关于 HMM 的模式识别学习报告

何峙　21215512 大数据与人工智能

引言

模式识别这门课，从最开始的决策论，再到极大似然估计、贝叶斯估计、非参数估计等各种估计技术，归根到底，无不源于贝叶斯公式：

$$P\left(w_i \mid \varphi\right)=\frac{P\left(p \mid w_i\right) \cdot P\left(w_i\right)}{\sum_j \cdot P\left(p \mid w_j\right) P\left(w_j\right)}$$

而隐马尔可夫模型（以下简称 HMM）是学习这么课程目前为止，对贝叶斯理论诠释得最综合的一个模型实例，是一种动态的贝叶斯网络，它也是令本人醍醐灌顶、感受最深的技术。特地深入学习，以此作为该课程学习的阶段性总结。

HMM 的一些研究现状

HMM 的现实应用研究相当广泛，尤其适合需要处理序列问题的场景，从语音识别、自然语言处理等基础学科，再到上层的金融、生物、机械工程、网络工程等应用领域均有涉及。

华南理工大学的萧超武[1]等提出的一种基于双层 HMM 的驾驶模式识别方法，认为驾驶行为是一系列驾驶意图的先后序列，利用传感器数据作为最初的观测数据输入到下层 HMM，训练出表征隐藏状态的驾驶操作序列（如猛加油、滑行、急刹车，等），再将这些驾驶操作序列作为最为上层 HMM 观测序列的输入，训练出表征上层隐藏序列的驾驶意图（准备、操作、恢复等状态），最后得出什么样的驾驶模式能更有效节能的结论，十分具有现实意义。

大连理工大学的瞿晓娟[2]等研究使用 HMM 构建驾驶疲劳识别模型。首先提取人体生物电信号——EGG，然后提取 EGG 的各个能量比特征指标作为观测序列，将隐藏状态分为“清醒”和“疲劳”两种类别，利用 Baum-Welch 算法对 HMM 进行训练，识别出驾驶人的疲劳状态水平，进一步完善了疲劳驾驶识别的工具集。

HMM 理论回顾

关于 HMM 的定义

HMM 是在马尔可夫链的基础上发展起来的，是一种序列生成模型。定义观测序列集合为 O = {o1, o2, o3, …, oM}，隐藏状态集为 S = {s1, s2, …, sN}，那么 HMM 的序列结构类似于概率图，如 Fig.1 所示。

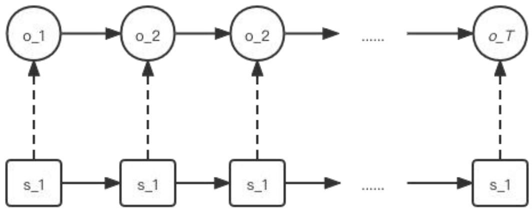


Fig.1 HMM 序列基本结构

Fig.1 的第一行横向箭头表示观测序列的生成，第二行横向箭头表示隐藏状态之间的转移，虚线箭头表示当前状态产生某个观测。

接着定义 HMM 的参数$\theta = \{A, B, \pi\}$，其中：

- A 为概率转移概率矩阵：
  $$A = [a_{ij}]_{N \times N},\ a_{ij} = P(i_{t+1} = sj \mid i_t = si)$$
  表示由状态 i 转移到状态 j 的概率，其中 i, j = 1, 2, …, N
- B 为观测生成概率矩阵：
  $$B = [b_j(k)]_{N \times M},\ b_j(k) = P(v_t = ok \mid i_t = sj)$$
  其中 k = 1, 2, …, M，而 j = 1, 2, …, N
- π为初始状态向量：
  $$\pi_i = P(i_1 = si),\quad \text{其中 } i = 1, 2, \dots, N$$

同时，HMM 作了两个基本假设：

1. 任意时刻 t 的状态仅取决于其前一时刻的状态，即：
   $$P(i_t \mid i_{t-1}, v_{t-1}, i_{t-2}, v_{t-2}, \dots, i_1, v_1) = P(i_t \mid i_{t-1})$$
2. 任意时刻的观测生成只取决于该时刻的状态，即：
   $$P(v_t \mid i_{t-1}, v_{t-1}, i_{t-2}, v_{t-2}, \dots, i_1, v_1) = P(v_t \mid i_t)$$

研究 HMM 的三个基本问题：

1. 估值问题

已知模型参数，给定观测序列，计算产生这个序列的概率。如果使用穷举法，其时间复杂度将达到 $O(T * N^T)$，其中 T 为序列长度，N 为状态数，计算量太大，一般不采用。使用前向算法或后向算法，可使时间复杂度降为 $O(T * N^2)$。

现以说明前向算法为例。先定义前向概率$\alpha_t(i)$：在已知模型参数条件下，在 t 时刻观察序列为 $v_1,v_2,v_3,\ldots,v_t$ 且此时状态为 si 的概率，即$\alpha_t(i) = P(v_1,v_2,v_3,\ldots,v_t,i_t = si \mid \theta)$

那么，前向算法过程可描述如下[3]：

输入：HMM 参数θ，观察序列 O

输出：观测序列概率 P(O|θ)

---

1. 初始化：t=1 时刻的前向概率：$\alpha_1(i)$
2. 迭代，对 t=1,2,…,T-1,求：

$$\alpha_{t+1}(i)=\left[\sum_{j=1}^{N} \partial_t(j) a_{ij}\right] b_i\left(V_{t+1}\right)$$

3. 终止：

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

后向算法类似，可参考本课程课件，这里不再赘述。

2. 解码问题

已知模型参数，给定观测序列，求这个观测序列对应的隐藏状态序列。解决此问题多使用 Viterbi 算法，它是基于一个动态规划算法，假设隐藏序列是一条最优路径（概率最大），那么其子路径（在 HMM 中即两个状态之间的转移路径）也是最优的。从 t=1 开始递推计算状态为 si 的各条分路径的最大概率，每个时刻选择出概率最大的路径，然后回溯，将路径上各个结点连接起来即得到最优路径，这条路径即为隐藏状态序列。具体算法描述可参考李航的《统计学习方法》第二版[3]。

3. 学习问题

此问题要求解 HMM 的参数。分成两种情况：

（1）给定观测序列及对应的隐藏状态序列。此时只需要进行监督学习即可，使用极大似然估计即可求得参数，如求状态转移概率:

$$\hat{a_{ij}}=\frac{|c_{ij}|}{\sum_{j=1}^{N}|c_{ij}|}$$

其中 c_{ij} 为出现状态 i 转移到状态 j 这种情况的频次。

（2）只给定观测序列，对应的隐藏状态序列未知。这时可采用 Baum-Welch 算法。该算法本质上是 EM 算法。指定参数的初始值θ_0，将其和观测序列代入 Q 函数，然后最大化这个函数，求得一组参数θ^*_k，使得 $P(O \mid \theta^*_k) > P(O \mid \theta^*_{k-1})$，不断迭代（其中 k 是迭代的轮次），使得模型参数收敛。具体算法实现过程可参考李航的《统计学习方法》第二版[3]。

HMM 上机实验

现以一个中文分词应用验证 HMM 的三个基本问题。假如现有一句话（字序列）：

“人们常说生活是一部教科书”，

已作分词如下：

['人们', '常', '说', '生活', '是', '一', '部', '教科书'],

定义状态集 S = {0, 1}，0 表示非终结字，1 表示终结字，那么对句子分词实际上就是找到句子中标记为 1 的那个字，然后切分即可：

“人/0 们/1 常/1 说/1 生/0 活/1 是/1 一/1 部/1 教/0 科/0 书/1”，

可将此表示为 Fig.2 的序列结构。

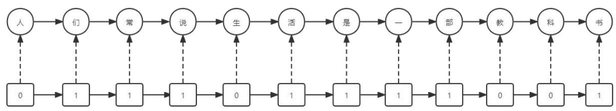
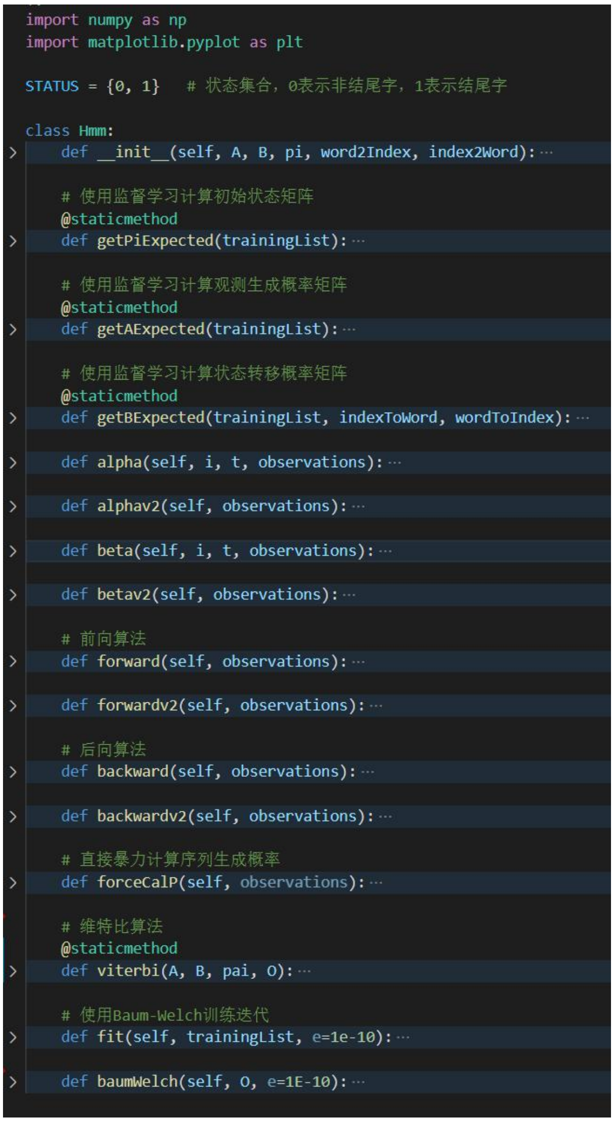


Fig.2 用 HMM 模型表示一句中文的分词结构

现通过编写代码方式验证这三个问题[1]。

---

[1] 相关代码：https://github.com/kevinva/hmmmmmmmmmm



```
import numpy as np
import matplotlib.pyplot as plt

STATUS = {0, 1}   # 状态集合，0表示非结尾字，1表示结尾字

class Hmm:
    def __init__(self, A, B, pi, word2Index, index2Word):···

    # 使用监督学习计算初始状态矩阵
    @staticmethod
    def getPiExpected(trainingList):···

    # 使用监督学习计算观测生成概率矩阵
    @staticmethod
    def getAExpected(trainingList):···

    # 使用监督学习计算状态转移概率矩阵
    @staticmethod
    def getBExpected(trainingList, indexToWord, wordToIndex):···

    def alpha(self, i, t, observations):···

    def alphav2(self, observations):···

    def beta(self, i, t, observations):···

    def betav2(self, observations):···

    # 前向算法
    def forward(self, observations):···

    def forwardv2(self, observations):···

    # 后向算法
    def backward(self, observations):···

    def backwardv2(self, observations):···

    # 直接暴力计算序列生成概率
    def forceCalP(self, observations):···

    # 维特比算法
    @staticmethod
    def viterbi(A, B, pai, O):···

    # 使用Baum-Welch训练迭代
    def fit(self, trainingList, e=1e-10):···

    def baumWelch(self, O, e=1E-10):···
```

Fig.3 HMM 代码图示

1. 学习问题

Table 1 由监督学习得出 HMM 的参数

π	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$
A	$\begin{bmatrix} 0.25 & 0.75 \\ 0.2857149 & 0.71428571 \end{bmatrix}$
B	$\begin{bmatrix} 0.25 & 0 & 0 & 0 & 0.25 & 0 & 0 \\ 0 & 0.125 & 0.125 & 0.125 & 0 & 0.125 & 0.125 \end{bmatrix}$

Table 2 前向算法和后向算法计算结果

	前向算法	后向算法
P(O | θ)	3.7272e-13	3.7272e-13

Table 3 估值问题中计算方式分别使用递归运算和矩阵计算的对比

	递归运算	矩阵运算
P(O | θ)	3.7272e-13	3.7272e-13
	0.010911 秒	0.000167 秒

Table 2 为分别使用前向算法和后向算法的计算结果，计算结果是一致的。

代码中分别使用了两种方式计算前向概率/后向概率：递归和矩阵运算。使用递归运算时，发现观测序列越长，计算速度越慢，譬如计算一个观测序列长度为 25 的前向概率，用递归方法，计算耗时竟然达到 48 秒之久。而改用 numpy 库的矩阵运算，计算速度有十分明显的提升。从 Table 3 可以看到，计算本实验的句子序列，使用矩阵运算比递归运算快几乎 66 倍，而两种方法的计算结果也是一致的。

2. 解码问题

Fig.4 使用 Viterbi 算法计算状态序列结果

使用 Table 1 的参数进行解码问题，得出结果如图 Fig.4 所示。对比 Fig.2，分词结果是一致的。

3. 学习问题

将句子“人们常说生活是一部教科书”直接输入到 HMM 模型中用 Baum-Welch 算法进行学习。

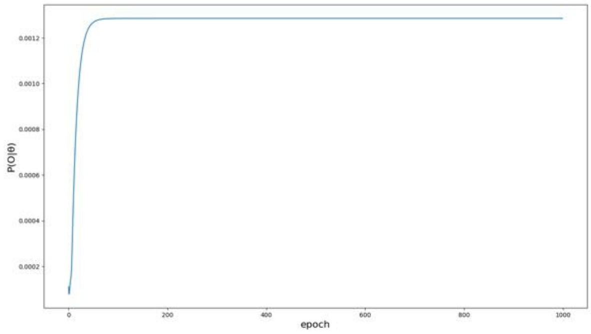


Fig.5 P(O | θ)不断迭代的值

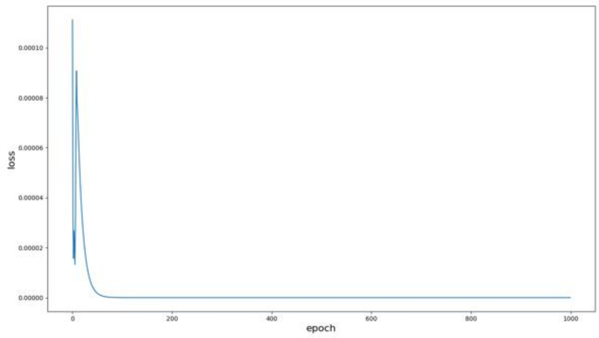


Fig.6 训练损失趋势

Fig.5 表示观测序列的概率随着模型参数不断迭代而不断增大，最终逼近一个概率最大值约为 0.0013，符合 EM 算法的表现形式，形象说明 Baum-Welch 算法本质上就是 EM 算法。

Fig.6 是参数迭代的损失趋势，Loss 的计算方式为：

$$\mathrm{Loss} = |P(O|\theta^*_k) - P(O|\theta^*_{k-1})|,$$

其中 k 为迭代轮数

跟 Fig.5 表示的结果是一致的，算法迭代大概 100 轮时已经收敛。

总结与展望

学习本课程，收获最大的不仅仅是一个个令人醍醐灌顶的算法，更重要的是对算法推导过程引起的思考。人工智能最核心的是模型的可解析性，而不是只弄懂表面的总体框架后就盲目猜测，盲目训练、调参。正如以上试验中文分词，为什么会有如此分词结果？它背后是贝叶斯理论的支撑，而实现这套理论就要通过动态规划、EM 算法等方式。另外，本课程也提供了不少解决问题的方法论，例如 HMM 的估值问题，有很多方法可以解决，如穷举法，一个个解去试，直到找到最好，但时间复杂度太高，实用性不强。这时不妨转变一下思路，使用迭代法，复杂度就降下来了，找到次优解或局部最优解也可以解决问题。

目前，关于 HMM 的学习上，本人还有几个问题有待解决：

1. 以上实验只是对一个样本进行学习，如果有多个训练观测序列，是否还能照搬 Baum-Welch 算法解决？
2. 进行分词时，观测序列可能很长，若每个汉字都对应一个 Unicode 编码，则有 65500 多个观测值，如何对如此长的序列进行有效学习？

相信持续通过对本课程进行深入学习，定能找到解决问题的思路。

参考文献

[1] 萧超武. 基于 HMM 的驾驶模式识别方法研究及应用[D]. 华南理工大学, 2015.

[2] 翟晓娟. 基于 HMM 的随机驾驶人疲劳状态识别研究[D]. 大连理工大学, 2019.

[3] 李航. 统计学习方法(第 2 版)[M]: 清华大学出版社, 2019.