偶尔作弊的赌场——隐马尔可夫中的动态规划

原创 苏欣妍 LOA算法学习笔记 2021-02-26 20:16

01 引言

隐马尔可夫模型（Hidden Markov Model，HMM）是关于时序的概率模型，是一个生成模型，由一个隐藏的马尔科夫链随机生成不可观测的状态序列描述。语音识别、基因预测等很多应用问题都可以建模成隐马尔可夫模型的解码问题。本文将以偶尔作弊的赌场为例介绍 HMM 解码问题中涉及的动态规划算法。

1.1 隐马尔可夫模型的定义

在探究这个问题之前，首先了解一下隐马尔可夫模型的定义形式。隐马尔可夫模型由初始概率分布、状态转移概率分布以及观测概率分布确定，定义如下：设Q为所有可能的状态的集合，V是所有可能的观测的集合，其中，N是可能的状态数，M是可能的观测数。I是长度为T的状态序列，O是对应的观测序列。

$$
\begin{aligned}
Q &= \{q_1, q_2, \ldots, q_N\} \\
V &= \{v_1, v_2, \ldots, v_M\} \\
I &= \{i_1, i_2, \ldots, i_T\} \\
O &= \{o_1, o_2, \ldots, o_T\}
\end{aligned}
$$

A是转移概率矩阵，B是观测概率矩阵，π是初始状态概率向量。其中 a_{ij} 是在时刻t状态 q_i 条件下在时刻t+1转移到状态 q_j 的概率。其中 $b_j(k)$ 是在时刻t处于状态 q_j 的条件下生成观测状态 v_k 的概率。

$$
\begin{aligned}
A &= [a_{ij}]_{N*N}, & a_{ij} &= P(i_{t+1} = q_j | i_t = q_i) \\
B &= [b_j(k)]_{N*M}, & b_j(k) &= P(o_t = v_k | i_t = q_j) \\
\pi &= (\pi_i), & \pi_i &= P(i_1 = q_i) \\
O &= \{o_1, o_2, \ldots, o_T\}
\end{aligned}
$$

因此，隐马尔可夫模型可以用三元符号表示，即（A，B，π），即隐马尔可夫模型的三要素。

1.2 隐马尔可夫模型的基本假设

齐次马尔科夫假设，即假设隐马尔可夫链在任意时刻的状态只依赖于其前一时刻的状态，与其他时刻的状态及观测无关；观测独立性假设，即假设任意时刻的观测只依赖于该时刻的马尔可夫链的状态，与其他时刻的观测及状态无关。

1.3 隐马尔可夫模型的解码问题

解码问题也称为预测问题。即已知观测序列，求解在条件概率最大的情况下，最有可能对应的隐状态序列。

02 问题描述

2.1 问题叙述

一家赌场里有两个骰子，一个是公平的骰子（Fair dice，记为 F），其掷出 6 种点数的概率都是1/6；另一个骰子是灌铅的（Loaded dice，记为 L），其掷出 5 点和 6 点的概率是3/10，掷出其他点数的概率是1/10（如图 2.1）。赌场有时使用公平的骰子，有时则会换成灌铅的骰子。

我们假设赌场按照如下的方式使用两个骰子：在第一次投掷时，赌场选择公平骰子的概率是3/5，选择灌铅骰子的概率是2/5；在后续投掷时，如果上一次使用的是公平骰子，则赌场下一轮使用灌铅骰子的概率是 0.2；如果上一轮使用的是灌铅骰子，则下一轮使用公平骰子的概率是 0.1。

我们考虑如下问题：当观察到6次投掷的点数 1, 3, 4, 5, 5, 6时，能否推测出每一次投掷最可能使用的是哪个骰子，图解此问题表示为图2.2。

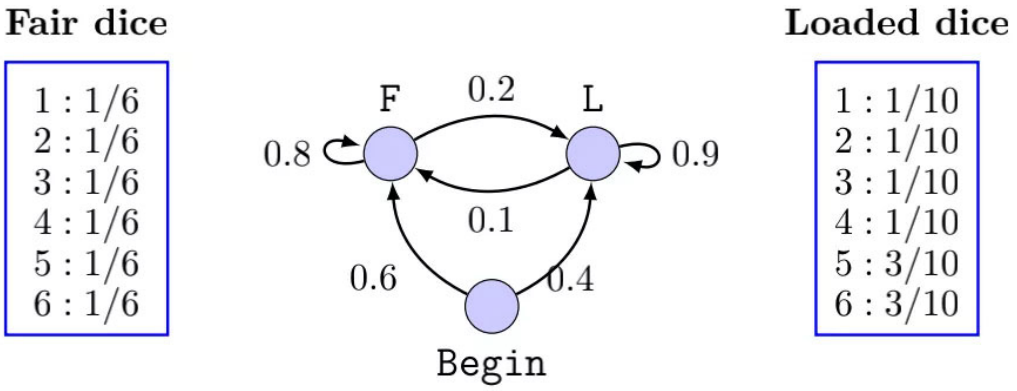

图2.1

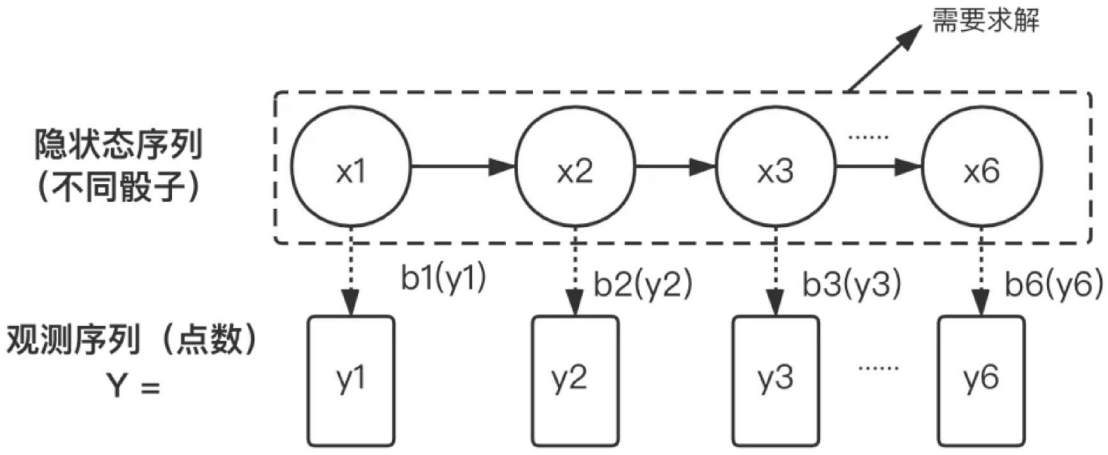

图2.2

2.2 问题建模

使用转移概率矩阵A，观测概率矩阵B和初始状态概率向量π以及观测序列O建模这个问题为：

$$A = \begin{bmatrix} 0.8 & 0.2 \\ 0.1 & 0.9 \end{bmatrix}$$

$$B = \begin{bmatrix} \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{3}{10} & \frac{3}{10} \end{bmatrix}$$

$$\boldsymbol{\pi} = \begin{bmatrix} 0.6 & 0.4 \end{bmatrix}$$

$$O = \begin{bmatrix} 1 & 3 & 4 & 5 & 5 & 6 \end{bmatrix}$$

03 解决方案

如果之前没有接触过隐马尔可夫模型，可以用有向图求解最长路径的思想理解这个问题。如图3.1，若求解图中的最大路径，利用动态规划，可写出最优子结构的递推关系式：

$$dilg(v)=max(u,v)=max(E\{dilg(u)+w(u, v)\})$$

结点u是指向结点v的前一结点，w(u,v)是u和v的权重，E表示长度。如果把这里的结点看成不同时刻的隐状态以及对应观测状态的乘积，结点之间的权重即为不同隐状态之间的转移概率，那么从前向后可以一步步推出路径的最大可能，最终得到一个从起点连接每一个终点的m条路径(假设有m个终点)。下面详细分析多步决策过程。

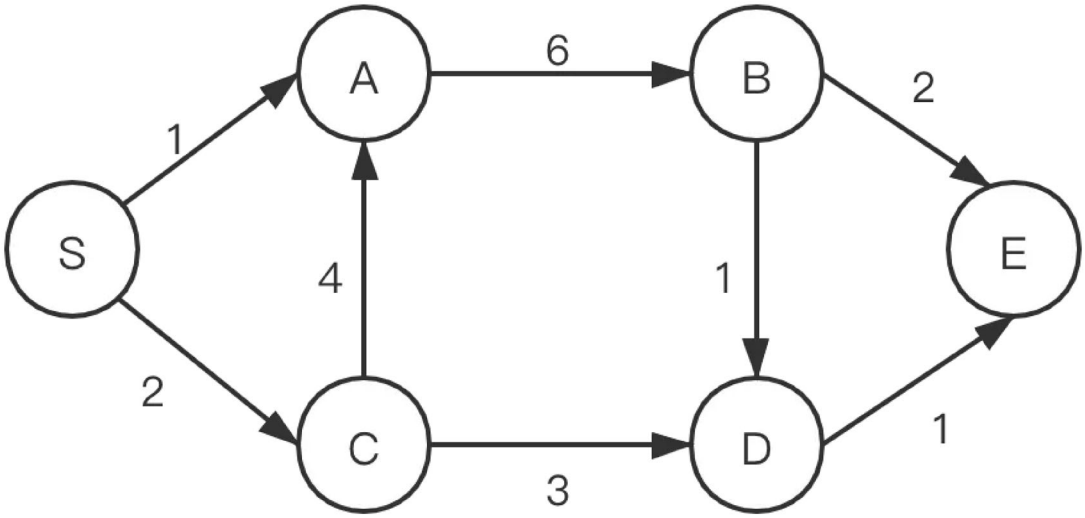

图3.1

1) 最简单的case会不会做?

对于最简单的实例 n = 1 来说，我们只需要比较 πFbF(y1) 和 πLbL(y1) 的大小即可确定最可能的隐含状态；然而当 n 比较大时，n 个时刻的状态总数很大，我们采用分而治之的策略。注意到问题的完整解是所有隐含状态变量 $x_1, x_2, \ldots, x_n$ 的取值，我们可
以将求解过程描述成按照 $x_n, x_{n-1}, \ldots, x_1$ 的顺序, 从后往前逐个确定隐含状态变量
取值的多步决策过程。

2) n>2 的时候如何定义子问题?

这个问题的目标是求解:

$$OPT(y_1, \cdots, y_n) = \max_{x_1, \cdots, x_n} \pi_{x_1} b_{x_1}(y_1) \prod_{i=2}^{n} (a_{x_{i-1} x_i} b_{x_i}(y_i)).$$

a）将子问题直接定义为计算 $OPT(y_1, \ldots, y_i)$

思考过程： $OPT(y_1, \ldots, y_i)$ 假如将子问题定义为计算，将难以在子问题最优解之间构建递归表达式。由隐马尔可夫模型的基本假设可知，隐含状态变量 x_{i-1} 依赖于状态 x_i，因此我们对 x_i 做的决策将会影响对 x_{i-1} （以及前面所有 i-1 个状态变量）的决策(此时的决策顺序是从后到前)，导致无法保证子问题的最优解 $\arg\max x_1, \cdots, x_{n-1} P(x_1, y_1, \ldots, x_{n-1}, y_{n-1})$ 是原问题最优解 $\arg\max x_1, \cdots, x_n P(x_1, y_1, \ldots, x_n, y_n)$ 的一部分。

b）将子问题定义得更为细致

在第 i 步，我们确定隐含状态变量xi的取值，需要在2个决策项 $x_i = F$ 和 $x_i = L$ 中做出选择，并使用 $b_{x_i}(y_i) a_{x_i x_{i+1}}$ 计算其收益。我们还需要单独处理i=1和i=6时的情况。采用上述方法可以将原始问题的优化目标函数等价成求解所有决策步骤的收益乘积。即确定隐含状态 $x_1, \ldots, x_{i-1}$，使得当 $x_i = k$ 时， $P(x_1, y_1, \ldots, x_i, y_i)$ 最大，将此最大值记为 $V_i(k)$，即:

$$V_i(k) = \max_{x_1, x_2, \ldots, x_{i-1}} P(x_1, y_1, \ldots, x_i = k, y_i)$$

3) 递推关系式

结合之前最大路径求解的思想， $V_i(k)$ 是求结束于 $x_i = k$ 状态的概率最大状态路径；假如这条路径经过 $x_{i-1} = l$ 顶点，则其前i -1个顶点必定构成结束于 $x_{i-1} = l$ 的概率最大状态路径。下面定义关于结束于 $x_i = k$ 状态的最优子结构:

$$V_i(k) = \max \begin{cases} \pi_k b_k(y_1) & i = 1 \\ \max_{l \in \{F,L\}} V_{i-1}(l) a_{lk} b_k(y_i) & else \end{cases}$$

$$OPT(y_1, \ldots, y_n) = \max\{V_n(F), V_n(L)\}.$$

每一步记录下上一步中到达概率最大状态的隐状态结点，最后的时候回溯即可得到最优隐状态序列。

4) python代码实现：

如果采用迭代技术来计算 $V_i(k)$，对应的算法称为 Viterbi 算法，具体实现如下。

```
import numpy as np
def viterbi(trainsition_probability,emission_probability,pi,obs_seq,hidden_s):
    #convert to the form of np.array
    trainsition_probability=np.array(trainsition_probability)
    emission_probability=np.array(emission_probability)
    pi=np.array(pi)

    # the result should be a matrix of Row*Col
    # row represents the length of hidden states
    # col represents the length of observed sequence
    Row = np.array(trainsition_probability).shape[0]
    Col = len(obs_seq)
    # initialize the result matrix and the backtrace matrix
    F=np.zeros((Row,Col))
    B=np.zeros((Row,Col))
    # the first col of F equals pi*bi(x_1), where x_1=obs_seq[0]
    F[:,0]=pi*np.transpose(emission_probability[:,obs_seq[0]-1])
    # t = 2,3,...T-1(col-1)
    for t in range(1,Col):
        list_max=[]
        for n in range(Row):
            # (Vt-1(n)*anj)
            list_x=list(np.array(F[:,t-1])*np.transpose(trainsition_probability[:,n]))
            # get the max probability of (Vt-1(n)*anj)
            list_p=[]
            for i in list_x:
                list_p.append(i*1000000)
            B[n][t]= list_p.index(max(list_p))
            list_max.append(max(list_p)/1000000)
        # max(Vt-1(n)*anj)*bj(x_t)
        F[:,t]=np.array(list_max)*np.transpose(emission_probability[:,obs_seq[t]-1])
    # backtrace
    list_last =  list(F[:,Col-1])
    # Z_T = argmax_i(deta_T(i))
    i_T = list_last.index(max(list_last))
    path = [0]*Col
```

```
    i_t = i_T
    for t in reversed(range(Col-1)):
        # Zt = fai_[t+1](Z_[t+1])
        i_t = int(B[i_t][t+1])
        path[t]=hidden_s[i_t]
    path[-1] = hidden_s[i_T]

    print('hidden_state_seq\n',path)
    return F,B

if __name__=='__main__':
    # hidden states
    hidden_s=['F','L']
    # initial states
    pi=[0.6,0.4]
    # trainsition matirx
    trainsion_probility=[[0.8,0.2],[0.1,0.9]]
    # emission matrix
    emission_probility=[[1/6,1/6,1/6,1/6,1/6,1/6],[1/10,1/10,1/10,1/10,3/10,3/10]]
    # observed seq
    obs_seq=[1,3,4,5,5,6]
    # final result
    F,B = viterbi(trainsion_probility,emission_probility,pi,obs_seq,hidden_s)
    print('result matirx\n',F)
    print('backtrace matrix\n',B)
```

时间复杂度：相当于对数组的遍历 $O(T \cdot N^2)$
空间复杂度：用到了几个数组存储数据 $O(N \cdot T)$
运行结果：

```
/data-trans/Viterbi.py
hidden_state_seq
 ['F', 'F', 'F', 'L', 'L', 'L']
result matirx
 [[1.00000000e-01 1.33333333e-02 1.77777778e-03 2.37037037e-04
  3.16049383e-05 4.21399177e-06]
 [4.00000000e-02 3.60000000e-03 3.24000000e-04 1.06666667e-04
  2.88000000e-05 7.77600000e-06]]
backtrace matrix
 [[0. 0. 0. 0. 0. 0.]
 [0. 1. 1. 0. 1. 1.]]
```

04 隐马尔可夫问题的扩展

上文分析的赌场问题是隐马尔可夫模型中的预测问题，与其他两个问题**概率计算问题**和**参数学习问题**并称为隐马尔可夫模型的3个基本问题。概率计算问题是给定模型和观测序列，计算观测序列出现的概率，该问题涉及的前向算法和

后向算法也可以利用上文提到的动态规划的思想进行思考，只不过求解的是全部路径之和，而不是一条路径。已知观测序列，估计模型参数，使得在该模型下观测序列概率最大，用极大似然估计方法估计参数为参数学习问题。

05 结语

隐马尔可夫模型在本学期多门课上被提及，于是笔者结合动态规划的思想，对维特比算法进行进一步的思考。正如卜老师所言，遇到任何一个问题，我们不应该轻易按照通用答案的思路思考，局限自己的思维；而应该有更多自己的想法，才会真正理解这个问题。最后，希望这篇文章能给您带来帮助。作为算法上的小白，错误在所难免，希望您在评论区不吝告知，将不胜感激。

参考文献：

1. https://zhuanlan.zhihu.com/p/144448849
2. 《算法讲义——关于求解问题的十八讲》.卜东坡、张家琳

喜欢此内容的人还喜欢

LOA公众号关闭通知
LOA算法学习笔记

我们在20个城市，邀请你探讨"什么是人生终极命题？"｜706流动客厅
共鸣读书社