



NLP

# 自然语言处理技术基础

Natural Language Processing, NLP

网络空间安全与计算机学院

# 第4章 隐马尔科夫模型

4.1 马尔科夫模型

4.2 隐马尔科夫模型(HMM)

4.3 HMM的三个基本问题

## 4.1 马尔科夫模型

**随机过程** Stochastic Process :

也称**随机函数**，是随时间而随机变化的过程。

例如：

某商店在从时间 $t_0$ 到时间 $t_K$ 这段时间内接待顾客的人数，是依赖于时间 $t$ 的一组随机变量，即随机过程。

马尔科夫模型描述了一类重要的随机过程。

将无规则的运动用数学描述出来，对现实生产、生活有着巨大的指导意义。

# 马尔可夫假设 (Markov Assumption)

本周天气:

周二天气 与 周一天气相关;

周三天气 与 周一、周二天气相关;

.....

周日天气 与 周一、周二、周三、周四、周五、周六天气相关。

## 马尔科夫假设:

明天天气只依赖于今天天气, 和昨天天气没有任何关系。

1. 这个假设可以大大的简化问题。
2. 这个假设可能是一个糟糕的假设, 因为很多重要的信息都丢失了。

# 马尔可夫过程 (Markov process)

- “ $t$ 时刻的状态只与 $t-1$ 时刻状态有关”的性质，称为马尔可夫性（或无后效性）。
- 具有这种性质的过程称为马尔可夫过程。
- 时间、状态都是离散的马尔可夫过程称为马尔可夫链。



# 马尔可夫模型 (Markov Model)

一阶马尔科夫过程，独立于时间t的随机过程：

$$p(q_i = s_j | q_{i-1} = s_i) = \alpha_{ij}, \quad 1 \leq i, j \leq N$$

则得到马尔科夫模型

$\alpha_{ij}$  : 状态转移概率

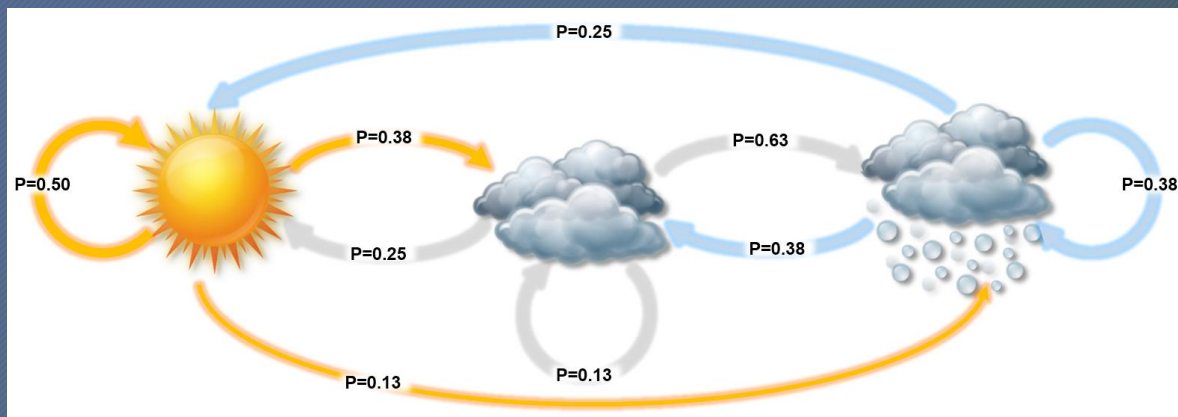
N个状态的一阶马尔科夫模型有N\*N个种状态转移

# 状态转移矩阵

N个状态的一阶马尔科夫模型有N\*N个种状态转移

天气有三种状态：晴朗、多云、下雨

状态转移关系图：



状态转移矩阵：

	晴朗	多云	下雨
晴朗	0.50	0.375	0.125
多云	0.25	0.125	0.625
下雨	0.25	0.375	0.375

## 4.2 隐马尔科夫模型(HMM)

**某人无法直观的观察天气情况，  
但是能观察海藻。**

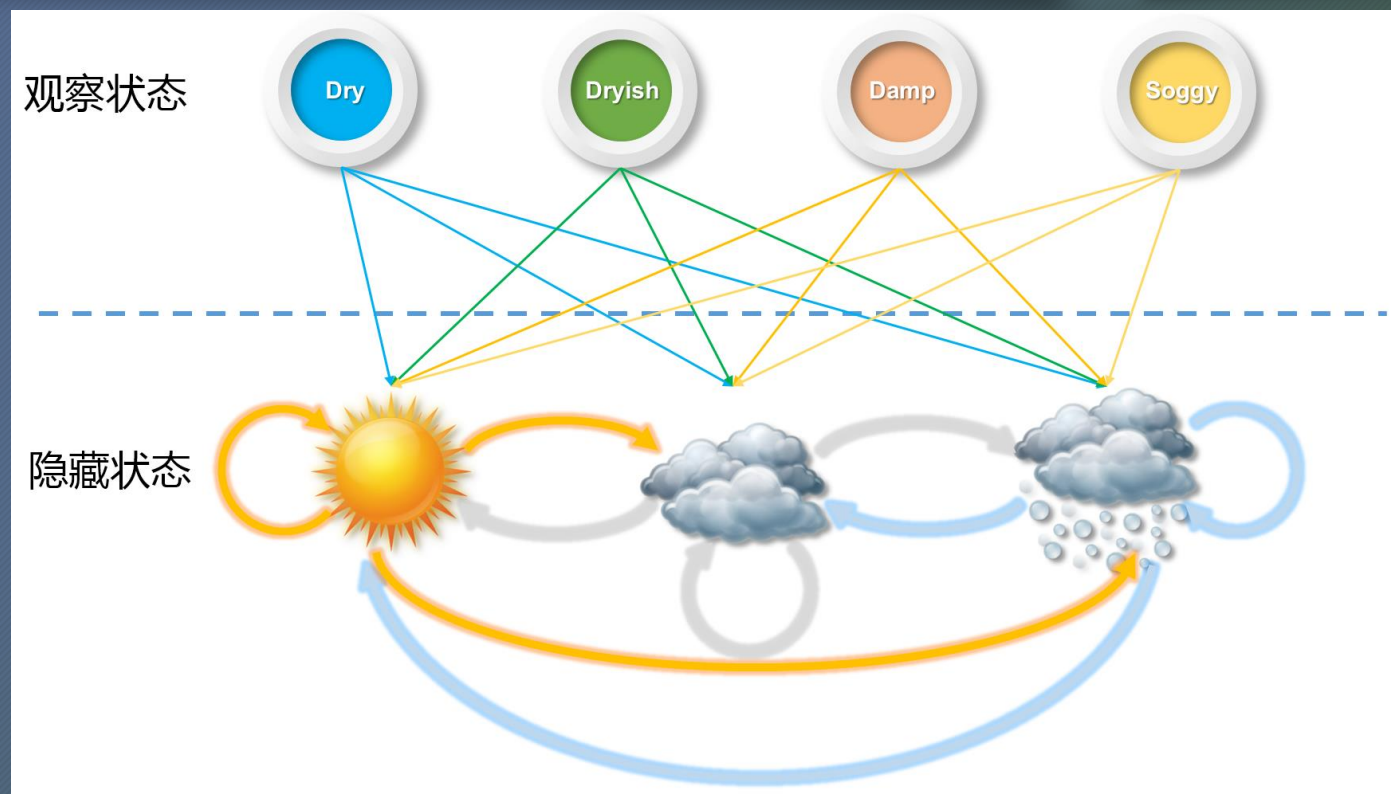
假设：海藻的状态和天气的情况相关的。

状态集合：

1. 观察的状态集合（海藻的状态）
2. 隐藏的状态集合（天气的状况）

**隐马尔可夫模型：**

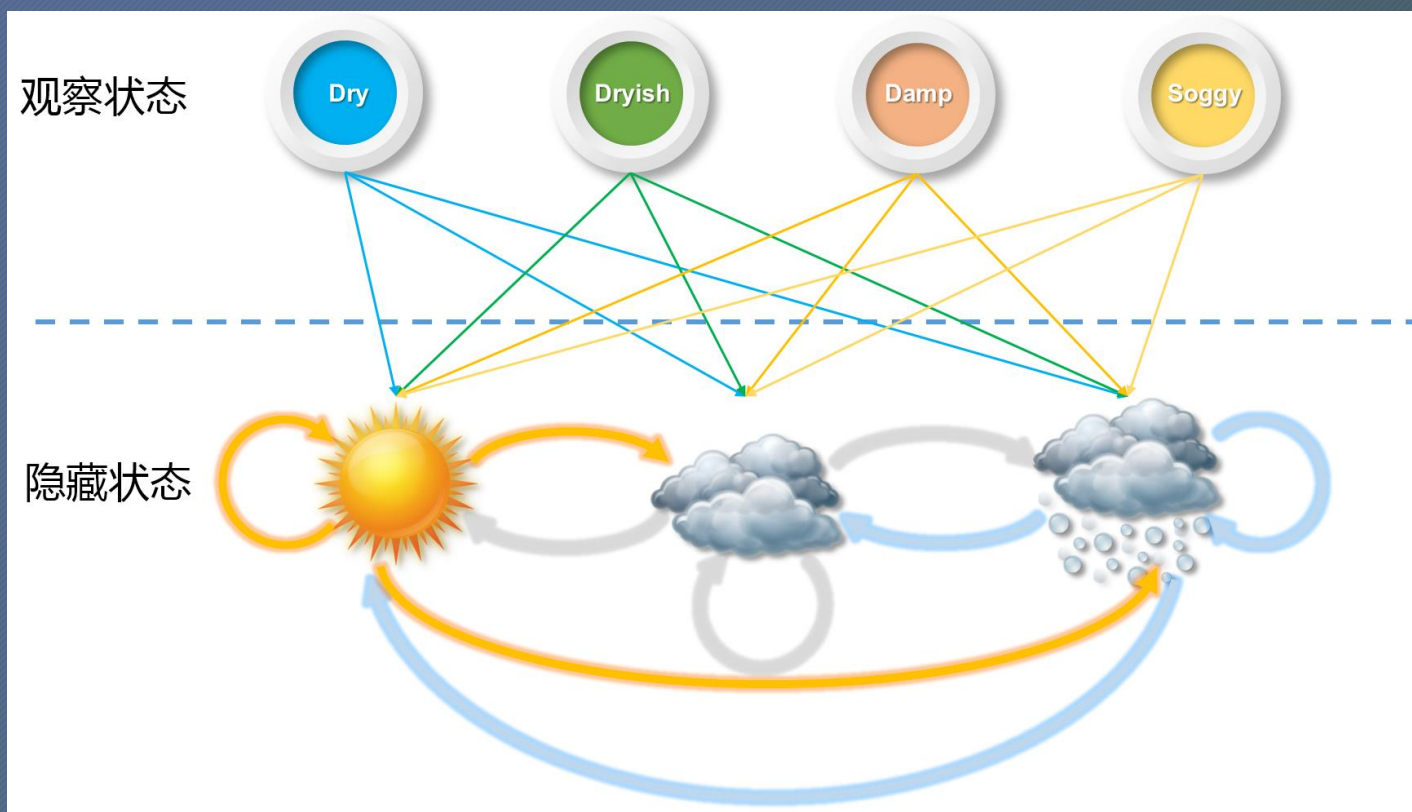
**根据海藻的状况和马尔科夫假设预测天气。**





# 发射概率矩阵

	干燥	稍干	潮湿	湿透
晴朗	0.60	0.20	0.15	0.05
多云	0.25	0.25	0.25	0.25
下雨	0.05	0.10	0.35	0.50



## 4.3 HMM的三个基本问题

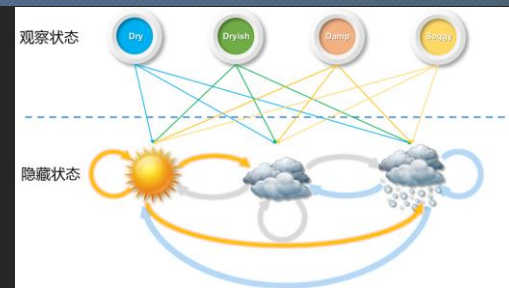
4.3.1 求解观察值序列的概率

4.3.2 确定最优状态序列

4.3.3 HMM的参数估计



# HMM的三个基本问题



## 1. 给定**模型**，如何有效计算产生**观测序列**的概率？

- 根据HMM得到一个可观察状态序列的概率( 评价 )
- **前向算法**(Forward Algorithm)、**后向算法**(Backward Algorithm)

## 2. 给定**模型**和**观测序列**，如何找到最匹配的**状态序列**？

- 找到一个隐藏状态的序列使得这个序列产生一个可观察状态序列的概率最大( 解码 )
- **维特比算法**(Viterbi Algorithm)

## 3. 给定**观测序列**，如何调整**模型参数**使得该序列出现的概率最大？

- 根据一个可以观察到的状态序列集, 产生一个HMM ( 学习 )
- **最大期望算法** (Expectation-Maximization algorithm, EM) 、**鲍姆-韦尔奇算法**(Baum-Welch Algorithm)



## 4.3.1 求解观察值序列的概率

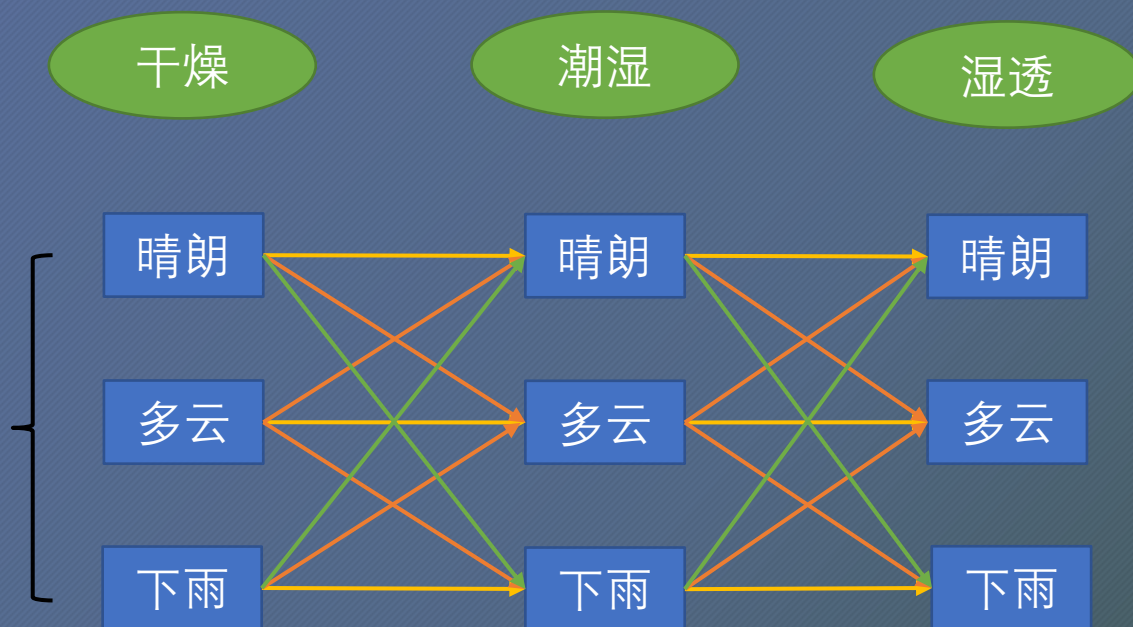
观察值序列

干燥

潮湿

湿透

隐状态



所有可能的天气序列，共27种。

状态转移矩阵:

	晴朗	多云	下雨
晴朗	0.50	0.375	0.125
多云	0.25	0.125	0.625
下雨	0.25	0.375	0.375

发射概率矩阵:

	干燥	稍干	潮湿	湿透
晴朗	0.60	0.20	0.15	0.05
多云	0.25	0.25	0.25	0.25
下雨	0.05	0.10	0.35	0.50



## 4.3.1 求解观察值序列的概率

初始概率矩阵  $\pi$  :

晴朗	多云	下雨
(0.63	0.17	0.20)

发射概率矩阵  $b$  :

	干燥	稍干	潮湿	湿透
晴朗	0.60	0.20	0.15	0.05
多云	0.25	0.25	0.25	0.25
下雨	0.05	0.10	0.35	0.50

状态转移矩阵  $a$  :

	晴朗	多云	下雨
晴朗	0.50	0.375	0.125
多云	0.25	0.125	0.625
下雨	0.25	0.375	0.375

$P(\text{干燥, 潮湿, 湿透}) =$

$$\sum_{k=0}^{26} P(\text{干燥, 潮湿, 湿透}, Q_k)$$

$Q_k$ : 所有可能的天气序列, 共27种。

数学推导过程参考教材54页

$Q_i = (\text{下雨, 多云, 晴朗})$

$P(\text{干燥, 潮湿, 湿透, 下雨, 多云, 晴朗}) =$

$$[\pi_{\text{下雨}} \times b_{\text{下雨, 干燥}}] \times [a_{\text{下雨, 多云}} \times b_{\text{多云, 潮湿}}] \times [a_{\text{多云, 晴朗}} \times b_{\text{晴朗, 湿透}}] =$$
$$[0.20 \times 0.05] \times [0.625 \times 0.25] \times [0.375 \times 0.05] =$$
$$2.9 \times 10^{-5}$$

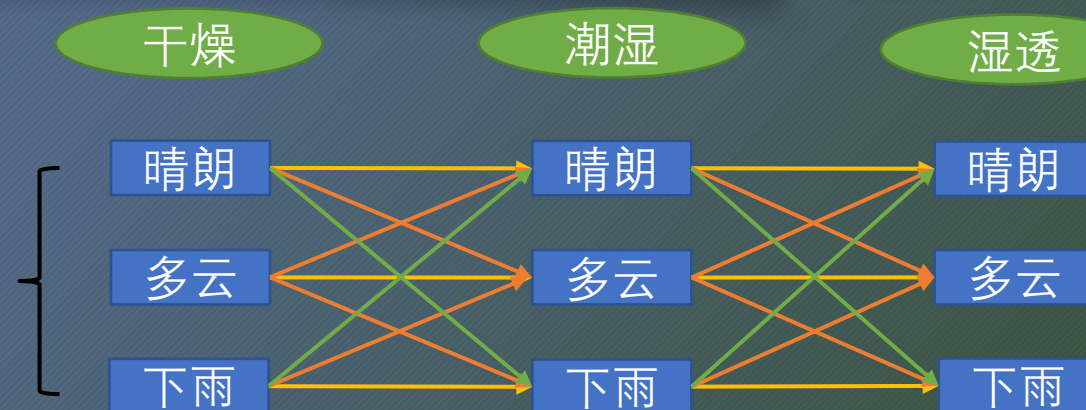
观察值序列

干燥

潮湿

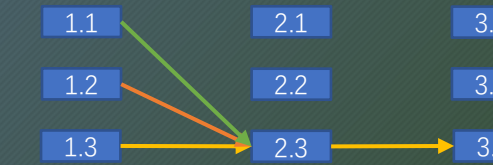
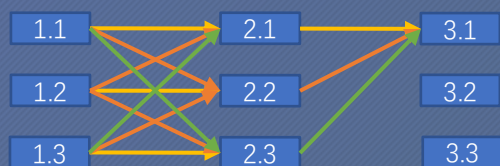
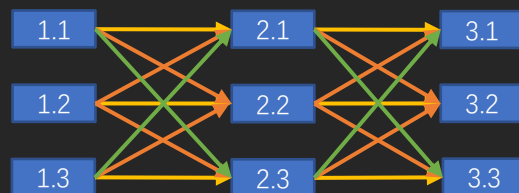
湿透

隐状态



**存在问题:** 隐藏状态序列个数呈指数级增长; 有许多重复计算的部分

# 计算过程分解



重复计算的解决方案：  
前向算法

包括公共计算部分  
利用前向变量记录

# 前向算法 (Forward Algorithm)

① 初始化:

$$\alpha_1(i) = \pi_i b_i(o_1), \quad 1 \leq i \leq N$$

② 递推计算:

$$\alpha_{t+1}(j) = \left( \sum_{i=1}^N \alpha_t(i) a_{ij} \right) b_j(o_{t+1}), \quad 1 \leq j \leq N, 1 \leq t \leq T-1$$

③ 求和结束:

$$p(O|\mu) = \sum_{i=1}^N \alpha_T(i)$$



① 初始化:  $\alpha_1(i) = \pi_i b_i(o_1), \quad 1 \leq i \leq N$

② 递归计算:

$$\alpha_{t+1}(j) = \left( \sum_{i=1}^N \alpha_t(i) a_{ij} \right) b_j(o_{t+1}), \quad 1 \leq j \leq N, 1 \leq t \leq T-1$$

③ 求和结束:  $p(O|\mu) = \sum_{i=1}^N \alpha_T(i)$

## 前向算法 举例

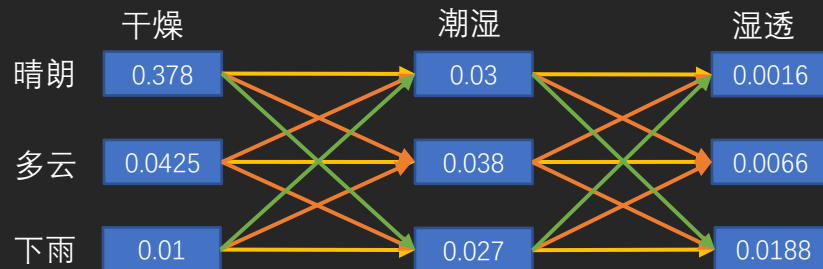
初始概率矩阵  $\pi$ :

晴朗	多云	下雨
(0.63	0.17	0.20)

$$\alpha_1(\text{晴朗}) = \pi_{\text{晴朗}} b_{\text{晴朗}}(\text{干燥}) = 0.63 \times 0.60 = 0.378$$

$$\alpha_1(\text{多云}) = \pi_{\text{多云}} b_{\text{多云}}(\text{干燥}) = 0.17 \times 0.25 = 0.0425$$

$$\alpha_1(\text{下雨}) = \pi_{\text{下雨}} b_{\text{下雨}}(\text{干燥}) = 0.20 \times 0.05 = 0.01$$



发射概率矩阵  $b$ :

	干燥	稍干	潮湿	湿透
晴朗	0.60	0.20	0.15	0.05
多云	0.25	0.25	0.25	0.25
下雨	0.05	0.10	0.35	0.50

状态转移矩阵  $a$ :

	晴朗	多云	下雨
晴朗	0.50	0.375	0.125
多云	0.25	0.125	0.625
下雨	0.25	0.375	0.375

$$\begin{aligned} \alpha_2(\text{晴朗}) &= \left[ \alpha_1(\text{晴朗}) a_{\text{晴朗}, \text{晴朗}} + \alpha_1(\text{多云}) a_{\text{多云}, \text{晴朗}} + \alpha_1(\text{下雨}) a_{\text{下雨}, \text{晴朗}} \right] \times b_{\text{晴朗}}(\text{潮湿}) \\ &= [0.378 \times 0.5 + 0.0425 \times 0.25 + 0.01 \times 0.25] \times 0.15 \approx 0.03 \end{aligned}$$

$$\alpha_2(\text{多云}) = \left[ \alpha_1(\text{晴朗}) a_{\text{晴朗}, \text{多云}} + \alpha_1(\text{多云}) a_{\text{多云}, \text{多云}} + \alpha_1(\text{下雨}) a_{\text{下雨}, \text{多云}} \right] \times b_{\text{多云}}(\text{潮湿})$$

$$\alpha_2(\text{下雨}) = \left[ \alpha_1(\text{晴朗}) a_{\text{晴朗}, \text{下雨}} + \alpha_1(\text{多云}) a_{\text{多云}, \text{下雨}} + \alpha_1(\text{下雨}) a_{\text{下雨}, \text{下雨}} \right] \times b_{\text{下雨}}(\text{潮湿})$$



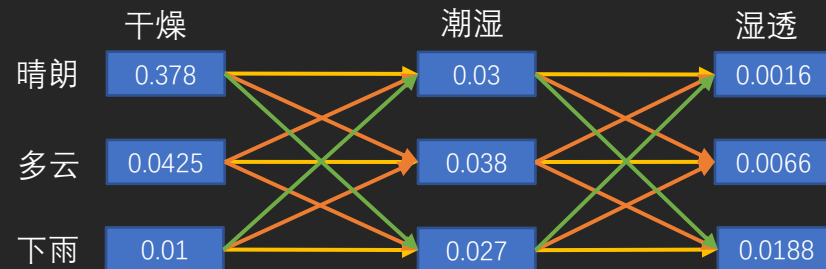
① 初始化:  $\alpha_1(i) = \pi_i b_i(o_1), \quad 1 \leq i \leq N$

② 递归计算:

$$\alpha_{t+1}(j) = \left( \sum_{i=1}^N \alpha_t(i) a_{ij} \right) b_j(o_{t+1}), \quad 1 \leq j \leq N, 1 \leq t \leq T-1$$

③ 求和结束:  $p(O|\mu) = \sum_{i=1}^N \alpha_T(i)$

## 前向算法 举例



初始概率矩阵  $\pi$ :

晴朗	多云	下雨
(0.63	0.17	0.20)

状态转移矩阵  $a$ :

	晴朗	多云	下雨
晴朗	0.50	0.375	0.125
多云	0.25	0.125	0.625
下雨	0.25	0.375	0.375

发射概率矩阵  $b$ :

	干燥	稍干	潮湿	湿透
晴朗	0.60	0.20	0.15	0.05
多云	0.25	0.25	0.25	0.25
下雨	0.05	0.10	0.35	0.50

$$\alpha_3(\text{晴朗}) = \left[ \alpha_2(\text{晴朗})a_{\text{晴朗}, \text{晴朗}} + \alpha_2(\text{多云})a_{\text{多云}, \text{晴朗}} + \alpha_2(\text{下雨})a_{\text{下雨}, \text{晴朗}} \right] \times b_{\text{晴朗}}(\text{湿透})$$

$$\alpha_3(\text{多云}) = \left[ \alpha_2(\text{晴朗})a_{\text{晴朗}, \text{多云}} + \alpha_2(\text{多云})a_{\text{多云}, \text{多云}} + \alpha_2(\text{下雨})a_{\text{下雨}, \text{多云}} \right] \times b_{\text{多云}}(\text{湿透})$$

$$\alpha_3(\text{下雨}) = \left[ \alpha_2(\text{晴朗})a_{\text{晴朗}, \text{下雨}} + \alpha_2(\text{多云})a_{\text{多云}, \text{下雨}} + \alpha_2(\text{下雨})a_{\text{下雨}, \text{下雨}} \right] \times b_{\text{下雨}}(\text{湿透})$$

$$\alpha = \alpha_3(\text{晴朗}) + \alpha_3(\text{多云}) + \alpha_3(\text{下雨}) = 0.0016 + 0.0066 + 0.0188 = 0.027$$

# Python实现前向算法

```
def hmm_forward(A, B, pi, O):
    T = len(O)
    N = len(A[0])

    #step1 初始化
    print("====1. 初始化====")
    alpha = [[0]*T for _ in range(N)]
    for i in range(N):
        alpha[i][0] = pi[i]*B[i][O[0]]
        print('alpha 1 (', i, ') = ', alpha[i][0])

    #step2 计算alpha(t)
    print("\n====2. 递归====")
    for t in range(1, T):
        for i in range(N):
            temp = 0
            for j in range(N):
                temp += alpha[j][t-1]*A[j][i]
            alpha[i][t] = temp*B[i][O[t]]
            print('alpha', t+1, ' (', i, ') = ', alpha[i][t])
        print("-----")

    #step3
    print("\n====3. 求和====")
    proba = 0
    for i in range(N):
        proba += alpha[i][-1]
    print("alpha = ", proba)

    return
```

```
A = [[0.5, 0.375, 0.125], [0.25, 0.125, 0.625], [0.25, 0.375, 0.375]] #状态转移矩阵
B = [[0.60, 0.20, 0.15, 0.05], [0.25, 0.25, 0.25, 0.25], [0.05, 0.10, 0.35, 0.5]] #发射概率矩阵
pi = [0.63, 0.17, 0.20] #初始概率矩阵
```

```
O = [0, 2, 3] #0干燥、1稍干、2潮湿、3湿透
hmm_forward(A, B, pi, O)
```

```
====1. 初始化====
alpha 1 ( 0 ) = 0.378
alpha 1 ( 1 ) = 0.0425
alpha 1 ( 2 ) = 0.010000000000000002
```

```
====2. 递归====
alpha 2 ( 0 ) = 0.03031875
alpha 2 ( 1 ) = 0.037703125
alpha 2 ( 2 ) = 0.027146875
-----
```

```
alpha 3 ( 0 ) = 0.0015685937499999997
alpha 3 ( 1 ) = 0.006565625
alpha 3 ( 2 ) = 0.0187671875
-----
```

```
====3. 求和====
alpha = 0.026901406250000003
```

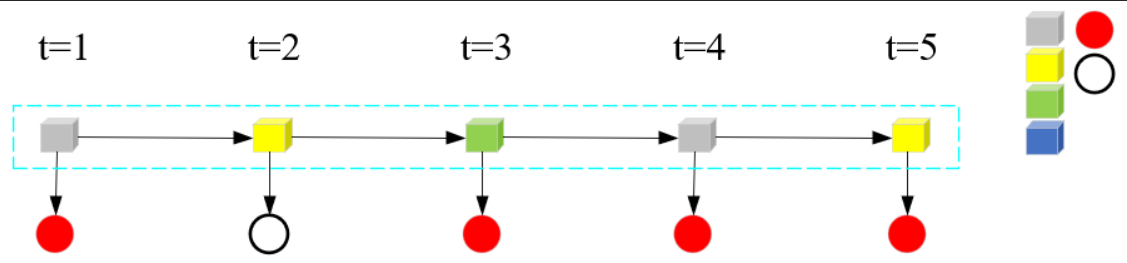
# 后向算法

- 后向算法与前向算法类似
- 前向算法和后向算法相结合的方法



观察者只能观测到球的颜色的序列，观测不到球是从哪个盒子取出的

# 红白球



O=(红, 白, 红, 红, 红)

- ◆ 桌上有4个外观一模一样的盒子，摆放一排
- ◆ 每个盒子装有10个一模一样的圆球
- ◆ 每个球只能是红色或白色

开始，从4个盒子里以等概率随机选取一个盒子，从这个盒子里随机抽出一个球，记录其颜色，放回；

然后，从当前盒子随机转移到下一个盒子，规则是：

如果当前盒子是盒子1，那么下一个盒子一定是盒子2，  
如果当前盒子是盒子2或者3，那么分别以概率0.4和0.6转移到左边或者右边盒子，  
如果当前盒子是4，那么各以0.5概率停留在盒子4或者转移到盒子3；

确定转移盒子之后，再从这个盒子随机抽出一个球，记录其颜色，放回；  
如此重复

隐藏状态：Q={盒子1, 盒子2, 盒子3, 盒子4}, N=4  
观测状态：V={红, 白}, M=2

盒子	1	2	3	4
红球数	5	3	6	8
白球数	5	7	4	2

初始概率分布： $\pi = (0.25, 0.25, 0.25, 0.25)^T$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.4 & 0 & 0.6 & 0 \\ 0 & 0.4 & 0 & 0.6 \\ 0 & 0 & 0.5 & 0.5 \end{bmatrix}$$

状态转移概率分布A

$$B = \begin{bmatrix} 0.5 & 0.5 \\ 0.3 & 0.7 \\ 0.6 & 0.4 \\ 0.8 & 0.2 \end{bmatrix}$$

观测（发射）概率分布B



# HMM的3个问题:

## 概率计算问题, 学习问题, 预测问题

概率计算问题描述如下:

已知一个隐马尔可夫模型  $\lambda = (A, B, \pi)$ , 已知一个观测序列  $O = (o_1, o_2, \dots, o_T)$

问题: 请计算在模型  $\lambda$  下观测序列  $O$  出现的概率  $P(O|\lambda)$

(1) 初值

$$\alpha_1(i) = \pi_i b_i(o_1), \quad i = 1, 2, \dots, N$$

(2) 递推 对  $t = 1, 2, \dots, T-1$ ,

$$\alpha_{t+1}(i) = \left[ \sum_{j=1}^N \alpha_t(j) a_{ji} \right] b_i(o_{t+1}), \quad i = 1, 2, \dots, N$$

(3) 终止

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

盒子	1	2	3	4
红球数	5	3	6	8
白球数	5	7	4	2

求观测序列概率:

给定HMM  $\lambda = (A, B, \pi)$

已知观测序列  $O = (\text{红}, \text{白}, \text{红})$ ,  $T=3$ ,

请用前向算法求  $P(O|\lambda)$

# HMM的3个问题：

概率计算问题， 学习问题， 预测问题

也称：解码(decoding)问题

已知隐马尔科夫模型 $\lambda=(A, B, \pi)$ ， 一个观测序列 $O=(o_1, o_2, \dots, o_T)$

问题： 求给定观测序列条件概率 $P(I|O)$ 最大的状态序列 $I=(i_1, i_2, \dots, i_T)$

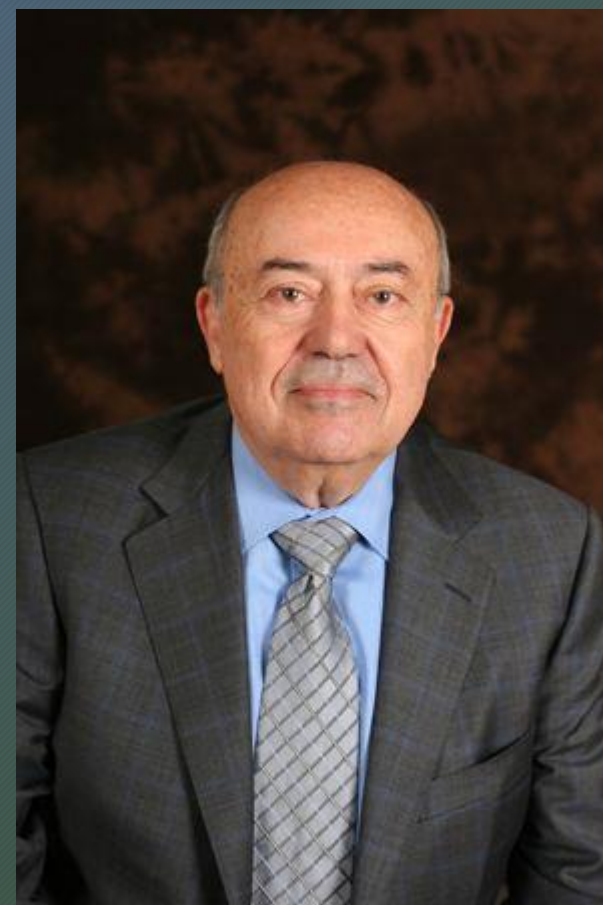
注： 即给定模型 $\lambda$ 和观测序列 $O$ ， 求最有可能的对应的状态序列。

例： 已知观测序列 $O=(\text{干燥}, \text{潮湿}, \text{湿透})$ ； 那么， 这三天最可能的天气序列是什么？

**维特比算法 ( Viterbi Algorithm ) (Viterbi, 1967)**

# 安德鲁·维特比 Andrew J. Viterbi

- CDMA之父、IEEE Fellow
  - 1935年出生在意大利
  - 1967年发明维特比算法，用来对卷积码数据进行译码
  - 1985年作为参与者之一创建了高通公司
  - 2007年获得美国国家科学奖章
- 
- 维特比算法(Viterbi Algorithm)是一种动态规划算法。
  - 用于在数字通信链路中解卷积以消除噪音。
  - 被广泛应用于CDMA和GSM数字蜂窝网络、拨号调制解调器、卫星、深空通信和802.11无线网络中解卷积码。
  - 现常用于语音识别、关键字识别、计算语言学和生物信息学中。



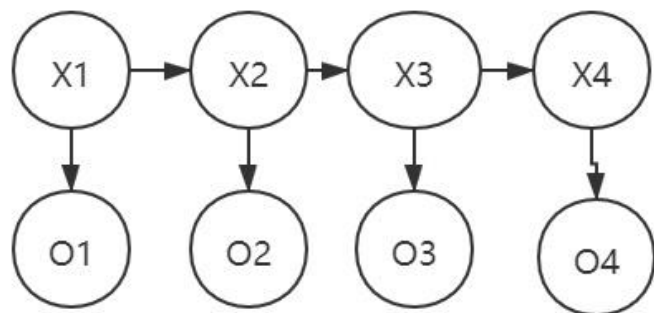


# 维特比算法求解HMM问题

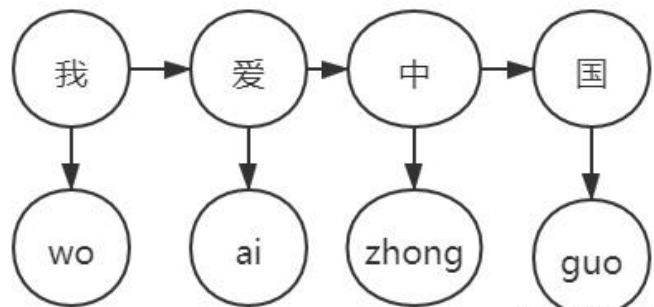
<https://www.zhihu.com/question/20136144/answer/1318908972>

问题描述如下：

首先，我们已经知道状态序列X会产生观测序列O：

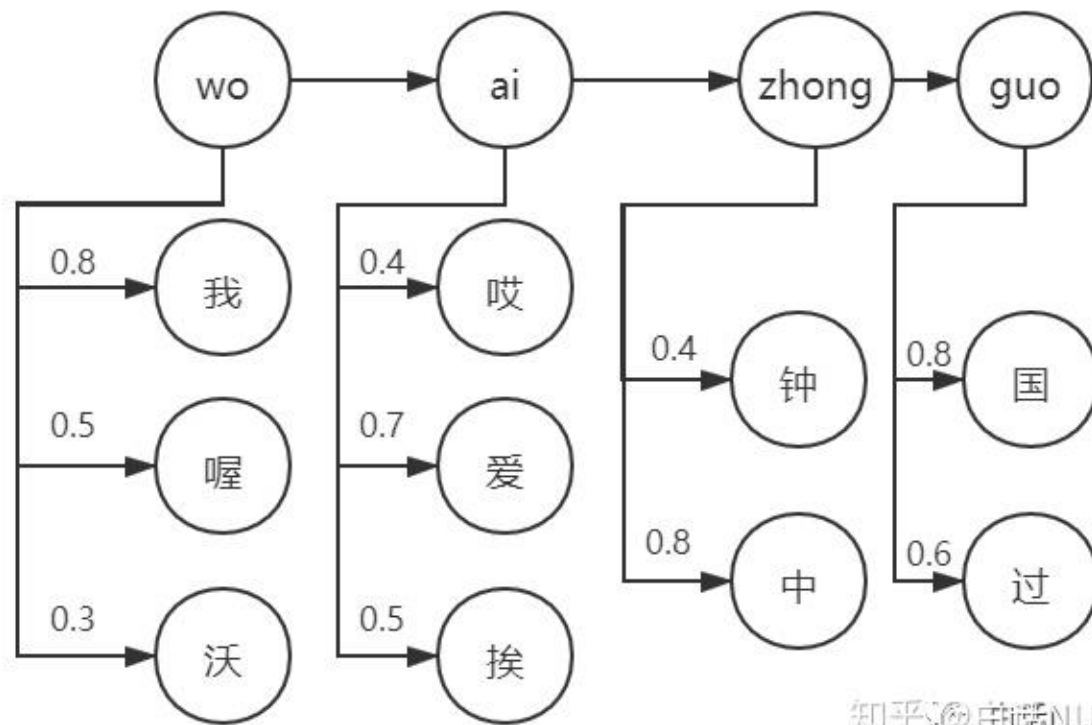


打个比方通过拼音识别汉字，我们观测到的是拼音



知乎 @白话NLP

观测序列O对应的状态序列X有很多种可能，每种都有概率，如下所示，比如wo可能有三种可能：喔、我、沃

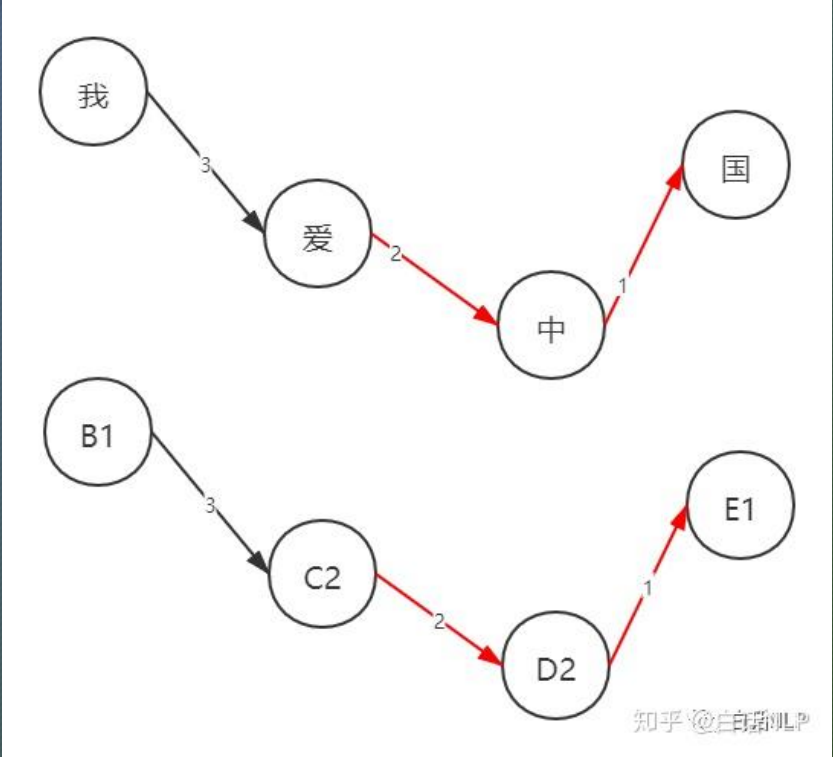
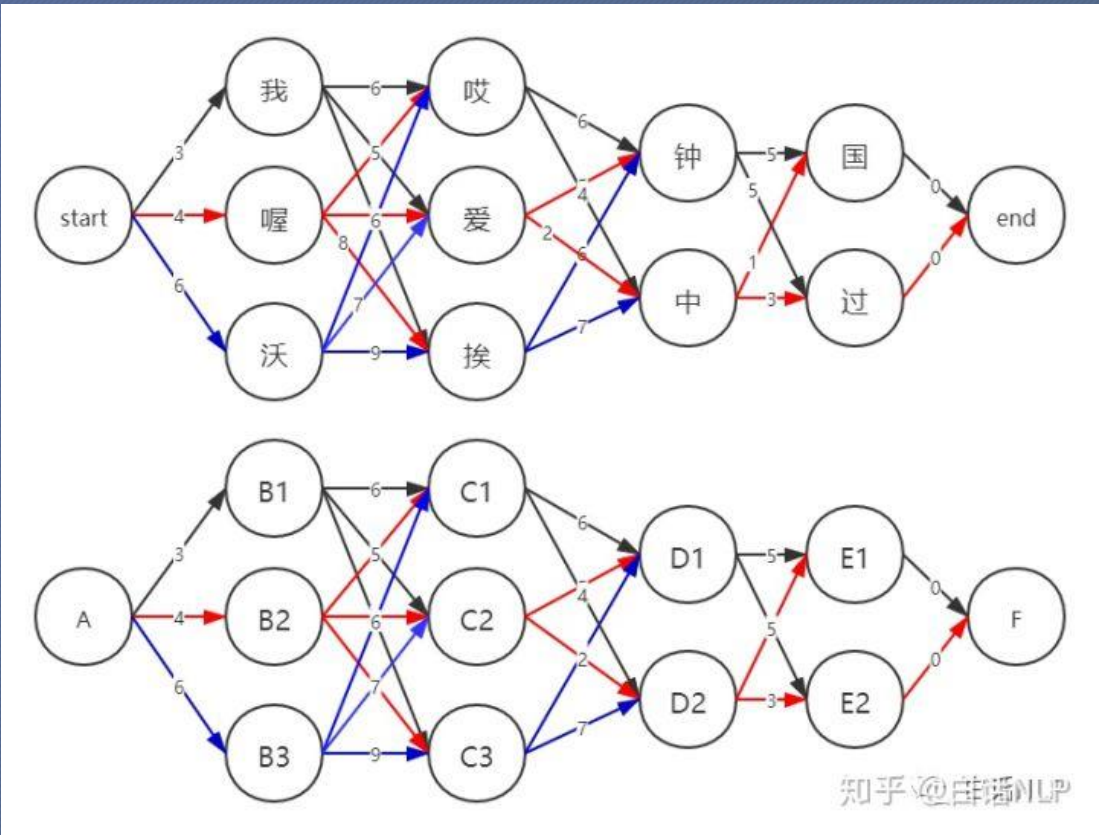


知乎 @白话NLP



# 维特比算法求解HMM问题

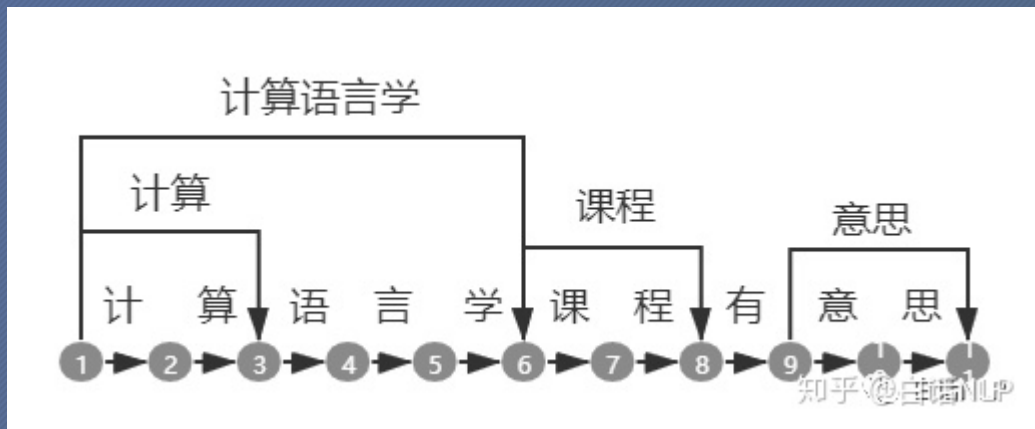
那么其实问题就变成了最大概率问题，把概率最大的理解为路径最短，转换为了求解最短路径问题：（为了方便标记，标记为了下图中的字母）



最终得到了A->E的最短路径A->B1->C2->D2->E1，至此，找到了wo ai zhong guo对应的概率最大的中文汉字组合为：我爱中国。

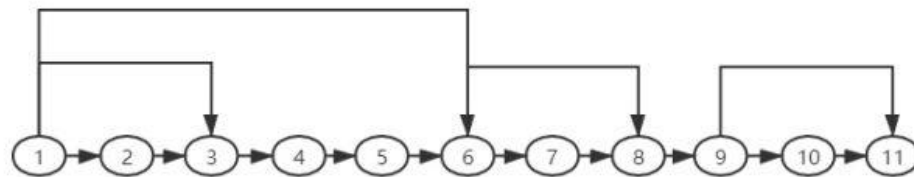
# 维特比算法与分词

利用维特比算法进行分词，和使用最短路径Dijkstra算法进行分词类似，也是最短路径法的分词：  
如下，假设各点之间权重为1：



最短路径为1->6->8->9->11,  
分词结果为[计算语言学、课程、有、意思]

维特比算法过程为：



1->2:  
最短为1

1->3:  
1->2->3为2  
1->3为1  
最短为1

1->4:  
1->3最短+3->4=1+1=2

1->5:  
1->4最短+4->5=2+1=3

1->6:  
1->5最短+5->6=3+1=4  
1->6为1  
最短为1

1->7:  
1->6最短+6->7=1+1=2

1->8:  
1->7最短+7->8=2+1=3  
1->6最短+6->8=1+1=2  
最短为2

1->9:  
1->8最短+8->9=2+1=3

1->10:  
1->9最短+9->10=3+1=4

1->11:  
1->10最短+10->11=4+1=5  
1->9最短+9->11=3+1=4

## 4.3.2 确定最优状态序列

给定模型和观测序列，如何找到最匹配的状态序列？

### 维特比算法 ( Viterbi Algorithm )

①初始化：

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

$$\psi(i) = 0$$

②递推计算：

$$\delta_{t+1}(i) = \max_j [\delta_t(j) \times a_{ji}] \times b_i(o_{t+1}), \quad 1 \leq i \leq N, 1 \leq t \leq T - 1$$

$$\psi_{t+1}(i) = \operatorname{argmax}_j [\delta_t(j) \times a_{ji}]$$

③路径回溯：

$$\hat{q}_T = \operatorname{argmax}_i \delta_T(i)$$

$$\hat{q}_{t-1} = \psi_t(\hat{q}_t), \quad 2 \leq t \leq T$$

$y = \operatorname{argmax} f(t)$  代表：y 是 f(t) 函数中，会产生最大 output 的那个参数 t。即，反向指针寻找最佳前驱节点。



①初始化:  $\delta_1(i) = \pi_i b_i(o_1), 1 \leq i \leq N$   
 $\psi(i) = 0$

②递归计算:  $\delta_{t+1}(i) = \max_j [\delta_t(j) \times a_{ji}] \times b_i(o_{t+1}), 1 \leq i \leq N, 1 \leq t \leq T-1$   
 $\psi_{t+1}(i) = \arg\max_j [\delta_t(j) \times a_{ji}] \times b_i(o_{t+1})$

③路径回溯:  $\hat{q}_{t-1} = \psi_t(\hat{q}_t), 2 \leq t \leq T$   
 $\hat{q}_T = \arg\max_i \delta_T(i)$

## 4.3.2 确定最优状态序列

状态转移矩阵  $a$ :

	晴朗	多云	下雨
晴朗	0.50	0.375	0.125
多云	0.25	0.125	0.625
下雨	0.25	0.375	0.375

发射概率矩阵  $b$ :

	干燥	稍干	潮湿	湿透
晴朗	0.60	0.20	0.15	0.05
多云	0.25	0.25	0.25	0.25
下雨	0.05	0.10	0.35	0.50

初始概率矩阵  $\pi$ :

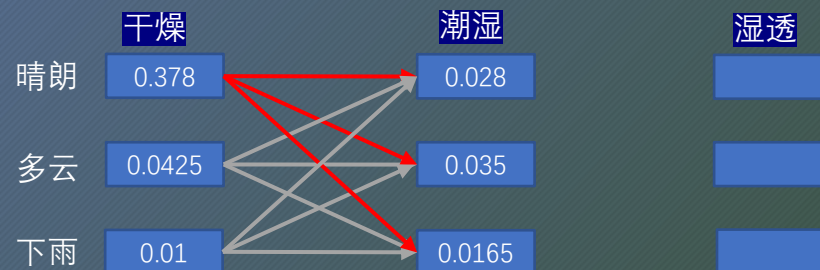
晴朗	多云	下雨
0.63	0.17	0.20

例: 已知观测序列  $O=(\text{干燥}, \text{潮湿}, \text{湿透})$ ; 那么, 这三天最可能的天气序列是什么?

$$\delta_1(\text{晴朗}) = \pi_{\text{晴朗}} b_{\text{晴朗}}(\text{干燥}) = 0.63 \times 0.60 = 0.378$$

$$\delta_1(\text{多云}) = \pi_{\text{多云}} b_{\text{多云}}(\text{干燥}) = 0.17 \times 0.25 = 0.0425$$

$$\delta_1(\text{下雨}) = \pi_{\text{下雨}} b_{\text{下雨}}(\text{干燥}) = 0.20 \times 0.05 = 0.01$$



$$\begin{aligned} \delta_2(\text{晴朗}) &= \max [\delta_1(\text{晴朗}) a_{\text{晴朗}, \text{晴朗}}, \delta_1(\text{多云}) a_{\text{多云}, \text{晴朗}}, \delta_1(\text{下雨}) a_{\text{下雨}, \text{晴朗}}] \times b_{\text{晴朗}}(\text{潮湿}) \\ &= \max [0.378 \times 0.5, 0.0425 \times 0.25, 0.01 \times 0.25] \times 0.15 \\ &= \max [0.189, 0.010625, 0.0025] \times 0.15 = 0.189 \times 0.15 = 0.02835 \approx 0.028 \end{aligned}$$

$\psi_2(\text{晴朗}) = \text{晴朗}$

同理可得:

$$\delta_2(\text{多云}) = \max [0.14175, 0.0053313, 0.00375] \times 0.25 = 0.035$$

$\psi_2(\text{多云}) = \text{晴朗}$

$$\delta_2(\text{下雨}) = \max [0.04725, 0.0265625, 0.00375] \times 0.35 = 0.0165$$

$\psi_2(\text{下雨}) = \text{晴朗}$

①初始化:  $\delta_1(i) = \pi_i b_i(o_1), 1 \leq i \leq N$   
 $\psi(i) = 0$

②递归计算:  $\delta_{t+1}(i) = \max_j [\delta_t(j) \times a_{ji}] \times b_i(o_{t+1}), 1 \leq i \leq N, 1 \leq t \leq T-1$   
 $\psi_{t+1}(i) = \operatorname{argmax}_j [\delta_t(j) \times a_{ji}] \times b_i(o_{t+1})$

③路径回溯:  $\hat{q}_{t-1} = \psi_t(\hat{q}_t), 2 \leq t \leq T$   
 $\hat{q}_T = \operatorname{argmax}_i \delta_T(i)$

## 4.3.2 确定最优状态序列

状态转移矩阵  $a$ :

	晴朗	多云	下雨
晴朗	0.50	0.375	0.125
多云	0.25	0.125	0.625
下雨	0.25	0.375	0.375

发射概率矩阵  $b$ :

	干燥	稍干	潮湿	湿透
晴朗	0.60	0.20	0.15	0.05
多云	0.25	0.25	0.25	0.25
下雨	0.05	0.10	0.35	0.50

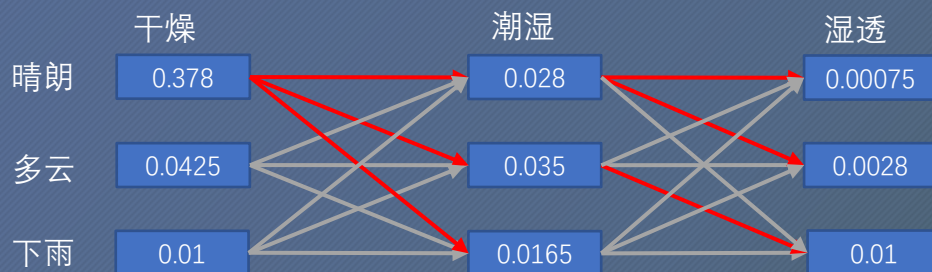
初始概率矩阵  $\pi$ :

晴朗	多云	下雨
0.63	0.17	0.20

$$\begin{aligned} \delta_3(\text{晴朗}) &= \max \left[ \delta_2(\text{晴朗})a_{\text{晴朗}, \text{晴朗}}, \delta_2(\text{多云})a_{\text{多云}, \text{晴朗}}, \delta_2(\text{下雨})a_{\text{下雨}, \text{晴朗}} \right] \times b_{\text{晴朗}}(\text{湿透}) \\ &= \max[0.03 \times 0.5, 0.035 \times 0.25, 0.0165 \times 0.25] \times 0.05 \\ &= \max[\mathbf{0.015}, 0.00875, 0.004125] \times 0.05 = 0.015 \times 0.05 = 0.00075 \quad \psi_3(\text{晴朗}) = \text{晴朗} \end{aligned}$$

同理可得:

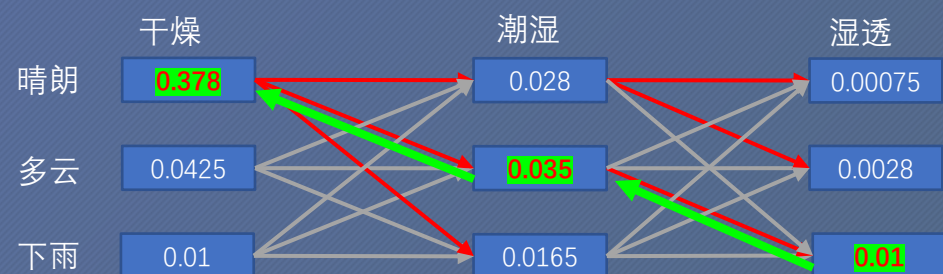
$$\begin{aligned} \delta_3(\text{多云}) &= \max[\mathbf{0.01125}, 0.004375, 0.00624375] \times 0.25 = 0.00028 \quad \psi_3(\text{多云}) = \text{晴朗} \\ \delta_3(\text{下雨}) &= \max[0.00375, \mathbf{0.021875}, 0.00624375] \times 0.50 = 0.01 \quad \psi_3(\text{下雨}) = \text{多云} \end{aligned}$$





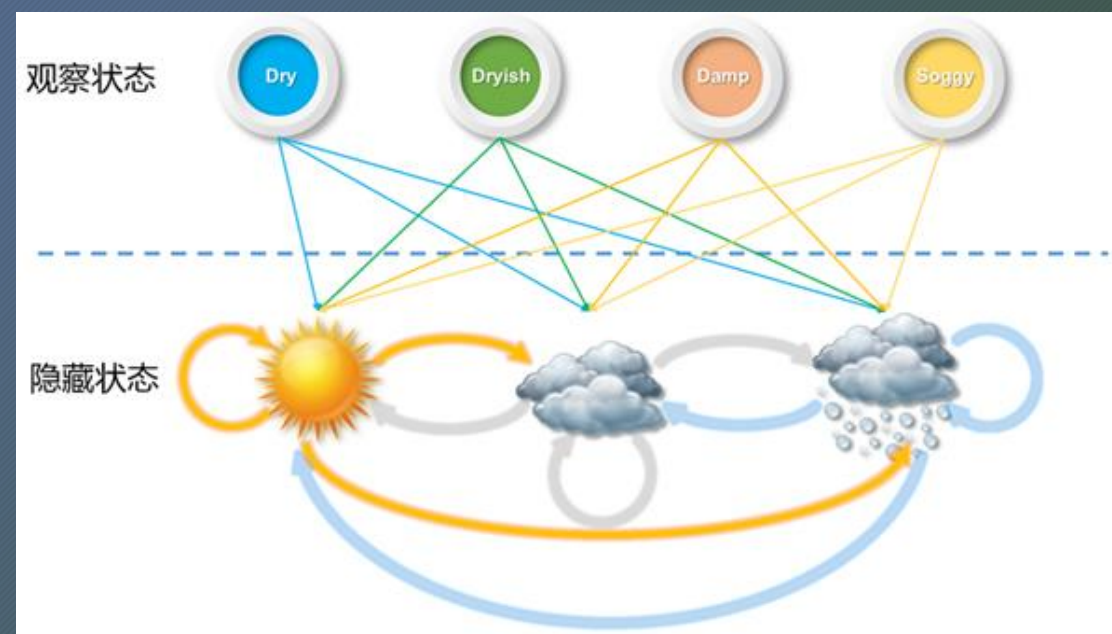
# 维特比算法 (Viterbi Algorithm)

③路径回溯:  $\hat{q}_T = \operatorname{argmax}_i \delta_T(i)$   $\hat{q}_{t-1} = \psi_t(\hat{q}_t), 2 \leq t \leq T$



因此，观测值序列：“干燥，潮湿，湿透”

对应的 最优状态序列为：“晴朗，多云，下雨”





```
def hmm_viterbi(A, B, pi, O):
    T = len(O)
    N = len(A[0])

    delta = [[0]*N for _ in range(T)]
    psi = [[0]*N for _ in range(T)]

    #step1: init
    print("=====1. 初始化=====")
    for i in range(N):
        delta[0][i] = pi[i]*B[i][O[0]]
        psi[0][i] = 0
        print('delta 1 (', i, ')=' , delta[0][i])

    #step2: iter
    print("\n=====2. 递推, 对 t=2, 3, ..., T=====")
    for t in range(1, T):
        for i in range(N):
            temp, maxindex = 0, 0
            for j in range(N):
                res = delta[t-1][j]*A[j][i]
                if res > temp:
                    temp = res
                    maxindex = j

            delta[t][i] = temp*B[i][O[t]] #delta
            psi[t][i] = maxindex
            print('delta', t+1, '(', i, ')=' , delta[t][i])
            print('最大值的位置: ', psi[t][i])
        print("-----")

    #end
    p = max(delta[-1])
    for i in range(N):
        if delta[-1][i] == p:
            i_T = i
            print('最后时刻T, 最大值位置是: ', i_T)

    #step3: backtrack
    print("\n=====3. 路径回溯=====")
    path = [0]*T
    i_t = i_T
    for t in reversed(range(T-1)):
        i_t = psi[t+1][i_t]
        path[t] = i_t
    path[-1] = i_T
    print("最优状态序列: ", path)

    return
```

# Source Code Of Viterbi

```
pi = [0.63, 0.17, 0.20] #初始概率矩阵
A = [[0.5, 0.375, 0.125], [0.25, 0.125, 0.625], [0.25, 0.375, 0.375]] #状态转移矩阵
B = [[0.60, 0.20, 0.15, 0.05], [0.25, 0.25, 0.25, 0.25], [0.05, 0.10, 0.35, 0.5]] #发射概率矩阵
O = [0, 2, 3] #0干燥、1稍干、2潮湿、3湿透
hmm_viterbi(A, B, pi, O)
```

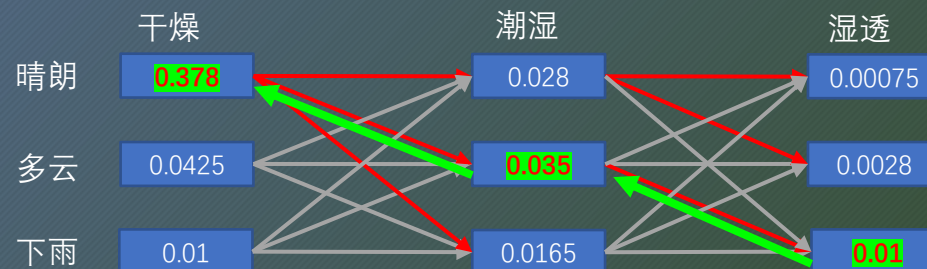
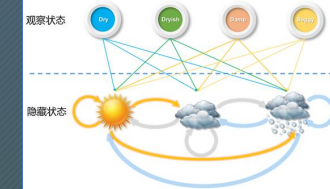
```
=====1. 初始化=====
delta 1 ( 0 ) = 0.378
delta 1 ( 1 ) = 0.0425
delta 1 ( 2 ) = 0.010000000000000002

=====2. 递推, 对 t=2, 3, ..., T=====
delta 2 ( 0 ) = 0.02835
最大值的位置: 0
delta 2 ( 1 ) = 0.0354375
最大值的位置: 0
delta 2 ( 2 ) = 0.0165375
最大值的位置: 0

delta 3 ( 0 ) = 0.0007087500000000001
最大值的位置: 0
delta 3 ( 1 ) = 0.0026578125
最大值的位置: 0
delta 3 ( 2 ) = 0.01107421875
最大值的位置: 1

最后时刻T, 最大值位置是: 2

=====3. 路径回溯=====
最优状态序列: [0, 1, 2]
```



最优状态序列为: “晴朗, 多云, 下雨”

# HMM的3个问题:

概率计算问题, 学习问题, 预测问题

已知: 一个观测序列 $O=(o_1, o_2, \dots, o_T)$

问题: 请估计隐马尔科夫模型 $\lambda=(A, B, \pi)$ 的参数, 使得在该模型下观测序列概率 $P(O|\lambda)$ 最大

注: 即用最大似然估计的方法估计参数

鲍姆-韦尔奇算法 (Baum-Welch Algorithm 1972) – 作者: Leonard E. Baum和Lloyd R. Welch  
也称: 前向-后向算法

给定观测序列 $O=\{o_1, o_2, \dots, o_T\}$ , 估计模型 $\lambda=(A, B, \pi)$ 的参数

## 4.3.3 HMM的参数估计

给定**观测序列**，如何调整**模型参数**使得该序列出现的概率最大？

鲍姆-韦尔奇算法 (Baum-Welch Algorithm)

Baum-Welch算法是EM算法的特例之一

### 最大期望算法 (Expectation-Maximization algorithm, EM)

吴军博士在《数学之美》一书中将该算法誉为“上帝的算法”

- 通过迭代进行**极大似然估计 (MLE)** 的优化算法
- 由E步 (Expectation step) 和M步 (Maximization step) 交替组成
- **算法的收敛性**可以确保迭代至少逼近局部极大值
- 可应用于**隐马尔可夫模型 (HMM) 的参数估计**
- EM算法应用极其广泛，在机器学习领域大量的应用，包括K-Means聚类算法、鲍姆-韦尔奇算法等



# EM算法基本思想

- ①初始时 ( $t=0$ ) 随机给模型参数赋值, 得到模型  $\theta_t$ 。
- ②根据  $\theta_t$  得到模型中隐变量取各个状态的期望值。
- ③用期望值代替实际次数可以得到模型参数的新估计值  $\theta_{t+1}$ 。
- ④用  $\theta_{t+1}$  代替  $\theta_t$  重复执行步骤②, 直到参数收敛。

# EM算法 例子

- ① 初始时 ( $t=0$ ) 随机给模型参数赋值, 得到模型  $\theta_t$ 。
- ② 根据  $\theta_t$  得到模型中隐变量取各个状态的期望值。
- ③ 用期望值代替实际次数可以得到模型参数的新估计值  $\theta_{t+1}$ 。
- ④ 用  $\theta_{t+1}$  代替  $\theta_t$  重复执行步骤②, 直到参数收敛。

## 有两枚硬币 $c_1$ 、 $c_2$

- 每一步选择一枚硬币投掷3次
- 选择  $c_1$  的概率为:  $\lambda$ ; 选择  $c_2$  的概率为:  $1 - \lambda$
- $c_1$  的正面概率为:  $h_1$ ,  $c_2$  的正面概率为:  $h_2$
- H代表正面 (Head), T代表背面 (Tail)
- 待估计模型参数:  $\theta = (\lambda, h_1, h_2)$

结果	硬币
HHH	$c_1$
TTT	$c_2$
HHH	$c_1$
TTT	$c_2$

$$\Theta = (\lambda, h_1, h_2)$$

- ① 初始时 (t=0) 随机给模型参数赋值, 得到模型  $\theta_t$ 。
- ② 根据  $\theta_t$  得到模型中隐变量取各个状态的期望值。
- ③ 用期望值代替实际次数可以得到模型参数的新估计值  $\theta_{t+1}$ 。
- ④ 用  $\theta_{t+1}$  代替  $\theta_t$  重复执行步骤②, 直到参数收敛。

结果	硬币
HHH	$c_1$
TTT	$c_2$
HHH	$c_1$
TTT	$c_2$

- ① 随机给模型参数赋值:  $\theta_0 = (0.3, 0.3, 0.6)$ ;  $\lambda = 0.3$ ,  $h_1 = 0.3$ ,  $h_2 = 0.6$
- ② 根据  $\theta_t$  得到模型中隐变量取各个状态的期望值。

$$\tilde{p}_1 = \frac{P(\langle HHH, c_1 \rangle)}{P(\langle HHH \rangle)} = \frac{P(c_1) \times P(\langle HHH | c_1 \rangle)}{P(\langle HHH \rangle)} = \frac{P(c_1) \times P(\langle HHH | c_1 \rangle)}{P(\langle HHH, c_1 \rangle) + P(\langle HHH, c_2 \rangle)} = \frac{\lambda \times (h_1 \times h_1 \times h_1)}{\lambda \times (h_1 \times h_1 \times h_1) + (1 - \lambda) \times (h_2 \times h_2 \times h_2)} \approx 0.0508$$

$$1 - \tilde{p}_1 = 0.9492$$

	HHH(第1轮)	TTT(第2轮)	HHH(第3轮)	TTT(第4轮)
$c_1$	0.0508	0.6967	0.0508	0.6967
$c_2$	0.9492	0.3033	0.9492	0.3033



	HHH(第1轮)	TTT(第2轮)	HHH(第3轮)	TTT(第4轮)
$c_1$	0.0508	0.6967	0.0508	0.6967
$c_2$	0.9492	0.3033	0.9492	0.3033

- ① 初始时 (t=0) 随机给模型参数赋值, 得到模型  $\theta_t$ 。
- ② 根据  $\theta_t$  得到模型中隐变量取各个状态的期望值。
- ③ 用期望值代替实际次数可以得到模型参数的新估计值  $\theta_{t+1}$ 。
- ④ 用  $\theta_{t+1}$  代替  $\theta_t$  重复执行步骤②, 直到参数收敛。

③用期望值代替实际次数可以得到模型参数的新估计值  $\theta_{t+1}$

$$\theta_1 = (\lambda', h_1', h_2')$$

$$\lambda' = \frac{0.0508 + 0.6967 + 0.0508 + 0.6967}{4} \approx 0.3738$$

$$h_1' = p(H/c_1) = \frac{0.0508 \times (3/3) + 0.6967 \times (0/3) + 0.0508 \times (3/3) + 0.6967 \times (0/3)}{0.0508 + 0.6967 + 0.0508 + 0.6967} \approx 0.0680$$

$$h_2' = p(H/c_2) = \frac{0.9492 \times (3/3) + 0.3033 \times (0/3) + 0.9492 \times (3/3) + 0.3033 \times (0/3)}{0.9492 + 0.3033 + 0.9492 + 0.3033} \approx 0.7578$$

$$\lambda = 0.5, \quad h_1 = 1, \quad h_2 = 0$$

④用 $\theta_{t+1}$ 代替 $\theta_t$ 重复执行步骤②，直到参数收敛

结果	硬币
HHH	$c_1$
TTT	$c_2$
HHH	$c_1$
TTT	$c_2$

- ① 初始时 (t=0) 随机给模型参数赋值，得到模型  $\theta_0$ 。
- ② 根据  $\theta_t$  得到模型中隐变量取各个状态的期望值。
- ③ 用期望值代替实际次数可以得到模型参数的新估计值  $\theta_{t+1}$ 。
- ④ 用  $\theta_{t+1}$  代替  $\theta_t$  重复执行步骤②，直到参数收敛。

	$\lambda$	$h_1$	$h_2$
$\theta_0$	0.3000	0.3000	0.6000
$\theta_1$	0.3738	0.0680	0.7578
$\theta_2$	0.4859	0.0004	0.9722
$\theta_3$	0.5000	0.0000	1.0000

	HHH (第1轮)	TTT (第2轮)	HHH (第3轮)	TTT (第4轮)
$c_1$	0.0508	0.6967	0.0508	0.6967
$c_2$	0.9492	0.3033	0.9492	0.3033

$c_1$	0.0004	0.9714	0.0004	0.9714
$c_2$	0.9996	0.0286	0.9996	0.0286

$c_1$	0.0000	1.0000	0.0000	1.0000
$c_2$	1.0000	0.0000	1.0000	0.0000

$c_1$	0.0000	1.0000	0.0000	1.0000
$c_2$	1.0000	0.0000	1.0000	0.0000

$\theta$  于 (0.5, 0, 1) 处收敛

选择  $c_1$  的概率为:  $\lambda = 0.5$   
 $c_1$  的正面概率为:  $h_1 = 0$   
 $c_2$  的正面概率为:  $h_2 = 1$

结果与初始值设定有关

# 更换样本

$\langle HHH \rangle, \langle TTT \rangle, \langle HHH \rangle, \langle TTT \rangle, \langle HHH \rangle$

选择 $c_1$ 的概率为:  $\lambda = 0.4$

$c_1$ 的正面概率为:  $h_1 = 0$

$c_2$ 的正面概率为:  $h_2 = 1$

$\langle HHT \rangle, \langle TTT \rangle, \langle HHH \rangle, \langle TTT \rangle$

选择 $c_1$ 的概率为:  $\lambda = 0.497$

$c_1$ 的正面概率为:  $h_1 = 0$

$c_2$ 的正面概率为:  $h_2 = 0.8248$



# 计算过程总结：估算步骤 和 最大化步骤 交替

- ① 初始时 (t=0) 随机给模型参数赋值, 得到模型  $\theta_t$ 。
- ② 根据  $\theta_t$  得到模型中隐变量取各个状态的期望值  $\tilde{p}_i$
- ③ 用期望值代替实际次数可以得到模型参数的新估计值  $\theta_{t+1}$   $\lambda'$ ,  $h_1'$ ,  $h_2'$ 。
- ④ 用  $\theta_{t+1}$  代替  $\theta_t$  重复执行步骤②, 直到参数收敛。

## 估算步骤 Expectation step

$$\tilde{p}_i = \frac{P(X_i = \langle Y_i, c_1 \rangle | \theta)}{P(X_i = \langle Y_i, c_1 \rangle | \theta) + P(X_i = \langle Y_i, c_2 \rangle | \theta)}$$

## 最大化步骤 Maximization step

$$\lambda' = \frac{\sum_{i=1}^n \tilde{p}_i}{n}$$

$$h_1' = \frac{\sum_{i=1}^n \frac{H_i}{3} \tilde{p}_i}{\sum_{i=1}^n \tilde{p}_i}$$

$$h_2' = \frac{\sum_{i=1}^n \frac{H_i}{3} (1 - \tilde{p}_i)}{\sum_{i=1}^n (1 - \tilde{p}_i)}$$

# 鲍姆-韦尔奇算法

也称：前向-后向算法

- ① 初始时 (t=0) 随机给模型参数赋值, 得到模型  $\theta_t, \lambda, h_1, h_2$
- ② 根据  $\theta_t$  得到模型中隐变量取各个状态的期望值  $\tilde{P}_i$
- ③ 用期望值代替实际次数可以得到模型参数的新估计值  $\theta_{t+1}, \lambda', h_1', h_2'$ 。
- ④ 用  $\theta_{t+1}$  代替  $\theta_t$  重复执行步骤②, 直到参数收敛。

①初始化。模型  $\mu_i$  ( $i = 0$ ) , 随机给参数  $\pi_i$ 、 $a_{ij}$ 、 $b_j(k)$  赋值, 使其满足如下约束:

$$\sum_{i=1}^N \pi_i = 1 \quad \sum_{j=1}^N a_{ij} = 1 \quad \sum_{k=1}^M b_j(k) = 1$$

②EM计算 (递推)

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

E步骤: 由模型  $\mu_i$  计算期望值:  $p(q_t = s_i | O, \mu) = \gamma_t(i) = \sum_{j=1}^N \xi_t(i, j)$

$$\gamma_t(i) = \frac{P(i_t = q_i, O; \lambda)}{P(O; \lambda)} = \frac{\alpha_t(i) \beta_t(i)}{\sum_{j=1}^N \alpha_t(j) \beta_t(j)}$$

M步骤: 由期望值重新估算  $\pi_i', a_{ij}', b_j(k)'$ , 得到  $\mu_{i+1}$

③循环计算

$\mu_{i+1}$  替代  $\mu_i$ , 重复②, 直到参数收敛

$$\pi_i = \gamma_1(i)$$
$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$
$$b_j(k) = \frac{\sum_{t=1}^T \gamma_t(j) I(o_t = v_k)}{\sum_{t=1}^T \gamma_t(j)}$$

为了方便计算, 定义  $\xi$  的变量。  
 $\xi$  两个参数: 前向变量  $\alpha$  和后向变量  $\beta$ 。  
前向变量  $\alpha$  和后向变量  $\beta$   
由前向算法和后向算法分别计算获得。

<https://blog.csdn.net/firpark/article/details/54934112>

<http://www.hankcs.com/ml/em-algorithm-and-its-generalization.html>

# EM算法的简明实现

## 双硬币模型

假设有两枚硬币A、B，以相同的概率随机选择一个硬币，进行如下的抛硬币实验：  
共做5次实验，每次实验独立的抛十次，结果如图中a所示，例如某次实验产生了H、T、T、T、H、H、T、H、T、H，H代表正面朝上。

假设试验数据记录员可能是实习生，业务不一定熟悉，造成a和b两种情况


a表示实习生记录了详细的试验数据，我们可以观测到试验数据中每次选择的是A还是B

b表示实习生忘了记录每次试验选择的是A还是B，我们无法观测实验数据中选择的硬币是哪一个

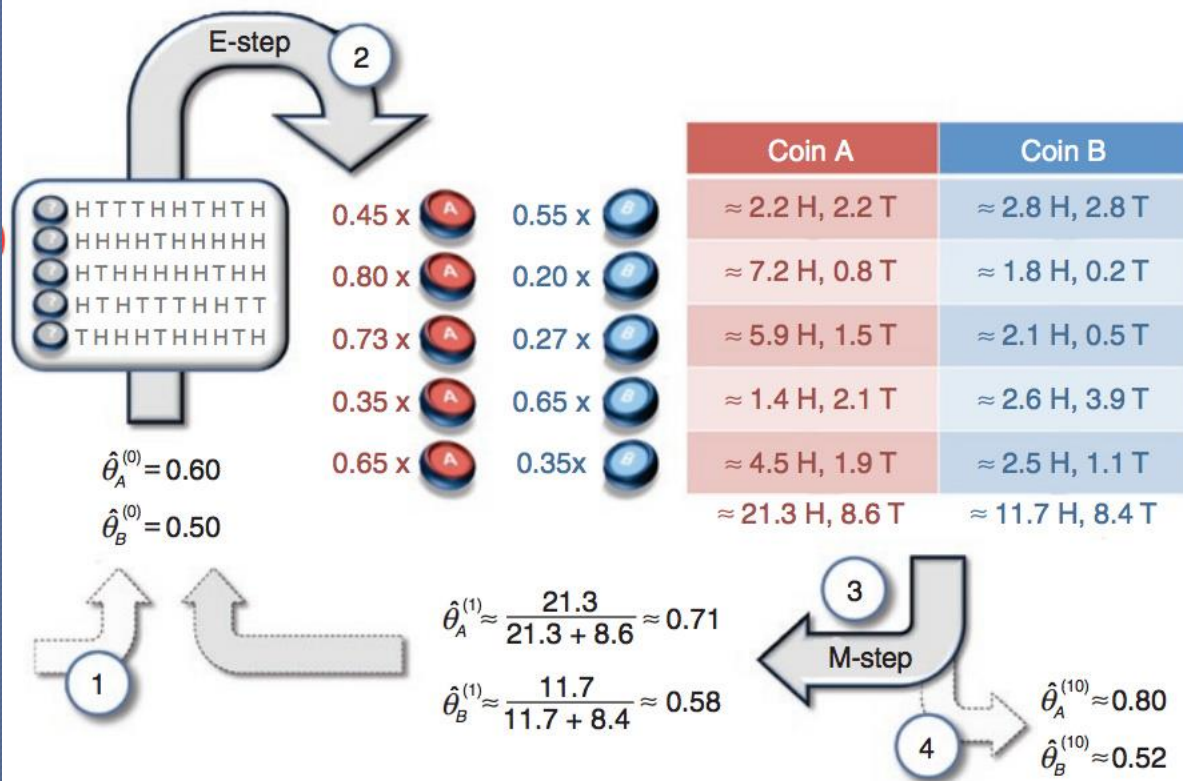
问在两种情况下分别如何估计两个硬币正面出现的概率？



## a Maximum likelihood

	Coin A	Coin B	
 H T T T H H T H T H		5 H, 5 T	
 H H H H T H H H H H	9 H, 1 T		$\hat{\theta}_A = \frac{24}{24 + 6} = 0.80$
 H T H H H H H T H H	8 H, 2 T		$\hat{\theta}_B = \frac{9}{9 + 11} = 0.45$
 H T H T T T H H T T		4 H, 6 T	
 T H H H T H H H T H	7 H, 3 T		
5 sets, 10 tosses per set	24 H, 6 T	9 H, 11 T	

## b Expectation maximization



a情况相信大家都很熟悉，既然能观测到试验数据是哪枚硬币产生的，就可以统计正反面的出现次数，直接利用最大似然估计即可。

b情况就无法直接进行最大似然估计了，只能用EM算法，接下来用简明EM算法Python实现。

<http://www.hankcs.com/ml/em-algorithm-and-its-generalization.html>

# 鲍姆-韦尔奇算法 实现

#前向-后向算法(Baum-Welch算法):由 EM算法 & HMM 结合形成

def baum\_welch(self, O, e=0.05):

row=self.A.shape[0]  
col=len(O)

done=False

while not done:

zeta=numpy.zeros((row,row,col-1))

alpha=self.forward(O)

beta=self.backward(O)

#EM算法: 由 E-步骤 和 M-步骤 组成

#E-步骤: 计算期望值zeta和gamma

for t in range(col-1):

#分母部分

denominator=numpy.dot(numpy.dot(alpha[:,t],self.A)\*self.B[:,O[t+1]],beta[:,t+1])

for i in range(row):

#分子部分以及zeta的值

numerator=alpha[i,t]\*self.A[i,:]\*self.B[:,O[t+1]]\*beta[:,t+1]

zeta[i,:,t]=numerator/denominator

gamma=numpy.sum(zeta,axis=1)

final\_numerator=(alpha[:,col-1]\*beta[:,col-1]).reshape(-1,1)

final=final\_numerator/numpy.sum(final\_numerator)

gamma=numpy.hstack((gamma,final))

#M-步骤: 重新估计参数Pi,A,B

newPi=gamma[:,0]

newA=numpy.sum(zeta,axis=2)/numpy.sum(gamma[:, :-1],axis=1)

newB=numpy.copy(self.B)

b\_denominator=numpy.sum(gamma,axis=1)

temp\_matrix=numpy.zeros((1,len(O)))

for k in range(self.B.shape[1]):

for t in range(len(O)):

if O[t]==k:

temp\_matrix[0][t]=1

newB[:,k]=numpy.sum(gamma\*temp\_matrix,axis=1)/b\_denominator

#终止阈值

if numpy.max(abs(self.Pi-newPi))<e and numpy.max(abs(self.A-newA))<e and numpy.max(abs(self.B-newB))<e:

done=True

self.A=newA

self.B=newB

self.Pi=newPi

return

if \_\_name\_\_ == "\_\_main\_\_":

#初始化, 随机的给参数A, B, Pi赋值

status=["晴天","多云","下雨"]

observations=["干燥","稍干","潮湿","湿透"]

A={"晴天":{"晴天":0.5,"多云":0.3,"下雨":0.2},  
"多云":{"晴天":0.5,"多云":0.2,"下雨":0.3},  
"下雨":{"晴天":0.5,"多云":0.2,"下雨":0.3}}

B={"晴天":{"干燥":0.4,"稍干":0.1,"潮湿":0.3,"湿透":0.2},  
"多云":{"干燥":0.1,"稍干":0.5,"潮湿":0.2,"湿透":0.2},  
"下雨":{"干燥":0.1,"稍干":0.5,"潮湿":0.2,"湿透":0.2}}

Pi=[0.5,0.1,0.4]

O=[1,2,0,2,3,0]

A=matrix(A,status,status)

B=matrix(B,status,observations)

hmm=HMM(A,B,Pi)

hmm.baum\_welch(O)

print('====状态转移矩阵 A =====')

print(hmm.A)

print('\n=====发射概率矩阵 B =====')

print(hmm.B)

print('\n=====初始概率矩阵 Pi =====')

print(hmm.Pi)

====状态转移矩阵 A =====

[[0.06436744 0.58109045 0.1262727 ]

[0.00164891 0.1972305 0.36716668]

[0.03917907 0.42395656 0.78928973]]

====发射概率矩阵 B =====

[[7.97673617e-05 8.94890208e-01 9.80080354e-01 1.00000000e+00]

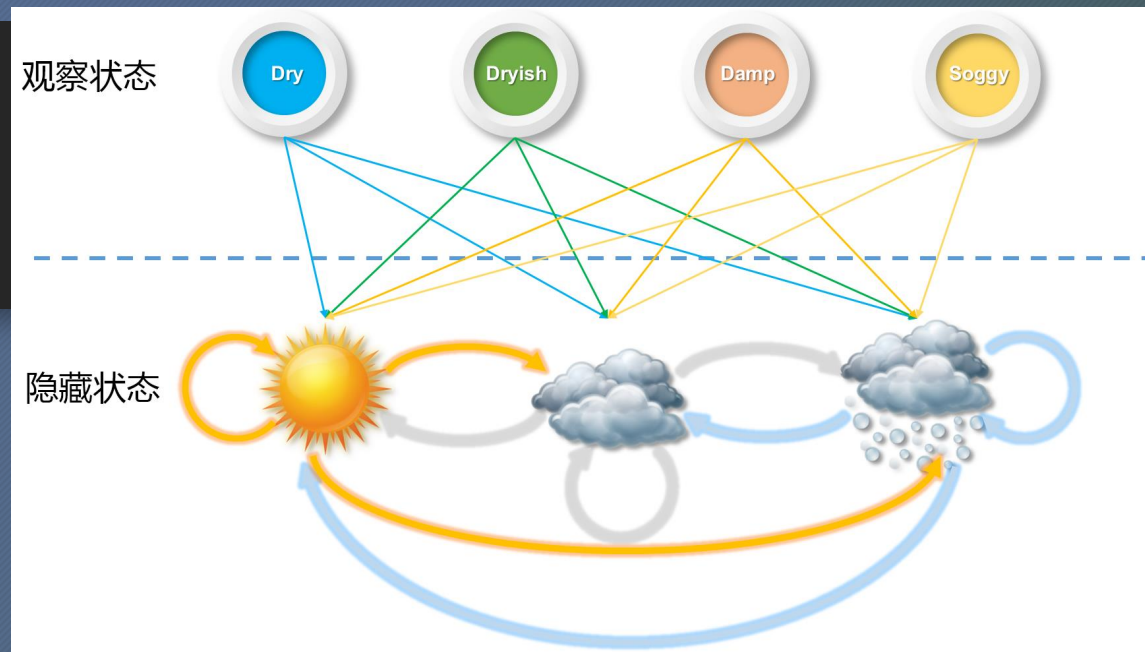
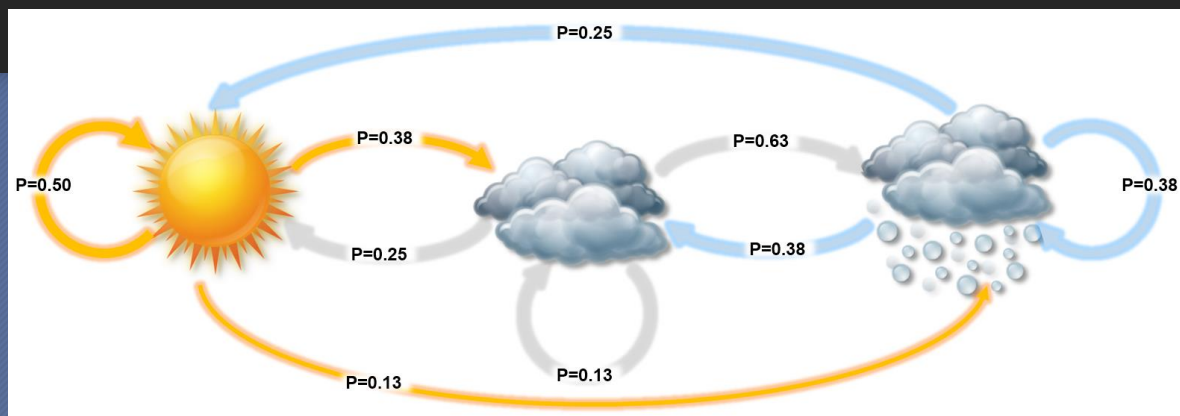
[2.84528326e-01 2.84627013e-01 8.58619759e-01 1.00000000e+00]

[4.63297590e-01 4.63344821e-01 7.74473877e-01 1.00000000e+00]]

====初始概率矩阵 Pi =====

[9.99693891e-01 1.44784079e-04 1.61324889e-04]

# HMM总结



1. 根据HMM得到一个可观察状态序列的概率( 评价 )

前向算法(Forward Algorithm)、后向算法(Backward Algorithm)

2. 找到一个隐藏状态的序列使这个序列产生一个可观察状态序列的概率最大( 解码 )

维特比算法(Viterbi Algorithm)

3. 根据一个可以观察到的状态序列集, 产生一个HMM ( 学习 )

最大期望算法 (Expectation-Maximization algorithm, EM) 、鲍姆-韦尔奇算法(Baum-Welch Algorithm)





THE END