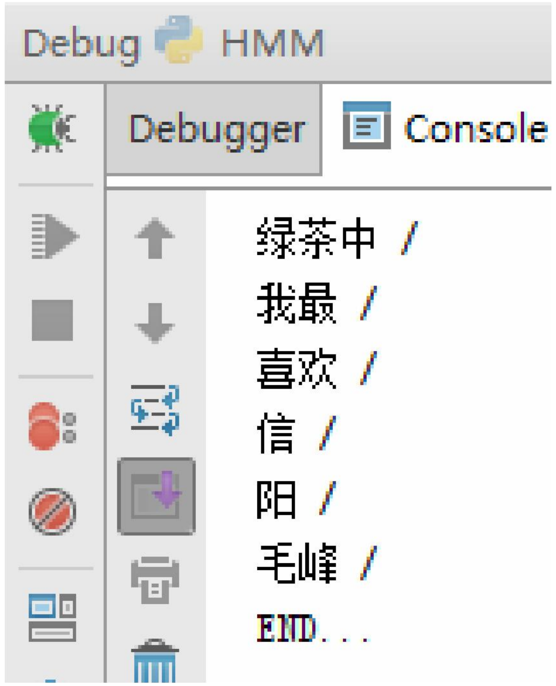
隐马尔科夫模型

七月算法 邹博

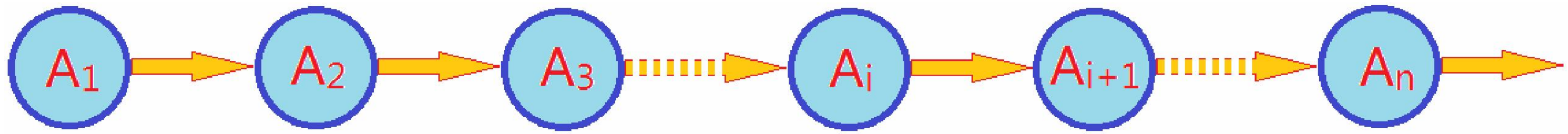
2015年12月12日

HMM用于中文分词

- 全国销量领先的红罐凉茶改成广州恒大
- 绿茶中我最喜欢信阳毛峰



特殊的贝叶斯网络



- M个离散结点形成一条链；
  - 若每一个结点有K个状态，则需要K-1+(M-1)K(K-1)个参数——O(M)；
  - 如果是全连接，需要K^M-1个参数——O(K^M)；
- 这个网络被称作**马尔科夫模型**：
  - 若状态A_i确定，则A_{i+1}只与A_i有关，与$A_1,\ldots,A_{i-1}$无关。

$$p(\theta^{(t+1)}|\theta^{(1)},\theta^{(2)},\ldots,\theta^{(t)})=p(\theta^{(t+1)}|\theta^{(t)})$$

马尔科夫随机过程

- 这个模型其实早已使用，并不陌生。
- 考察C语言提供的rand随机数函数。

伪随机

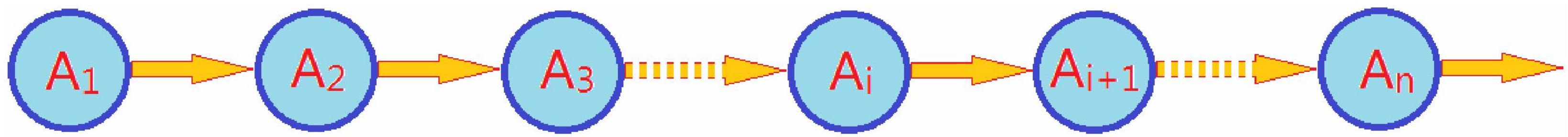
```c
/***
*int rand() - returns a random number
*
*Purpose:
*       returns a pseudo-random number 0 through 32767.
*
*Entry:
*       None.
*
*Exit:
*       Returns a pseudo-random number 0 through 32767.
*
*Exceptions:
*
*******************************************************************************/

int __cdecl rand (
        void
        )
{
#ifdef _MT

        _ptiddata ptd = _getptd();

        return( ((ptd->_holdrand = ptd->_holdrand * 214013L
            + 2531011L) >> 16) & 0x7fff );

#else  /* _MT */
        return(((holdrand = holdrand * 214013L + 2531011L) >> 16) & 0x7fff);
#endif  /* _MT */
}
```

辅助结构

```c
#ifndef _MT
static long holdrand = 1L;
#endif  /* _MT */

/***
*void srand(seed) - seed the random number generator
*
*Purpose:
*       Seeds the random number generator with the int given.  Adapted from the
*       BASIC random number generator.
*
*Entry:
*       unsigned seed - seed to seed rand # generator with
*
*Exit:
*       None.
*
*Exceptions:
*
*******************************************************************************/

void __cdecl srand (
        unsigned int seed
        )
{
#ifdef _MT

        _getptd()->_holdrand = (unsigned long)seed;

#else  /* _MT */
        holdrand = (long)seed;
#endif  /* _MT */
}
```

伪随机 $p(\theta^{(t+1)}|\theta^{(1)},\theta^{(2)},\ldots,\theta^{(t)}) = p(\theta^{(t+1)}|\theta^{(t)})$

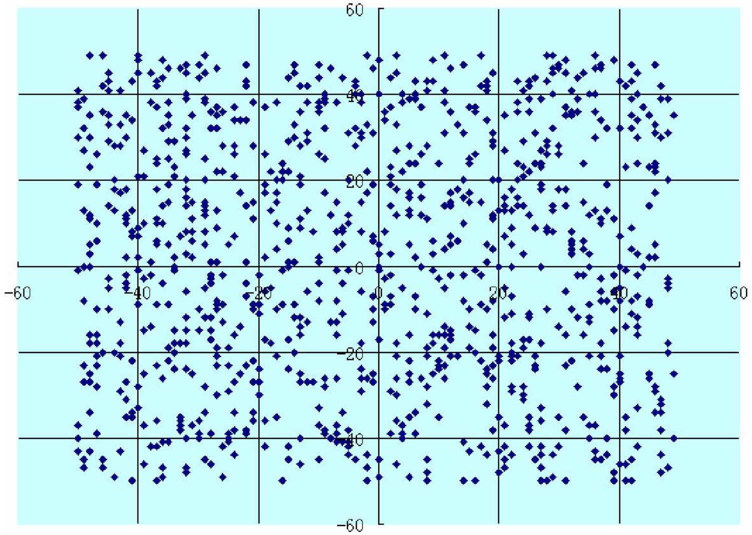
- 事先给定一个值x作为种子seed——若不指定，默认为1；
  - 先验的值x往往称为种子。
- 根据种子seed计算一个r，并且将r作为新的种子seed，返回r。
- 种子→x_1→x_2→x_3→x_4……
  - 马尔科夫随机过程



伪随机的效果：产生二维随机数

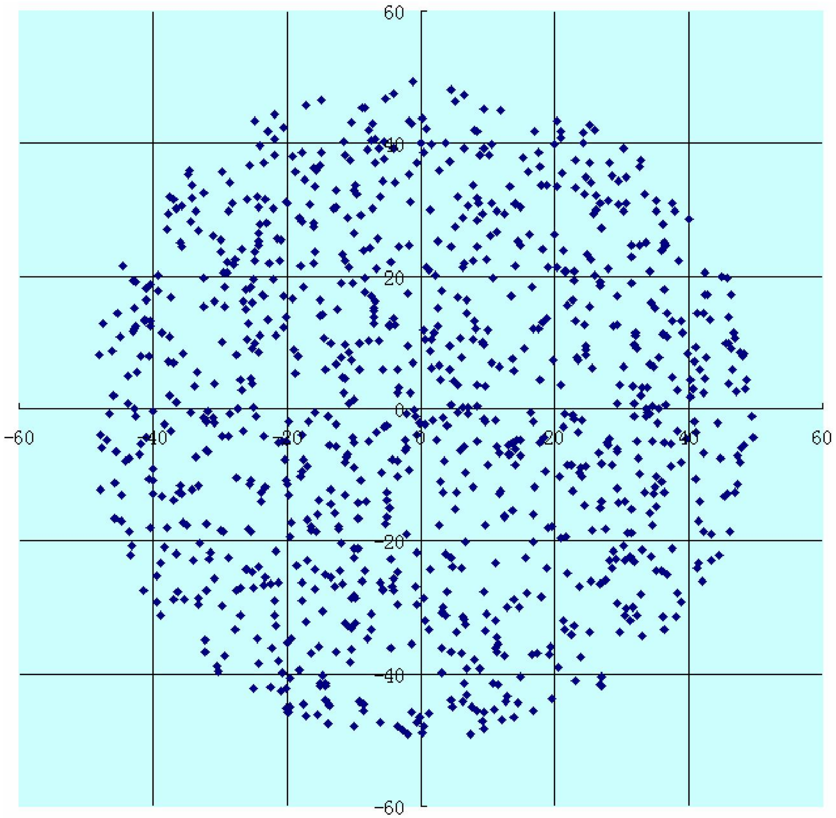
- 给定区间$[a_x,b_x]\times[a_y,b_y]$，使得二维随机点(x,y)落在等概率落在区间的某个点上。
- 因为两个维度是独立的，分别生成两个随机数即可。

```
int rand50()
{
    return rand() % 100 - 50;
}

int _tmain(int argc, _TCHAR* argv[])
{
    ofstream oFile;
    oFile.open(_T("D:\\rand.txt"));
    int x, y;
    for(int i = 0; i < 1000; i++)
    {
        x = rand50();
        y = rand50();
        oFile << x << '\t' << y << '\n';
    }
    oFile.close();
    return 0;
}
```

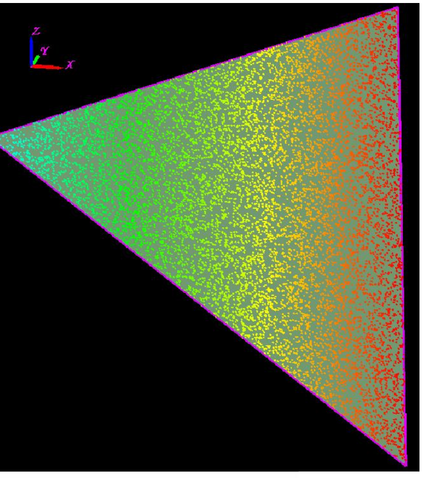


伪随机效果：产生圆内随机数

```
double rand2500()
{
    return rand() % 2500;
}

int _tmain(int argc, _TCHAR* argv[])
{
    ofstream oFile;
    oFile.open(_T("D:\\rand.txt"));
    double r,theta;
    double x,y;
    for(int i = 0; i < 1000; i++)
    {
        r = sqrt(rand2500());
        theta = rand();
        x = r*cos(theta);
        y = r*sin(theta);
        oFile << x << '\t' << y << '\n';
    }
    oFile.close();
    return 0;
}
```



伪随机应用：三角形内随机数




```cpp
void CRandomTriangle::CalcRotate()
{
    float f12 = CDelPoint::Distance2(m_point1, m_point2);
    float f23 = CDelPoint::Distance2(m_point2, m_point3);
    float f31 = CDelPoint::Distance2(m_point3, m_point1);
    if(f12 > f23)
    {
        if(f12 > f31)   //12最大
        {
            m_ptBase = m_point1;
            m_ptExtend = m_point2;
            m_ptExtend -= m_ptBase;
            m_ptHeight = m_point3;
            m_ptHeight -= m_ptBase;
        }
        else    //13最大
        {
            m_ptBase = m_point1;
            m_ptExtend = m_point3;
            m_ptExtend -= m_ptBase;
            m_ptHeight = m_point2;
            m_ptHeight -= m_ptBase;
        }
    }
    else
    {
        if(f23 > f31)   //23最大
        {
            m_ptBase = m_point2;
            m_ptExtend = m_point3;
            m_ptExtend -= m_ptBase;
            m_ptHeight = m_point1;
            m_ptHeight -= m_ptBase;
        }
        else    //13最大
        {
            m_ptBase = m_point1;
            m_ptExtend = m_point3;
            m_ptExtend -= m_ptBase;
            m_ptHeight = m_point2;
            m_ptHeight -= m_ptBase;
        }
    }
    CDelPoint pt(0,0,0);
    m_tsBig.SetPoint(pt, m_ptExtend, m_ptHeight);   //原三角形
    m_ptLeft0 = m_ptHeight;
    m_ptLeft0 /= 2;
    CDelPoint::CenterPoint(m_ptExtend, m_ptHeight, m_ptRight0);

    CDelPoint ptHeighto;
    CDelPoint::Orthogonal(m_ptExtend, m_ptHeight, ptHeighto);   //求向量m_ptHeight垂直m_ptExtend的分量

    m_tsLeft.SetPoint(pt, m_ptHeight, ptHeighto);   //左侧外面那个三角形

    pt = ptHeighto;
    pt += m_ptExtend;
    m_tsRight.SetPoint(m_ptHeight, m_ptExtend, pt); //右侧外面那个三角形

    m_ptHeight = ptHeighto; //垂直分量
}
```

```cpp
void CRandomTriangle::Random2(int nSize)
{
    CalcRotate();
    m_nSize = nSize;
    if(m_pRandomPoint)
        delete[] m_pRandomPoint;
    m_pRandomPoint = new CDelPoint[nSize];
    CDelPoint pt;
    for(int i = 0; i < nSize; i++)
    {
        pt.RandomInRectangle(m_ptExtend, m_ptHeight);
        if(m_tsBig.IsIn(pt))
        {
            pt += m_ptBase;
            m_pRandomPoint[i] = pt;
        }
        else if(m_tsLeft.IsIn(pt))
        {
            CDelPoint::MirrorPoint(pt, m_ptLeft0);
            pt += m_ptBase;
            m_pRandomPoint[i] = pt;
        }
        else if(m_tsRight.IsIn(pt))
        {
            CDelPoint::MirrorPoint(pt, m_ptRight0);
            pt += m_ptBase;
            m_pRandomPoint[i] = pt;
        }
    }
    CDelPoint::Save(m_pRandomPoint, m_nSize, _T("D:\\random.pt"), 0);
}
```

思考和发现

- 可以发现，满足马尔科夫模型的样本点 $s_1, s_2 .. s_n$，与"独立同分布"只弱一点：样本间只有相邻元素有相关关系，并且，有些以"独立同分布"为条件的定理，是可以放松到马尔科夫过程的。
- 如：大数定理。

马尔科夫过程版本的大数定理

Let $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(M)}$ be M values from a Markov chain that is *aperiodic*, *irreducible*, and *positive recurrent* (then the chain is ergodic), and $E[g(\boldsymbol{\theta})] < \infty$.

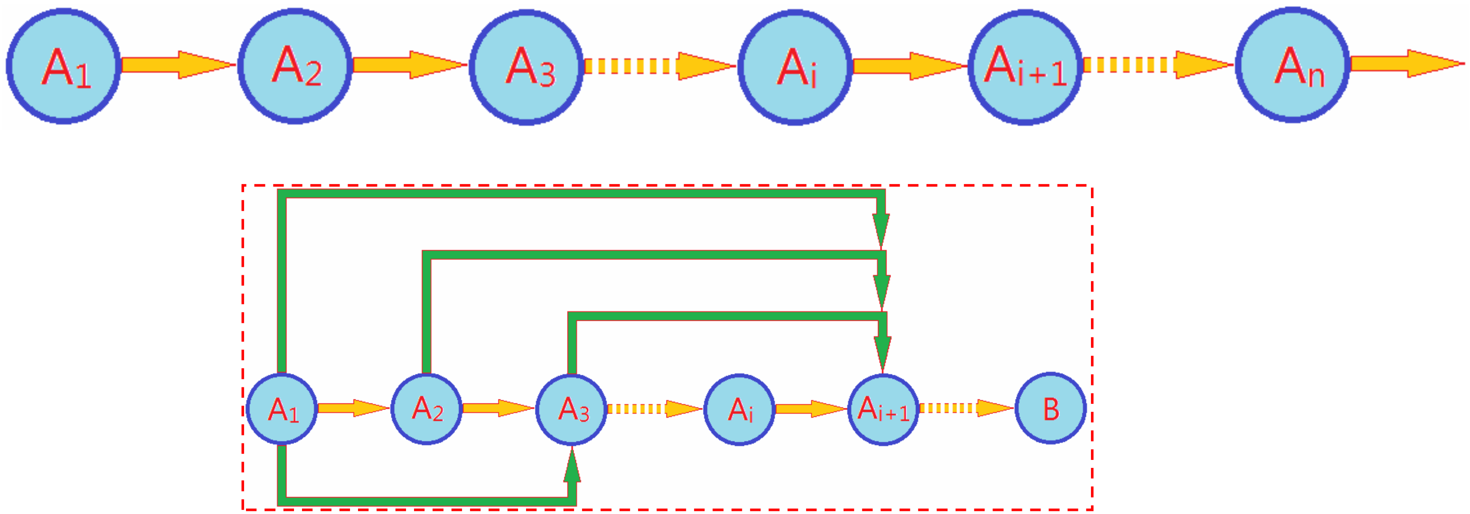
Then with probability 1,

$$\frac{1}{M}\sum_{i=1}^{M} g(\boldsymbol{\theta}_i) \to \int_{\Theta} g(\boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta}$$

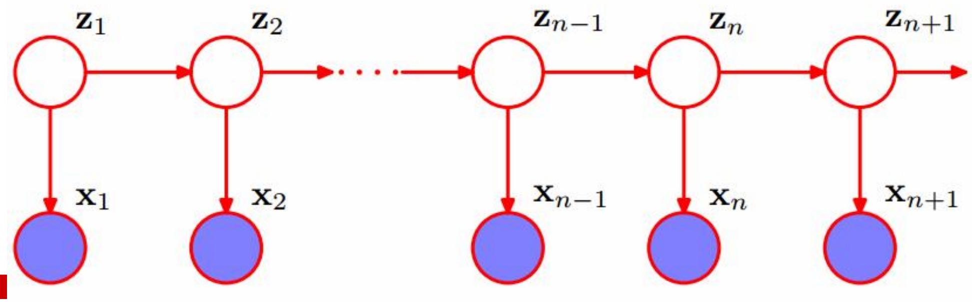
as $M \to \infty$, where π is the stationary distribution.

关于马尔科夫模型的思考

- 转移概率矩阵以及稳定概率分布
- MCMC随机模拟抽样方法
- 高阶Markov模型

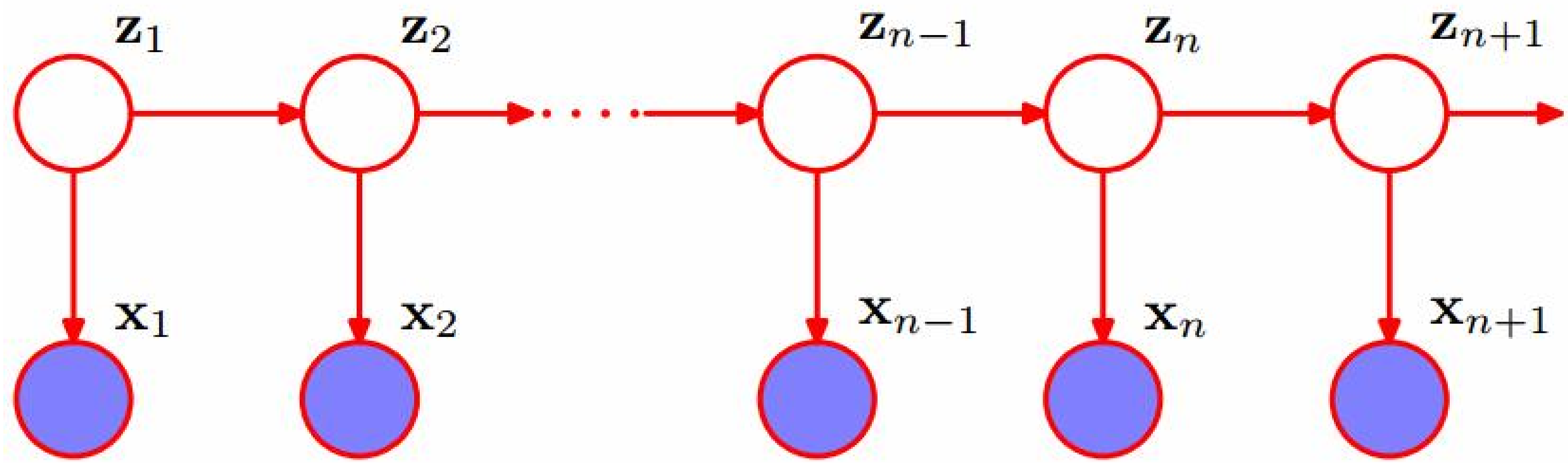


HMM定义



- 隐马尔科夫模型(HMM, Hidden Markov Model)可用标注问题，在语音识别、NLP、生物信息、模式识别等领域被实践证明是有效的算法。
- HMM是关于时序的概率模型，描述由一个隐藏的马尔科夫链随机生成不可观测的状态随机序列，再由各个状态生成一个观测而产生观测随机序列的过程。
- 隐马尔科夫模型随机生成的状态的序列，称为状态序列；每个状态生成一个观测，由此产生的观测随机序列，称为观测序列。
  - 序列的每个位置可看做是一个时刻。

隐马尔科夫模型的贝叶斯网络



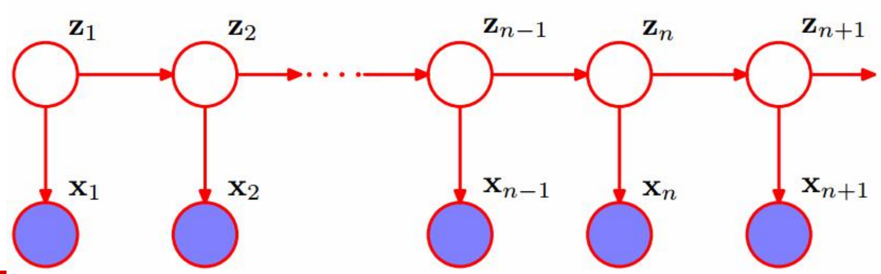
- 请思考：
  - 在z1、z2不可观察的前提下，x1和z2独立吗？x1和x2独立吗？

HMM的确定

- HMM由初始概率分布π、状态转移概率分布A以及观测概率分布B确定。

$$\lambda = (A, B, \pi)$$

HMM的参数



- Q是所有可能的状态的集合
  - N是可能的状态数
- V是所有可能的观测的集合
  - M是可能的观测数

$$Q = \{q_1, q_2, \ldots q_N\}$$

$$V = \{v_1, v_2, \ldots v_M\}$$

HMM的参数 $\lambda = (A, B, \pi)$

- I是长度为T的状态序列，O是对应的观测序列

$$I = \{i_1, i_2, \ldots i_T\} \qquad O = \{o_1, o_2, \ldots o_T\}$$

- A是状态转移概率矩阵

$$A = \left[a_{ij}\right]_{N \times N}$$

- 其中 $a_{ij} = P\left(i_{t+1} = q_j \middle| i_t = q_i\right)$
- a_{ij}是在时刻t处于状态q_i的条件下时刻t+1转移到状态q_j的概率。

HMM的参数 $\lambda=(A,B,\pi)$

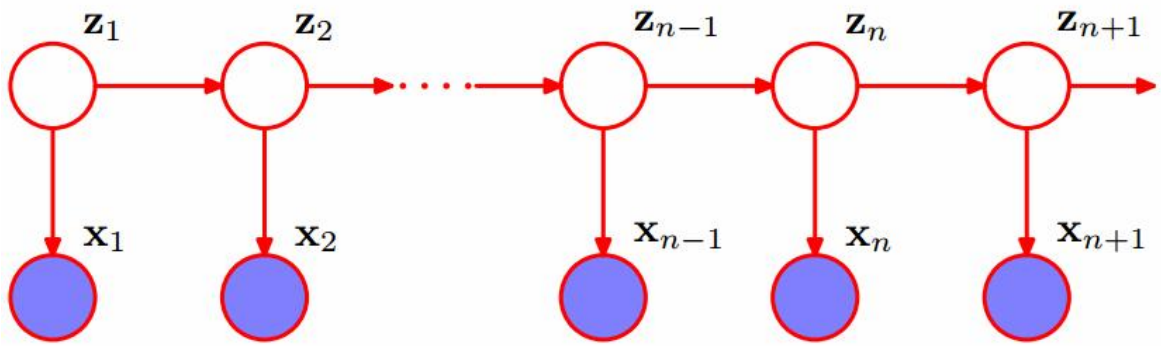
- B是观测概率矩阵 $B=\left[b_{ik}\right]_{N\times M}$
- 其中，$b_{ik}=P\left(o_t=v_k \middle| i_t=q_i\right)$
  - b_{ik}是在时刻t处于状态q_i的条件下生成观测v_k的概率。
- π是初始状态概率向量： $\pi=\left(\pi_i\right)$
- 其中，$\pi_i=P\left(i_1=q_i\right)$
  - π_i是时刻t=1处于状态qi的概率。

HMM的参数总结

- HMM由初始概率分布π(向量)、状态转移概率分布A(矩阵)以及观测概率分布B(矩阵)确定。π和A决定状态序列，B决定观测序列。因此，HMM可以用三元符号表示，称为HMM的三要素：

$$\lambda = (A, B, \pi)$$

HMM的两个基本性质



- 齐次假设：

$$P\left(i_t \mid i_{t-1}, o_{t-1}, i_{t-2}, o_{t-2} \cdots i_1, o_1\right) = P\left(i_t \mid i_{t-1}\right)$$

- 观测独立性假设：

$$P\left(o_t \mid i_T, o_T, i_{T-1}, o_{T-1} \cdots i_1, o_1\right) = P\left(o_t \mid i_t\right)$$

HMM举例

- 假设有3个盒子，编号为1、2、3，每个盒子都装有红白两种颜色的小球，数目如下：

盒子号	1	2	3
红球数	5	4	7
白球数	5	6	3

- 按照下面的方法抽取小球，得到球颜色的观测序列：
  - 按照 $\pi=(0.2,0.4,0.4)$ 的概率选择1个盒子，从盒子随机抽出1个球，记录颜色后放回盒子；
  - 按照某条件概率(下页)选择新的盒子，重复上述过程；
  - 最终得到观测序列："红红白白红"。

该示例的各个参数

- 状态集合：Q={盒子1，盒子2，盒子3}
- 观测集合：V={红，白}
- 状态序列和观测序列的长度T=5
- 初始概率分布 π ：
- 状态转移概率分布A：
- 观测概率分布B：

$$\pi = \begin{pmatrix} 0.2 \\ 0.4 \\ 0.4 \end{pmatrix} \qquad A = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.3 & 0.5 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{bmatrix} \qquad B = \begin{bmatrix} 0.5 & 0.5 \\ 0.4 & 0.6 \\ 0.7 & 0.3 \end{bmatrix}$$

思考：

- 在给定参数π、A、B的前提下，得到观测序列“红红白白红”的概率是多少？

HMM的3个基本问题

- 概率计算问题：前向-后向算法——动态规划
  - 给定模型 $\lambda=(A,B,\pi)$ 和观测序列 $O=\{o_1,o_2,\ldots o_T\}$，计算模型λ下观测序列O出现的概率P(O| λ)
- 学习问题：Baum-Welch算法(状态未知)——EM
  - 已知观测序列 $O=\{o_1,o_2,\ldots o_T\}$，估计模型 $\lambda=(A,B,\pi)$ 的参数，使得在该模型下观测序列P(O| λ)最大
- 预测问题：Viterbi算法——动态规划
  - 解码问题：已知模型 $\lambda=(A,B,\pi)$ 和观测序列 $O=\{o_1,o_2,\ldots o_T\}$ 求给定观测序列条件概率P(I|O, λ)最大的状态序列I

概率计算问题

- 直接算法
  - 暴力算法
- 前向算法
- 后向算法
  - 这二者是理解HMM的算法重点

直接计算法

- 按照概率公式，列举所有可能的长度为T的状态序列 $I=\{i_1,i_2,\ldots i_T\}$，求各个状态序列I与观测序列 $O=\{o_1,o_2,\ldots o_T\}$ 的联合概率P(O,I|λ)，然后对所有可能的状态序列求和，从而得到P(O|λ)

直接计算法

- 状态序列 $I=\{i_1,i_2,\ldots i_T\}$ 的概率是：

$$P(I|\lambda)=\pi_{i_1}a_{i_1i_2}a_{i_2i_3}\cdots a_{i_{T-1}i_T}$$

- 对固定的状态序列I，观测序列O的概率是：

$$P(O|I,\lambda)=b_{i_1o_1}b_{i_2o_2}\cdots b_{i_To_T}$$

$$P(I|\lambda) = \pi_{i_1} a_{i_1 i_2} a_{i_2 i_3} \cdots a_{i_{T-1} i_T}$$

直接计算法

$$P(O|I,\lambda) = b_{i_1 o_1} b_{i_2 o_2} \cdots b_{i_T o_T}$$

- O和I同时出现的联合概率是：

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

$$= \pi_{i_1} b_{i_1 o_1} a_{i_1 i_2} b_{i_2 o_2} \cdots a_{i_{T-1} i_T} b_{i_T o_T}$$

- 对所有可能的状态序列I求和，得到观测序列O的概率P(O| λ)

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

$$= \sum_{i_1, i_2, \cdots i_T} \pi_{i_1} b_{i_1 o_1} a_{i_1 i_2} b_{i_2 o_2} \cdots a_{i_{T-1} i_T} b_{i_T o_T}$$

直接计算法分析

- 对于最终式

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

$$=\sum_{i_1,i_2,\cdots i_T}\pi_{i_1}b_{i_1o_1}a_{i_1i_2}b_{i_2o_2}\cdots a_{i_{T-1}i_T}b_{i_To_T}$$

- 分析：加和符号中有2T个因子，I的遍历个数为N^T，因此，时间复杂度为$O(T\,N^T)$，复杂度过高。

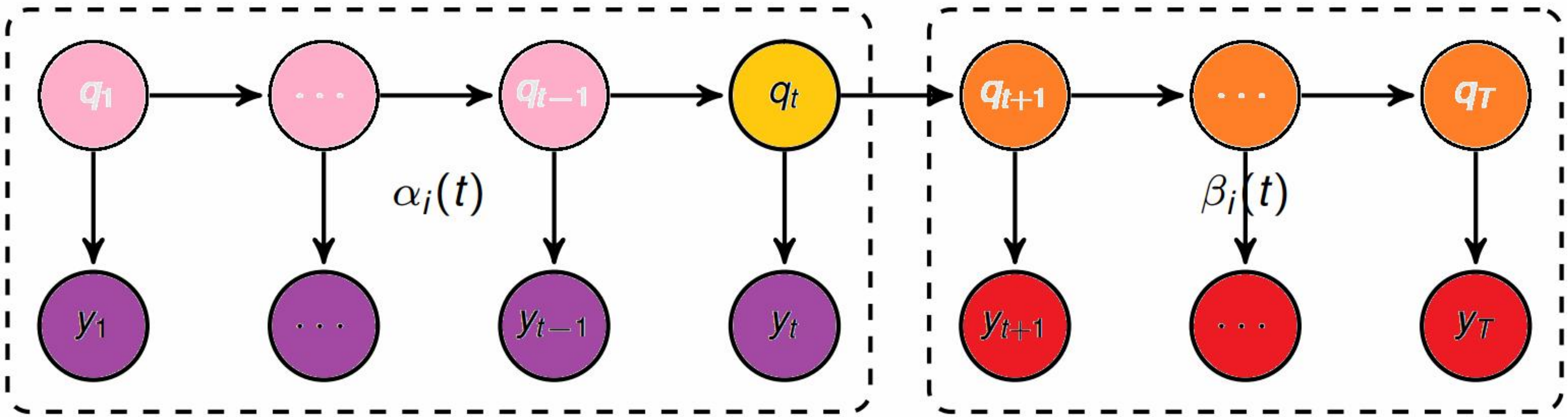
借鉴算法的优化思想

- 最长递增子序列
  - 给定一个长度为N的数组，求该数组的一个最长的单调递增的子序列(不要求连续)。
  - 数组：5, 6, 7, 1, 2, 8 的LIS：5, 6, 7, 8
- 最大连续子数组
  - 给定一个长度为N的数组，求该数组中连续的一段数组(子数组)，使得该子数组的和最大。
    - 数组： 1, -2, 3, 10, -4, 7, 2, -5，
    - 最大子数组：3, 10, -4, 7, 2
- KMP中next数组的计算

模式串	a	b	a	a	b	c	a	b	a
next	-1	0	0	1	1	2	0	1	2

定义：前向-后向

$$\alpha_i(t) = p(y_1, y_2, \dots y_t, q_t = i|\lambda) \qquad \beta_i(t) = p(y_{t+1}, \dots, y_T|q_t = i, \lambda)$$



前向算法

- 定义：给定λ，定义到时刻t部分观测序列为o1,o2…ot且状态为qi的概率称为前向概率，记做：

$$\alpha_t(i) = P(o_1, o_2, \cdots o_t, i_t = q_i | \lambda)$$

  - 可以递推计算前向概率 $\alpha_t(i)$ 及观测序列概率 $P(O|\lambda)$

前向算法 $\alpha_t(i) = P(o_1, o_2, \cdots o_t, i_t = q_i | \lambda)$

- 初值： $\alpha_1(i) = \pi_i b_{io_1}$

- 递推： 对于t=1,2…T-1

$$\alpha_{t+1}(i) = \left(\sum_{j=1}^{N} \alpha_t(j) a_{ji} \right) b_{io_{t+1}}$$

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

前向算法 $\alpha_t(i) = P(o_1, o_2, \cdots o_t, i_t = q_i | \lambda)$

- 思考：前向概率算法的时间复杂度是O(TN^2)

- 重点考察第二步：
  - 递推：对于t=1,2…T-1

$$\alpha_{t+1}(i) = \left(\sum_{j=1}^{N} \alpha_t(j) a_{ji} \right) b_{io_{t+1}}$$

例：盒子球模型

- 考察盒子球模型，计算观测向量O="红白红"的出现概率。

$$\pi = \begin{pmatrix} 0.2 \\ 0.4 \\ 0.4 \end{pmatrix} \qquad A = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.3 & 0.5 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{bmatrix} \qquad B = \begin{bmatrix} 0.5 & 0.5 \\ 0.4 & 0.6 \\ 0.7 & 0.3 \end{bmatrix}$$

解：盒子球模型

$$\pi = \begin{pmatrix} 0.2 \\ 0.4 \\ 0.4 \end{pmatrix} \quad A = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.3 & 0.5 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{bmatrix} \quad B = \begin{bmatrix} 0.5 & 0.5 \\ 0.4 & 0.6 \\ 0.7 & 0.3 \end{bmatrix}$$

- 计算初值

$$\alpha_1(1) = \pi_1 b_{1o_1} = 0.2 \times 0.5 = 0.1$$

$$\alpha_1(2) = \pi_2 b_{2o_1} = 0.4 \times 0.4 = 0.16$$

$$\alpha_1(3) = \pi_3 b_{3o_1} = 0.4 \times 0.7 = 0.28$$

解

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

$$\alpha_1(1)=\pi_1 b_{1o_1}=0.2\times 0.5=0.1$$
$$\alpha_1(2)=\pi_2 b_{2o_1}=0.4\times 0.4=0.16$$
$$\alpha_1(3)=\pi_3 b_{3o_1}=0.4\times 0.7=0.28$$

递推

$$\alpha_2(1)=\left(\sum_{j=1}^{N}\alpha_1(j)a_{j1}\right)b_{1o_2}$$
$$=(0.1\times 0.5+0.16\times 0.3+0.28\times 0.2)\times 0.5$$
$$=0.077$$

$$\alpha_2(2)=0.1104$$

$$\alpha_2(3)=0.0606$$

$$\alpha_3(1)=0.04187$$
$$\alpha_3(2)=0.03551$$
$$\alpha_3(3)=0.05284$$

解：盒子球模型

$$\alpha_3(1) = 0.04187$$
$$\alpha_3(2) = 0.03551$$
$$\alpha_3(3) = 0.05284$$

- 最终

$$P(O|\lambda) = \sum_{i=1}^{3} \alpha_3(i)$$
$$= 0.04187 + 0.03551 + 0.05284$$
$$= 0.13022$$

后向算法

- 定义：给定λ，定义到时刻t状态为qi的前提下，从t+1到T的部分观测序列为$o_{t+1},o_{t+2}\cdots o_T$的概率为后向概率，记做：

$$\beta_t(i)=P\left(o_{t+1},o_{t+2},\cdots o_T \middle| i_t=q_i,\lambda\right)$$

- 可以递推计算后向概率$\beta_t(i)$及观测序列概率$P(O|\lambda)$

后向算法 $\beta_t(i)=P(o_{t+1},o_{t+2},\cdots o_T|i_t=q_i,\lambda)$

- 初值： $\beta_T(i)=1$
- 递推：对于t=T-1,T-2…,1

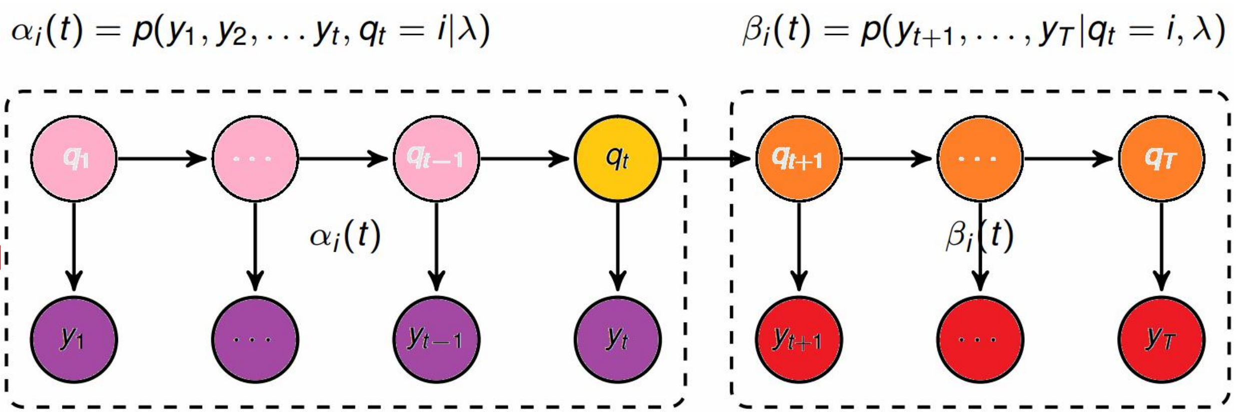
$$\beta_t(i)=\left(\sum_{j=1}^{N}a_{ij}b_{jo_{t+1}}\beta_{t+1}(j)\right)$$

- 最终： $P(O|\lambda)=\sum_{i=1}^{N}\pi_i b_{io_1}\beta_1(i)$

后向算法的说明

- 为了计算在时刻t状态为qi条件下时刻t+1之后的观测序列为$o_{t+1},o_{t+2}\ldots o_T$的后向概率$\beta_t(i)$，只需要考虑在时刻t+1所有可能的N个状态qj的转移概率(a_{ij}项)，以及在此状态下的观测o_{t+1}的观测概率(b_{jot+1}项)，然后考虑状态qj之后的观测序列的后向概率$\beta_{t+1}(j)$

前后向关系



- 根据定义：

$$
\begin{aligned}
& P(i_t = q_i, O|\lambda) \\
&= P(O|i_t = q_i, \lambda)P(i_t = q_i|\lambda) \\
&= P(o_1, \cdots o_t, o_{t+1}, \cdots o_T|i_t = q_i, \lambda)P(i_t = q_i|\lambda) \\
&= P(o_1, \cdots o_t|i_t = q_i, \lambda)P(o_{t+1}, \cdots o_T|i_t = q_i, \lambda)P(i_t = q_i|\lambda) \\
&= P(o_1, \cdots o_t, i_t = q_i|\lambda)P(o_{t+1}, \cdots o_T|i_t = q_i, \lambda) \\
&= \alpha_t(i)\beta_t(i)
\end{aligned}
$$

单个状态的概率 $P(i_t = q_i, O|\lambda) = \alpha_t(i)\beta_t(i)$

- 求给定模型λ和观测O，在时刻t处于状态qi的概率。
- 记： $\gamma_t(i) = P(i_t = q_i | O, \lambda)$

单个状态的概率

- 根据前向后向概率的定义，

$$P(i_t=q_i,O|\lambda)=\alpha_t(i)\beta_t(i)$$

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

$$\gamma_t(i)=\frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}=\frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)}$$

γ 的意义

- 在每个时刻t选择在该时刻最有可能出现的状态i_t^*，从而得到一个状态序列$I^*=\{i_1^*, i_2^* \cdots i_T^*\}$，将它作为预测的结果。
- 给定模型和观测序列，时刻t处于状态qi的概率为：

$$\gamma_t(i)=\frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}=\frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)}$$

两个状态的联合概率

- 求给定模型λ和观测O，在时刻t处于状态qi并且时刻t+1处于状态qj的概率。

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

两个状态的联合概率

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

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

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

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

期望

- 在观测O下状态i出现的期望：

$$\sum_{t=1}^{T} \gamma_t(i)$$

- 在观测O下状态i转移到状态j的期望：

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

学习算法

- 若训练数据包括观测序列和状态序列，则HMM的学习非常简单，是监督学习；
- 若训练数据只有观测序列，则HMM的学习需要使用EM算法，是非监督学习。

大数定理

- 假设已给定训练数据包含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，可以直接利用Bernoulli大数定理的结论“频率的极限是概率”，给出HMM的参数估计。

监督学习方法

- 转移概率aij的估计：
  - 设样本中时刻t处于状态i时刻t+1转移到状态j的频数为Aij，则

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

- 观测概率bik的估计：
  - 设样本中状态i并观测为k的频数为Bik，则

$$\hat{b}_{ik} = \frac{B_{ik}}{\sum_{k=1}^{M} B_{ik}}$$

- 初始状态概率πi的估计为S个样本中初始状态为qi的概率。

Baum-Welch算法

- 若训练数据只有观测序列，则HMM的学习需要使用EM算法，是非监督学习。

附：EM算法整体框架

Repeat until convergence {

(E-step) For each i, set

$$Q_i(z^{(i)}) := p(z^{(i)}|x^{(i)};\theta).$$

(M-step) Set

$$\theta := \arg\max_{\theta} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

}

Baum-Welch算法

- 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$\ln P(O,I|\lambda)$
- 假设$\bar{\lambda}$是HMM参数的当前估计值，λ为待求的参数。

$$Q(\lambda,\bar{\lambda})=\sum_I\left(\ln P(O,I|\lambda)\right)P\left(I|O,\bar{\lambda}\right)$$

$$=\sum_I \ln P(O,I|\lambda)\frac{P\left(O,I|\bar{\lambda}\right)}{P\left(O,\bar{\lambda}\right)}$$

$$\propto\sum_I \ln P(O,I|\lambda)P\left(O,I|\bar{\lambda}\right)$$

EM过程

- 根据 $P(O,I|\lambda)=P(O|I,\lambda)P(I|\lambda)$

$$=\pi_{i_1}b_{i_1o_1}a_{i_1i_2}b_{i_2o_2}\cdots a_{i_{T-1}i_T}b_{i_To_T}$$

- 函数可写成

$$Q(\lambda,\bar{\lambda})=\sum_I \ln P(O,I|\lambda)P(O,I|\bar{\lambda})$$

$$=\sum_I \ln \pi_{i_1} P(O,I|\bar{\lambda})$$

$$+\sum_I\left(\sum_{t=1}^{T-1}\ln a_{i_t i_{t+1}}\right)P(O,I|\bar{\lambda})$$

$$+\sum_I\left(\sum_{t=1}^{T}\ln b_{i_t o_t}\right)P(O,I|\bar{\lambda})$$

极大化

- 极大化Q，求得参数A,B,π
- 由于该三个参数分别位于三个项中，可分别极大化

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

- 注意到πi满足加和为1，利用拉格朗日乘子法，得到：

$$\sum_{i=1}^{N} \ln \pi_i P(O,i_1 = i|\bar{\lambda}) + \gamma\left(\sum_{i=1}^{N} \pi_i - 1\right)$$

初始状态概率 $\sum_{i=1}^{N} \ln \pi_i P(O, i_1 = i | \bar{\lambda}) + \gamma \left(\sum_{i=1}^{N} \pi_i - 1 \right)$

- 对上式相对于πi求偏导，得到：

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

- 对i求和，得到：

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

- 从而得到初始状态概率：

$$\pi_i = \frac{P(O, i_1 = i | \bar{\lambda})}{P(O | \bar{\lambda})} = \frac{P(O, i_1 = i | \bar{\lambda})}{\sum_{i=1}^{N} P(O, i_1 = i | \bar{\lambda})} = \frac{\gamma_1(i)}{\sum_{i=1}^{N} \gamma_1(i)}$$

转移概率和观测概率

- 第二项可写成：

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

- 仍然使用拉格朗日乘子法，得到

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

- 同理，得到：

$$b_{ik}=\frac{\sum_{t=1}^{T}P(O,i_t=i|\bar{\lambda})I(o_t=v_k)}{\sum_{t=1}^{T}P(O,i_t=i|\bar{\lambda})}=\frac{\sum_{t=1,o_t=v_k}^{T}\gamma_t(i)}{\sum_{t=1}^{T}\gamma_t(i)}$$

预测算法

- 近似算法
- Viterbi算法

预测的近似算法

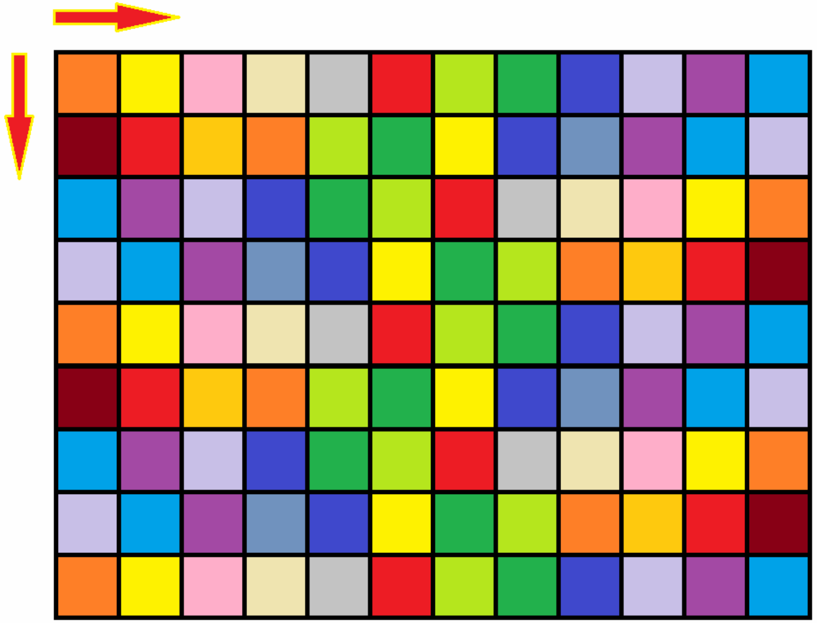
- 在每个时刻t选择在该时刻最有可能出现的状态i_t^*，从而得到一个状态序列$I^*=\{i_1^*, i_2^* \cdots i_T^*\}$，将它作为预测的结果。
- 给定模型和观测序列，时刻t处于状态qi的概率为：

$$\gamma_t(i)=\frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}=\frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)}$$

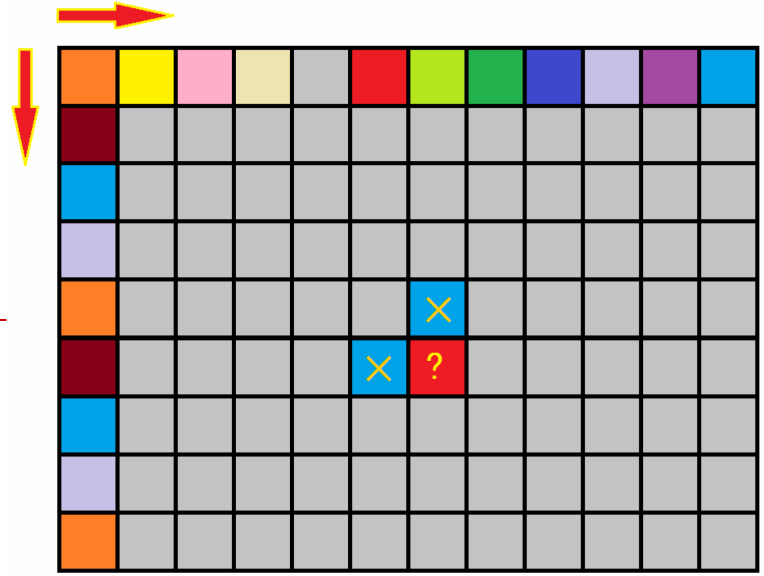
- 选择概率最大的i作为最有可能的状态
  - 会出现此状态在实际中可能不会发生的情况

算法：走棋盘/格子取数

- 给定m*n的矩阵，每个位置是一个非负整数，从左上角开始，每次只能朝右和下走，走到右下角，求总和最小的路径。



问题分析



- 走的方向决定了同一个格子不会经过两次。
  - 若当前位于(x,y)处，它来自于哪些格子呢？
  - dp[0,0]=a[0,0] / 第一行(列)累积
  - dp[x,y] = min(dp[x-1,y]+a[x,y],dp[x,y-1]+a[x,y])
  - 即：dp[x,y] = min(dp[x-1,y],dp[x,y-1]) +a[x,y]
- 思考：若将上述问题改成"求从左上到右下的**最大路径**"呢？

Viterbi算法

- Viterbi算法实际是用动态规划解HMM预测问题，用DP求概率最大的路径(最优路径)，这是一条路径对应一个状态序列。
- 定义变量$\delta_t(i)$：在时刻t状态为i的所有路径中，概率的最大值。

Viterbi算法

- 定义：

$$\delta_t(i) = \max_{i_1, i_2, \ldots i_{t-1}} P(i_t = i, i_{t-1}, \ldots i_1, o_t, \ldots o_1 | \lambda)$$

- 递推：

$$\delta_1(i) = \pi_i b_{io_1}$$

$$\delta_{t+1}(i) = \max_{i_1, i_2, \ldots i_t} P(i_{t+1} = i, i_t, \ldots i_1, o_{t+1}, \ldots o_1 | \lambda)$$

$$= \max_{1 \le j \le N} \left(\delta_t(j) a_{ji} \right) b_{io_{t+1}}$$

- 终止：

$$P^* = \max_{1 \le i \le N} \delta_T(i)$$

例

- 考察盒子球模型，观测向量O=“红白红”，试求最优状态序列。

$$\pi=\begin{pmatrix}0.2\\0.4\\0.4\end{pmatrix}\qquad A=\begin{bmatrix}0.5&0.2&0.3\\0.3&0.5&0.2\\0.2&0.3&0.5\end{bmatrix}\qquad B=\begin{bmatrix}0.5&0.5\\0.4&0.6\\0.7&0.3\end{bmatrix}$$

解：观测向量O=“红白红”

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

- 初始化：
- 在t=1时，对于每一个状态i，求状态为i观测到o1=红的概率，记此概率为 $\delta_1(t)$

$$\delta_1(i) = \pi_i b_{io_1} = \pi_i b_{i红}$$

- 求得 $\delta_1(1)=0.1$
- $\delta_1(2)=0.16$
- $\delta_1(3)=0.28$

解：观测向量O=“红白红” $A=\begin{bmatrix}0.5 & 0.2 & 0.3\\ 0.3 & 0.5 & 0.2\\ 0.2 & 0.3 & 0.5\end{bmatrix}$ $B=\begin{bmatrix}0.5 & 0.5\\ 0.4 & 0.6\\ 0.7 & 0.3\end{bmatrix}$

- 在t=2时，对每个状态i，求在t=1时状态为j观测为红并且在t=2时状态为i观测为白的路径的最大概率，记概率为δ $_2$(t)，则：

$$\delta_{t+1}(i)=\max_{1\le j\le 3}\left(\delta_1(j)a_{ji}\right)b_{io_2}=\max_{1\le j\le 3}\left(\delta_1(j)a_{ji}\right)b_{i白}$$

- 求得

$$\delta_2(1)=\max_{1\le j\le 3}\left(\delta_1(j)a_{j1}\right)b_{i白}$$

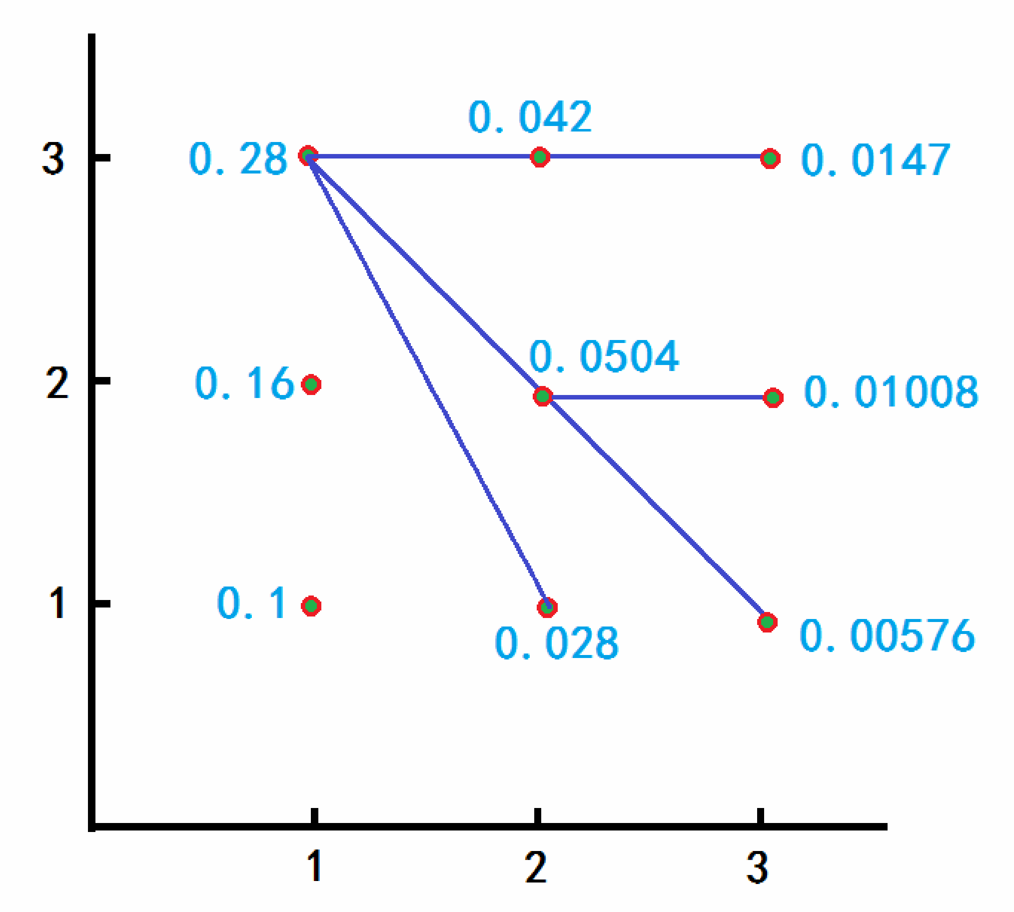
$$=\max\{0.10\times0.5,0.16\times0.3,0.28\times0.2\}\times0.5=0.028$$

- 同理：
  - $\delta_2(2)=0.0504$
  - $\delta_2(3)=0.042$

解：观测向量O="红白红"

- 同理，求得
- $\delta_3(1)=0.00756$
- $\delta_3(2)=0.01008$
- $\delta_3(3)=0.0147$
- 从而，最大是$\delta_3(3)=0.0147$，根据每一步的最大，得到序列是(3,3,3)

求最优路径图解



Baum-Welch code：初始化

```python
if __name__ == "__main__":
    # 初始化pi,A,B
    pi = [random.random() for x in range(4)]    # 初始分布
    log_normalize(pi)
    A = [[random.random() for y in range(4)] for x in range(4)] # 转移矩阵
    A[0][0] = A[0][3] = A[1][0] = A[1][3]\
        = A[2][1] = A[2][2] = A[3][1] = A[3][2] = 0 # 不可能事件
    B = [[random.random() for y in range(65536)] for x in range(4)]
    for i in range(4):
        log_normalize(A[i])
        log_normalize(B[i])
    baum_welch(pi, A, B)
    save_parameter(pi, A, B)
```

Baum-Welch：主函数

```python
def baum_welch(pi, A, B):
    f = file(".\\text\\1.txt")
    sentence = f.read()[3:].decode('utf-8') # 跳过文件头
    f.close()
    T = len(sentence)   # 观测序列
    alpha = [[0 for i in range(4)] for t in range(T)]
    beta = [[0 for i in range(4)] for t in range(T)]
    gamma = [[0 for i in range(4)] for t in range(T)]
    ksi = [[[0 for j in range(4)] for i in range(4)] for t in range(T-1)]
    for time in range(100):
        calc_alpha(pi, A, B, sentence, alpha)    # alpha(t,i):给定lamda，在时刻t的状态为i
        calc_beta(pi, A, B, sentence, beta)      # beta(t,i): 给定lamda和时刻t的状态i，观
        calc_gamma(alpha, beta, gamma)           # gamma(t,i): 给定lamda和O，在时刻t状态位
        calc_ksi(alpha, beta, A, B, sentence, ksi)    # ksi(t,i,j): 给定lamda和O，在时刻t
        bw(pi, A, B, alpha, beta, gamma, ksi, sentence) #baum_welch算法
```

前向-后向

```python
def calc_alpha(pi, A, B, o, alpha):
    for i in range(4):
        alpha[0][i] = pi[i] + B[i][ord(o[0])]
    T = len(o)
    temp = [0 for i in range(4)]
    del i
    for t in range(1, T):
        for i in range(4):
            for j in range(4):
                temp[j] = (alpha[t-1][j] + A[j][i])
            alpha[t][i] = log_sum(temp)
            alpha[t][i] += B[i][ord(o[t])]
```

```python
def calc_beta(pi, A, B, o, beta):
    T = len(o)
    for i in range(4):
        beta[T-1][i] = 1
    temp = [0 for i in range(4)]
    del i
    for t in range(T-2, -1, -1):
        for i in range(4):
            beta[t][i] = 0
            for j in range(4):
                temp[j] = A[i][j] + B[j][ord(o[t+1])] + beta[t+1][j]
            beta[t][i] += log_sum(temp)
```

隐状态概率 – 隐状态转移概率

```
def calc_gamma(alpha, beta, gamma):
    for t in range(len(alpha)):
        for i in range(4):
            gamma[t][i] = alpha[t][i] + beta[t][i]
        s = log_sum(gamma[t])
        for i in range(4):
            gamma[t][i] -= s
```

```
def calc_ksi(alpha, beta, A, B, o, ksi):
    T = len(alpha)
    temp = [0 for x in range(16)]
    for t in range(T-1):
        k = 0
        for i in range(4):
            for j in range(4):
                ksi[t][i][j] = alpha[t][i] + A[i][j] + B[j][ord(o[t+1])] + beta[t+1][j]
                temp[k] =ksi[t][i][j]
                k += 1
        s = log_sum(temp)
        for i in range(4):
            for j in range(4):
                ksi[t][i][j] -= s
```

EM迭代

```python
def bw(pi, A, B, alpha, beta, gamma, ksi, o):
    T = len(alpha)
    for i in range(4):
        pi[i] = gamma[0][i]
    s1 = [0 for x in range(T-1)]
    s2 = [0 for x in range(T-1)]
    for i in range(4):
        for j in range(4):
            for t in range(T-1):
                s1[t] = ksi[t][i][j]
                s2[t] = gamma[t][i]
            A[i][j] = log_sum(s1) - log_sum(s2)
    s1 = [0 for x in range(T)]
    s2 = [0 for x in range(T)]
    for i in range(4):
        for k in range(65536):
            valid = 0
            for t in range(T):
                if ord(o[t]) == k:
                    s1[valid] = gamma[t][i]
                    valid += 1
                s2[t] = gamma[t][i]
            if valid == 0:
                B[i][k] = infinite
            else:
                B[i][k] = log_sum(s1[:valid]) - log_sum(s2)
```

Viterbi

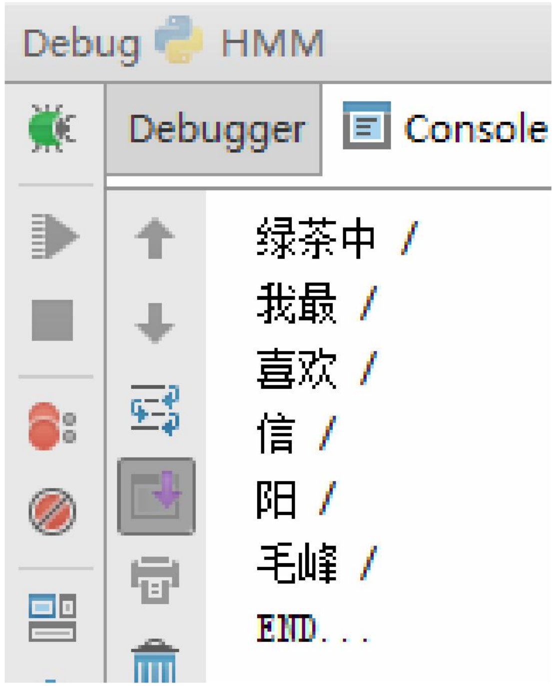
```python
def viterbi(pi, A, B, o):
    T = len(o)   # 观测序列
    delta = [[0 for i in range(4)] for t in range(T)]
    pre = [[0 for i in range(4)] for t in range(T)]  # 前一个状态
    for i in range(4):
        delta[0][i] = pi[i] + B[i][ord(o[0])]
    for t in range(1, T):
        for i in range(4):
            delta[t][i] = delta[t-1][0] + A[0][i]
            for j in range(1,4):
                vj = delta[t-1][j] + A[j][i]
                if delta[t][i] < vj:
                    delta[t][i] = vj
                    pre[t][i] = j
            delta[t][i] += B[i][ord(o[t])]
    decode = [-1 for t in range(T)]  # 解码：回溯查找最大路径
    q = 0
    for i in range(1, 4):
        if delta[T-1][i] > delta[T-1][q]:
            q = i
    decode[T-1] = q
    for t in range(T-2, -1, -1):
        q = pre[t+1][q]
        decode[t] = q
    return decode
```

分词

```python
def segment(sentence, decode):
    N = len(sentence)
    i = 0
    while i < N:  #B/M/E/S
        if decode[i] == 0 or decode[i] == 1:  # Begin
            j = i+1
            while j < N:
                if decode[j] == 2:
                    break
                j += 1
            print sentence[i:j+1], "/"
            i = j+1
        elif decode[i] == 3 or decode[i] == 2:  # single
            print sentence[i:i+1], "/"
            i += 1
        else:
            print 'Error:', i, decode[i]
            i += 1
```

测试

- 全国销量领先的红罐凉茶改成广州恒大
- 绿茶中我最喜欢信阳毛峰



参考文献

- 统计学习方法，李航著，清华大学出版社，2012年
- Pattern Recognition and Machine Learning Chapter 13, Bishop M, Springer-Verlag, 2006
- A Tutorial on Learning With Bayesian Networks, David Heckerman, 1996
- Radiner L,Juang B. An introduction of hidden markov Models. IEEE ASSP Magazine, January 1986

我们在这里

7 七月算法 http://www.julyedu.com/
- 视频/课程/社区

- 七月题库APP：Android/iOS
  - http://www.julyapp.com/
- 微博
  - @研究者July
  - @七月题库
  - @邹博_机器学习
- 微信公众号
  - julyedu

感谢大家！

恳请大家批评指正！