隐马尔可夫学习笔记（一）

隐马尔可夫模型学习笔记

前言

学习隐马尔可夫模型时，最大的困难便是一堆公式与实际问题对应不上号。原因可能还是在于对概率论的理解太表面，且隐马尔可夫模型考虑了时间因素，显然这样的随机过程一时半会是难以形象的理解的。因此，本文采用先举例，后定义公式的方式来学习隐马尔可夫模型。

思考

- 隐马尔可夫链是什么，用来解决一些什么具体问题?
- 隐马尔可夫链的基本假设是什么？
- 隐马尔可夫模型的3个基本问题
    - 概率计算问题对应的算法如何实现？
    - 预测问题对应的算法如何实现？
    - 学习问题对应的算法如何实现？（算法模型比较复杂，将分篇阐述）

概念阐述

隐马尔可夫模型

当然，我们还是首先需要知道隐马尔可夫模型（HMM）在统计学习中的地位和应用。参考书本《统计学习方法》的介绍，隐马尔可夫是可用于**标准问题**的统计学习模型，描述由隐藏的马尔可夫链随机生成观测序列的过程，属于生成模型。本文首先介绍隐马尔可夫模型的基本概念，然后分别叙述隐马尔可夫模型的概率计算**算法**、学习算法以及预测算法。隐马尔可夫模型在**语音识别**、**自然语言**处理、生物信息、模式识别等领域有着广泛的应用。

标注问题

问题阐述

参看《统计学习方法》信息抽取例子。

> 输入：At Microsoft Research, we have an insatiable curiosity and the desire to create new technology that will help define the computing experience.
> 输出：At/O Microsoft/B Research/E, we/O have/O an/O insatiable/B curiosity/E and/O the/O desire/BE to/O create/O new/B technology/E that/O will/O help/O define/O the/O computing/B experience/E.

这是在干嘛。。。如果我们是人，这个问题是说，给你一个输入，该输入为一句完整的句子，我们要从中找出名词短语来（PS：/后的符号是一个标注，由计算输出，暂时可以不去看它）。OK，这有什么困难的，根据我们所学的英语知识，寻找连续两个名词，再人为的判断下它是否符合为意义的名词即可。再做进一步退化，我们不知道该单词的意思，只知道词性，我们该怎么做？行吧，标出该句子中每个单词的词性，如介词，副词，名词，连续2个or N个名词组成的词组可能为名词短语。可并不是所有的名词短语是有意义的啊，我们还是需要大量的英文知识储藏才能解决这个问题。

现在我们来考虑计算机如何解决该问题。目前来看，计算机并不知道各英语单词的词性，这不难，给它一个词性对应表，让它暴力判断下即可。可问题又来了，哪怕在成千上万的英语单词中，能够寻找到对应的词性，可连续2个or N个单词组成的词组如何判断是有意义的呢？显然，想要再找对应的名词短语映射表，那简直就是天文数字，无从下手。

机械的求解方法无法解决，这时候统计学就派上了用场，标注问题便是解决该问题的有效手段。标注问题分为学习和标注两个过程，首先给定一个训练数据集

$$T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

这里，$x_i = (x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(n)})^T, i = 1, 2, \ldots, N$，是输入观测序列，$y_i = (y_i^{(1)}, y_i^{(2)}, \ldots, y_i^{(n)})^T$ 是相应的输出标记序列，n是序列的长度，对不同样本可以有不同的值。

继续回到上述问题，假设我们有了一堆输入输出数据，标注问题便能够通过数学概率模型来建立一种学习模型，训练出有效的参数，根据这个模型来预测输入值对应的输出值。如下图：

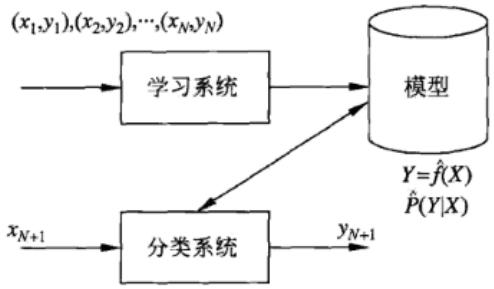


图 1.4 分类问题

那么该问题就完全就可以用统计学习的方法来解决，我们假设要对文章进行标注。那么每个英文单词是一个观测，英文句子是一个观测序列，标记表示名词名词短语的“开始”、“结束”、“其他”（分别以B,E,O表示），标记序列表示英文句子中基本名词短语的所在位置。

也就是说，针对观测序列，即我们的输入语句，我们可以对应的给每个单词进行标注，标注符号可以自由定义，套用前面最蠢的办法，我们可以训练出一个词性标注器，即at对应的介词，Microsoft对应的名词，再来判断名词短语。但既然我们可以人为定义标注符号，解决该问题并不需要词性标注这一过程，而是直接根据学习系统去判断名词短语的开始和结束，或者其他。只要输入的训练数据均符合上述约定即可。

OK，我们解释了标注问题用来解决什么样的实际问题，但刚才提到的数学模型却没有明确，其背后使用了统计概率的相关知识，而一个最常见的模型便是我们的隐马尔可夫模型。相比信息抽取的例子，要理解隐马尔可夫模型是干嘛的，显然还是困难的，我们不妨引用一篇关于HMM博文（请点击这里）的例子来进一步阐述隐马尔可夫模型的原理。

隐马尔可夫模型定义

刚才我们说了信息抽取的例子，其实仔细回想一下，为何需要隐马尔可夫模型，“隐”这个概念是相当重要的。藏于物理世界中的往往是表象，而真正的隐藏状态是不可见的，我们往往需要通过观察大量的表象来总结规律，从而能够确定事物的隐含状态，达到认识事物的本质。而事物的隐含状态与事物的表象存在着一定的关联关系，我们可以通过统计的方法来确立它们之间的关联。如观察海藻湿度的状态，能否确定是雨天还是晴天。这个例子可能偏颇，需明确一下，海藻的湿度为可见状态，而天气状态为隐藏状态（假设盲人能够感知海藻湿度，却无法观察天气）。

1.海藻模型

我们先来定义观测概率矩阵：

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

其中，

$$b_j(k) = P(o_t = v_k | i_t = q_j), k = 1, 2, \ldots, M; j = 1, 2, \ldots N$$

是在时刻t处于状态q_j的条件下生成观测v_k的概率。观测概率矩阵的定义考虑了时间t的因素，但实际上在分析问题时，我们做了简化，即在任何时刻，我的观测概率矩阵不随时间变化，因此我们在理解观测概率矩阵时，可以把t给去掉。M是我们的观测状态总数，N是我们隐藏状态总数。

回归海藻例子，刚才提到了我们可以观察海藻的湿度来确定隐含状态。那么我们现定义海藻湿度分为四个等级为”dry”,”dryish”,”damp”,”soggy”.显而易见，不同的天气状况我们能观测到海藻不同湿度的概率是不同的，以表格的形式如下：

	dry	dryish	damp	soggy
sunny	0.60	0.20	0.15	0.05
cloudy	0.25	0.25	0.25	0.25
rainy	0.05	0.10	0.35	0.50

如果仔细观察$b_j(k)$的定义，上述表格所列出的概率，其实就是我们的观测概率矩阵。因此，$b_j(k)$可以这样理解，j为不同的天气状态，k为不同的海藻湿度等级。而条件概率，无非是定义了在t时刻某个q_j状态下，某个v_k状态在t时刻的概率，由于观测概率矩阵不考虑时间因素，所以$b_j(k)$也不会随着时间发生变化。写成矩阵的形式为：

$$B = \begin{pmatrix} 0.60 & 0.20 & 0.15 & 0.05 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.05 & 0.10 & 0.35 & 0.50 \end{pmatrix}$$

隐马尔可夫模型中还定义了状态概率矩阵：

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

其中，

$$a_{ij} = P(i_{t+1} = q_j | i_t = q_i), i = 1, 2, \ldots N; j = 1, 2, \ldots, N$$

是在时刻处于状态q_i的条件下在时刻t+1转移到状态q_j的概率。

我们先来看看最一般的马尔科夫定义，马尔可夫链是随机变量$X_1, X_2, \ldots X_N$的一个数列。这些变量的范围，即他们所有可能取值的集合，被称为“状态空间”，而X_n的值则是在时间n的状态。如果X_{n+1}对于过去状态的条件概率分布仅是X_n的一个函数，则

$$P(X_{n+1}|X_0, \ldots, X_n) = P(X_{n+1}|X_n)P(X_n|X_{n-1}) \cdots P(X_1|X_0)$$

这式子在理论上是最完美的，可是在实际运用过程中却令人非常的头疼，第一，假设我们要计算某个连续序列的概率，这些连续序列的概率随着长度的增加将不断减小，最后概率值小到根本无意义。第二，每次计算都要记录前n次状态，当序列长度一增，数据量上去后，计算量相当大。试想，一个关于天气状态的时间序列，漫长的状态转移是由地球诞生那年开始的，如果计算出现序列的概率，那要何年马月才能算得清楚。

所以隐马尔可夫模型做了最基本的假设，即当前状态只与前一个状态有关。即

$$P(X_{n+1}|X_0, \ldots, X_n) = P(X_{n+1}|X_n)$$

换句话说，今天的天气状态只受昨天天气状态的影响，此处+1的含义为+1天，当然你又会想了，在不同的时间段，每次+1天的状转移矩阵都是一样的嘛？好吧，按实际情况来说，万年前晴天转雨天的概率显然和如今晴天转雨天是不一样的，但隐马尔可夫模型又做了一个假设，不动性假设（状态概率转移矩阵与时间无关）。即

$$P(X_{i+1}|X_i) = P(X_{j+1}|X_j), \forall i, j$$

此处的i，j在海藻模型中表示的是地球时间第i天和地球时间第j天。

有了时间不动性假设，我们就可以统计概率了，即假设历史上只出现了三种天气状态，分别为”sunny”,”cloudy”,’rainy’.那么根据某个时间段内，可以统计出如”sunny”→”cloudy”的频率，这里的频率随着统计量可以理解为概率。那么，我们又可以列一个表格来描述从不同的状态转移至另一状态，如下表

	sunny	cloudy	rainy
sunny	0.500	0.375	0.125
cloudy	0.250	0.125	0.625
rainy	0.250	0.375	0.375

咱们再来看看a_{ij}的定义，某个时刻为q状态时，到下一时刻q的概率。这说的不就是上述表格对应的从行到列的概率。因此，也搞清楚了a_{ij}为

$$A = \begin{pmatrix} 0.500 & 0.375 & 0.125 \\ 0.250 & 0.125 & 0.625 \\ 0.250 & 0.375 & 0.375 \end{pmatrix}$$

这里我更喜欢把a_{ij}看成$a_{i\to j}$，即表示从i到j的概率，反之是没有意义的。不行你把列加一加看看是否符合概率和为1？

这时有了状态转移概率和观测概率矩阵，我们可以搬出书中关于隐马尔可夫模型的定义了。等一下，作为一个系统我该从哪里开始运行？或者计算从哪里开始算起呢？显然我们还缺一个要素，初始状态概率向量π，即$\pi = (\pi_i)$

$$\pi_i = P(i_1 = q), i = 1, 2, \ldots, N$$

是时刻$t = 1$出去状态q的概率。

看定义一目了然了，地球诞生之初，我咋知道是什么天气状态？由此，我们需要人为给定一个初始向量如

$$\pi = (1.0, 0.0, 0.0)$$

或者

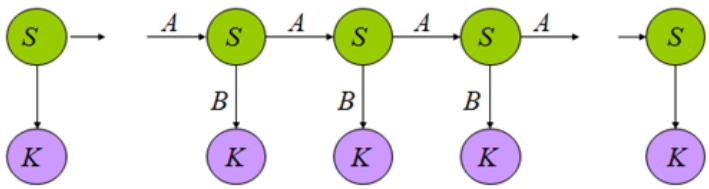
$$\pi = (0.6, 0.2, 0.2)$$

好了，一个完整的隐马尔可夫模型已经定义完毕了，回归书上的数学定义重新梳理一遍。隐马尔可夫模型由初始状态概率向量π、状态转移概率矩阵A和观测概率矩阵B决定。π和A决定状态序列，B决定观测序列。因此，隐马尔可夫模型λ可以用三元符号表示，即

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

A, B, π称为隐马尔可夫模型的三要素。

隐马尔可夫模型是关于时序的概率模型，描述由一个隐藏的马尔可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而产生观测随机序列的过程。隐藏的马尔可夫链随机生成的状态的序列，称为状态序列；每个状态生成一个观测，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。



如上图所示，第一层为不可观测状态，第二层为观测状态。在海藻模型中，盲人每天第一件事情便是感知海藻的湿度，即每天记录海藻的状态。假设盲人连续记录了三天，海藻的观测序列为(“dry”,”damp”,”soggy”)，那由我们的隐马尔可夫模型知道，第一天海藻状态为”dry”可能最佳隐藏状态为”sunny”；

第二天海藻状态为”damp”可能最佳隐藏状态为”cloudy”；第三天海藻状态为”soggy”可能最佳隐藏状态为”rainy”。这是根据人为经验去判断的，但whatever，人其实就是一个自我学习系统，我们训练出的规则可能相对简单，由此我们得到了天气的隐藏状态序列为(“sunny”,”cloudy”,”rainy”)。

继续隐马尔可夫模型的形式定义：

设Q是所有可能的状态集合，V是所有可能的观测的集合。

$$Q = \{q_1, q_2, \dots, q_N\}, V = \{v_1, v_2, \dots, v_M\}$$

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。

$$I = \{i_1, i_2, \dots, i_T\}, V = \{o_1, o_2, \dots, o_T\}$$

由上述的海藻模型，以及盲人的记录可以得到，$I = \{sunny, cloudy, rainy\}, O = \{dry, damp, soggy\}$.

你可能疑问了O/I，对应的观测状态O下，I序列的概率一定是最大嘛？即$P(I|O)$ 的概率一定是最大吗？或者你又想了，为什么在某天开始的观察序列一定是$O = \{dry, damp, soggy\}$从而引申出另外一个问题，假设给出了$\lambda = (A, B, \pi)$ 的模型，你咋确定从某一时刻起一定是该观测序列，而不是其他。即求$P(O|\lambda)$的概率。

这里便引出了隐马尔可夫模型的3个基本问题的其中两个，概率计算问题和预测问题。

隐马尔可夫3个基本问题

- 概率计算问题。给定模型$\lambda = (A, B, \pi)$ 和观测序列$O = \{o_1, o_2, \dots, o_T\}$，计算在模型$\lambda$下观测序列$O$出现的概率$P(O|\lambda)$.
- 预测问题，也称为解码问题。已知模型$\lambda = (A, B, \pi)$ 和观测序列$O = \{o_1, o_2, \dots, o_T\}$，求给定观测序列条件概率$P(I|O)$ 最大的状态序列$I = \{i_1, i_2, \dots, i_T\}$. 即给定观测序列，求最有可能的对应的状态序列。
- 学习问题。已知观测序列$O = \{o_1, o_2, \dots, o_T\}$，估计模型$\lambda = (A, B, \pi)$ 参数，使得在该模型下观测序列概率$P(O|\lambda)$最大。即用极大似然估计的方法估计参数。

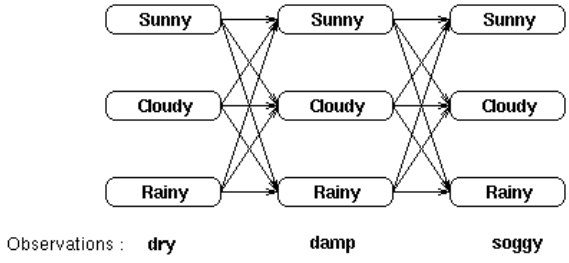
算法

概率计算问题

海藻模型中提出了在已知模型$\lambda = (A, B, \pi)$ 求解$P(O|\lambda)$的问题。我们就给它来求解一把。

1.穷举搜索

问题求解当没有好的办法时，我们骑着驴也要找马，咋办呢，穷举呗。刚才在海藻模型中，根据观察序列寻找了一个隐藏序列。该概率可以表示为$P(O|I, \lambda)$。显然I 序列的个数不只一个，3种天气状态，对应3天序列，I 的总数为$3 \times 3 \times 3 = 27$个，假设有N中状态，对应T个观察序列，那么I 的总数为N^T个。



从图中也能看的非常清楚，27种序列一一列举出来后，进行分别计算。对固定的状态序列$I = \{i_1, i_2, \dots, i_T\}$，观测序列$O = \{o_1, o_2, \dots, o_T\}$ 的概率是$P(O|I, \lambda)$，即

$$P(O|I, \lambda) = b_1(o_1)b_2(o_2) \cdots b_T(o_T)$$

因为这里是条件概率，即我们假定了每个隐藏序列是已知的，所以观测到的海藻状态跟状态转移矩阵没有关系，只跟它的观测概率分布有关。如第一天sunny，第二天sunny，第三天sunny均已知，那么

$$P(O = \{dry, damp, soggy\}|I = \{sunny, sunny, sunny\}, \lambda) = b_{sunny}(dry)b_{sunny}(damp)b_{sunny}(soggy) = 0.60 * 0.15 * 0.05 = 0.0045$$

那么问题来了，我们实际并不清楚$I = \{sunny, sunny, sunny\}$出现的概率是多少，这又怎么求解呢？即计算$P(I|\lambda)$出现的概率，但不是又状态转移矩阵嘛，只要知道初始状态概率向量π，我们就能够从某个时刻推断一连串隐藏状态序列的概率了，即

$$P(I|\lambda) = \pi_{i_1} a_{i_1 i_2} a_{i_2 3} \cdots a_{i_{T-1} i_T}$$

在这个计算公式中，对马尔可夫链的一阶假设得到了很好的简化。比如，第一天为sunny的情况的概率为初始概率已知，第二天为sunny的概率只跟第一天的状态有关，就是我们的状态转移矩阵sunny→sunny的概率，第三天仍为sunny的概率也只跟前一天相关，即sunny→sunny的概率，所以隐藏状态从一个状态转移至另一个状态每次只要乘以某个状态转移矩阵中的某个值即可。否则，第三天跟第一天和第二天的状态相关，公式表示为：

$$P(I|\lambda) = \pi_{i_1} a_{i_1 i_2} P(i_3|i_2, i_1)$$

很明显$P(i_3|i_2, i_1)$的概率是无法求得的（我们不能根据一阶的状态转移矩阵去求解二阶马尔科夫模型）。除非我们在做历史统计时，统计二阶转移概率，试想一下，当隐藏状态序列为T时，我们就要分别统计$T-1, T-2, \ldots, 1$阶的状态转移概率，当序列T为无穷时，这已经无法完成了。

有了I出现的概率，和在I出现的条件下，O出现的概率后，我们就可以求得联合概率$P(O, I|\lambda)$，即

$$P(O, I|\lambda) = P(O|I, \lambda)P(I|\lambda)$$
$$= \pi_{i_1} b_1(o_1) a_{i_1 i_2} b_2(o_2) \cdots a_{i_{T-1} i_T} b_T(o_T)$$

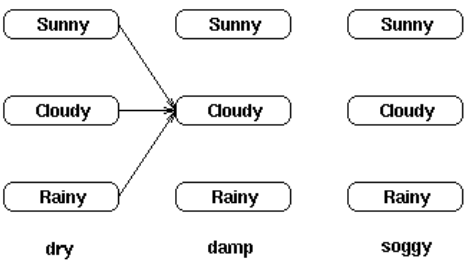
然后，对所有可能的状态序列I求和，得到观测序列O的概率$P(O|\lambda)$，即

$$P(O|\lambda) = \sum_I P(O|I, \lambda)P(I|\lambda)$$
$$= \sum_{i_1, i_2 \ldots, i_T} \pi_{i_1} b_1(o_1) a_{i_1 i_2} b_2(o_2) \cdots a_{i_{T-1} i_T} b_T(o_T)$$

利用上述公式计算量很大，是$O(TN^T)$阶的，这种算法在数据量大的情况下，是不可行的。其实从上图也能看出，这种计算方法，对每条路径都会计算一次，算法不记忆先前计算的结果，算法传播到第三个状态后，一切推倒重新计算。

在程序中有空间换时间的思想，其实在数学公式里也是通用的，我们可以采用中间变量来记录结果，假设能够利用该中间变量来计算后续节点，那么很明显可以极大的简化计算次数。因为，我们无需重新计算先前路径。

2.递归思想之前向算法



在这之前我们需要定义节点之间路径的数学含义和节点的有向边汇聚的数学含义。这是我们后续做理论推到的基础，也是理解前向算法和后向算法等价的基础。假设sunny有初始值$sunny = 0.6$，指向cloudy，且该路径附上权值$path_{sunny \to cloudy} = 0.375$，那么经过该路径连接的两个节点关系为

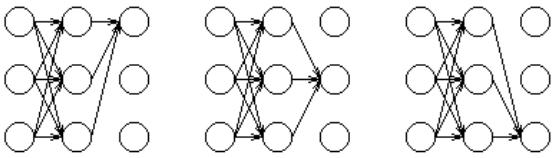
$cloudy = sunny * path_{sunny \to cloudy}$即有向路径可以代表“乘法”。同样的，指向cloudy有多条路径，即$cloudy = 0.2, rainy = 0.2$我们定义有向边汇聚到同一点，则该点表示为

$$cloudy = sunny * path_{sunny \to cloudy} + cloudy * path_{cloudy \to cloudy} + rainy * path_{rainy \to cloudy}$$

也就是说有向边的汇聚可以定义为各元素的“加法”法则。有了上述的约定之后，我们才可以做进一步的推导，递归式的寻找才有了理论依据。根据该节点有向边和有向边汇聚的性质，我们可以推得，soggy对应的sunny值可以由第二状态中

$$sunny * path_{sunny,sunny} + cloudy * path_{cloudy,sunny} + rainy * path_{rainy,sunny}$$

得到。同理，soggy下cloudy的状态可以由第二状态中$sunny, cloud, rainy$以及对应的有向边权值得到。如下图所示，



很明显，我们在上述推导过程中已经找到了递推式了。我们再来看看公式$P(O|\lambda)$，

$$P(O|\lambda) = \sum_{I} P(O|I,\lambda)P(I|\lambda)$$
$$= \sum_{i_1,i_2\ldots,i_T} \pi_i b_1(o_1) a_{i_1 i_2} b_2(o_2) \cdots a_{i_{T-1} i_T} b_T(o_T)$$

这公式描述的不就是所有节点有向边和有向边汇聚这张图嘛！无非就是找到其中的某几项能够表示成前一个节点的总和，再根据这个（总和*路径）就得到了下一节点和。答案渐渐明朗了，其实这中间变量就是我们的节点，那么我们来看看在隐马尔可夫模型中该节点具体是怎么算的。给定马尔科夫模型λ，定义到时刻t部分观测序列为$o_1, o_2, \ldots, o_t$且状态为q_i的概率为前向概率，记作

$$\alpha_t(i) = P(o_1, o_2, \ldots, o_t, i_t = q_i|\lambda)$$

刚才说了节点就是我们的$\alpha_t(i)$，即表示为在t时刻，状态为i的中间变量。由此我们可以根据有向边加权路径和有向边汇聚准则可以求得t+1时刻的$\alpha_{t+1}(i)$的值，即

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

根据上图的完全定义，其实不应该包含有$b_i(o_{t+1})$，但这要根据实际的隐马尔可夫模型挂钩，所以就带上它并不影响递归的推导过程。这里的$\alpha_t(j)$就是第t层各节点求得的值，而a_{ij}不就是我们的有向边权值嘛。既然有了递归表达式，但它不属于我们的最终形态，物理含义还是相对比较模糊的，即$\alpha_t(i)$这中间变量在隐马尔可夫模型中到底是个什么东西？

书中直接给出了定义：

$$\alpha_t(i) = P(o_1, o_2, \ldots, o_t, i_t = q_i|\lambda)$$

我们根据定义式来证明递推式是否成立，

$$\begin{aligned}\alpha_t(i) &= P(o_1, o_2, \ldots, o_t, i_t|\lambda) \\ &= P(o_1, o_2, \ldots, o_t|i_t, \lambda)P(i_t|\lambda) \\ &= P(o_1, o_2, \ldots, o_{t-1}|o_t, i_t, \lambda)P(o_t|i_t, \lambda)P(i_t|\lambda) \\ &= [\sum_{j=1,2,\ldots N} P(o_1, o_2, \ldots, o_{t-1}, j_{t-1}|o_t, i_t, \lambda)P(i_t|j_{t-1}, \lambda)]P(o_t|i_t, \lambda) \\ &= [\sum_{j=1}^{N} \alpha_{t-1}(j)a_{ji}]b_t(o_t)\end{aligned}$$

得证。式子$P(o_1, o_2, \ldots, o_{t-1}, j_{t-1}|o_t, i_t, \lambda)$中$o_1, o_2, \ldots, o_{t-1}$与$o_t, i_t$相互独立，因此，上式中的条件只与$\lambda$有关。有了$\alpha_t(i)$的概率公式，想要得到$P(O|\lambda)$,只要对最终时刻进行累加即可：

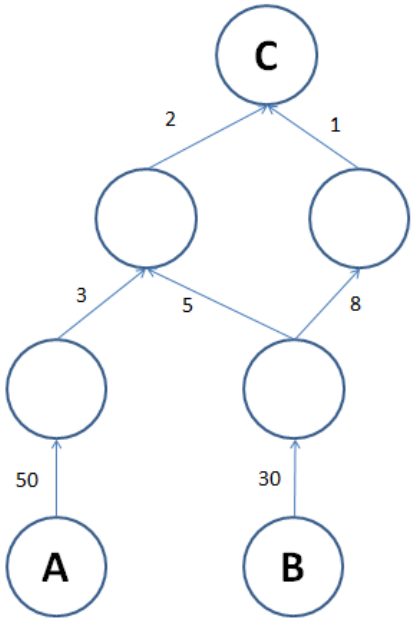
$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

综上所处，前向算法的计算思路已经阐述完毕，主要就是由中间节点来临时保存计算结果，在最后时刻对节点进行求解过程中，利用上一层保存的节点值来进一步求解，从而递归的进行计算，降低计算的复杂度。这样，利用前向概率计算$P(O|\lambda)$的计算量是$O(N^2T)$阶的。

3.反向算法

反向算法也是一种求解$P(O|\lambda)$的一种方法，但它的物理含义却并没有那么清晰，或者说根本很难确定它实际的物理含义，但我们可以简单证明下反向算法和前向算法所得出的结果是等价的。知乎上同样有一篇关于反向传播答疑贴，请参看这里

我们先来举一个简单的例子，假如我们知道C商户有A类货物和B类货物，A类货物50元/斤，B类货物30元/斤，A类货物卖出了3斤，卖给了2个人。B类货物分别卖出了5斤给2个人，8斤给1个人，请问C商户总过卖了多少钱。小学数学题，答案很简单，$C = 50 * 3 * 2 + (30 * 5 * 2 + 30 * 8 * 1) = 840$元



其实，除了直接进行求解外，我们可以把这个问题用节点来进行建模，如上图，我们根据有向边和有向边汇聚建立了如上各节点，同样的，我们能求出$C = 840$元。好了，我们现在要开始反向传播算法了，令C=1，注意这里的节点资源均为1，即刚才的前向算法是令A=1，B=1，求出C。现在反向传播的做法便是令C=1，OK，现在跟着剪头相反方向进行计算，由上至下，第二层节点的值分别为2和1，第三层节点的值分别2*3和$(2 * 5 + 1 * 8)$, 最后一层节点$A = 6 * 50 = 300, B = 18 * 30 = 540$,由此得A货物卖出了300元，B货物卖出了540元，总共也卖出了840元。所以说，不管自下而上求出各节点，还是自上而下求出的各节点，只要最终对各节点和进行累加，所得到的结果是一样的。

我们来看看隐马尔可夫模型反向算法是如何定义的，给定隐马尔可夫模型λ，定义在时刻t状态为q的条件下，从t+1到T的部分观测序列为$o_{t+1}, o_{t+2}, \ldots, o_T$的概率为后项概率，记作

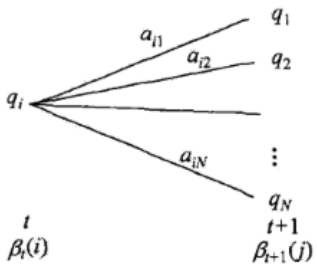
$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots, o_T|i_t = q, \lambda)$$

可以用递推的方法求得后向概率$\beta_t(i)$及观察序列概率$P(O|\lambda)$.

算法（观测序列概率的后向算法）

> 输入：隐马尔可夫模型λ，观测序列O;
> 输出：观测序列概率$P(O|\lambda)$
> (1)
>
> $$\beta_T(i) = 1, i = 1, 2, \ldots, N$$
>
> 类似于在商户卖货问题中令最终节点C=1
>
> (2) 对$t = T-1, T-2, \ldots, 1$
>
> $$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}\beta_{t+1}(j)), i = 1, 2, \ldots, N$$
>
> 其中a_{ij}直观的理解，就是对有向边进行了反向，而β就是反向算法过程中的中间变而已，我们也给它赋予了实际的物理含义。
>
> (3)
>
> $$P(O|\lambda) = \sum_{i=1}^{N} \pi_i b_i(o_1)\beta_1(i)$$
>
> 类似于对商户问题的起始节点进行求和。

我们再来理解下$\beta_t(i)$的定义，状态$i_t = q_i$为$\beta_t(i)$的条件了，这里和前向算法中存在了微小的差异，来看书上的图：



从图中其实可以很明显的看出，虽然是反向算法，但我们可以从前向的角度去理解该式子，由于t时刻，不同的q_i对应的a_{ij}是不同的，所以$P(o_{t+1}, o_{t+2}, \ldots, o_T|\lambda)$除了跟模型参数有关，还跟$q_i$相关。

步骤1.初始化后向概率，对最终时刻的所有状态q_i规定$\beta_T(i) = 1$. 步骤2.是后向概率的递推公式。如上图所示，为了计算在时刻t状态为q_i条件下时刻t+1之后的观测序列为$o_{t+1}, o_{t+2}, \ldots, o_T$的后向概率$\beta_t(i)$，只需要考虑在时刻t+1所有可能的N个状态$q_j$的转移概率，以及在此状态下的观测$o_{t+1}$的观测概率，然后考虑状态$q_j$之后的观测序列的后向概率。步骤3.求$P(O|\lambda)$的思路与步骤2.一致，只是初始概率$\pi_i$代替转移概率。

预测问题

回归实际问题，我们现在能够根据给定的模型参数λ计算出$P(O|\lambda)$的值，本文对标注问题进行阐述时引出了信息抽取的一个例子，它就是典型的预测问题，给定一连串输入序列，我们需要对每个单词进行标注，从而使得整个隐藏序列对于观测序列来讲概率最大。即$P(I|O)$最大。

1.维特比算法

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

我们先来看《统计学习算法》中，对维特比算法的人工计算实例，从而加深对维特比算法的理解。

例. 输入模型$\lambda = (A, B, \pi)$

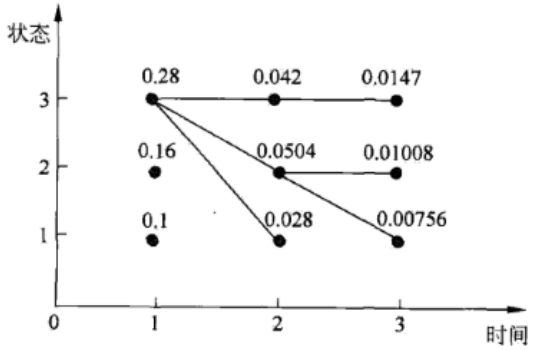
$$A = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.3 & 0.5 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.5 & 0.5 \\ 0.4 & 0.6 \\ 0.7 & 0.3 \end{bmatrix}$$

$$\pi = (0.2, 0.4, 0.4)^T$$

已知观测序列O={红，白，红}，试求最优状态序列，即最优路径$I^* = (i_1^*, i_2^*, i_3^*)$

假设我们对动态规划的原理不甚了解，没关系，我们只需要知道动态规划的其中一个原理，即如果最优路径在时刻t通过节点i_t^*，那么这一路径从节点i_t^*到终点i_T^*的部分路径，对于从i_t^*到i_T^*所有可能的部分路径来说，必须是最优的。



在t=1时刻，对于每一个状态$i, i = 1, 2, 3$,求状态为i观测o_1为红的概率，为了方便书写，我们记作$\delta_1(i)$，则

$$\delta_1(i) = \pi_i b_i(o_1) = \pi_i b_i(\text{红}), i = 1, 2, 3$$

代入实际数据

$$\delta_1(1) = 0.10, \delta_1(2) = 0.16, \delta_1(3) = 0.28$$

对于t=1时刻，每一个隐藏状态i，可能出现红色的概率已经有了，这跟我们的初始状态有关，跟转移概率无关。现在，很直观的想法，在t=2时刻，对于不同的隐藏状态，有三条路径同时达到，此处不再是计算三条路径的概率和了，而是取其中概率最大的一条，代表从t时刻转移至t+1时刻，某一隐藏状态序列i_t, i_{t+1}出现o_t, o_{t+1}的概率最大。因此，用公式表达便是：

$$\delta_2(i) = \max_{1 \le j \le 3} [\delta_1(j) a_{ji}] b_i(o_2)$$

计算：

$$\delta_2(1) = \max_{1 \le j \le 3} [\delta_1(j) a_{j1}] b_1(o_2)$$
$$= \max_j \{0.10 * 0.5, 0.16 * 0.3, 0.28 * 0.2\} * 0.5 = 0.028$$

同理，$\delta_2(2) = 0.0504, \delta_2(3) = 0.042$

在t=3时刻，

$$\delta_3(i) = \max_{1 \le j \le 3}[\delta_2(j)a_{ji}]b_i(o_3)$$

$$\delta_3(1) = 0.00756$$

$$\delta_3(2) = 0.01008$$

$$\delta_3(3) = 0.0147$$

以P^*表示最优路径的概率，则

$$P^* = \max_{1 \le i \le 3} \delta_3(i) = 0.0147$$

那么最优路径的终点是i_3^*:

$$i_3^* = \arg\max_i[\delta_3(i)] = 3$$

咦，少量了一个记录路径经过节点的变量了，没错，所以书中还定义了

$$\psi_t(i) = \arg\max_{1 \le j \le N}[\delta_{t-1}a_{ji}], i = 1, 2, .., N$$

有了该函数的定义，我们就能根据计算出来的最大概率路径，一路找回所有的节点了，即

$$在t = 2时，\ i_2^* = \psi_3(i_3^*) = \psi_3(3) = 3$$

$$在t = 1时，\ i_1^* = \psi_2(i_2^*) = \psi_2(3) = 3$$

于是求得最优路径，即最优状态序列$I* = (i_1^*, i_2^*, i_3^*) = (3, 3, 3)$

由此回过头来再看看书中更一般的定义，首先导入变量δ和ψ.（求解最优路径的中间变量和用来记录路径节点的指针）定义在时刻t状态为i的所有单个路径$(i_1, i_2, \ldots, i_t)$中概率最大值为

$$\delta_t(i) = \max_{i_1, i_2 \ldots, i_{t-1}} P(i_t = i, i_{t-1}, \ldots, i_1, o_t, \ldots, o_1|\lambda), i = 1, 2, \ldots, N$$

简化公式为$\delta_t(i) = \max_{i_1, i_2 \ldots, i_{t-1}} P(I, O|\lambda), i = 1, 2, \ldots, N$，这里是求$I, O$的联合概率密度最大，但仔细思考为什么这就等效于求解$P(I|O)$概率最大呢？

其实条件概率和联合概率之间有它们的关系，省略共同的影响参数λ，可得

$$P(I, O) = P(I|O)P(O)$$

这里的变量为I对应的隐藏状态序列，我们把公式进行变换一下就得

$$P(I|O) = \frac{P(I,O)}{P(O)}$$

变量为I，但由公式已知，决定状态序列O下的隐藏序列I的概率，由联合概率和O本身的概率所决定，但在针对预测问题时，我们是假定我们得到了O序列，所以可以理解为$P(O) = 1$，我们无需匹配O, I使得$P(I|O)$最大，所以对联合概率求关于变量序列I的最大概率最终就相当于求解$P(I|O)$的概率最大。

由定义可得变量δ的递推公式：

$$\delta_{t+1}(i) = \max_{1 \le j \le N}[\delta_t(j)a_{ji}]b_i(o_{t+1}), i = 1, 2, \ldots, N, t = 1, 2, \ldots, T-1$$

定义在时刻t状态为i的所有单个路径$(i_1, i_2, \ldots, i_{t-1}, i_t)$中概率最大的路径的第t-1个节点为：

$$\psi_t(i) = \arg \max_{1 \le j \le N}[\delta_{t-1}a_{ji}], i = 1, 2, .., N$$

由此可得维特比算法。

算法（维特比算法）

输入：模型$\lambda = (A, B, \pi)$和观测$O = (o_1, o_2, \ldots, o_t)$；
输出：最优路径$I^* = (i_1^*, i_2^*, \ldots, i_T^*)$
(1) 初始化

$$\delta_1(i) = \pi_i b_i(o_1), i = 1, 2, \ldots, N$$

$$\psi_1(i) = 0, i = 1, 2, \ldots, N$$

(2) 递推。对$t = 2, 3, \ldots, T$

$$\delta_t(i) = \max_{1 \le j \le N}[\delta_{t-1}(j)a_{ji}]b_i(o_t), i = 1, 2, \ldots, N$$

$$\psi_t(i) = \arg \max_{1 \le j \le N}[\delta_{t-1}a_{ji}], i = 1, 2, .., N$$

(3) 终止

$$P^* = \max_{1 \le i \le N} \delta_T(i)$$

$$i_T^* = \arg \max_{1 \le i \le N}[\delta_T(i)]$$

最优路径回溯。对$t = T-1, T-2, \ldots, 1$

$$i_t^* = \psi_{t+1}(i_{t+1}^*)$$

求得最优路径$I^* = (i_1^*, i_2^*, \ldots, i_T^*)$

在预测问题上，我们用到了贝叶斯的思考哲学，更多的内容请参看博文-数学之美番外篇：平凡而又神奇的贝叶斯方法。针对学习算法，由于它内容繁杂且用到EM算法，因此待我学习完EM算法后，再进行梳理。

Code Time

在海藻模型中我们定义了一堆状态转移矩阵，观测概率矩阵和初始状态概率向量。在**Python**中数据如下：（打开文本编辑器，新建文本test.py）

```
observations = ('dry','dryish','damp','soggy')

start_probability ={'sunny':0.6,'cloudy':0.2,'rainy':0.2}

transition_probability = {
    'sunny':{'sunny':0.5,'cloudy':0.375,'rainy':0.125},
    'cloudy':{'sunny':0.25,'cloudy'0.125:,'rainy':0.625},
    'rainy':{'sunny':0.25,'cloudy':0.375,'rainy':0.375}
}

emission_probability ={
    'sunny':{'dry':0.6,'dryish':0.2,'damp':0.15,'soggy':0.05},
    'cloudy':{'dry':0.25,'dryish':0.25,'damp':0.25,'soggy':0.25},
    'rainy':{'dry':0.05,'dryish':,0.1'damp':0.35,'soggy':0.50}
}
```

定义HMM模型：（打开文本编辑器，新建文件hmm.py）

```
class HMM:
    def __init__(self,A,B,pi):
        self.A =A
        self.B =B
        self.pi =pi
```

前向算法：（在文件hmm.py中添加）

```
def _forward(self,obs_seq):
        # 取A = N x N
        N = self.A.shape[0]
        T = len(obs_seq)

        F = zeros((N,T))

        # alpha = pi*b
        F[:,0] = self.pi *self.B[:,obs_seq[0]]

        for t in range(1,T):
            for n in range(N):
                # 计算第t时，第n个状态的前向概率
                F[n,t] = dot(F[:,t-1], (self.A[:,n])) * self.B[n, obs_seq[t]]
        return F
```

F 即为我们在前向算法中定义的$\alpha_t(i)$.

后向算法：（在文件hmm.py中添加）

```
def _backward(self,obs_seq):
        N = self.A.shape[0]
        T = len(obs_seq)

        X = zeros((N,T))
        # 表示X矩阵的最后一列
        X[:,-1:] =1

        for t in reversed(range(T-1)):
            for n in range(N):
                # 边权值为a_ji
                X[n,t] = sum(X[:,t+1] * self.A[n,:] * self.B[:,obs_seq[t+1]])

```

```
        return X
```

测试前向算法和后向算法数据的一致性：（在test.py中添加）

```
def gengerate_index_map(labels):
    index_label = {}
    label_index = {}

    i =0
    for l in labels:
        index_label[i] =l
        label_index[l] =i

        i+=1
    return label_index,index_label

states_label_index,states_index_label = gengerate_index_map(states)
observations_label_index,observations_index_label = gengerate_index_map(observations)

def convert_observations_to_index(observations,label_index):
    list =[]
    for o in observations:
        list.append(label_index[o])
    return list


def convert_map_to_matrix(map,label_index1,label_index2):
    m = empty((len(label_index1),len(label_index2)),dtype = float)
    for line in map:
        for col in map[line]:
            m[label_index1[line]][label_index2[col]] = map[line][col]
    return m

def convert_map_to_vector(map,label_index):
    v = empty(len(map),dtype = float)
    for e in map:
        v[label_index[e]] =map[e]
    return v

A = convert_map_to_matrix(transition_probability,states_label_index,states_label_index)
print (A)
B = convert_map_to_matrix(emission_probability,states_label_index,observations_label_index)
print (B)
observations_index = convert_observations_to_index(observations,observations_label_index)
print (observations_index)
pi = convert_map_to_vector(start_probability,states_label_index)
print (pi)

h = HMM(A,B,pi)
# 人为定义的海藻状态序列
obs_seq = ('dry','damp','soggy')
obs_seq_index = convert_observations_to_index(obs_seq,observations_label_index)

# 计算P(o|lambda)
F = h._forward(obs_seq_index)
print ("forward: P(O|lambda) = %f" %sum(F[:,-1]))
X = h._backward(obs_seq_index)
print ("backward: P(O|lambda) = %f" %sum(X[:,0]*pi*B[:,0]))
```

前向，后向算法输出的结果为：（答案一致）

```
forward: P(O|lambda) = 0.026441
backward: P(O|lambda) = 0.026441
```

维特比算法：（在hmm.py中添加）

```
def viterbi(self,obs_seq):

        N = self.A.shape[0]
```

```
        T = len(obs_seq)

        prev = zeros((T-1,N),dtype = int)

        V = zeros((N,T))
        V[:,0] = self.pi * self.B[:,obs_seq[0]]

        for  t in range(1,T):
            for n in range(N):
                # 计算delta(j)*a_ji
                seq_probs = V[:,t-1] * self.A[:,n] * self.B[n,obs_seq[t]]
                # 记录最大状态转移过程
                prev[t-1,n] = argmax(seq_probs)
                V[n,t] = max(seq_probs)
        return V,prev

    def build_viterbi_path(self,prev,last_state):
        """
        returns a state path ending in last_state in reverse order.
        """
        T = len(prev)
        yield(last_state)
        # 从T-1开始，每次下降1
        for i in range(T-1,-1,-1):
            yield(prev[i,last_state])
            last_state = prev[i,last_state]

    def state_path(self,obs_seq):
        V,prev = self.viterbi(obs_seq)

        # build state path with greatest probability
        last_state = argmax(V[:,-1])
        path = list(self.build_viterbi_path(prev,last_state))

        return V[last_state,-1],reversed(path)
```

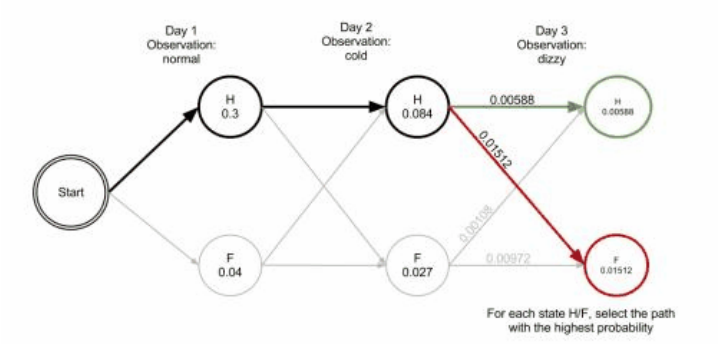
输出结果：（在test.py中添加）

```
# 计算P(I|o)
p,ss = h.state_path(obs_seq_index)
path = []
for s in ss:
    path.append(states_index_label[s])

print("最有可能的隐藏序列为：" ,path)
print("viterbi: P(I|O) =%f"% p)

result:
输入观察序列为：['dry','damp','soggy']
最有可能的隐藏序列为： ['sunny', 'cloudy', 'rainy']
viterbi: P(I|O) =0.010547
```

至此，维特比算法也呈现完毕，从输入和输出对应的关系来看，dry对应的sunny，damp对应的cloudy，soggy对应的rainy，还是相当符合实际情况滴。



关于隐马尔可夫模型概率计算问题和预测问题已经全部阐述完毕了，理论结合实践，在该模型的学习过程中，理解了反向算法和前向算法的一致性，在算法执行效率上，代替了暴力求解，采用递归思想和动态规划的原理来降低计算时间复杂度，理解了数学中的中间变量即为递归原型。

参考文献及推荐阅读

- 隐马尔可夫模型（HMM）攻略
- 隐马尔可夫模型 hacks 码农场
- 如何直观的解释back propagation算法
- 数学之美番外篇：平凡而又神奇的贝叶斯方法
- 李航. 统计学习方法[M]. 北京：清华大学出版社，2012