Viterbi算法——

给出一个观测序列o1,o2,o3 …，我们希望找到观测序列背后的隐藏状态序列s1, s2, s3, …；Viterbi以它的发明者名字命名，正是这样一种由动态规划的方法来寻找出现概率最大的隐藏状态序列（被称为Viterbi路径）的算法。

Viterbi算法的思想个人觉得和数据结构中图哪部分寻找最短路径的思想是一样的，只是图中节点间的距离是通过模型中的一些概率计算后得到的。

这里需要抄一点有关隐马可夫序列（HMM，Hidden Markov Model）的知识来解释一下观测序列和隐藏状态序列。首先从最简单的离Markov过程入手，我们知道，Markov随机过程具有如下的性质：在任意时刻，从当前状态转移到下一个状态的概率与当前状态之前的那些状态没有关系。所以，我们可以用一个状态转移概率矩阵来描述它。假设我们有n个离散状态S1, S2,…Sn，我们可以构造一个矩阵A，矩阵中的元素aij表示从当前状态Si下一时刻迁移到Sj状态的概率。但是在很多情况下，Markov模型中的状态是我们观察不到的。例如，容器与彩球的模型：有若干个容器，每个容器中按已知比例放入各色的彩球（这样，选择了容器后，我们可以用概率来预测取出各种彩球的可能性）；我们做这样的实验，实验者从容器中取彩球——先选择一个容器，再从中抓出某一个球，只给观察者看球的颜色；这样，每次取取出的球的颜色是可以观测到的，即o1, o2,…，**但是每次选择哪个容器是不暴露给观察者的，容器的序列就组成了隐藏状态序列S1, S2,…Sn**。这是一个典型的可以用HMM描述的实验。

HMM有几个重要的任务，其中之一就是期望通过观察序列来猜测背后最有可能的隐藏序列。在上面的例子中，就是找到我们在实验中最有可能选择到的容器序列。Viterbi正是用来解决这个问题的算法。HMM另外两个任务是：a) 给定一个HMM，计算一个观测序列出现的可能性；b)已知一个观测序列，HMM参数不定，如何优化这些参数使得观测序列的出现概率最大。解决前一个问题可以用与Viberbi结构非常类似的Forward算法来解决（实际上在下面合二为一），而后者可以用Baum-Welch/EM算法来迭代逼近。

从Wiki上抄一个例子来说明Viterbi算法。

假设你有一个朋友在外地，每天你可以通过电话来了解他每天的活动。他每天只会做三种活动之一——Walk, Shop, Clean。你的朋友从事哪一种活动的概率与当地的气候有关，这里，我们只考虑两种天气——Rainy, Sunny。我们知道，天气与运动的关系如下：

|  |  |  |
| --- | --- | --- |
|  | Rainy | Sunny |
| Walk | 0.1 | 0.6 |
| Shop | 0.4 | 0.3 |
| Clean | 0.5 | 0.1 |

例如，在下雨天出去散步的可能性是0.1。

而天气之前互相转换的关系如下，（从行到列）

|  |  |  |
| --- | --- | --- |
|  | Rainy | Sunny |
| Rainy | 0.7 | 0.3 |
| Sunny | 0.4 | 0.6 |

例如，从今天是晴天而明天就开始下雨的可能性是0.4 。

同时为了求解问题我们假设初始情况：通话开始的第一天的天气有0.6的概率是Rainy，有0.4概率是Sunny。OK，现在的问题是，**如果连续三天，你发现你的朋友的活动是：Walk->Shop->Clean；那么，如何判断你朋友那里这几天的天气是怎样的？**

解决这个问题的python伪代码如下：

def forward\_viterbi(y, X, sp, tp, ep):

   T = {}

   for state in X:

       ##          prob.      V. path  V. prob.

       T[state] = (sp[state], [state], sp[state])

   for output in y:

       U = {}

       for next\_state in X:

           total = 0

           argmax = None

           valmax = 0

           for source\_state in X:

               (prob, v\_path, v\_prob) = T[source\_state]

               p = ep[source\_state][output] \* tp[source\_state][next\_state]

               prob \*= p

               v\_prob \*= p

               total += prob

               if v\_prob > valmax:

                   argmax = v\_path + [next\_state]

                   valmax = v\_prob

           U[next\_state] = (total, argmax, valmax)

       T = U

   ## apply sum/max to the final states:

   total = 0

   argmax = None

   valmax = 0

   for state in X:

       (prob, v\_path, v\_prob) = T[state]

       total += prob

       if v\_prob > valmax:

           argmax = v\_path

           valmax = v\_prob

   return (total, argmax, valmax)

几点说明：

算法对于每一个状态要记录一个三元组：(prob, v\_path, v\_prob)，其中，prob是从开始状态到当前状态所有路径（不仅仅是最有可能的viterbi路径）的概率加在一起的结果（作为算法附产品，它可以输出一个观察序列在给定HMM下总的出现概率，即forward算法的输出），v\_path是从开始状态一直到当前状态的viterbi路径，v\_prob则是该路径的概率。算法开始，初始化T （T是一个Map，将每一种可能状态映射到上面所说的三元组上）三重循环，对每个一活动y，考虑下一步每一个可能的状态next\_state，并重新计算若从T中的当前状态state跃迁到next\_state概率会有怎样的变化。跃迁主要考虑天气转移（tp[source\_state][next\_state]）与该天气下从事某种活动（ep[source\_state][output]）的联合概率。所有下一步状态考虑完后，要从T中找出最优的选择viterbi路径——即概率最大的viterbi路径，即上面更新Map U的代码U[next\_state] = (total, argmax, valmax)。算法最后还要对T中的各种情况总结，对total求和，选择其中一条作为最优的viterbi路径。算法输出四个天气状态，这是因为，计算第三天的概率时，要考虑天气转变到下一天的情况。通过程序的输出可以帮助理解这一过程：

observation=Walk

        next\_state=Sunny

                state=Sunny

                        p=0.36

                        triple=(0.144,Sunny->,0.144)

                state=Rainy

                        p=0.03

                        triple=(0.018,Rainy->,0.018)

Update U[Sunny]=(0.162,Sunny->Sunny->,0.144)

next\_state=Rainy

                state=Sunny

                        p=0.24

                        triple=(0.096,Sunny->,0.096)

                state=Rainy

                        p=0.07

                        triple=(0.042,Rainy->,0.042)

        Update U[Rainy]=(0.138,Sunny->Rainy->,0.096)

observation=Shop

        next\_state=Sunny

                state=Sunny

                        p=0.18

                        triple=(0.02916,Sunny->Sunny->,0.02592)

                state=Rainy

                        p=0.12

                        triple=(0.01656,Sunny->Rainy->,0.01152)

        Update U[Sunny]=(0.04572,Sunny->Sunny->Sunny->,0.02592)

        next\_state=Rainy

                state=Sunny

                        p=0.12

                        triple=(0.01944,Sunny->Sunny->,0.01728)

                state=Rainy

                        p=0.28

                        triple=(0.03864,Sunny->Rainy->,0.02688)

        Update U[Rainy]=(0.05808,Sunny->Rainy->Rainy->,0.02688)

observation=Clean

        next\_state=Sunny

                state=Sunny

                        p=0.06

                        triple=(0.0027432,Sunny->Sunny->Sunny->,0.0015552)

                state=Rainy

                        p=0.15

                        triple=(0.008712,Sunny->Rainy->Rainy->,0.004032)

        Update U[Sunny]=(0.0114552,Sunny->Rainy->Rainy->Sunny->,0.004032)

        next\_state=Rainy

                state=Sunny

                        p=0.04

                        triple=(0.0018288,Sunny->Sunny->Sunny->,0.0010368)

                state=Rainy

                        p=0.35

                        triple=(0.020328,Sunny->Rainy->Rainy->,0.009408)

        Update U[Rainy]=(0.0221568,Sunny->Rainy->Rainy->Rainy->,0.009408)

final triple=(0.033612,Sunny->Rainy->Rainy->Rainy->,0.009408)所以，最终的结果是，朋友那边这几天最可能的天气情况是Sunny->Rainy->Rainy->Rainy，它有0.009408的概率出现。而我们算法的另一个附带的结论是，我们所观察到的朋友这几天的活动序列：Walk->Shop->Clean在我们的隐马可夫模型之下出现的总概率是0.033612。