到的概率 $\gamma_t(i)$。于是我们有了：

$$i_t^* = \arg\max_{1 \le i \le N}[\gamma_t(i)], t = 1, 2, \ldots, T \tag{24}$$

近似算法很简单，但时却不能保证预测的状态序列的整体时最有可能的状态序列，因为预测的状态序列中某些相邻的隐藏状态可能存在转移概率为 0 的情况。

为了解决这样的问题，维特比算法将 HMM 的状态序列作为一个整体来考虑。

4.2 维特比算法概述

维特比算法是一个解码问题的通用算法，本身基于的是动态规划的求序列最路径的方法。因为本身是动态规划算法，那么就需要寻得合适的局部状态以及局部状态的递推公式。在 HMM 中，维特比算法定义了两个局部状态用于递推。

第一个局部状态是在时刻 t 隐藏状态为 i 所有可能的隐藏状态转移路径 $i_1, i_2, \ldots, i_t$ 中的概率最大值。记为 $\delta_t(i)$, 表示为:

$$\begin{aligned} &\delta_t(i) = \max_{i_1, i_2, \ldots, i_{t-1}} P(i_t = i, i_1, i_2, \ldots, i_{t-1}, o_t, o_{t-1}, \ldots, o_1|\lambda) \\ &i = 1, 2, \ldots, N \end{aligned} \tag{25}$$

由上我们可以得到 $t+1$ 时刻的 δ 递推表达式：

$$\begin{aligned} \delta_{t+1}(i) &= \max_{i_1, i_2, \ldots, i_t} P(i_t = i, i_1, i_2, \ldots, i_t, o_{t+1}, o_t, \ldots, o_1|\lambda) \\ &= \max_{1 \le j \le N}[\delta_t(j)a_j i]b_i(o_{t+1}) \end{aligned} \tag{26}$$

第二个局部状态由第一个局部状态递推得到。我们定义在时刻 t 隐藏状态为 i 的所有单个状态转移路径 $(i_1, i_2, \ldots, i_{t-1}, i)$ 中概率最大的转移路径中第 $t-1$ 个节点的隐藏状态为 $\psi_t(i)$，其递推表达式可以表示为:

$$\psi_t(i) = \arg\max_{1 \le j \le N}[\delta_{t-1}(j)a_{ji}] \tag{27}$$

在有了这两个局部状态后，我们就可以从时刻 0 一直递推到时刻 T，然后利用 ψ 记录的前一个最可能的状态节点回溯，直到找到最优的隐藏状态序列。

4.3 维特比算法代码实现

以下为维特比算法的 python 代码实现

```
def viterbi_algorithm(A, B, PI, O):
    """
    :param A: 马尔科夫链的状态转移矩阵 A
```

```
    :param B: 观察状态生成矩阵 B
    :param PI: 初始概率分布 PI
    :param O: 观察序列 O={o1, o2,...,oT}
    :return I_: 最有可能的的隐藏状态序列
    """
    N = A.shape[0]
    M = B.shape[1]
    delta_1 = [PI[i]*B[i][O[0]] for i in range(0)]
    psi_1 = [0] * N
    delta = []
    psi = []
    delta.append(delta_1)
    psi.append(psi_1)
    def argmax(lst):
        return max(range(len(lst)),key=lst.__getitem)
    for t in range(1,len(O)):
        delta_t_i = []
        psi_t_i = []
        for i in range(N):
            for j in range(N):
                delta_t_i.append(delta[t-1][j]*A[j][i])
                psi_t_i.append(delta[t-1][j]*A[j][i])
            delta_t_i = max(delta_t_i)*B[i][O[t]]
            psi_t_i = argmax(psi_t_i)
            delta.append(delta_t_i)
            psi.append(psi_t_i)

    P_x = max(delta[-1])
    i_x_T = argmax(delta[-1])
    i_x_t = i_x_T
    I_ = []
    I_.append(i_x_t)
    for t in range(1, len(O)):
        i_x_t = psi[len(O)-t][i_x_t]
        I_.append(i_x_t)
    return I_
```

5 鲍姆-韦尔奇算法求解 HMM 参数

HMM 模型参数的求解问题在这三类问题中算是最复杂的。

5.1 HMM 模型参数求解概述

HMM 模型参数求解根据已知的条件可以分为两种情况。第一类情况相对简单，即已知 D 个长度为 T 的观测序列和对应的隐藏状态序列，即 $\{(O_1, I_1), (O_2, I_2), \ldots, (O_D, I_D)\}$ 是已知的，此时我们很容易的就可以用最大似然来求解模型参数。我们假设样本从隐藏状态 q_i 转移到 q_j 的频率计数是 A_{ij}，那么状态转移矩阵可以求得为：

$$A = [a_{ij}], where a_{ij} = \frac{A_{ij}}{\sum_{s=1}^{N} A_{is}} \tag{28}$$

假设样本隐藏状态为 q_i 且观测状态为 v_k 的频率计数为 B_{ik}，那么观测状态概率矩阵为:

$$B = [b_i(k)], where b_i(k) = \frac{B_{ik}}{\sum_{s=1}^{N} B_{is}} \tag{29}$$

假设所有样本中初始隐藏状态为 q_i 的计数频率为 $C(i)$，那么初始概率分布为：

$$\Pi = \pi(i) = \frac{C(i)}{\sum_{s=1}^{N} C(s)} \tag{30}$$

这类情况求解模型比较简单。但是在更多的时候，我们无法得到 HMM 样本观测序列对应的隐藏序列，只有 D 个长度为 N 的观测序列，即 $(O_1), (O_2), \ldots, (O_D)$ 是已知的，此时我们需要用到鲍姆-韦尔奇算法来求解。

5.2 鲍姆-韦尔奇算法原理

鲍姆-韦尔奇算法又叫广义前向后向算法。是 EM 算法的一种特殊情况。我们需要在 E 步求 (Expectation 步) 出联合分布 $P(O, I|\lambda)$ 基于条件概率 $P(I|O, \overline{\lambda})$ 的期望，其中 $\overline{\lambda}$ 为当前的模型参数，然后再在 M 步 (Maximization 步) 最大化这个期望，得到更新后的模型参数 λ。接着不停的进行 E 步和 M 步的 EM 迭代，直到模型参数收敛为止。首先看看 E 步，当前模型参数为 $\overline{\lambda}$，联合分布 $P(O, I|\lambda)$ 基于条件概率 $P(I|O, \overline{\lambda})$ 的期望表达为:

$$L(\lambda, \overline{\lambda}) = \sum_{d=1}^{D} \sum_{I} P(I|O, \overline{\lambda}) \log P(O, I|\lambda) \tag{31}$$

在 M 步，我们极大化上面的式子。然后得到更新后的模型参数如下：

$$\overline{\lambda} = \arg\max_{\lambda} \sum_{d=1}^{D} \sum_{I} P(I|O, \overline{\lambda} \log P(O, I|\lambda)) \tag{32}$$

5.3 鲍姆-韦尔奇算法的推导

我们的训练数据为 $(O_1, I_1), (O_2, I_2), \ldots, (O_D, I_D)$ 其中任意一个观测序列 $O_d = o_1^{(d)}, o_2^(d), \ldots, o_T^{(d)}$ 其对应的未知的隐藏状态序列表示为:$O_d = i_1^(d), i_2^(2), \ldots, i_T^(d)$。根据先前的推导我们有如下公式:
联合分布 $P(O, I|\lambda)$ 的表达式：

$$P(O, I|\lambda) = \pi_{i_1} b_{i_1}(o_1) a_{i_1 i_2} b_{i_2}(o_2) \ldots a_{i_{T-1} i_T} b_{i_T}(o_T) \tag{33}$$

我们的 E 步得到的期望表达式为:

$$L(\lambda, \overline{\lambda}) = \sum_{d=1}^{D} \sum_{I} P(I|O, \overline{\lambda}) \log P(O, I|\lambda) \tag{34}$$

M 步极大化的的模型参数式为:

$$\overline{\lambda} = \arg\max_{\lambda} \sum_{d=1}^{D} \sum_{I} P(I, O|\overline{\lambda} \log P(O, I|\lambda)) \tag{35}$$

由于 $P(I|O, \overline{\lambda}) = P(I, O|\overline{\lambda})/P(O|\overline{\lambda})$ 而 $P(O|\overline{\lambda})$ 是常数。所以将上面 $P(O, I|\lambda)$ 的表达式带入我们的极大化式子后再约去常数项，得到表达式如下:

$$\overline{\lambda} = \arg\max_{\lambda} \sum_{d=1}^{D} \sum_{I} P(O, I|\overline{\lambda}(\log \pi_{i_1} + \sum_{t=1}^{T-1} \log a_{i_t i_{t+1}} + \sum_{t=1}^{T} \log b_{i_t}(o_t)) \tag{36}$$

我们的隐藏模型参数 $\lambda = (A, B, \Pi)$, 因此下面我们只需要对上式中的 A, B, Π 求导即可得到我们更新的参数 $\overline{\lambda}$。

首先我们对模型参数 Π 求导，由于 Π 只出现在上面表达式的第一部分。所以在求导后只留下带 Π 项，于是我们极大化式子等效于:

$$\begin{aligned} \overline{\pi_i} &= \arg\max_{\pi_{i_1}} \sum_{d=1}^{D} \sum_{I} P(O, I|\overline{\lambda}(\log \pi_{i_1}) \\ &= \arg\max_{\pi_{i_1}} \sum_{d=1}^{D} \sum_{i=1}^{N} P(O, i_1^{(d)} = i|\overline{\lambda}(\log \pi_{i_1}) \end{aligned} \tag{37}$$

由于 π_i 满足约束条件 $\sum_{i=1}^{N} \pi_i = 1$ 所以我们可以构造出要极大化的拉格朗日函数（拉格朗日乘子法是一种寻找变量受一个或多个条件所限制的多元函数的极值的方法，后面会专门写一篇关于这个的）:

$$\arg\max_{\pi_{i_1}} \sum_{d=1}^{D} \sum_{i=1}^{N} P(O, i_1^{(d)} = i|\overline{\lambda} \log \pi_{i_1} + \gamma(\sum_{i=1}^{N} \pi_i - 1) \tag{38}$$

其中，γ 为拉格朗日系数。上式对 π_i 求偏导数并令结果为 0，我们得到:

$$\sum_{d=1}^{D} P(O, i_1^{(d)} = i|\overline{\lambda}) + \gamma(\sum_{i=1}^{N} \pi_i - 1) = 0 \tag{39}$$

令 i 分别等于从 1 到 N，从上式可以得到 N 个式子，对这 N 个式子求和可得:

$$\begin{aligned} \sum_{d=1}^{D} P(O|\overline{\lambda}) + \gamma = 0 \\ \gamma = -\sum_{d=1}^{D} P(O|\overline{\lambda}) \end{aligned} \tag{40}$$

从上式代入拉格朗日函数的导函数消去 γ 得到 pi_i 的表达式为:

$$\begin{aligned}\pi_i &= \frac{\sum_{d=1}^{D} P(O, i_1^{(d)} = i|\overline{\lambda})}{\sum_{d=1}^{D} P(O|\overline{\lambda})} \\ &= \frac{\sum_{d=1}^{D} P(O, i_1^{(d)} = i|\overline{\lambda})}{DP(O|\overline{\lambda})} \\ &= \frac{\sum_{d=1}^{D} P(i_1^{(d)} = i|O, \overline{\lambda})}{D} \\ &= \frac{\sum_{d=1}^{D} P(i_1^{(d)} = i|O^{(d)}, \overline{\lambda})}{D}\end{aligned} \tag{41}$$

利用前向概率的定义可得:

$$P(i_1^{(d)} = i|O^d, \overline{\lambda}) = \gamma_1^{(d)}(i) \tag{42}$$

因此在代入前向概率定义后得到最终 M 步 π_i 的迭代公式为:

$$\pi_i = \frac{\sum_{d=1}^{D} \gamma_1^{(d)}(i)}{D} \tag{43}$$

而 A 和 B 也可以通过相同的方式获得。A 可以通过 $\sum_{j=1}^{N} a_{ij} = 1$ 来约束。B 可以通过 $\sum_{k=1}^{M} b_j(o_t = v_k) = 1$ 来约束。A 的最大化函数表达式和拉格朗日函数以及最后得到的迭代公式如下:

$$\begin{aligned}\overline{a_{ij}} &= \arg\max_{a_{ij}} \sum_{d=1}^{D} \sum_{I} \sum_{t=1}^{T-1} P(O, I|\overline{\lambda}) \log a_{i_t i_{t+1}} \\ &= \arg\max_{\pi_i} \sum_{d=1}^{D} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{t=1}^{T-1} P(O, i_t^{(d)} = i, i_{t+1}^{(d)} = j|\overline{\lambda}) \log a_{ij}\end{aligned} \tag{44}$$

$$\arg\max_{a_{ij}} \sum_{d=1}^{D} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{t=1}^{T-1} P(O, i_t^{(d)} = i, i_t^{(d)} = i|\overline{\lambda}) \log a_{ij} + \xi(\sum_{j=1}^{N} a_{ij} - 1) \tag{45}$$

$$a_{ij} = \frac{\sum_{d=1}^{D} \sum_{t=1}^{T-1} \xi_t^(d)(i, j)}{\sum_{d=1}^{D} \sum_{t=1}^{T-1} \gamma_t^(d)(i)} \tag{46}$$

B 的最大化函数表达式和拉格朗日函数以及最后得到的迭代公式如下:

$$\begin{aligned}\overline{b_j(k)} &= \arg\max_{b_j(k)} \sum_{d=1}^{D} \sum_{I} \sum_{t=1}^{T} P(O, I|\overline{\lambda}) \log b_{i_t}(o_t) \\ &= \arg\max_{b_j(k)} \sum_{d=1}^{D} \sum_{j=1}^{N} \sum_{t=1}^{T} P(O, i_t^{(d)} = j|\overline{\lambda}) \log b_j(o_t)\end{aligned} \tag{47}$$

$$\arg\max_{b_j(k)} \sum_{d=1}^{D} \sum_{j=1}^{N} \sum_{t=1}^{T} P(O, i_t^{(d)} = j|\overline{\lambda}) \log b_j(o_t) + \gamma(\sum_{k=1}^{M} b_j(o_t = v_k) - 1) \tag{48}$$

$$b_j(k) = \frac{\sum_{d=1}^{D} \sum_{t=1, o_t^d = v_k}^{T} \gamma_t^(d)(i)}{\sum_{d=1}^{D} \sum_{t=1}^{T} \gamma_t^(d)(i)} \tag{49}$$

有了这些表达公式后，就可以迭代求解 HMM 模型参数了。

5.4 鲍姆-韦尔奇算法 python 实现

python 代码实现如下所示。

```
import numpy as np

def em_algorithm(O_d, N, M):
    """
    :param O: 观察序列 O_d={O1, O2,...,Od}
    :param N: 隐藏状态序列数量 N
    :param M: 观测状态序列数量 M
    :return A: 隐藏状态转移矩阵 A
    :return B: 观察状态概率矩阵 B
    :return PI: 初始状态转移矩阵 PI
    """
    A = np.random.rand(N, N)
    B = np.random.rand(N, M)
    PI = np.random.rand(N)
    gamma = []
    xi = []
    for D in O_d:
        gamma_d = []
        xi_d = []
        _, alpha_d = forward_algorithm(A, B, PI, D)
        _, beta_d = backward_algorithm(A, B, PI, D)
        for t in len(D):
            gamma_d_t = []
            xi_d_t = []
            for i in range(N):
                alpha_d_t_i = alpha_d[t][i]
                beta_d_t_i = beta_d[t][i]
                gamma_d_t_i = alpha_d_t_i*beta_d_t_i
                xi_d_t_i = []
                for j in range(N):
                    xi_d_t_ij = alpha_d_t_i*A[i][j]*b[j][D[t+1]]*beta_d[t+1][j]
                    xi_d_t_i.append(xi_d_t_ij)
```

```
                gamma_d_t.append(alpha_d_t_i)
                xi_d_t.append(xi_d_t_i)
            gamma_d.append(gamma_d_t)
            xi_d.append(xi_d_t)
        gamma_d = gamma_d/sum(gamma_d)
        xi_d = xi_d/sum(xi_d)
        gamma.append(gamma_d)
        xi.append(xi_d)
    for i in range(N):
        pi_i = sum([gamma[d][1][i] for d in range(len(O_d))])/len(O_d)
        PI[i] = pi_i
        for j in rang(N):
            a_ij = sum([(sum([xi[d][t][i][j] for t in range(len(d)-1)])) \
            for d in range(len(O_d))]) sum([(sum([gamma[d][t][i] for t in \
            range(len(d)-1)])) for d in range(len(O_d))])
            A[i][j] = a_ij
        for k in range(M):
            b_ik = sum([sum(sum(gamma[d][t][i] for t in range(len(d))\
            if O_d[d][t] == k)) for d in range(len(O_d))])\
            /sum([(sum([gamma[d][t][i] for t in range(len(d))])) for d in range(len(O_d))])
            B[i][k] = b_ik

    return A, B, PI
```