# （选）应用随机过程课程论文

课程号：754.007.201　时间：2020-2021学年第二学期

# 隐式马尔科夫模型(HMM)原理介绍

华光辉　经济统计1901　17066003

**摘　要**

隐马尔可夫模型背后的数学是由LEBaum和他的同事开发的。它与早期由RuslanL.Stratonovich提出的最优非线性滤波问题息息相关。在简单的马尔可夫模型（如马尔可夫链），所述状态是直接可见的观察者，因此状态转移概率是唯一的参数。在隐马尔可夫模型中，状态是不直接可见的，但输出依赖于该状态下，是可见的。每个状态通过可能的输出记号有了可能的概率分布。因此，通过一个HMM产生标记序列提供了有关状态的一些序列的信息。隐马尔可夫模型作为一种统计分析模型，创立于20世纪70年代。80年代得到了传播和发展，成为信号处理的一个重要方向，现已成功地用于生物信息学，语音识别，行为识别，文字识别以及故障诊断等领域。也是如今热门的机器学习领域的一个基础模型之一。本文通过一个具体实例介绍隐马尔可夫模型，然后介绍该模型中的三个基本问题，以及解决三个问题的方法以及原理

**关键词：**隐马尔可夫模型，前向后向算法，EM算法，维比特算法

## 1　问题与背景

### 1.1　问题

本文采用李航老师《统计学习方法》一文中的例子，
转移概率矩阵
观测概率矩阵
转移概率矩阵

表 1: 模型数据

| 盒子编号 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 红球数 | 5 | 3 | 6 | 8 |
| 白球数 | 5 | 7 | 4 | 2 |

表 2: 观测概率矩阵

| | $v_1$ | $v_2$ |
|---|---|---|
| $q_1$ | 0.5 | 0.5 |
| $q_2$ | 0.3 | 0.7 |
| $q_3$ | 0.6 | 0.4 |
| $q_4$ | 0.8 | 0.2 |

表 3: 模型数据

| | $q_1$ | $q_2$ | $q_3$ | $q_4$ |
|---|---|---|---|---|
| $q_1$ | 0 | 1 | 0 | 0 |
| $q_2$ | 0.4 | 0 | 0.6 | 0 |
| $q_3$ | 0 | 0.4 | 0 | 0.6 |
| $q_4$ | 0 | 0 | 0.5 | 0.5 |

## 1.2 符号解释

隐马尔可夫模型中的三个基本问题

表 4: 符号说明

| 属性 | 符号 |
|---|---|
| 所有可能状态的个数 | $N$ |
| 所有可能观测的个数 | $M$ |
| 状态序列的个数 | $T$ |
| 单次状态 | $q_i$ |
| 所有可能状态的集合 | $Q=\{q_1,q_2,\ldots,q_N\}$ |
| 单次观测 | $v_i$ |
| 所有可能观测的集合 | $V=\{v_1,v_2,\ldots,v_M\}$ |
| 状态序列 | $I=\{i_1,i_2,\ldots,i_T\}$ |
| 观测序列 | $O=\{o_1,o_2,\ldots,o_T\}$ |
| 状态转移概率 | $a_{ij}=P(i_{t+1}=q_j\|i_t=q_i)$ |
| 条件观测概率 | $b_j(k)=P(o_t=v_k\|i_t=q_j)$ |
| 状态转移矩阵 | $A=[a_{ij}]_{N\times N}$ |
| 观测矩阵 | $B=[b_j(k)]_{N\times M}$ |
| 初始状态概率 | $\pi_i=P(i_1=q_i)$ |
| 初始状态概率向量 | $\pi=(\pi_i)$ |
| 所有参数 | $\lambda=(A,B,\pi)$ |

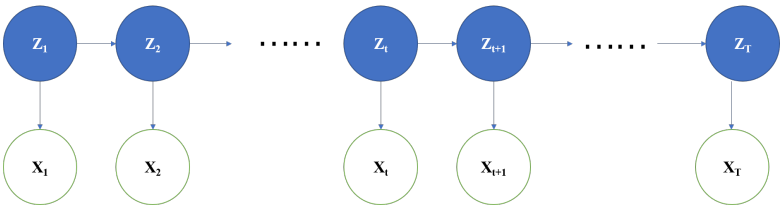


图 1: 示意图

## 参考文献

[1] 李航，统计学习方法，北京：清华大学出版社，2012.3。

[2] Weisong Zhao，HMM隐马尔可夫模型详解，https://blog.csdn.net/weixin_41923961/article/details/82750687，20(3)，2021。

# 附录

## 附录1. 原始数据

## 附录2. 随机模拟程序及运行结果

```
import numpy as np
pi = np.array([.25, .25, .25, .25])
A = np.array([
[0,  1,  0, 0],
[.4, 0, .6, 0],
[0, .4, 0, .6],
[0, 0, .5, .5]])
B = np.array([
[.5, .5],
[.3, .7],
[.6, .4],
[.8, .2]])

def get_data_with_dist(dist):
    r = np.random.rand()
    print(r)
    for i, p in enumerate(dist):
    if r < p: return i
    r -= p

def generate(T: int):
    z = get_data_with_dist(pi)
    x = get_data_with_dist(B[z])
    result = [x]
    for _ in range(T-1):
    z = get_data_with_dist(A[z])
    x = get_data_with_dist(B[z])
    result.append(x)
    return result

generate(3)
```