基于隐马尔可夫模型的天气预测算法

智能 2002　　王明强

202005100207

日期：May 15, 2023

摘　要

本文简要介绍了隐马尔科夫模型的研究进展，并介绍了隐马尔科夫模型的基本概念、训练算法与预测算法，其中，以 Baum-Welch 算法作为训练算法，以 Viterbi 算法作为预测算法。在这些算法的基础上，本文选取西安市连续二十天的天气情况作为训练数据，预测了未来 5 天的天气情况。最后，本文总结了这次实践的经验

关键词： 隐马尔可夫模型，Baum-Welch 算法，Viterbi 算法

1　引言

隐马尔可夫模型（Hidden Markov Model, HMM）是一种常见的统计模型，它建立在马尔可夫过程的基础上，并被广泛应用于语音识别、自然语言处理、信号处理、生物信息学、图像识别和行为识别等领域。近年来，HMM 在以下几个方面有了不少研究进展：

1. 深度学习中的应用

随着深度学习的兴起，研究者们开始将 HMM 与深度学习模型结合，这些模型与传统基于贝叶斯算法的 HMM 模型相比，具有更高的准确性和更好的鲁棒性。例如，深度置信网络（Deep Belief Network, DBN）可以用于提高 HMM 的特征提取和分类性能与总结。。

2. 参数估计的新方法

基于最小二乘和最小二乘法的 HMM 参数估计方法已经被广泛使用。在这些方法中，参数估计被转换为一个优化问题，并使用非线性最小二乘或非线性规划算法求解。这些方法可以在处理大规模数据时加快计算速度，提高模型准确性。

3. 非标准 HMM 模型的研究

研究者们开始探索各种非标准 HMM 模型，例如混合隐马尔可夫模型（Mixture HMM）和动态变化 HMM（Time-varying HMM），这些模型在某些应用场景中具有更高的灵活性和更好的预测性能。

4. 应用场景的扩展

HMM 已经被广泛应用于语音识别、自然语言处理、信号处理、生物信息学、图像识别和行为识别等领域。近年来，HMM 在其他领域的应用也越来越受到关注，例如金融、交通和能源等领域

2　隐马尔可夫模型

2.1　基本概念

隐马尔科夫模型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从而产生观测随机序列的过程，隐藏的马尔科夫链随机生成的状态

的序列，称为状态序列；每个状态生成一个规则，而由此产生的观测的随机序列称为观测序列。序列的每一个位置又可以看作是一个时刻。

HMM 的形式定义如下：

设 Q 是所有可能的状态集合，V 是所有可能的观测的集合：

$$Q = \{q_1, q_2, \cdots, q_N\}, V = \{v_1, v_2, \cdots, v_M\} \tag{2.1}$$

其中，N 是可能的状态数，M 是可能的观测数。

设 I 为长度为 T 的状态序列，O 是对应的观测序列：

$$I = \{i_1, i_2, \cdots, i_T\}, V = \{v_1, v_2, \cdots, v_T\} \tag{2.2}$$

设状态转移概率矩阵为 A：

$$A = [a_{ij}]_{N \times N} \tag{2.3}$$

其中，$i_{ij} = P(i_{t+1} = q_j \mid i_t = q_i), i = 1, 2, \cdots, N; j = 1, 2, \cdots, N$，表示在 t 时刻处于状态 q_i 条件下在时刻 $t+1$ 转移到状态 q_j 的概率。

观测概率矩阵为 B:

$$B = [b_j(k)]_{N \times M} \tag{2.4}$$

其中，$b_j(k) = P(o_t = v_k \mid i_t = q_j), k = 1, 2, \cdots, M; j = 1, 2, \cdots, N$，表示在时刻 t 处于状态 q_j 的条件下生成的观测 v_k 的概率。π 是初始态概率向量：

$$\pi = (\pi_i) \tag{2.5}$$

其中，$\pi_i = P(i_1 = q_i), i = 1, 2, \cdots, N$

隐马尔可夫模型由初始状态概率向量 π、状态转移矩阵 A 和观测概率矩阵 B 决定，初始状态概率向量 π、状态转移矩阵 A 决定了状态序列，观测概率矩阵 B 决定了观测序列。因此，隐马尔科夫模型 λ 如下：

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

A、B、π 称为隐马尔科夫模型的三要素。状态转移概率矩阵 A 与初始状态概率向量 π 确定了隐藏的马尔科夫链，生成不可观测的状态序列。观测概率矩阵 B 确定了如何从状态生成观测，与状态序列综合确定了如何产生观测序列。

2.2 基本假设

1. 齐次马尔可夫性假设

假设隐藏的马尔可夫链在任意时刻 t 的状态仅依赖前一时刻的状态，与七态状态及观测无关，也与时刻 t 无关，即：

$$P(i_t \mid i_{t-1}, o_{t-1}, \cdots, i_1, o_1) = P(i_t \mid i_{t-1}), t = 1, 2, \cdots, T \tag{2.6}$$

2. 观测独立性假设假设任意时刻的观测只依赖于该时刻的马尔科夫链状态，与其他观测和状态无关。

3 隐马尔科夫模型的训练与预测

3.1 前向算法

给定隐马尔可夫模型 λ 定义到时刻 t 部分观测序列为 $o_1, o_2, \cdots, o_t$，且状态为 q_i 的概率为前向概率，记作

$$\alpha_t(i) = P(o_1, o_2, \cdots, o_t, i_t = q_i \mid \lambda) \tag{3.1}$$

可以递推的计算出前向概率 $\alpha_t(i)$ 以及观测序列概率 $P(O \mid \lambda)$

算法

输入：λ、O

1. 初值：$\alpha_1(i) = \pi_i b_i(o_1), i = 1, 2, \cdots, N$

2. 迭代：

$$\alpha_{t+1}(i) = \left[\sum_{j=1}^{N} \alpha_t(j) a_{ji}\right] b_i(o_{t+1}), i = 1, 2, \cdots, N \tag{3.2}$$

3. 输出：

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{3.3}$$

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

3.2 后向算法

给定隐马尔可夫模型 λ，定义到时刻 t 状态为 q_i 的条件下，从 $t+1$ 到 T 的部分观测序列为 $o_1, o_2, \cdots, o_t$ 的概率为后向概率，记作

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \cdots, o_T \mid i_t = q_i, \lambda) \tag{3.4}$$

递推计算求得后向概率 $\beta_t(i)$ 与观测序列概率 $P(O \mid \lambda)$

算法

输入：λ、O

1. 初值：$\beta_T(i) = 1, i = 1, 2, \cdots, N$

2. 对 $t = T-1, T-2, \cdots, 1$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), i = 1, 2, \cdots, N \tag{3.5}$$

3. 输出：

$$P(O \mid \lambda) = \sum_{i=1}^{N} \pi_i b_i(o_1) \beta_1(i) \tag{3.6}$$

3.3 概率与期望值计算

利用前向概率与后向概率，可以得到关于单个状态和两个状态概率公式。给定模型 λ 和观测 O，在时刻 t 处于状态 q_i 的概率，记为

$$\gamma_t(i) = P(i_t = q_i \mid O, \lambda) \tag{3.7}$$

由定义可知：

$$\alpha_t(i)\beta_t(i) = P(i_t = q_i, O \mid \lambda) \tag{3.8}$$

由此得到

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \tag{3.9}$$

给定模型 λ 和观测 O，在时刻 t 处于状态 q_i 且在时刻 $t+1$ 处于状态 q_j 的概率，记为：

$$\xi_t(i, j) = P\left(i_t = q_i, i_{t+1=q_j} \mid O, \lambda\right) \tag{3.10}$$

通过前向概率与后向概率计算：

$$\begin{aligned} \xi_t(i, j) &= \frac{P\left(i_t = q_i, i_{t+1=q_j}, O \mid \lambda\right)}{P(O \mid \lambda)} \\ &= \frac{P\left(i_t = q_i, i_{t+1=q_j}, O \mid \lambda\right)}{\sum_{i=1}^{N} \sum_{j=1}^{N} P\left(i_t = q_i, i_{t+1=q_j}, O \mid \lambda\right)} \end{aligned} \tag{3.11}$$

3.4 Baum-Welch 算法

Baum-Welch 模型参数估计公式：

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{3.12}$$

$$b_j(k)^{(n+1)} = \frac{\sum_{t=1,o_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{3.13}$$

$$\pi_i = \gamma_1(i) \tag{3.14}$$

算法：

1. 输入观测数据：$O = (o_1, o_2, \cdots, o_T)$

2. 初始化：对 $n = 0$，选取 $a_{ij}^{(0)}, b_j(k)^{(0)}, \pi_i^{(0)}$，得到模型 $\lambda^{(0)} = \left(A^{(0)}, B^{(0)}, \pi^{(0)}\right)$

3. 开始迭代：对 $n = 1, 2, \cdots$:

$$a_{ij}^{(n+1)} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{3.15}$$

$$b_j(k)^{(n+1)} = \frac{\sum_{t=1,o_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{3.16}$$

$$\pi_i^{(n+1)} = \gamma_1(i) \tag{3.17}$$

右端各值按观测 $O = (o_1, o_2, \cdots, o_T)$ 和模型 $\lambda^{(n)} = \left(A^{(n)}, B^{(n)}, \pi^{(n)}\right)$

3. 输出：模型参数 $\lambda^{(n+1)} = \left(A^{(n+1)}, B^{(n+1)}, \pi^{(n+1)}\right)$

3.5 Viterbi 算法

维特比算法（Viterbi）实际是用动态规划解隐马尔科夫模型预测问题，即用动态规划求概率最大路径（最优）。这时一条路径对应着一个状态序列。

算法流程：

1. 输入：λ、O

2. 初始化：

$$\delta_1(i) = \pi_i b_i\left(o_1\right), i = 1, 2, \cdots, N \tag{3.18}$$

3. 开始迭代：

$$\begin{aligned} \delta_t(i) &= \max_{1 \le j \le N} \left[\delta_{t-1}(j) a_{ji}\right] b_i\left(o_t\right), i = 1, 2, \cdots, N \\ \Psi_t(i) &= \arg \max_{1 \le j \le N} \left[\delta_{t-1}(j) a_{ji}\right], i = 1, 2, \cdots, N \end{aligned} \tag{3.19}$$

4. 输出：

$$\begin{aligned} P^* &= \max_{1 \le j \le N} \delta_T(i) \\ i_T^* &= \arg \max_{1 \le j \le N} [\delta_T(i)] \end{aligned} \tag{3.20}$$

4. 最优路径回溯：

对 $t = T-1, T-2, \cdots, 1,$

$$i_t^* = \Psi_{t+1}(i_{t+1}^*) \tag{3.21}$$

5. 输出：最优路径 $I^* = (i_1^*, i_2^*, \cdots, i_T^*)$

4 基于隐马尔科夫模型的天气预测实例

4.1 数据准备

选取西安市近 20 天的天气，由于天气情况复杂，为简化模型，将天气分为晴、雨、多云、阴四种天气状况，如下表：

表 1: 西安近 20 天天气

日期	天气情况	日期	天气情况
04-01	多云	04-06	晴
04-02	雨	04-07	阴
04-03	雨	04-08	晴
04-04	阴	04-09	多云
04-05	雨	04-10	阴
04-11	多云	04-16	晴
04-12	多云	04-17	多云
04-13	雨	04-18	多云
04-14	多云	04-19	多云
04-15	多云	04-20	多云

根据经验，设初始状态概率为：

$$\pi = \Big(0.25, 0.25, 0.25, 0.25\Big)^T$$

状态转移矩阵为：

$$A = \begin{pmatrix} 0.25 & 0.25 & 0.25 & 0.3 \\ 0. & 0.3 & 0.35 & 0.35 \\ 0.2 & 0.4 & 0.25 & 0.2 \\ 0.3 & 0.2 & 0.25 & 0.25 \end{pmatrix}$$

状态观测矩阵为：

$$B = \begin{pmatrix} 0.25 & 0.25 & 0.25 & 0.3 \\ 0 & 0.5 & 0.25 & 0.25 \\ 0 & 0.4 & 0.45 & 0.2 \\ 0.3 & 0.2 & 0.25 & 0.25 \end{pmatrix}$$

4.2 实验结果

利用 python 编程，对之后五天的天气情况进行预测，并与实际天气进行比较，得到以下结果：

表 2: 实验结果

日期	04-21	04-22	04-23	04-24	04-25
预测天气	雨	雨	雨	阴	雨
实际天气	雨	雨	雨	阴	多云

从表中可以看出在 5 天的预测中，有一天的实际天气与预测结果不符。这说明预测的准确率较低，需要更多准确的数据来提升准确率。其次，本文中所用的参数为笔者经验所设置，并未在大量数据的基础上计算得到，因此也导致了预测结果的不准确。

5 经验与总结

通过本次的实践，笔者得到些许经验：

1. 数据处理十分重要。在隐马尔可夫模型的实验中，数据的质量对模型的好坏产生非常重要的影响，它影响了参数的设置与模型的训练。因此，要仔细审查原始数据的质量，采用合适的方法对数据进行处理。

2. 特征工程是提升模型精度的关键。样本集的特征影响着模型的训练和测试过程，因此必须对不同特征进行测试，挑选出与模型最佳兼容的特征。特征的选择可以采用相关系数、信息增益、卡方检验等统计方法进行选择。在天气预测中，天气的状态可以增加更为丰富的内容，如当日的湿度、空气质量指数、温度等，天气变化是一个非常复杂的变化过程，需要更多的数据进行训练以及更多的状态观测等。

3. 模型调参是重要的过程。隐马尔可夫模型中有多个参数可供调整，例如隐藏状态数目、初始概率、转移概率、发射概率等。因此，在建模过程中需要不断调整参数，以获取最佳的模型。

6 参考文献

[1] 谢超成. 基于隐马尔科夫模型及多类映射的人脸识别 [D]. 电子科技大学,2011.

[2] 杉山将. 统计机器学习 [M]. 机械工业出版社, 2018.

附录 A　代码实现

Listing 1: main.py

```
import numpy as np
from HMM import HMM

# 随机生成二十天的天气数据
weather_data = np.array([3, 1, 1, 2, 1, 0, 2, 0, 3, 2, 3, 3, 1, 3, 3, 0, 3, 3, 3,
    3])
print(weather_data)
# 初始化初始隐状态概率
startprob = np.array([0.25, 0.25, 0.25,0.25])

# 初始化状态转移矩阵
transmat = np.array([[0.25, 0.25, 0.25, 0.3],
                     [0.0, 0.3,0.35, 0.35],
                     [0.2, 0.4, 0.25, 0.2],
                     [0.3, 0.2, 0.25,0.25]])
observation = np.array([[0.25, 0.25, 0.25, 0.3],
                     [0, 0.5,0.25, 0.25],
                     [0, 0.4, 0.45, 0.2],
                     [0.3, 0.2, 0.25,0.25]])
# 构造 HMM 模型
model = HMM(num_states=4, num_obs=4, pi=startprob, A=transmat, B=observation)

# 使用 Baum-Welch 算法对模型参数进行训练
model.baum_welch_train(weather_data)

# 预测未来五天的天气情况
future_data = np.array([1, 1, 1, 2, 3]) # 未来 5 天都是晴天
predicted_weather = model.viterbi_predict(future_data)

# 将状态转换成具体的天气情况
weather_dict = {0: "晴天", 1: "雨天", 2:'阴天', 3:'多云'}
predicted_weather_text = [weather_dict[status] for status in predicted_weather]

# 输出预测结果
print("未来五天的天气情况预测为: ", predicted_weather_text)
```

Listing 2: HMM.py

```
import numpy as np


class HMM:
    def __init__(self, num_states, num_obs, pi=None, A=None, B=None):
        self.num_states = num_states
```

```
        self.num_obs = num_obs

        if pi is None:
            self.pi = np.ones(num_states) / num_states
        else:
            self.pi = pi

        if A is None:
            self.A = np.ones((num_states, num_states)) / num_states
        else:
            self.A = A

        if B is None:
            self.B = np.ones((num_states, num_obs)) / num_obs
        else:
            self.B = B

    def forward(self, obs):
        T = len(obs)
        alpha = np.zeros((T, self.num_states))
        alpha[0, :] = self.pi * self.B[:, obs[0]]
        for t in range(1, T):
            alpha[t, :] = np.dot(alpha[t - 1, :], self.A) * self.B[:, obs[t]]
        return alpha

    def backward(self, obs):
        T = len(obs)
        beta = np.zeros((T, self.num_states))
        beta[T - 1, :] = 1
        for t in range(T - 2, -1, -1):
            beta[t, :] = np.dot(self.A, self.B[:, obs[t + 1]] * beta[t + 1, :])
        return beta

    def baum_welch_train(self, obs, num_iters=100):
        T = len(obs)
        for n in range(num_iters):
            alpha = self.forward(obs)
            beta = self.backward(obs)
            xi = np.zeros((T - 1, self.num_states, self.num_states))
            for t in range(T - 1):
                xi[t, :, :] = self.A * np.outer(alpha[t, :], beta[t + 1, :] * self.
                    B[:, obs[t + 1]])
                xi[t, :, :] /= np.sum(xi[t, :, :])
            self.A = np.sum(xi, axis=0) / np.sum(xi, axis=(0, 1)).reshape(-1, 1)
            gamma = alpha * beta / np.sum(alpha * beta, axis=1).reshape(-1, 1)
            self.pi = gamma[0, :]
            self.B = np.zeros((self.num_states, self.num_obs))
```

```
            for i in range(self.num_states):
                for k in range(self.num_obs):
                    mask = (obs == k)
                    self.B[i, k] = np.sum(gamma[mask, i]) / np.sum(gamma[:, i])

    def viterbi_predict(self, obs):
        T = len(obs)
        delta = np.zeros((T, self.num_states))
        psi = np.zeros((T, self.num_states), dtype=int)
        delta[0, :] = self.pi * self.B[:, obs[0]]
        for t in range(1, T):
            for j in range(self.num_states):
                aux = delta[t - 1, :] * self.A[:, j]
                delta[t, j] = np.max(aux) * self.B[j, obs[t]]
                psi[t, j] = np.argmax(aux)
        path = np.zeros(T, dtype=int)
        path[T - 1] = np.argmax(delta[T - 1, :])
        for t in range(T - 2, -1, -1):
            path[t] = psi[t + 1, path[t + 1]]
        return path
```

附录 B　程序运行结果

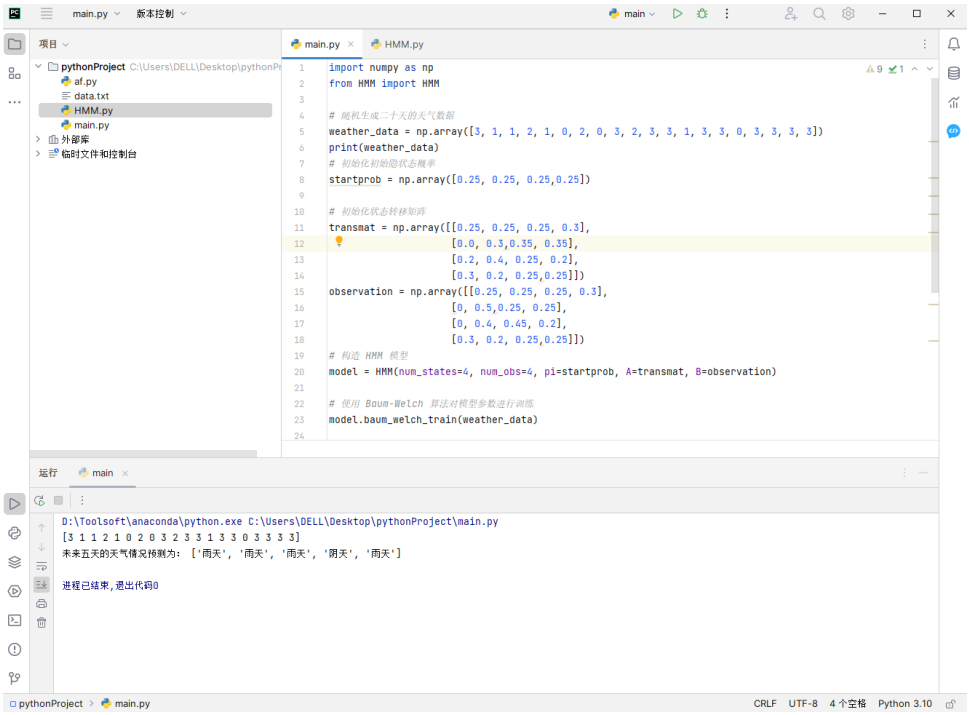

图 1: 程序运行结果