University of Chinese Academy of Sciences

模式识别与机器学习

081203M04004H

Chap 10 课程作业解答

2022 年 12 月 26 号

Professor: 黄庆明

学生: 周胤昌

学号: 202228018670052

学院: 网络空间安全学院

所属专业: 网络空间安全

方向: 安全协议理论与技术

Problem 1

假设我们要采用 HMM 实现一个英文的词性标注系统, 系统中共有 20 种词性, 则状态转移矩阵 $\boldsymbol{A}$ 的大小为 ______.

Solution: 由于系统中共有 20 种词性, 因此 Markov 状态节点的个数就是 20, 于是状态转移矩阵的大小为 20×20.

Problem 2

已知以下贝叶斯网络 (如图1中所示), 包含 7 个变量, 即 Season(季节)、Flu(流感)、Dehydration(脱水)、Chills(发冷)、Headache(头疼)、Nausea(恶心)、Dizziness(头晕), 则下列条件独立成立的是 ( ).

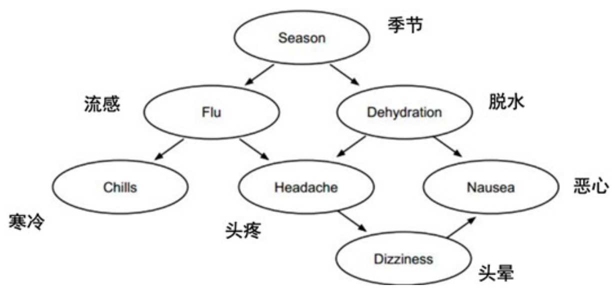

图 1: 贝叶斯网络

(A) . $Season \perp Chills | Flu$ (B) . $Season \perp Chills$ (C) . $Season \perp Headache | Flu$

Solution: 为了叙述方便, 我们不妨先对上述贝叶斯网络中的各节点进行拓扑排序, 具体如下图2所示:

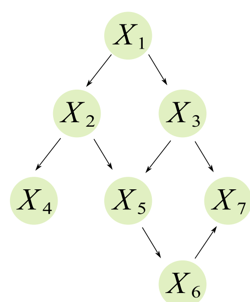

图 2: 简化后的贝叶斯网络

上述选项即变为

$$(A) . X_1 \perp X_4 | X_2 \quad (B) . X_1 \perp X_4 \quad (C) . X_1 \perp X_5 | X_2$$

显然 B 选项错误, 先看 C 选项: 给定 X_2 时, 检查 X_1, X_5 的可达性. 利用快速检验准则: 从 X_1 出发, 通过 X_3, 即可到达 X_5, 因此 C 项错误. 再看 A 项, 利用准则可以看出, 从 X_1 出发, 要么在 $X_1 \to X_2 \to X_4$ 这条路线上被反弹回 X_1. 要么先经过 $X_1 \to X_3 \to X_5$ 到达 X_5, 但是球在路线 $X_5 \to X_2 \to X_4$ 上被 X_2 截止了, 总之到达不了 X_4; 从 X_4 出发, 类似的, 也是到达不了 X_1. 因此, $X_1 \perp X_4 | X_2$(即 A 项正确).

Problem 3

已知以下贝叶斯网络 (如图3中所示), 包含 4 个二值变量, 则该网络一共有 (　) 个参数.

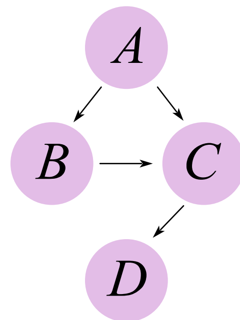

图 3: 贝叶斯网络

Solution: 先写出联合概率分布

$$p(A, B, C, D) = p(A) \cdot p(B|A) \cdot p(C|A, B) \cdot p(D|C)$$

因此网络参数共有 $2^0 + 2^1 + 2^2 + 2^1 = 9$ 个, 于是选 C 项.

Problem 4

假设你有 3 个盒子, 每个盒子里都有一定数量的苹果和橘子. 每次随机选择一个盒子, 然后从盒子里选一个水果, 并记录你的发现 (a 代表苹果, o 代表橘子). 不幸的是, 你忘了写下你所选的盒子, 只是简单的记下了苹果和橘子. 假设每个盒子中水果数量如下:

- 盒子 1: 2 个苹果, 2 个橘子;
- 盒子 2: 3 个苹果, 1 个橘子;
- 盒子 3: 1 个苹果, 3 个橘子;

(1). 请用 HMM 模型描述上述过程;
(2). 请给出水果序列 $\boldsymbol{x} = (a, a, o, o, o)$ 对应的最佳盒子序列.

Solution: (1). 将盒子视作隐变量 (即状态节点), 拿出来的水果视作观测变量 (即输出节点). 因为每次都是随机选取的盒子, 因此初始状态的概率分布应为均匀分布, 即 $\boldsymbol{\pi} = \begin{pmatrix}1/3 & 1/3 & 1/3\end{pmatrix}^{\mathrm{T}}$. 因为每次都是以均匀分布抽取盒子的且 $a_{ij} = p\left(y_{t+1} = s_j | y_t = s_i\right)$, 因此盒子间的状态转移矩阵为

$$\boldsymbol{A} = \left(a_{ij}\right)_{N \times N} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{pmatrix}$$

由于 $b_{ij} = p\left(x_t = o_j | y_t = s_i\right)$, 因此发射概率矩阵 (给定盒子时, 选择每种水果的概率) 为

$$\boldsymbol{B} = \left(b_{ij}\right)_{N \times M} = \begin{pmatrix} 1/2 & 1/2 \\ 3/4 & 1/4 \\ 1/4 & 3/4 \end{pmatrix}$$

(2). Viterbi 解码算法如下:

$$\text{动态规划}: V_1(j) = \pi_j b_j(x_1), \begin{cases} V_t(j) = b_j(x_t) \cdot \max\limits_{1\le i\le N} \{a_{ij} V_{t-1}(i)\} \\ \psi_t(j) = \arg\max\limits_{1\le i\le N} \{a_{ij} V_{t-1}(i)\} \end{cases}, \text{其中} 1 \le j \le N$$

$$\text{反向回溯}: P^* = \max_{1\le i\le N} V_T(i), \begin{cases} i_T^* = \arg\max\limits_{1\le i\le N} V_T(i) \\ i_t^* = \psi_{t+1}(i_{t+1}^*) \end{cases}$$

- 当 $t = 1$ 时, 已知 $x_1 = a$, 所以初始化如下:

$$\begin{aligned} V_1(1) &= \pi_1 b_1(a) = 1/3 \cdot 1/2 = 1/6, \psi_1(1) = 0 \\ V_1(2) &= \pi_2 b_2(a) = 1/3 \cdot 3/4 = 1/4, \psi_1(2) = 0 \\ V_1(3) &= \pi_3 b_3(a) = 1/3 \cdot 1/4 = 1/12, \psi_1(3) = 0 \end{aligned}$$

- 当 $t = 2$ 时, 已知 $x_2 = a$, 所以有:

$$\begin{aligned} V_2(1) &= b_1(a) \cdot 1/3 \cdot 1/4 = 1/24, \psi_2(1) = 2 \\ V_2(2) &= b_2(a) \cdot 1/3 \cdot 1/4 = 1/16, \psi_2(2) = 2 \\ V_2(3) &= b_3(a) \cdot 1/3 \cdot 1/4 = 1/48, \psi_2(3) = 2 \end{aligned}$$

- 当 $t = 3$ 时, 已知 $x_3 = o$, 所以有:

$$\begin{aligned} V_3(1) &= b_1(o) \cdot 1/3 \cdot 1/16 = 1/96, \psi_3(1) = 2 \\ V_3(2) &= b_2(o) \cdot 1/3 \cdot 1/16 = 1/192, \psi_3(2) = 2 \\ V_3(3) &= b_3(o) \cdot 1/3 \cdot 1/16 = 1/64, \psi_3(3) = 2 \end{aligned}$$

- 当 $t = 4$ 时, 已知 $x_4 = o$, 所以有:

$$\begin{aligned} V_4(1) &= b_1(o) \cdot 1/3 \cdot 1/64 = 1/384, \psi_4(1) = 3 \\ V_4(2) &= b_2(o) \cdot 1/3 \cdot 1/64 = 1/768, \psi_4(2) = 3 \\ V_4(3) &= b_3(o) \cdot 1/3 \cdot 1/64 = 1/256, \psi_4(3) = 3 \end{aligned}$$

- 当 $t = 5$ 时, 已知 $x_5 = o$, 所以有:

$$\begin{aligned} V_5(1) &= b_1(o) \cdot 1/3 \cdot 1/256 = 1/1536, \psi_5(1) = 3 \\ V_5(2) &= b_2(o) \cdot 1/3 \cdot 1/256 = 1/3072, \psi_5(2) = 3 \\ V_5(3) &= b_3(o) \cdot 1/3 \cdot 1/256 = 1/1024, \psi_5(3) = 3 \end{aligned}$$

迭代过程终止, 且 $P^* = \max\limits_{1\le i\le 3} V_T(i) = V_5(3) = \dfrac{1}{1024}$. 回溯过程为:

$$i_5^* = 3, i_4^* = \psi_5(3) = 3, i_3^* = \psi_4(3) = 3, i_2^* = \psi_3(3) = 2, i_1^* = \psi_2(2) = 2$$

因此回溯出最优路径为 $\boldsymbol{y} = \{2, 2, 3, 3, 3\}$.

Problem 5

给定如图4所示的 HMM

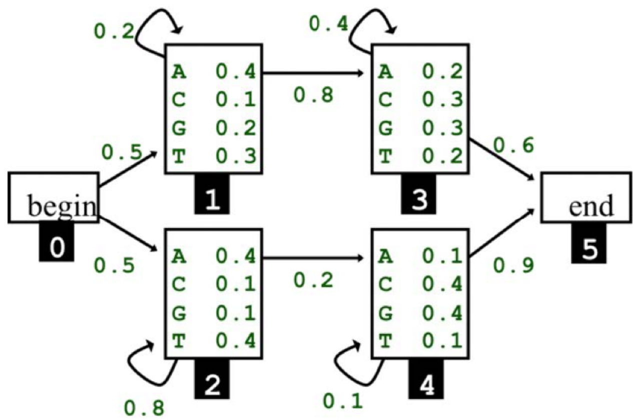

图 4: HMM 示意图

(1). 采用前向算法计算序列 AGTT 出现概率;
(2). 计算观测 TATA 最可能的序列.
Solution: (1). 由于初始状态节点已经确定为"0"了, 所以初始状态的概率分布为

$$\boldsymbol{\pi} = (1,0,0,0,0,0)^{\mathrm{T}}$$

转移概率矩阵 $\boldsymbol{A}$ 和发射概率矩阵 $\boldsymbol{B}$(把 begin 和 end 也看做观测状态且放到 $\boldsymbol{B}$ 的最后两列) 分别为:

$$\boldsymbol{A} = \begin{pmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0.8 & 0 & 0 \\ 0 & 0 & 0.8 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0.4 & 0 & 0.6 \\ 0 & 0 & 0 & 0 & 0.1 & 0.9 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \boldsymbol{B} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0.4 & 0.1 & 0.2 & 0.3 & 0 & 0 \\ 0.4 & 0.1 & 0.1 & 0.4 & 0 & 0 \\ 0.2 & 0.3 & 0.3 & 0.2 & 0 & 0 \\ 0.1 & 0.4 & 0.4 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

α 递归计算的前向算法为:

$$\alpha_1(j) = \pi_j b_j(x_1), \alpha_t(j) = b_j(x_t) \cdot \sum_{i=1}^{N} a_{ij}\alpha_{t-1}(i), \text{其中} 1 \le j \le N$$

为叙述方便, 将状态"0,1,2,3,4,5"记作状态"1,2,3,4,5,6". 此时观测序列为 $\{\text{begin}, A, G, T, T, \text{end}\}$. 因此初始化 ($t = 1$ 时), 则有 $x_1 = \text{begin}$ 且:

$$\alpha_1(1) = \pi_1 b_1(x_1) = 1 \cdot 1 = 1, \alpha_1(2) = \alpha_1(3) = \alpha_1(4) = \alpha_1(5) = \alpha_1(6) = 0$$

当 $t = 2$ 时, 则有 $x_2 = A, \alpha_2(j) = b_j(A) \cdot \sum_{i=1}^{6} a_{ij}\alpha_1(i)$ 且:

$$\alpha_2(1) = 0, \alpha_2(2) = 0.2, \alpha_2(3) = 0.2, \alpha_2(4) = 0, \alpha_2(5) = 0, \alpha_2(6) = 0$$

当 $t = 3$ 时, 则有 $x_3 = G, \alpha_3(j) = \left(\sum_{i=1}^{6} \alpha_2(i) a_{ij}\right) b_j(G)$ 且:

$$\alpha_3(1) = 0, \alpha_3(2) = 0.008, \alpha_3(3) = 0.016, \alpha_3(4) = 0.048, \alpha_3(5) = 0.016, \alpha_3(6) = 0$$

当 $t=4$ 时, 则有 $x_4=T, \alpha_4(j)=\left(\sum_{i=1}^{6}\alpha_3(i)a_{ij}\right)b_j(T)$ 且:

$$\alpha_4(1)=0, \alpha_4(2)=0.00048, \alpha_4(3)=0.00512, \alpha_4(4)=0.00512, \alpha_4(5)=0.00048, \alpha_4(6)=0$$

当 $t=5$ 时, 则有 $x_5=T, \alpha_5(j)=\left(\sum_{i=1}^{6}\alpha_4(i)a_{ij}\right)b_j(T)$ 且:

$$\alpha_5(1)=0, \alpha_5(2)=0.0000288, \alpha_5(3)=0.00164, \alpha_5(4)=0.000486, \alpha_5(5)=0.000107, \alpha_5(6)=0$$

当 $t=6$ 时, 则有 $x_6=\text{end}, \alpha_6(j)=\left(\sum_{i=1}^{6}\alpha_5(i)a_{ij}\right)b_j(\text{end})$ 且:

$$\alpha_6(1)=\alpha_5(2)=\alpha_5(3)=\alpha_5(4)=\alpha_5(5)=0, \alpha_5(6)=0.000388$$

因此序列 $(x_1,x_2,x_3,x_4,x_5,x_6)=(\text{begin,A,G,T,T,end})$ 的出现概率为

$$p(x_1,x_2,x_3,x_4,x_5,x_6|\boldsymbol{A},\boldsymbol{B},\boldsymbol{\pi})=\sum_{j=1}^{6}\alpha_6(j)=0.000388$$

(2). 注意到观测序列为 $\{\text{begin}, T, A, T, A, \text{end}\}$, Viterbi 解码算法如下:

动态规划 : $V_1(j)=\pi_j b_j(x_1), \begin{cases} V_t(j)=b_j(x_t)\cdot \max\limits_{1\le i\le N}\{a_{ij}V_{t-1}(i)\} \\ \psi_t(j)=\arg\max\limits_{1\le i\le N}\{a_{ij}V_{t-1}(i)\} \end{cases}$,其中$1\le j\le N$

反向回溯 : $P^*=\max\limits_{1\le i\le N}V_T(i), \begin{cases} i_T^*=\arg\max\limits_{1\le i\le N}V_T(i) \\ i_t^*=\psi_{t+1}\left(i_{t+1}^*\right) \end{cases}$

- 当 $t=1$ 时, 已知 $x_1=\text{begin}$, 所以初始化如下:

$$V_1(1)=1, V_1(2)=V_1(3)=V_1(4)=V_1(5)=V_1(6)=0$$
$$\psi_1(1)=\psi_1(2)=\psi_1(3)=\psi_1(4)=\psi_1(5)=\psi_1(6)=0$$

- 当 $t=2$ 时, 已知 $x_2=T$, 所以有:

$$V_2(1)=0, V_2(2)=0.15, V_2(3)=0.2, V_2(4)=0, V_2(5)=0, V_2(6)=0$$
$$\psi_2(1)=1, \psi_2(2)=1, \psi_2(3)=1, \psi_2(4)=1, \psi_2(5)=1, \psi_2(6)=1$$

- 当 $t=3$ 时, 已知 $x_3=A$, 所以有:

$$V_3(1)=0, V_3(2)=0.012, V_3(3)=0.064, V_3(4)=0.024, V_3(5)=0.004, V_3(6)=0$$
$$\psi_3(1)=1, \psi_3(2)=2, \psi_3(3)=3, \psi_3(4)=2, \psi_3(5)=3, \psi_3(6)=1$$

- 当 $t=4$ 时, 已知 $x_4=T$, 所以有:

$$V_4(1)=0, V_4(2)=0.00072, V_4(3)=0.02048, V_4(4)=0.00192, V_4(5)=0.00128, V_4(6)=0$$
$$\psi_4(1)=1, \psi_4(2)=2, \psi_4(3)=3, \psi_4(4)=2, \psi_4(5)=3, \psi_4(6)=4$$

- 当 $t=5$ 时, 已知 $x_5=T$, 所以有:

$$V_5(1)=0, V_5(2)=5.76\times10^{-5}, V_5(3)=0.0065536, V_5(4)=0.0001536, V_5(5)=0.0004096, V_5(6)=0$$
$$\psi_5(1)=1, \psi_5(2)=2, \psi_5(3)=3, \psi_5(4)=4, \psi_5(5)=3, \psi_5(6)=5$$

- 当 $t=6$ 时, 已知 $x_6=\text{end}$, 所以有:

$$V_6(1)=0, V_6(2)=0, V_6(3)=0, V_6(4)=0, V_6(5)=0, V_6(6)=0.00036864$$
$$\psi_6(1)=1, \psi_6(2)=2, \psi_6(3)=3, \psi_6(4)=4, \psi_6(5)=3, \psi_6(6)=5$$

迭代过程终止, 且 $P^*=\max\limits_{1\le i\le4}V_T^i=V_6(6)=0.00036864, i_6^*=6$, 于是回溯过程如下:

$$i_6^*=\arg\max_{1\le i\le6}V_6(i)=6, i_5^*=\psi_6\left(i_6^*\right)=\psi_6(6)=5, i_4^*=\psi_5\left(i_5^*\right)=\psi_5(5)=3,$$
$$i_3^*=\psi_4\left(i_4^*\right)=\psi_4(3)=3, i_2^*=\psi_3\left(i_3^*\right)=\psi_3(3)=3, i_1^*=\psi_2\left(i_2^*\right)=\psi_2(3)=1$$

因此回溯得到的最优序列为 $1\to3\to3\to3\to5\to6$, 将标号对应回去则**真正的最优序列**为 $0\to2\to2\to2\to4\to5$, 且对应的概率值为 0.00036864.

接下来我们需要通过实验代码来验证上述计算的正确性, 因此我们需要先给出其伪代码:

Algorithm 1 HMM 状态序列解码的 Viterbi 算法

Input: 转移概率矩阵 $\boldsymbol{A}$, 发射概率矩阵 B, 初始概率分布 $\boldsymbol{\pi}$

Output: HMM 的最优路径

1: **Viterbi**$(\boldsymbol{A},\boldsymbol{B},\boldsymbol{\pi})$**:**
2: **for** $j\leftarrow1; j\le N; j\leftarrow j+1$ **do**
3: $\quad\psi_1(j)\leftarrow0;$
4: $\quad V_1(j)\leftarrow\pi_j b_j(x_1);$ ▷ $t=1$ 时的初始化, 其中 $b_j(x_t)$ 为状态节点 j 输出 x_t 的概率
5: **end for**
6: **for** $t\leftarrow2; t\le T; t\leftarrow t+1$ **do** ▷ 前向动态规划
7: $\quad$**for** $j\leftarrow1; j\le N; j\leftarrow j+1$ **do**
8: $\quad\quad V_t(j)=b_j(x_t)\cdot\max\limits_{1\le i\le N}\left\{a_{ij}V_{t-1}(i)\right\};$
9: $\quad\quad \psi_t(j)=\arg\max\limits_{1\le i\le N}\left\{a_{ij}V_{t-1}(i)\right\};$
10: $\quad$**end for**
11: **end for**
12: $P^*\leftarrow\max\limits_{1\le i\le N}V_T(i), i_T^*\leftarrow\arg\max\limits_{1\le i\le N}V_T(i);$ ▷ 初始化反向回溯
13: **for** $t\leftarrow T-1; t\ge1; t\leftarrow t-1$ **do**
14: $\quad i_t^*=\psi_{t+1}\left(i_{t+1}^*\right);$ ▷ 反向状态回溯
15: **end for**
16: **return** 最优路径 $\left\{i_1^*, i_2^*, \cdots, i_T^*\right\};$
17: **end {Viterbi}**

现在分析一下算法的时间复杂度[1], $T(N,T)=O(N)+O(T\cdot N\cdot N)+O(N)=O(T\cdot N^2)$, 显然是多项式时间内可计算的. 有了上述伪码, 则可以写出本题的 C++ 代码 (如后页所示):

[1] 伪码的 8,9,12 行需要一次 for 循环 (循环变量为 i) 来求出最大值和所在索引, 需要消耗 $O(N)$ 的时间.

```cpp
#include <bits/stdc++.h>
using namespace std;

int Idx(string x) {
    int res;
    if (x == "A") res = 0;
    else if (x == "C") res = 1;
    else if (x == "G") res = 2;
    else if (x == "T") res = 3;
    else if (x == "begin") res = 4;
    else if (x == "end") res = 5;
    return res;
}

vector<int> Viterbi( //Viterbi 动态规划解码算法
    vector<string> &X, vector<double> &pi,
    vector<vector<double>> &A, vector<vector<double>> &B
) {
    int T = X.size(), N = A.size(), M = B[0].size();
    vector<vector<double>> V(T, vector<double>(N, 0));
    vector<vector<int>> psi(T, vector<int>(N, 0));
    for(int j = 0; j < N; j++) {
        V[0][j] = pi[j] * B[j][Idx(X[0])];
        psi[0][j] = 0;
    }
    for(int t = 1; t < T; t++) {
        for(int j = 0; j < N; j++) {
            double temp = A[0][j] * V[t - 1][0];
            int idx = 0;
            for(int i = 1; i < N; i++) {
                if(A[i][j] * V[t - 1][i] > temp) {
                    temp = A[i][j] * V[t - 1][i];
                    idx = i;
                }
            }
            V[t][j] = B[j][Idx(X[t])] * temp;
            psi[t][j] = idx;
        }
    }
    double P = V[T - 1][0];
    vector<int> path(T);
    for(int i = 1; i < N; i++) {
        if(V[T - 1][i] > P) {
            P = V[T - 1][i];
            path[T - 1] = i;
        }
    }
    for(int t = T - 2; t >= 0; t--) {
        path[t] = psi[t + 1][path[t + 1]];
    }
    return path;
}
```

主函数编码如下:

```cpp
int main() {
    vector<string> X = {"begin", "T", "A", "T", "A", "end"};
    vector<double> pi = {1, 0, 0, 0, 0, 0};
    vector<vector<double>> A = {
        {0, 0.5, 0.5, 0, 0, 0},
        {0, 0.2, 0, 0.8, 0, 0},
        {0, 0, 0.8, 0, 0.2, 0},
        {0, 0, 0, 0.4, 0, 0.6},
        {0, 0, 0, 0, 0.1, 0.9},
        {0, 0, 0, 0, 0, 1}
    };
    vector<vector<double>> B = {
        {0, 0, 0, 0, 1, 0},
        {0.4, 0.1, 0.2, 0.3, 0, 0},
        {0.4, 0.1, 0.1, 0.4, 0, 0},
        {0.2, 0.3, 0.3, 0.2, 0, 0},
        {0.1, 0.4, 0.4, 0.1, 0, 0},
        {0, 0, 0, 0, 0, 1}
    };
    vector<int> path = Viterbi(X, pi, A, B);
    cout << " 最优路径为: " << endl;
    for(int i = 0; i < path.size(); i++) {
        cout << path[i] << " ";
    }
}
```

相应的输出为:

```
开始运行...
最优路径为: 0 2 2 2 4 5
运行结束
```

显然这与我们的手动计算结果是相同的, 证明了我们算法实现的正确性和手算的正确性.
并且我们可以利用 python 中的 hmmlearn 库来进行编码并输出最优序列, 具体代码如后页所示:

```python
# pip3 install hmmlearn
import numpy as np
from hmmlearn import hmm

states = ["0", "1", "2", "3", "4", "5"]
n_states = len(states)

observations = ["A", "C", "G", "T", "begin", "end"]
n_observations = len(observations)

start_probability = np.array([1, 0, 0, 0, 0, 0])

transition_probability = np.array([
    [0, 0.5, 0.5, 0, 0, 0],
    [0, 0.2, 0, 0.8, 0, 0],
    [0, 0, 0.8, 0, 0.2, 0],
    [0, 0, 0, 0.4, 0, 0.6],
    [0, 0, 0, 0, 0.1, 0.9],
    [0, 0, 0, 0, 0, 1]
])

emission_probability = np.array([
    [0, 0, 0, 0, 1, 0],
    [0.4, 0.1, 0.2, 0.3, 0, 0],
    [0.4, 0.1, 0.1, 0.4, 0, 0],
    [0.2, 0.3, 0.3, 0.2, 0, 0],
    [0.1, 0.4, 0.4, 0.1, 0, 0],
    [0, 0, 0, 0, 0, 1]
])

model = hmm.CategoricalHMM(n_components=n_states)
model.startprob_=start_probability
model.transmat_=transition_probability
model.emissionprob_=emission_probability

seen = np.array([[4, 3, 0, 3, 0, 5]]).T
logprob, box = model.decode(seen, algorithm="viterbi")
print(" 观测序列为:", ", ".join(map(lambda x: observations[x[0]], seen)))
print(" 最优路径为:", ", ".join(map(lambda x: states[x], box)))
```

上述代码的运行结果为:

```
观测序列为: begin, T, A, T, A, end
最优路径为: 0, 2, 2, 2, 4, 5
```

由此可见, 自行编写的算法跟 hmmlearn 算法库的运行结果是一致的.

Problem 6

请编写 **Problem 4** 中的 C++ 程序和 Python 程序, 并展示运行结果.

Solution: C++ 代码如下所示:

```cpp
#include <bits/stdc++.h>
using namespace std;

int Idx(string x) {
    int res;
    if (x == "Apple") res = 0;
    else if (x == "Orange") res = 1;
    return res;
}

vector<int> Viterbi( //Viterbi 动态规划解码算法
    vector<string> &X, vector<double> &pi,
    vector<vector<double>> &A, vector<vector<double>> &B
) {
    int T = X.size(), N = A.size(), M = B[0].size();
    vector<vector<double>> V(T, vector<double>(N, 0));
    vector<vector<int>> psi(T, vector<int>(N, 0));
    for(int j = 0; j < N; j++) {
        V[0][j] = pi[j] * B[j][Idx(X[0])];
        psi[0][j] = 0;
    }
    for(int t = 1; t < T; t++) {
        for(int j = 0; j < N; j++) {
            double temp = A[0][j] * V[t - 1][0];
            int idx = 0;
            for(int i = 1; i < N; i++) {
                if(A[i][j] * V[t - 1][i] > temp) {
                    temp = A[i][j] * V[t - 1][i];
                    idx = i;
                }
            }
            V[t][j] = B[j][Idx(X[t])] * temp;
            psi[t][j] = idx;
        }
    }
    double P = V[T - 1][0];
    vector<int> path(T);
    for(int i = 1; i < N; i++) {
        if(V[T - 1][i] > P) {
            P = V[T - 1][i];
            path[T - 1] = i;
        }
    }
    for(int t = T - 2; t >= 0; t--) {
        path[t] = psi[t + 1][path[t + 1]];
    }
    return path;
}
```

主函数编码如下:

```cpp
int main() {
    vector<string> X = {"Apple", "Apple", "Orange", "Orange", "Orange"};
    vector<double> pi = {1.0 / 3, 1.0 / 3, 1.0 / 3};
    vector<vector<double>> A = {
        {1.0 / 3, 1.0 / 3, 1.0 / 3},
        {1.0 / 3, 1.0 / 3, 1.0 / 3},
        {1.0 / 3, 1.0 / 3, 1.0 / 3}
    };
    vector<vector<double>> B = {
        {1.0 / 2, 1.0 / 2},
        {3.0 / 4, 1.0 / 4},
        {1.0 / 4, 3.0 / 4}
    };
    vector<int> path = Viterbi(X, pi, A, B);
    cout << " 最优路径为: " ;
    for(int i = 0; i < path.size(); i++) {
        cout << path[i] + 1 << " ";
    }
}
```

代码输出结果为 (即最优序列为盒子 2→ 盒子 2→ 盒子 3→ 盒子 3→ 盒子 3):

```
开始运行...
最优路径为: 2 2 3 3 3
运行结束
```

Problem 4 和 **Problem 5** 的 α 前向递归算法的 C++ 代码如下所示:

```cpp
double Forward_Alg( //alpha 前向递归算法
    vector<string> &X, vector<double> &pi,
    vector<vector<double>> &A, vector<vector<double>> &B
) {
    int T = X.size(), N = A.size(), M = B[0].size();
    vector<vector<double>> alpha(T, vector<double>(N, 0));
    for(int j = 0; j < N; j++) {
        alpha[0][j] = pi[j] * B[j][Idx(X[0])];
    }
    for(int t = 1; t < T; t++) {
        for(int j = 0; j < N; j++) {
            double temp = 0;
            for(int i = 0; i < N; i++) {
               temp += A[i][j] * alpha[t - 1][i];
            }
            alpha[t][j] = B[j][Idx(X[t])] * temp;
        }
    }
    return accumulate(alpha[T - 1].begin(), alpha[T - 1].end(), 0.0);
}
```

主函数中添加如下代码并获得输出 (显然跟手算结果是一样的, 因此代码正确):

```
vector<string> S = {"begin", "A", "G", "T", "T", "end"};
double prob = Forward_Alg(S, pi, A, B);
cout << " 序列 S 的出现概率为" << prob << endl;
开始运行...
序列 S 的出现概率为 0.00038832
运行结束
```

同时, Viterbi 解码算法的 Python 代码如下:

```python
import numpy as np
from hmmlearn import hmm

states = [" 盒子 1", " 盒子 2", " 盒子 3"]
n_states = len(states)

observations = ["Apple", "Orange"]
n_observations = len(observations)

start_probability = np.array([1.0/3, 1.0/3, 1.0/3])

transition_probability = np.array([
    [1.0/3, 1.0/3, 1.0/3],
    [1.0/3, 1.0/3, 1.0/3],
    [1.0/3, 1.0/3, 1.0/3]
])

emission_probability = np.array([
    [1.0/2, 1.0/2],
    [3.0/4, 1.0/4],
    [1.0/4, 3.0/4]
])

model = hmm.CategoricalHMM(n_components=n_states)
model.startprob_=start_probability
model.transmat_=transition_probability
model.emissionprob_=emission_probability

seen = np.array([[0, 0, 1, 1, 1]]).T
logprob, box = model.decode(seen, algorithm="viterbi")
print(" 观测序列为:", ", ".join(map(lambda x: observations[x[0]], seen)))
print(" 最优路径为:", ", ".join(map(lambda x: states[x], box)))
```

代码输出为:

```
观测序列为: Apple, Apple, Orange, Orange, Orange
最优路径为: 盒子 2, 盒子 2, 盒子 3, 盒子 3, 盒子 3
```