

# 史上最详细最容易理解的HMM文章



2012年10月15日 13:52:18

标签：自然语言处理 / 算法 / algorithm / library / vb / 脚本

10

45370

<http://v.52nlp.cn/hmm-learn-best-practices-four-hidden-markov-models>[wik 个比较好的HMM例子](#)[分 尔科夫模型](#)

1(隐马尔科夫模型)是自然语言处理中的一个基本模型，用途比较广泛，如汉语分词、词性标注及语音识别等，在NLP中占有很重要的地位。网上关于HMM的介绍讲解文档很多，我自己当时开始看的时候也有点稀里糊涂。后来看到wiki上有个关于HMM的例子才如醍醐灌顶，忽然间明白HMM的三大问题是怎么回事了。例子我借助中文wiki重新翻译了一下，并对三大基本问题进行说明，希望对读者朋友有所帮助：

Alice 和Bob是好朋友，但是他们离得比较远，每天都是通过电话联系对方那天作了什么。Bob仅仅对三种活动感兴趣：公园散步、购物以及清理房间。他选择做什么事情只凭当天天气。Alice对于Bob所住的地方的天气情况并不了解，但是知道总的趋势。在Bob告诉Alice每天所做的事情基础上，Alice想要猜测Bob所在地的天气情况。

Alice认为天气的运行就像一个马尔可夫链。其有两个状态“雨”和“晴”，但是无法直接观察它们，也就是说，它们对于Alice是隐藏的。每天，Bob有一定的概率进行下列活动：“散步”，“购物”，或“清理”。因为Bob会告诉Alice他的活动，所以这些活动就是Alice的观察数据。这个系统就是一个**隐马尔可夫模型HMM**。

Alice知道这个地区的总的天气趋势，并且平时知道Bob会做的事情。也就是说这个隐马尔可夫模型的参数是已知的。可以用程序语言(Python)写下来：

```
// 状态数曰 两个状态 · 雨或晴
```



MachineLearning-ZJU

原创	粉丝	喜欢	评论
50	170	94	61

等级： **博客 5**      访问量：21万+

积分：2321      排名：1万+

排名	制造商	测试值/峰值	实验室
1	IBM Earth Simulator/ S120	35868.00 44068.00	日本/2002
2	Bowling-Packer ASCI Q - AlphaServer 385/45/ 8192	13880.00 20480.00	美国/2002
3	Linux Network Intel Linux Cluster Xeon 2.4 GHz - Quadrics/ 2304	7638.00	爱特纳-利普莫尔国家实验室 11060.00 美国/2002
4	IBM ASCI White, SP Power3 375 MHz/ 8192	7304.00 12288.00	爱特纳-利普莫尔国家实验室 美国/2000
5	IBM SP Power3 375 MHz 16 way/ 6400	6984.00 7304.00	爱特纳-利普莫尔国家实验室 美国/2002
6	IBM eSeries Cluster Xeon 3.4 GHz - Quadrics/ 1900	6588.00 8216.00	爱特纳-利普莫尔国家实验室 美国/2003
7	Fujitsu	5406.00	日本国家航空实验室

## 计算机学校排名



## 他的最新文章

[更多文章](#)

九度 1339

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



```
// 每个状态下可能的观察值
observations = ('walk', 'shop', 'clean')

//初始状态空间的概率分布
start_probability = {'Rainy': 0.6, 'Sunny': 0.4}

// 与时间无关的状态转移概率矩阵
transition_probability = {
    'Rainy': {'Rainy': 0.7, 'Sunny': 0.3},
    'Sunny': {'Rainy': 0.4, 'Sunny': 0.6},
}

// 给定状态下，观察值概率分布,发射概率
emission_probability = {
    'Rainy': {'walk': 0.1, 'shop': 0.4, 'clean': 0.5},
    'Sunny': {'walk': 0.6, 'shop': 0.3, 'clean': 0.1},
}
```

在这些代码中,start\_probability代表了Alice对于Bob第一次给她打电话时的天气情况的不确定性(Alice知道的只是那个地方平均起来下雨多些).在这里,这个特定的概率分布并非平衡的,平衡概率应该接近（在给定变迁概率的情况下）{'Rainy': 0.571, 'Sunny': 0.429}。 transition\_probability 表示马尔可夫链下的天气变迁情况,在这个例子中,如果今天下雨,那么明天天晴的概率只有30%.代码emission\_probability 表示了Bob每天作某件事的概率.如果下雨,有 50% 的概率他在清理房间;如果天晴,则有60%的概率他在外头散步。

Alice和Bob通了三天电话后发现第一天Bob去散步了，第二天他去购物了，第三天他清理房间了。Alice现在有两个问题：这个观察序列“散步、购物、清理”的总的概率是多少？(注：这个问题对应于HMM的基本问题之一：已知HMM模型 $\lambda$ 及观察序列O，如何计算 $P(O|\lambda)$ )？最能解释这个观察序列的状态序列（晴/雨）又是什么？（注：这个问题对应HMM基本问题之二：给定观察序列 $O=O_1,O_2,...,O_T$ 以及模型 $\lambda$ ,如何选择一个对应的状态序列 $S = q_1,q_2,...,q_T$ ，使得S能够最为合理的解释观察序列O？）

至于HMM的基本问题之三：如何调整模型参数,使得 $P(O|\lambda)$ 最大？这个问题事实上就是给出很多个观察序列值，来训练以上几个参数的问题。

九度 1206  
九度 1205

文章分类

ACM	6篇
算法	18篇
机器学习	11篇
学习与生活	5篇
九度OJ	19篇
南洋理工ACM	1篇

展开

文章存档

2014年5月	21篇
2013年6月	3篇
2013年1月	1篇
2012年10月	6篇
2012年7月	2篇
2012年6月	13篇



展开


他的热门文章

## HMM学习最佳范例与崔晓源的博客

分类 [隐马尔科夫模型](#)

“HMM学习最佳范例”与“崔晓源的博客”本来是不搭边的，由于自己花了几几乎一个晚上浏览崔师兄的博客，没有时间写文章了，所以最终决定放在这里做成大杂烩，不过我觉得这个大杂烩还是有点价值的。

 说HMM，通过Google Analytics 发现，读者经常通过与“隐马尔科夫模型、HMM”相关的关键字访问52nlp的，因为这里曾经写了一篇简单的介绍HMM的文章。事实上，对于HMM，由于自己没有直接实践过，仅停留在“纸上得来”的程度，心里也虚。某天赶巧遇到了国外这个专门介绍HMM及其相关算法的主页：<http://www.comp.leeds.ac.uk/roger/Hidde>  
 [Models/html\\_dev/main.html](#)

 图文并茂还动感十足，写得又通俗易懂，可以说是我见到过的介绍HMM最好的范例了。读完了立即有翻译全文的冲动，一方面可以给有需要的读者以帮助，另一方面翻译虽然耗时，但却能仔细把握一下比较细节的地方，这也是我翻译“自然语言处理”的初衷。不过Google了一下，发现已经有人把这件事做了，这个人就是崔晓源，中文译名是“隐马尔科夫模型HMM自学”。

原计划这一篇博客题目为“HMM学习最佳范例”的，准备介绍这个英文主页和崔晓源的翻译，出于尊重译者劳动的缘故，Google“隐马尔科夫模型HMM自学”，可是发现其被转载了很多，除了知道译者是“崔晓源”外，原始出处似乎被丢失了。不过最终还是找到了原始出处：

[http://blogcui.spaces.live.com/blog/cns!46BDB23E24219CE9!144.entry?\\_c=BlogPart](http://blogcui.spaces.live.com/blog/cns!46BDB23E24219CE9!144.entry?_c=BlogPart)

其实就是崔师兄在msn spaces上的博客，仔细对比了一下原文，发现崔师兄的这个翻译是一个简化和缩略版本，有些地方只是概况性的翻译了一下，省去了一些内容，所以这个介绍HMM主页还有再翻译的必要。有可能的话，今后在52nlp上我会慢慢翻译HMM这个系列。

对比完之后，我就浏览开他的博客了，没想到，一发而不可收，主要在于他的博客多数内容都很有价值。博客最集中更新的一段时间，是在05年5月到9月份他在MSRA-NLC组实习的时候，不过可惜的是，现在几乎不更新了，但是技术博客的好处再于其有效期更长，所以许多东西仍很可以参考，读者如果对NLP，IR或者机器学习感兴趣，不妨按时间顺序读读他的日志，定会有不小收获的，这里绝非广告。他目前在MSRA工作，以下是他的“About me”：

”A man full of enthusiasm for advanced technology business, natrua language processing, IR and search engine technology. I’m finding my way. You never do a bad job only if you choose the right time. So keep your pace accordingly.“

他在！五！星！级！的！发！现！这！个！并！不！如！器！器！器！的！精！神！博！客！

## 互信息概念与定理

 23341

kD-tree 的C语言实现 带有史上最全的注释和解释

 17139

史上最全最完整栈应用 C语言 源代码 可直接运行

 15520

The EM(Expectation–Maximization) Algorithm 详解

 10034

严蔚敏 数据结构 课本中 栈应用 走迷宫 C语言 完整版 源代码和注释 可直接执行

 9560

基于半边数据结构（翼边数据结构）的Euler操作来实现扫成 通过OpenGL进行CAD.

 6698

用C++写的矩阵处理函数 包括求逆、转置、乘积等等

 6289

经典算法研究：模式匹配（子串匹配）之KMP 算法（C语言实现版）

 6150

最大连续子序列之和

 5616

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



## 几种不同程序语言的HMM版本

### 分类 隐马尔科夫模型

👍 : 得来终觉浅，绝知此事要躬行”，在继续翻译《[HMM学习最佳范例](#)》之前，这里先补充几个不同程序语言实现的HMM版本，主要参考了维基百科。读者有兴趣的话可以研究一下代码，这样对于HMM的学习会深刻很多！

#### C语言：

##### 1、 HTK ( Hidden Markov Model Toolkit )：

📖 是英国剑桥大学开发的一套基于C语言的隐马尔科夫模型工具箱，主要应用于语音识别、语音合成的研究，也被用在其他领域，如字符识别和DNA排序等。HTK是重量级的HMM版本。

💬 主页：<http://htk.eng.cam.ac.uk/>

##### 2、 GHMM Library：

The General Hidden Markov Model library (GHMM) is a freely available LGPL-ed C library implementing efficient data structures and algorithms for basic and extended HMMs.

GHMM主页：<http://www.ghmm.org/>

##### 3、 UMDHMM ( Hidden Markov Model Toolkit )：

Hidden Markov Model (HMM) Software: Implementation of Forward-Backward, Viterbi, and Baum-Welch algorithms.

这款属于轻量级的HMM版本。

UMDHMM主页：<http://www.kanungo.com/software/software.html>

#### Java版：

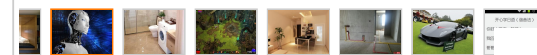
##### 4、 Jahmm Java Library (general-purpose Java library)：

Jahmm (pronounced “jam”), is a Java implementation of Hidden Markov Model (HMM) related algorithms. It's been designed to be easy to use (e.g. simple things are simple to program) and general purpose.

Jahmm主页：<http://code.google.com/p/jahmm/>



## 人工智能网站



## 联系我们



请扫描二维码联系客服

✉ webmaster@csdn.net

☎ 400-660-0108

💬 QQ客服 🗣 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



**Malab版：**

## 5、 Hidden Markov Model (HMM) Toolbox for Matlab：

This toolbox supports inference and learning for HMMs with discrete outputs (dhmm's), Gaussian outputs (ghmm's), or mixture of Gaussians output (mhmm's).

Matlab-HMM主页：<http://www.cs.ubc.ca/~murphyk/Software/HMM/hmm.html>

**Common Lisp版：**

## 6、 Common HMM Library (HMM Library for Common Lisp)：

Simple Hidden Markov Model library for ANSI Common Lisp. Main structures and basic algorithms implemented. Performance comparable to C code. It's licensed under LGPL.

Common HMM主页：<http://www.ashrentum.net/jmcejuela/programs/cl-hmm/>

**Haskell版：**

## 7、 hmm package ( A Haskell library for working with Hidden Markov Models )：

A simple library for working with Hidden Markov Models. Should be usable even by people who are not familiar with HMMs. Includes implementations of Viterbi's algorithm and the forward algorithm.

Haskell-HMM主页：<http://hackage.haskell.org/cgi-bin/hackage-scripts/package/hmm>

注：Haskell是一种纯函数式编程语言，它的命名源自美国数学家Haskell Brooks Curry，他在数学逻辑方面上的工作使得函数式编程语言有了广泛的基础。

是否还有C++版、Perl版或者Python版呢？欢迎补充！

**HMM学习最佳范例一：介绍**

隐马尔科夫模型（HMM）依然是读者访问“我爱自然语言处理”的一个热门相关关键词，我曾在《[HMM学习最佳范例与崔晓源的博客](#)》中介绍过国外的一个不错的[HMM学习教程](#)，并且国内崔晓源师兄有一个相应的[翻译版本](#)，不过这个版本比较简化和粗略，有些地方只是概况性的翻译了一下，省去了一些内容，所以从今天开始计划在52nlp上系统的重新翻译这个学习教程，希望对大家有点用。

列，句子中的词语顺序和口语单词中的音素序列等等，事实上任何领域中的一系列事件都有可能产生有用的模式。

考虑一个简单的例子，有人试图通过一片海藻推断天气——民间传说告诉我们‘湿透的’海藻意味着潮湿阴雨，而‘干燥的’海藻则意味着阳光灿烂。如果它处于一个中间状态（‘有湿气’），我们就无法确定天气如何。然而，天气的状态并没有受限于海藻的状态，所以我们可以观察的基础上预测天气是雨天或晴天的可能性。另一个有用的线索是前一天的天气状态（或者，至少是它的可能状态）——通过综合昨天的天气及相应观察到的海藻状态，我们有可能更好的预测今天的天气。



本教程中我们将考虑的一个典型的系统类型。

10

首先，我们将介绍产生概率模式的系统，如晴天及雨天间的天气波动。



，我们将会看到这样一个系统，我们希望预测的状态并不是观察到的——其底层系统是隐藏的。在上面的例子中，观察到的序列将是海藻而隐藏的系统将是实际的天气。



，我们会利用已经建立的模型解决一些实际的问题。对于上述例子，我们想知道：

1. 给出一个星期每天的海藻观察状态，之后的天气将会是什么？



定一个海藻的观察状态序列，预测一下此时是冬季还是夏季？直观地，如果一段时间内海藻都是干燥的，那么这段时间很可能是夏季，反之，如果一段时间内海藻都是潮湿的，那么这段时间可能是冬季。

## 二、生成模式（Generating Patterns）

### 1、确定性模式（Deterministic Patterns）

考虑一套交通信号灯，灯的颜色变化序列依次是红色-红色/黄色-绿色-黄色-红色。这个序列可以作为一个状态机器，交通信号灯的不同状态都紧跟着上一个状态。

注意每一个状态都是唯一的依赖于前一个状态，所以，如果交通灯为绿色，那么下一个颜色状态将始终是黄色——也就是说，该系统是确定性的。确定性系统相对比较容易理解和分析，因为状态间的转移是完全已知的。

### 2、非确定性模式（Non-deterministic patterns）

为了使天气那个例子更符合实际，加入第三个状态——多云。与交通信号灯例子不同，我们并不期望这三个天气状态之间的变化是确定性的，但是我们依然希望对这个系统建模以便生成一个天气变化模式（规律）。

一种做法是假设模型的当前状态仅仅依赖于前面的几个状态，这被称为马尔科夫假设，它极大地简化了问题。显然，



譬如风力、气压等则没有考虑。在这个例子以及其他相似的例子中，这样的假设显然是不现实的。然而，由于这样经过简化的系统可以用来分析，我们常常接受这样的知识假设，虽然它产生的某些信息不完全准确。

一个马尔科夫过程是状态间的转移仅依赖于前 $n$ 个状态的过程。这个过程被称之为 $n$ 阶马尔科夫模型，其中 $n$ 是影响下一个状态选择的（前） $n$ 个状态。最简单的马尔科夫过程是一阶模型，它的状态选择仅与前一个状态有关。这里要注意它与确定性系统并不相同，因为下一个状态的选择由相应的概率决定，并不是确定性的。



是天气例子中状态间所有可能的一阶状态转移情况：

10

有 $M$ 个状态的一阶马尔科夫模型，共有 $M^2$ 个状态转移，因为任何一个状态都有可能是所有状态的下一个转移状态。每一个状态转移都有一个概率值，称为状态转移概率——这是从一个状态转移到另一个状态的概率。所有的 $M^2$ 个概率可以用一个 $M \times M$ 转移矩阵表示。注意这些概率并不随时间变化而不同——这是一个非常重要（但常常不符合实际）的假设。

下面的状态转移矩阵显示的是天气例子中可能的状态转移概率：



也就是说，如果昨天是晴天，那么今天是晴天的概率为0.5，是多云的概率为0.375。注意，每一行的概率之和为1。要初始化这样一个系统，我们需要确定起始日天气的（或可能的）情况，定义其为一个初始概率向量，称为向量。

也就是说，第一天为晴天的概率为1。

现在我们定义一个一阶马尔科夫过程如下：

**状态：**三个状态——晴天，多云，雨天。

**向量：**定义系统初始化时每一个状态的概率。

**状态转移矩阵：**给定前一天天气情况下的当前天气概率。

任何一个可以用这种方式描述的系统都是一个马尔科夫过程。

### 3、总结

我们尝试识别时间变化中的模式，并且为了达到这个目我们试图对这个过程建模以便产生这样的模式。我们使用了离散时间点、离散状态以及做了马尔科夫假设。在采用了这些假设之后，系统产生了这个被描述为马尔科夫过程的模式，它包含了一个向量（初始概率）和一个状态转移矩阵。关于假设，重要的一点是状态转移矩阵并不随时间的改变而改变——

这个矩阵在整个系统的生命周期中是不变的

## 1、马尔科夫过程的局限性

在某些情况下，我们希望找到的模式用马尔科夫过程描述还显得不充分。回顾一下天气那个例子，一个隐士也许不能够直接获取到天气的观察情况，但是他有一些水藻。民间传说告诉我们水藻的状态与天气状态有一定的概率关系——天气和水藻的状态是紧密相关的。在这个例子中我们有两组状态，观察的状态（水藻的状态）和隐藏的状态（天气的状态）。我们希望为隐士设计一种算法，在不能够直接观察天气的情况下，通过水藻和马尔科夫假设来预测天气。

一个更实际的问题是语音识别，我们听到的声音是来自于声带、喉咙大小、舌头位置以及其他一些东西的组合结果。所以，因素相互作用产生一个单词的声音，一套语音识别系统检测的声音就是来自于个人发音时身体内部物理变化所引起的不断改变的声音。

语音识别装置工作的原理是将内部的语音产出看作是隐藏的状态，而将声音结果作为一系列观察的状态，这些由语音生成并且最好的近似了实际（隐藏）的状态。在这两个例子中，需要着重指出的是，隐藏状态的数目与观察状态的数目是不同的。一个包含三个状态的天气系统（晴天、多云、雨天）中，可以观察到4个等级的海藻湿润情况（干、稍干、潮湿、湿润）；纯粹的语音可以由80个音素描述，而身体的发音系统会产生出不同数目的声音，或者比80多，或者比80少。

在这种情况下，观察到的状态序列与隐藏过程有一定的概率关系。我们使用隐马尔科夫模型对这样的过程建模，这个模型包含了一个底层隐藏的随时间改变的马尔科夫过程，以及一个与隐藏状态某种程度相关的可观察到的状态集合。

## 2、隐马尔科夫模型（Hidden Markov Models）

下图显示的是天气例子中的隐藏状态和观察状态。假设隐藏状态（实际的天气）由一个简单的一阶马尔科夫过程描述，那么它们之间都相互连接。

隐藏状态和观察状态之间的连接表示：在给定的马尔科夫过程中，一个特定的隐藏状态生成特定的观察状态的概率。这很清晰的表示了‘进入’一个观察状态的所有概率之和为1，在上面这个例子中就是 $\Pr(\text{Obs}|\text{Sun})$ ,  $\Pr(\text{Obs}|\text{Cloud})$  及  $\Pr(\text{Obs}|\text{Rain})$ 之和。（对这句话我有点疑惑？）

除了定义了马尔科夫过程的概率关系，我们还有另一个矩阵，定义为混淆矩阵（confusion matrix），它包含了给定一个隐藏状态后得到的观察状态的概率。对于天气例子，混淆矩阵是：

注意矩阵的每一行之和是1。



以和隐藏状态的数码不同。

我们使用一个隐马尔科夫模型（HMM）对这些例子建模。这个模型包含两组状态集合和三组概率集合：

\* 隐藏状态：一个系统的（真实）状态，可以由一个马尔科夫过程进行描述（例如，天气）。

\* 观察状态：在这个过程中‘可视’的状态（例如，海藻的湿度）。

\* 向量：包含了（隐）模型在时间 $t=1$ 时一个特殊的隐藏状态的概率（初始概率）。

\* 状态转移矩阵：包含了一个隐藏状态到另一个隐藏状态的概率

 输出矩阵：包含了给定隐马尔科夫模型的某一个特殊的隐藏状态，观察到的某个观察状态的概率。

因此一个隐马尔科夫模型是在一个标准的马尔科夫过程中引入一组观察状态，以及其与隐藏状态间的一些概率关系。

#### 四、 尔科夫模型（Hidden Markov Models）

##### 1、 Definition of a hidden Markov model）

一个隐马尔科夫模型是一个三元组（ $\lambda$ , A, B）。

 始化概率向量；

：状态转移矩阵；

：混淆矩阵；

在状态转移矩阵及混淆矩阵中的每一个概率都是时间无关的——也就是说，当系统演化时这些矩阵并不随时间改变。

实际上，这是马尔科夫模型关于真实世界最不现实的一个假设。

##### 2、应用（Uses associated with HMMs）

一旦一个系统可以作为HMM被描述，就可以用来解决三个基本问题。其中前两个是模式识别的问题：给定HMM求一个观察序列的概率（评估）；搜索最有可能生成一个观察序列的隐藏状态训练（解码）。第三个问题是给定观察序列生成一个HMM（学习）。

###### a) 评估（Evaluation）

考虑这样的问题，我们有一些描述不同系统的隐马尔科夫模型（也就是一些 $(\lambda, A, B)$ 三元组的集合）及一个观察序列。我们想知道哪一个HMM最有可能产生了这个给定的观察序列。例如，对于海藻来说，我们也许会有一个“夏季”模型和一个“冬季”模型，因为不同季节之间的情况是不同的——我们也许想根据海藻湿度的观察序列来确定当前的季节。


我们使用前向算法（forward algorithm）来计算给定隐马尔科夫模型（HMM）后的一个观察序列的概率，并因此选择


建模时。一个观察序列从一个发音单词中形成，并且通过寻找对于此观察序列最有可能的隐马尔科夫模型（HMM）识别这个单词。


b) 解码（Decoding）

**给定观察序列搜索最可能的隐藏状态序列。**

另一个相关问题，也是最感兴趣的一个，就是搜索生成输出序列的隐藏状态序列。在许多情况下我们对于模型中的隐藏状态更感兴趣，因为它们代表了一些更有价值的东西，而这些东西通常不能直接观察到。

 海藻和天气这个例子，一个盲人隐士只能感觉到海藻的状态，但是他更想知道天气的情况，天气状态在这里就是隐藏状态。

 使用Viterbi 算法（Viterbi algorithm）确定（搜索）已知观察序列及HMM下最可能的隐藏状态序列。

 Viterbi算法（Viterbi algorithm）的另一广泛应用是自然语言处理中的词性标注。在词性标注中，句子中的单词是观察状态。词性（语法类别）是隐藏状态（注意对于许多单词，如wind，fish拥有不止一个词性）。对于每句话中的单词，通过搜索最可能的隐藏状态，我们就可以在给定的上下文中找到每个单词最可能的词性标注。

c) 学习（Learning）

**根据观察序列生成隐马尔科夫模型。**

第三个问题，也是与HMM相关的问题中最难的，根据一个观察序列（来自于已知的集合），以及与其有关的一个隐藏状态集，估计一个最合适的隐马尔科夫模型（HMM），也就是确定对已知序列描述的最合适的 $(A, B)$ 三元组。

当矩阵A和B不能够直接被（估计）测量时，前向-后向算法（forward-backward algorithm）被用来进行学习（参数估计），这也是实际应用中常见的情况。

3、总结（Summary）

由一个向量和两个矩阵 $(A, B)$ 描述的隐马尔科夫模型对于实际系统有着巨大的价值，虽然经常只是一种近似，但它们却是经得起分析的。隐马尔科夫模型通常解决的问题包括：

1. 对于一个观察序列匹配最可能的系统——评估，使用前向算法（forward algorithm）解决；
2. 对于已生成的一个观察序列，确定最可能的隐藏状态序列——解码，使用Viterbi 算法（Viterbi algorithm）解决；
3. 对于已生成的观察序列，决定最可能的模型参数——学习，使用前向-后向算法（forward-backward algorithm）解决。

**五、前向算法（Forward Algorithm）**

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

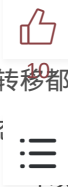
登录

注册



## 1. 穷举搜索 ( Exhaustive search for solution )

给定隐马尔科夫模型，也就是在模型参数 ( , A, B) 已知的情况下，我们想找到观察序列的概率。还是考虑天气这个例子，我们有一个用来描述天气及与它密切相关的海藻湿度状态的隐马尔科夫模型(HMM)，另外我们还有一个海藻的湿度状态观察序列。假设连续3天海藻湿度的观察结果是 ( 干燥、湿润、湿透 ) ——而这三天每一天都可能是晴天、多云或下雨，对于观察序列以及隐藏的状态，可以将其视为网格：

 中的每一列都显示了可能的天气状态，并且每一列中的每个状态都与相邻列中的每一个状态相连。而其状态间的转移都由状态转移矩阵提供一个概率。在每一列下面都是某个时间点上的观察状态，给定任一个隐藏状态所得到的观察状态概率由混淆矩阵提供。

可以看出，一种计算观察序列概率的方法是找到每一个可能的隐藏状态，并且将这些隐藏状态下的观察序列概率相加。上面那个 ( 天气 ) 例子，将有  $3^3 = 27$  种不同的天气序列可能性，因此，观察序列的概率是：

$$\Pr(\text{dry,damp,soggy} \mid \text{HMM}) = \Pr(\text{dry,damp,soggy} \mid \text{sunny,sunny,sunny}) + \Pr(\text{dry,damp,soggy} \mid \text{sunny,sunny,cloudy}) + \Pr(\text{dry,damp,soggy} \mid \text{sunny,sunny,rainy}) + \dots + \Pr(\text{dry,damp,soggy} \mid \text{rainy,rainy,rainy})$$

用这种方式计算观察序列概率极为昂贵，特别对于大的模型或较长的序列，因此我们可以利用这些概率的时间不变性来减少问题的复杂度。

## 2. 使用递归降低问题复杂度

给定一个隐马尔科夫模型 ( HMM )，我们将考虑递归地计算一个观察序列的概率。我们首先定义局部概率 ( partial probability )，它是到达网格中的某个中间状态时的概率。然后，我们将介绍如何在  $t=1$  和  $t=n(>1)$  时计算这些局部概率。

假设一个  $T$ -长观察序列是：

### 2a. 局部概率('s)

考虑下面这个网格，它显示的是天气状态及对于观察序列干燥，湿润及湿透的一阶状态转移情况：

我们可以将计算到达网格中某个中间状态的概率作为所有到达这个状态的可能路径的概率求和问题。

例如， $t=2$  时位于“多云”状态的局部概率通过如下路径计算得出：

$$t(j) = \Pr(\text{观察状态} | \text{隐藏状态}j) \times \Pr(t\text{时刻所有指向}j\text{状态的路径})$$

对于最后的观察状态，其局部概率包括了通过所有可能的路径到达这些状态的概率——例如，对于上述网格，最终的局部概率通过如下路径计算得出：

由此可见，对于这些最终局部概率求和等价于对于网格中所有可能的路径概率求和，也就求出了给定隐马尔科夫模型(HMM)的观察序列概率。



10

给出了一个计算这些概率的动态示例。

2b. 计算 $t=1$ 时的局部概率's



按如下公式计算局部概率：



$$t(j) = \Pr(\text{观察状态} | \text{隐藏状态}j) \times \Pr(t\text{时刻所有指向}j\text{状态的路径})$$



当 $t=1$ 时，没有任何指向当前状态的路径。故 $t=1$ 时位于当前状态的概率是初始概率，即 $\Pr(\text{state}|t=1)=P(\text{state})$ ，因此

的局部概率等于当前状态的初始概率乘以相关的观察概率：



所以初始时刻状态 $j$ 的局部概率依赖于此状态的初始概率及相应时刻我们所见的观察概率。

2c. 计算 $t>1$ 时的局部概率's

我们再次回顾局部概率的计算公式如下：

$$t(j) = \Pr(\text{观察状态} | \text{隐藏状态}j) \times \Pr(t\text{时刻所有指向}j\text{状态的路径})$$

我们可以假设（递归地），乘号左边项“ $\Pr(\text{观察状态} | \text{隐藏状态}j)$ ”已经有了，现在考虑其右边项“ $\Pr(t\text{时刻所有指向}j\text{状态的路径})$ ”。

为了计算到达某个状态的所有路径的概率，我们可以计算到达此状态的每条路径的概率并对它们求和，例如：

计算所需要的路径数目随着观察序列的增加而指数级递增，但是 $t-1$ 时刻's给出了所有到达此状态的前一路径概率，因此，我们可以通过 $t-1$ 时刻的局部概率定义 $t$ 时刻's，即：

故我们所计算的这个概率等于相应的观察概率（亦即， $t+1$ 时在状态 $j$ 所观察到的符号的概率）与该时刻到达此状态的概率总和——这来自于上一步每一个局部概率的计算结果与相应的状态转移概率乘积后再相加——的乘积。

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

[登录](#)
[注册](#)


$t=2$ 时刻的's，通过 $t=2$ 时刻的's计算 $t=3$ 时刻的's等等直到 $t=T$ 。给定隐马尔科夫模型(HMM)的观察序列的概率就等于 $t=T$ 时刻的局部概率之和。

## 2d.降低计算复杂度

我们可以比较通过穷举搜索（评估）和通过递归前向算法计算观察序列概率的时间复杂度。

我们有一个长度为 $T$ 的观察序列 $O$ 以及一个含有 $n$ 个隐藏状态的隐马尔科夫模型 $l=(A,B)$ 。



10

公式



公式

我们所观察到的概率求和——注意其复杂度与 $T$ 成指数级关系。相反的，使用前向算法我们可以利用上一步计算的信息应地，其时间复杂度与 $T$ 成线性关系。

注意：穷举搜索的时间复杂度是，前向算法的时间复杂度是，其中 $T$ 指的是观察序列长度， $N$ 指的是隐藏状态数目。



## 3.总结

我们的目标是计算给定隐马尔科夫模型HMM下的观察序列的概率—— $\Pr(\text{observations } |)$ 。

我们首先通过计算局部概率（'s）降低计算整个概率的复杂度，局部概率表示的是 $t$ 时刻到达某个状态 $s$ 的概率。

$t=1$ 时，可以利用初始概率(来自于 $P$ 向量)和观察概率 $\Pr(\text{observation}|\text{state})$ （来自于混淆矩阵）计算局部概率；而 $t>1$ 时的局部概率可以利用 $t-1$ 时的局部概率计算。

因此，这个问题是递归定义的，观察序列的概率就是通过依次计算 $t=1,2,\dots,T$ 时的局部概率，并且对于 $t=T$ 时所有局部概率's相加得到的。

注意，用这种方式计算观察序列概率的时间复杂度远远小于计算所有序列的概率并对其相加（穷举搜索）的时间复杂度。

我们使用前向算法计算 $T$ 长观察序列的概率:

其中 $y$ 的每一个是观察集合之一。局部（中间）概率('s)是递归计算的，首先通过计算 $t=1$ 时刻所有状态的局部概率：

然后对于每个 $t=2, \dots, T$ ，计算每个状态的局部概率，并求和得到下一个时间步的局部概率。

也就是当前状态相应的观察概率与所有到达该状态的路径概率之积，其递归地利用了上一个时间点已经计算好的一些值。

最后，给定HMM, 观察序列的概率等于T时刻所有局部概率之和：

再重复说明一下，每一个局部概率（ $t > 2$  时）都由前一时刻的结果计算得出。

对于“天气”那个例子，下面的图表显示了 $t = 2$ 为状态为多云时局部概率的计算过程。这是相应的观察概率 $b$ 与前一时刻的



10

总结 (Summary)



使用前向算法来计算给定隐马尔科夫模型（HMM）后的一个观察序列的概率。它在计算中利用递归避免对网格所有路径进行穷举计算。



这种算法，可以直接用来确定对于已知的一个观察序列，在一些隐马尔科夫模型（HMMs）中哪一个HMM最好的描述了它——先用前向算法评估每一个（HMM），再选取其中概率最高的一个。

首先需要说明的是，本节不是这个系列的翻译，而是作为前向算法这一章的补充，希望能从实践的角度来说明前向算法。除了用程序来解读hmm的前向算法外，还希望将原文所举例子的的问题拿出来和大家探讨。

文中所举的程序来自于UMDHMM这个C语言版本的HMM工具包，具体见《[几种不同程序语言的HMM版本](#)》。先说明一下UMDHMM这个包的基本情况，在linux环境下，进入umdhmm-v1.02目录，“make all”之后会产生4个可执行文件，分别是：

**genseq:** 利用一个给定的隐马尔科夫模型产生一个符号序列（Generates a symbol sequence using the specified model sequence using the specified model）

**testfor:** 利用前向算法计算 $\log \text{Prob}(\text{观察序列} | \text{HMM模型})$ （Computes  $\log \text{Prob}(\text{observation} | \text{model})$  using the Forward algorithm.）

**testvit:** 对于给定的观察符号序列及HMM，利用Viterbi 算法生成最可能的隐藏状态序列（Generates the most like state sequence for a given symbol sequence, given the HMM, using Viterbi）

**esthmm:** 对于给定的观察符号序列，利用BaumWelch算法学习隐马尔科夫模型HMM（Estimates the HMM from a given symbol sequence using BaumWelch）

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册





## HMM 文件格式：

---

M= number of symbols

N= number of states

A:

a<sub>11</sub> a<sub>12</sub> ... a<sub>1N</sub>



a<sub>21</sub> a<sub>22</sub> ... a<sub>2N</sub>

10

....



....

....



a<sub>N1</sub> a<sub>N2</sub> ... a<sub>NN</sub>

B:



b<sub>11</sub> b<sub>12</sub> ... b<sub>1M</sub>

b<sub>21</sub> b<sub>22</sub> ... b<sub>2M</sub>

....

....

....

b<sub>N1</sub> b<sub>N2</sub> ... b<sub>NM</sub>

pi:

pi<sub>1</sub> pi<sub>2</sub> ... pi<sub>N</sub>

---

## HMM文件举例：

---

M= 2

N= 3

A:

0.333 0.333 0.333

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



0.333 0.333 0.333

B:

0.5 0.5

0.75 0.25

0.25 0.75

$\pi$ :



0.333 0.333 0.333

10



序列文件格式：



$T$ =sequence length



$o_1 o_2 o_3 \dots o_T$

观察序列文件举例：

$T = 10$

1 1 1 1 2 1 2 2 2 2

对于前向算法的测试程序testfor来说，运行：

```
testfor model.hmm (HMM文件) obs.seq (观察序列文件)
```

就可以得到观察序列的概率结果的对数值，这里我们在testfor.c的第58行对数结果的输出下再加一行输出：

```
fprintf(stdout, "prob(O| model) = %fn", proba);
```

就可以输出运用前向算法计算观察序列所得到的概率值。至此，所有的准备工作已结束，接下来，我们将进入具体的程序解读。

首先，需要定义HMM的数据结构，也就是HMM的五个基本要素，在UMDHMM中是如下定义的（在hmm.h中）：

```

int N; /* 隐藏状态数目;Q={1,2,...,N} */
int M; /* 观察符号数目; V={1,2,...,M}*/
double **A; /* 状态转移矩阵A[1..N][1..N]. a[i][j] 是从t时刻状态i到t+1时刻状态j的转移概率 */
double **B; /* 混淆矩阵B[1..N][1..M]. b[j][k]在状态j时观察到符合k的概率。*/
double *pi; /* 初始向量pi[1..N] , pi[i] 是初始状态概率分布 */
} HMM.

```



前向算法程序示例如下（在forward.c中）：

```

/*
 * 参数说明：
 * 'phmm'：已知的HMM模型；T：观察符号序列长度；
 * 'O'：观察序列；**alpha：局部概率；*pprob：最终的观察概率
 */
void forward(HMM *phmm, int T, int *O, double **alpha, double *pprob)
{
    int i, j; /* 状态索引 */
    int t; /* 时间索引 */
    double sum; /*求局部概率时的中间值 */

    /* 1. 初始化：计算t=1时刻所有状态的局部概率： */
    for (i = 1; i <= phmm->N; i++)
        alpha[1][i] = phmm->pi[i] * phmm->B[i][O[1]];

    /* 2. 归纳：递归计算每个时间点，t=2，...，T时的局部概率 */
    for (t = 1; t < T; t++)
    {
        for (j = 1; j <= phmm->N; j++)
        {

```

```

        sum += alpha[t][i]* (phmm->A[i][j]);
        alpha[t+1][j] = sum*(phmm->B[j][O[t+1]]);
    }
}

/* 3. 终止：观察序列的概率等于T时刻所有局部概率之和*/
double prob = 0.0;
for (t = 1; t <= phmm->N; t++)
    *pprob += alpha[t][i];
}

```



节我将用这个程序来验证英文原文中所举前向算法演示例子的问题。



在4M这个翻译系列的原文中，作者举了一个前向算法的交互例子，这也是这个系列中比较出彩的地方，但是，在具体运行这个例子的时候，却发现其似乎有点问题。

先说一下如何使用这个[交互例子](#)，运行时需要浏览器支持java，我用的是firefox。首先在Set按钮前面的对话框里上观察序列，如“Dry,Damp, Soggy”或“Dry Damp Soggy”，观察符号间用逗号或空格隔开；然后再点击Set按钮，这样就初始化了观察矩阵；如果想得到一个总的结果，即Pr(观察序列|隐马尔科夫模型)，就点旁边的Run按钮；如果想一步一步观察计算过程，即每个节点的局部概率，就单击旁边的Step按钮。

原文交互例子（即天气这个例子）中所定义的已知隐马尔科夫模型如下：

- 1、隐藏状态 (天气)：Sunny，Cloudy，Rainy；
- 2、观察状态（海藻湿度）：Dry，Dryish，Damp，Soggy；
- 3、初始状态概率：Sunny（0.63），Cloudy（0.17），Rainy（0.20）；
- 4、状态转移矩阵：

```

weather today
Sunny Cloudy Rainy

weather Sunny 0.500 0.375 0.125
vesterday Cloudy 0.250 0.125 0.625

```

5、混淆矩阵：

	observed states
	Dry Dryish Damp Soggy
	Sunny 0.60 0.20 0.15 0.05
hidden states	Cloudy 0.25 0.25 0.25 0.25
	Rainy 0.05 0.10 0.35 0.50



10

为了UMDHMM也能运行这个例子，我们将上述天气例子中的隐马尔科夫模型转化为如下的UMDHMM可读的HMM文

件v .hmm：



M= 4

N= 3



A:

0.500 0.375 0.125

0.250 0.125 0.625

0.250 0.375 0.375

B:

0.60 0.20 0.15 0.05

0.25 0.25 0.25 0.25

0.05 0.10 0.35 0.50

pi:

0.63 0.17 0.20

在运行例子之前，如果读者也想观察每一步的运算结果，可以将umdhmm-v1.02目录下forward.c中的void Forward(...)函数替换如下：

```
void Forward(HMM *phmm, int T, int *O, double **alpha, double *pprob)
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



```

int t; /* time index */

double sum; /* partial sum */

/* 1. Initialization */
for (i = 1; i <= phmm->N; i++)
{
    alpha[1][i] = phmm->pi[i] * phmm->B[i][O[1]];
    printf( "a[1][%d] = pi[%d] * b[%d][%d] = %f * %f = %f\n", i, i, O[i], phmm->pi[i], phmm->B[i][O[1]], alpha[1][i] );
}

/* Induction */
for (t = 1; t < T; t++)
{
    for (j = 1; j <= phmm->N; j++)
    {
        sum = 0.0;
        for (i = 1; i <= phmm->N; i++)
        {
            sum += alpha[t][i] * (phmm->A[i][j]);
            printf( "a[%d][%d] * A[%d][%d] = %f * %f = %f\n", t, i, i, j, alpha[t][i], phmm->A[i][j], alpha[t][i] * (phmm->A[i][j]));
        }
        printf( "sum = %f\n", sum );
        alpha[t+1][j] = sum * (phmm->B[j][O[t+1]]);
        printf( "a[%d][%d] = sum * b[%d][%d] = %f * %f = %f\n", t+1, j, j, O[t+1], sum, phmm->B[j][O[t+1]], alpha[t+1][j] );
    }
}

```



```

/* 3. Termination */

*pprob = 0.0;
for (i = 1; i <= phmm->N; i++)
{
    *pprob += alpha[T][i];
    printf( "alpha[%d][%d] = %f\n", T, i, alpha[T][i] );
    printf( "pprob = %f\n", *pprob );
}
}

```



完毕之后，重新“make clean”，“make all”，这样新的testfor可执行程序就可以输出前向算法每一步的计算结果。



现在我们就用testfor来运行原文中默认给出的观察序列“Dry,Damp,Soggy”，其所对应的UMDHMM可读的观察序列文件t

est1



T=3

1 3 4

好了，一切准备工作就绪，现在就输入如下命令：

```
testfor weather.hmm test1.seq > result1
```

result1就包含了所有的结果细节：

Forward without scaling

$$a[1][1] = \pi[1] * b[1][1] = 0.630000 * 0.600000 = 0.378000$$

$$a[1][2] = \pi[2] * b[2][3] = 0.170000 * 0.250000 = 0.042500$$

$$a[1][3] = \pi[3] * b[3][4] = 0.200000 * 0.050000 = 0.010000$$

...


**pprob = 0.026901**

...

黑体部分是最终的观察序列的概率结果，即本例中的 $\Pr(\text{观察序列}|\text{HMM}) = 0.026901$ 。

但是，在原文中点Run按钮后，结果却是：Probability of this model = 0.027386915。

这其中的差别到底在哪里？我们来仔细观察一下中间运行过程：


在初始化亦 $t=1$ 时刻的局部概率计算两个是一致的，没有问题。但是， $t=2$ 时，在隐藏状态“Sunny”的局部概率是不一致的。 原文给出的例子的运行结果是：

<sup>10</sup>  
Alpha = (((0.37800002\*0.5) + (0.0425\*0.375) + (0.010000001\*0.125)) \* 0.15) = 0.03092813

 4DHMM给出的结果是：

  $1] * A[1][1] = 0.378000 * 0.500000 = 0.189000$

sum = 0.189000

  $2] * A[2][1] = 0.042500 * 0.250000 = 0.010625$

sum = 0.199625

$a[1][3] * A[3][1] = 0.010000 * 0.250000 = 0.002500$

sum = 0.202125

$a[2][1] = \text{sum} * b[1][3] = 0.202125 * 0.150000 = 0.030319$

区别就在于状态转移概率的选择上，原文选择的是状态转移矩阵中的第一行，而UMDHMM选择的则是状态转移矩阵中的第一列。如果从原文给出的状态转移矩阵来看，第一行代表的是从前一时刻的状态“Sunny”分别到当前时刻的状态“Sunny”，“Cloudy”，“Rainy”的概率；而第一列代表的是从前一时刻的状态“Sunny”，“Cloudy”，“Rainy”分别到当前时刻状态“Sunny”的概率。这样看来似乎原文的计算过程有误，读者不妨多试几个例子看看，前向算法这一章就到此为止了。

HMM学习最佳范例六：维特比算法


寻找最可能的隐藏状态序列(Finding most probable sequence of hidden states)


## 1. 穷举搜索


我们使用下面这张网格图片来形象化的说明隐藏状态和观察状态之间的关系：

我们可以通过列出所有可能的隐藏状态序列并且计算对于每个组合相应的观察序列的概率来找到最可能的隐藏状态序列。最可能的隐藏状态序列是使下面这个概率最大的组合：

$\Pr(\text{观察序列} | \text{隐藏状态的组合})$

，对于网格中所显示的观察序列，最可能的隐藏状态序列是下面这些概率中最大概率所对应的那个隐藏状态序列：  
10

 y,damp,soggy | sunny,sunny,sunny),  $\Pr(\text{dry,damp,soggy} | \text{sunny,sunny,cloudy})$ ,  $\Pr(\text{dry,damp,soggy} | \text{sunny,sunny,rainy})$ , ...  
Pr(sunny,damp,soggy | rainy,rainy,rainy)

 方法是可行的，但是通过穷举计算每一个组合的概率找到最可能的序列是极为昂贵的。与前向算法类似，我们可以利用这些概率的时间不变性来降低计算复杂度。

## 2. 降低复杂度

给定一个观察序列和一个隐马尔科夫模型（HMM），我们将考虑递归地寻找最有可能的隐藏状态序列。我们首先定义局部概率，它是到达网格中的某个特殊的中间状态时的概率。然后，我们将介绍如何在 $t=1$ 和 $t=n(>1)$ 时计算这些局部概率。

这些局部概率与前向算法中所计算的局部概率是不同的，因为它们表示的是时刻 $t$ 时到达某个状态最可能的路径的概率，而不是所有路径概率的总和。

### 2a. 局部概率's和局部最佳途径

考虑下面这个网格，它显示的是天气状态及对于观察序列干燥，湿润及湿透的一阶状态转移情况：

对于网格中的每一个中间及终止状态，都有一个到达该状态的最可能路径。举例来说，在 $t=3$ 时刻的3个状态中的每一个都有一个到达此状态的最可能路径，或许是这样的：

我们称这些路径局部最佳路径(partial best paths)。其中每个局部最佳路径都有一个相关联的概率，即局部概率或。与前向算法中的局部概率不同，是到达该状态（最可能）的一条路径的概率。

因而 $(i,t)$ 是 $t$ 时刻到达状态 $i$ 的所有序列概率中最大的概率，而局部最佳路径是得到此最大概率的隐藏状态序列。对于每一个可能的 $(i,t)$ 值来说，这一概率（及局部最佳路径）均存在

特别地，在 $t=T$ 时每一个状态都有一个局部概率和一个局部最佳路径。这样我们就可以通过选择此时刻包含最大局部概率的状态及其相应的局部最佳路径来确定全局最佳路径（最佳隐藏状态序列）。

## 2b. 计算 $t=1$ 时刻的局部概率's

我们计算的局部概率是作为最可能到达我们当前位置的路径的概率（已知的特殊知识如观察概率及前一个状态的概率）。当 $t=1$ 的时候，到达某状态的最可能路径明显是不存在的；但是，我们使用 $t=1$ 时的所处状态的初始概率及相应的观察概率的观察概率计算局部概率；即



10

——与前向算法类似，这个结果是通过初始概率和相应的观察概率相乘得出的。



## 2c. 计算 $t>1$ 时刻的局部概率's



我们来展示如何利用 $t-1$ 时刻的局部概率计算 $t$ 时刻的局部概率。

考虑如下的网格：



我们考虑计算 $t$ 时刻到达状态 $X$ 的最可能的路径；这条到达状态 $X$ 的路径将通过 $t-1$ 时刻的状态 $A$ ， $B$ 或 $C$ 中的某一个。

因此，最可能的到达状态 $X$ 的路径将是下面这些路径的某一个

（状态序列），...， $A$ ， $X$

（状态序列），...， $B$ ， $X$

或 （状态序列），...， $C$ ， $X$

我们想找到路径末端是 $AX$ 、 $BX$ 或 $CX$ 并且拥有最大概率的路径。

回顾一下马尔科夫假设：给定一个状态序列，一个状态发生的概率只依赖于前 $n$ 个状态。特别地，在一阶马尔可夫假设下，状态 $X$ 在一个状态序列后发生的概率只取决于之前的一个状态，即

$\Pr(\text{到达状态A最可能的路径}) \cdot \Pr(X | A) \cdot \Pr(\text{观察状态} | X)$

与此相同，路径末端是 $AX$ 的最可能的路径将是到达 $A$ 的最可能路径再紧跟 $X$ 。相似地，这条路径的概率将是：

$\Pr(\text{到达状态A最可能的路径}) \cdot \Pr(X | A) \cdot \Pr(\text{观察状态} | X)$


因此，到达状态 $X$ 的最可能路径概率是：


其中第一项是 $t-1$ 时刻的局部概率，第二项是状态转移概率以及第三项是观察概率。

这里，我们假设前一个状态的知识（局部概率）是已知的，同时利用了状态转移概率和相应的观察概率之积。然后，我们就可以在其中选择最大的概率了（局部概率）。

## 2d.反向指针，'s

考虑下面这个网格

 一个中间及终止状态我们都知道了局部概率， $(i, t)$ 。然而我们的目标是在给定一个观察序列的情况下寻找网格中最可能隐藏状态序列——因此，我们需要一些方法来记住网格中的局部最佳路径。

 回顾一下我们是如何计算局部概率的，计算 $t$ 时刻的's我们仅仅需要知道 $t-1$ 时刻的's。在这个局部概率计算之后，就有可能知道前一时刻哪个状态生成了 $(i, t)$ ——也就是说，在 $t-1$ 时刻系统必须处于某个状态，该状态导致了系统在 $t$ 时刻到达状态 $i$ 的。这种记录（记忆）是通过对每一个状态赋予一个反向指针完成的，这个指针指向最优的引发当前状态的前一时刻一个状态。

 上，我们可以写成如下的公式

其中 $\text{argmax}$ 运算符是用来计算使括号中表达式的值最大的索引 $j$ 的。

请注意这个表达式是通过前一个时间步骤的局部概率's和转移概率计算的，并不包括观察概率（与计算局部概率's本身不同）。这是因为我们希望这些's能回答这个问题“如果我在这里，最可能通过哪条路径到达下一个状态？”——这个问题与隐藏状态有关，因此与观察概率有关的混淆（矩阵）因子是可以被忽略的。

## 2e.维特比算法的优点

使用Viterbi算法对观察序列进行解码有两个重要的优点：

1. 通过使用递归减少计算复杂度——这一点和前向算法使用递归减少计算复杂度是完全类似的。
2. 维特比算法有一个非常有用的性质，就是对于观察序列的整个上下文进行了最好的解释（考虑）。事实上，寻找最可能的隐藏状态序列不止这一种方法，其他替代方法也可以，譬如，可以这样确定如下的隐藏状态序列：

其中

这里，采用了“自左向右”的决策方式进行一种近似的判断，其对于每个隐藏状态的判断是建立在前一个步骤的判断的

相反，维特比算法在确定最可能的终止状态前将考虑整个观察序列，然后通过指针“回溯”以确定某个隐藏状态是否是最可能的隐藏状态序列中的一员。这是非常有用的，因为这样就可以孤立序列中的“噪音”，而这些“噪音”在实时数据中是很常见的。

### 3.小结

维特比算法提供了一种有效的计算方法来分析隐马尔科夫模型的观察序列，并捕获最可能的隐藏状态序列。它利用递归计算量，并使用整个序列的上下文来做判断，从而对包含“噪音”的序列也能进行良好的分析。

在使用时，维特比算法对于网格中的每一个单元(cell)都计算一个局部概率，同时包括一个反向指针用来指示最可能的到达该单元的路径。当完成整个计算过程后，首先在终止时刻找到最可能的状态，然后通过反向指针回溯到 $t=1$ 时刻，这样回溯上的状态序列就是最可能的隐藏状态序列了。

#### 1、算法的形式化定义

维特比算法可以形式化的概括为：

令：每一个 $i, i = 1, \dots, n$ ，令：

——这一步是通过隐藏状态的初始概率和相应的观察概率之积计算了 $t=1$ 时刻的局部概率。

对于 $t=2, \dots, T$ 和 $i=1, \dots, n$ ，令：

——这样就确定了到达下一个状态的最可能路径，并对如何到达下一个状态做了记录。具体来说首先通过考察所有的转移概率与上一步获得的最大的局部概率之积，然后记录下其中最大的一个，同时也包含了上一步触发此概率的状态。

令：

——这样就确定了系统完成时( $t=T$ )最可能的隐藏状态。

对于 $t=T-1, \dots, 1$

令：

——这样就可以按最可能的状态路径在整个网格回溯。回溯完成时，对于观察序列来说，序列 $i_1 \dots i_T$ 就是生成此观察序列的最可能的隐藏状态序列。



## 2. 计算单独的's和's

维特比算法中的局部概率's的计算与前向算法中的局部概率's的很相似。下面这幅图表显示了's和's的计算细节，可以对比一下[前向算法3](#)中的计算局部概率's的那幅图表：

唯一不同的是前向算法中计算局部概率's时的求和符号( $\sum$ )在维特比算法中计算局部概率's时被替换为 $\max$ ——这一个重要的不同也说明了在维特比算法中我们选择的是到达当前状态的最可能路径，而不是总的概率。我们在维特比算法中维护一个“反向指针”记录了到达当前状态的最佳路径，即在计算's时通过 $\text{argmax}$ 运算符获得。

10

总结(Summary)

对于一个特定的隐马尔科夫模型，维特比算法被用来寻找生成一个观察序列的最可能的隐藏状态序列。我们利用概率的归一化变体，通过避免计算网格中每一条路径的概率来降低问题的复杂度。维特比算法对于每一个状态( $t > 1$ )都保存了一个反向指针( $\text{backpointer}$ )，并在每一个状态中存储了一个局部概率( $\text{local\_prob}$ )。

局部概率是由反向指针指示的路径到达某个状态的概率。

当 $t=T$ 时，维特比算法所到达的这些终止状态的局部概率's是按照最优（最可能）的路径到达该状态的概率。因此，选择其中最大的一个，并回溯找出所隐藏的状态路径，就是这个问题的最佳答案。

关于维特比算法，需要着重强调的一点是它不是简单的对于某个给定的时间点选择最可能的隐藏状态，而是基于全局序列做决策——因此，如果在观察序列中有一个“非寻常”的事件发生，对于维特比算法的结果也影响不大。

这在语音处理中是特别有价值的，譬如当某个单词发音的一个中间音素出现失真或丢失的情况时，该单词也可以被识别出来。

仍然需要说明的是，本节不是这个系列的翻译，而是作为维特比算法这一章的补充，将UMDHMM这个C语言版本的HMM工具包中的维特比算法程序展示给大家，并运行包中所附带的例子。关于UMDHMM这个工具包的介绍，大家可以参考[前向算法4](#)中的介绍。

**维特比算法程序示例如下（在viterbi.c中）：**

```
void Viterbi(HMM *phmm, int T, int *O, double **delta, int **psi, int *q, double *pprob)
{
    int i, j; /* state indices */
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

[登录](#)
[注册](#)


```
int maxvalind;

double maxval, val;

/* 1. Initialization */

for (i = 1; i <= phmm->N; i++)
{
    delta[1][i] = phmm->pi[i] * (phmm->B[i][O[1]]);
    psi[1][i] = 0;
}

/* Recursion */
for (t = 2; t <= T; t++)
{
    for (j = 1; j <= phmm->N; j++)
    {
        maxval = 0.0;
        maxvalind = 1;
        for (i = 1; i <= phmm->N; i++)
        {
            val = delta[t-1][i]*(phmm->A[i][j]);
            if (val > maxval)
            {
                maxval = val;
                maxvalind = i;
            }
        }

        delta[t][j] = maxval*(phmm->B[j][O[t]]);
    }
}
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



```

    }

}

/* 3. Termination */

*pprob = 0.0;
*best = 1;
for (i = 1; i <= phmm->N; i++)
{
    if (delta[T][i] > *pprob)
    {
        *pprob = delta[T][i];
        q[T] = i;
    }
}

/* 4. Path (state sequence) backtracking */

for (t = T - 1; t >= 1; t--)
    q[t] = psi[t+1][q[t+1]];
}

```

在UMDHMM包中所生成的4个可执行程序中，testvit是用来测试维特比算法的，对于给定的观察符号序列及HMM，利用Viterbi 算法生成最可能的隐藏状态序列。这里我们利用UMDHMM包中test.hmm和test.seq来测试维特比算法，关于这两个文件，具体如下：

**test.hmm：**

---

M= 2

N= 2

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



0.333 0.333 0.333

0.333 0.333 0.333

0.333 0.333 0.333

B:

0.5 0.5

0.75 0.25



10

0.25 0.75

pi:



0.333 0.333 0.333



eq :



T= 10

1 1 1 1 2 1 2 2 2 2

对于维特比算法的测试程序testvit来说，运行：

```
testvit test.hmm test.seq
```

结果如下：

Viterbi using direct probabilities

Viterbi MLE log prob = -1.387295E+01

Optimal state sequence:

T= 10

2 2 2 2 3 2 3 3 3 3

Viterbi using log probabilities

T= 10

2 2 2 2 3 2 3 3 3 3

The two log probabilities and optimal state sequences  
should be identical (within numerical precision).



10

“2 2 2 2 3 2 3 3 3 3”就是最终所找到的隐藏状态序列。好了，维特比算法这一章就到此为止了。

HMM学习最佳范例七：前向-后向算法1



分类 隐马尔科夫模型



七、前向-后向算法(Forward-backward algorithm)



根 序列生成隐马尔科夫模型(Generating a HMM from a sequence of observations)

与HMM模型相关的“有用”的问题是评估（前向算法）和解码（维特比算法）——它们一个被用来测量一个模型的相对适用性，另一个被用来推测模型隐藏的部分在做什么（“到底发生了什么”）。可以看出它们都依赖于隐马尔科夫模型（HMM）参数这一先验知识——状态转移矩阵，混淆（观察）矩阵，以及向量（初始化概率向量）。

然而，在许多实际问题的情况下这些参数都不能直接计算的，而要需要进行估计——这就是隐马尔科夫模型中的学习问题。前向-后向算法就可以以一个观察序列为基础来进行这样的估计，而这个观察序列来自于一个给定的集合，它所代表的是一个隐马尔科夫模型中的一个已知的隐藏集合。

一个例子可能是一个庞大的语音处理数据库，其底层的语音可能由一个马尔可夫过程基于已知的音素建模的，而其可以观察的部分可能由可识别的状态（可能通过一些矢量数据表示）建模的，但是没有（直接）的方式来获取隐马尔科夫模型（HMM）参数。

前向-后向算法并非特别难以理解，但自然地比前向算法和维特比算法更复杂。由于这个原因，这里就不详细讲解前向-后向算法了（任何有关HMM模型的参考文献都会提供这方面的资料的）。

总之，前向-后向算法首先对于隐马尔科夫模型的参数进行一个初始的估计（这很可能是完全错误的），然后通过对于给定的数据评估这些参数的价值并减少它们所引起的错误来重新修订这些HMM参数。从这个意义上讲，它是以一种梯度

型的近似估计)，又计算生成此模型最终状态的“后向”概率（给定当前模型的近似估计）。这些都可以利用递归进行有利地计算，就像我们已经看到的。可以通过利用近似的HMM模型参数来提高这些中间概率进行调整，而这些调整又形成了前向-后向算法迭代的基础。

注：关于前向-后向算法，原文只讲了这么多，后继我将按自己的理解补充一些内容。

了解前向-后向算法，首先需要了解两个算法：后向算法和EM算法。后向算法是必须的，因为前向-后向算法就是利用前向算法与后向算法中的变量因子，其得名也由于此；而EM算法不是必须的，不过由于前向-后向算法是EM算法的一个特例，因此了解一下EM算法也是有好处的，说实话，对于EM算法，我也是云里雾里的。好了，废话少说，我们先谈谈后向算法。

### 1、后向算法 (Backward algorithm)

如果理解了前向算法，后向算法也是比较好理解的，这里首先重新定义一下前向算法中的局部概率 $a_t(i)$ ，称其为前向概率，这也是为前向-后向算法做点准备：

相似地，我们也可以定义一个后向变量 $B_t(i)$ （同样可以理解为一个局部概率）：

后向变量(局部概率)表示的是已知隐马尔科夫模型及 $t$ 时刻位于隐藏状态 $S_i$ 这一事实，从 $t+1$ 时刻到终止时刻的局部观察序列的概率。同样与前向算法相似，我们可以从后向前（故称之为后向算法）递归地计算后向变量：

1) 初始化，令 $t=T$ 时刻所有状态的后向变量为1：

2) 归纳，递归计算每个时间点， $t=T-1, T-2, \dots, 1$ 时的后向变量：

这样就可以计算每个时间点上所有的隐藏状态所对应的后向变量，如果需要利用后向算法计算观察序列的概率，只需将 $t=1$ 时刻的后向变量（局部概率）相加即可。下图显示的是 $t+1$ 时刻与 $t$ 时刻的后向变量之间的关系：

上述主要参考自HMM经典论文《A tutorial on Hidden Markov Models and selected applications in speech recognition》。下面我们给出利用后向算法计算观察序列概率的程序示例，这个程序仍然来自于UMDHMM。



```

void Backward(HMM *phmm, int T, int *O, double **beta, double *pprob)
{
    int i, j; /* state indices */
    int t; /* time index */
    double sum;

    /* 1. Initialization */
    for (i = 1; i <= phmm->N; i++)
        beta[T][i] = 1.0;

    /* 2. Induction */
    for (t = T - 1; t >= 1; t--)
    {
        for (i = 1; i <= phmm->N; i++)
        {
            sum = 0.0;
            for (j = 1; j <= phmm->N; j++)
                sum += phmm->A[i][j] *
                    (phmm->B[j][O[t+1]]) * beta[t+1][j];

            beta[t][i] = sum;
        }
    }

    /* 3. Termination */
    *pprob = 0.0;
    for (i = 1; i <= phmm->N; i++)
        *pprob += beta[1][i];
}

```

前向-后向算法是Baum于1972年提出来的，又称之为Baum-Welch算法，虽然它是EM(Expectation-Maximization)算法的一个特例，但EM算法却是于1977年提出的。那么为什么说前向-后向算法是EM算法的一个特例呢？这里有两点需要说明一下。

第一，1977年A. P. Dempster、N. M. Laird、D. B. Rubin在其论文“Maximum Likelihood from Incomplete Data via the EM Algorithm”中首次提出了EM算法的概念，但是他们也在论文的介绍中提到了在此之前就有一些学者利用了EM算法的思想解决了一些特殊问题，其中就包括了Baum在70年代初期的相关工作，只是这类方法没有被总结而已，他们的工作就是对这类解决问题的方法在更高的层次上定义了一个完整的EM算法框架。

对于前向-后向算法与EM算法的关系，此后在许多与HMM或EM相关的论文里都被提及，其中贾里尼克（Jelinek）老先生在1997所著的书“Statistical Methods for Speech Recognition”中对于前向-后向算法与EM算法的关系进行了完整的描述。读者有兴趣的话可以找来这本书读读。

EM算法的讲解，网上有很多，这里我就不献丑了，直接拿目前搜索“EM算法”在Google排名第一的文章做了参考。读者不要拍砖：

EM算法是Dempster, Laird, Rubin于1977年提出的求参数极大似然估计的一种方法，它可以从非完整数据集中对参数进行极大似然估计，是一种非常简单实用的学习算法。这种方法可以广泛地应用于处理缺损数据，截尾数据，带有讨厌数据等所谓的不完全数据(incomplete data)。

假定集合 $Z = (X, Y)$ 由观测数据 $X$ 和未观测数据 $Y$ 组成， $Z = (X, Y)$ 和 $X$ 分别称为完整数据和不完整数据。假设 $Z$ 的联合概率密度被参数化地定义为 $P(X, Y|\theta)$ ，其中 $\theta$ 表示要被估计的参数。 $\theta$ 的最大似然估计是求不完整数据的对数似然函数 $L(X; \theta)$ 的最大值而得到的：

$$L(\theta; X) = \log p(X|\theta) = \int \log p(X, Y|\theta) dY; \quad (1)$$

EM算法包括两个步骤：由E步和M步组成，它是通过迭代地最大化完整数据的对数似然函数 $L_c(X; \theta)$ 的期望来最大化不完整数据的对数似然函数，其中：

$$L_c(X; \theta) = \log p(X, Y|\theta); \quad (2)$$

假设在算法第 $t$ 次迭代后 $\theta$ 获得的估计记为 $\theta(t)$ ，则在 $(t+1)$ 次迭代时，

E-步：计算完整数据的对数似然函数的期望，记为：

$$Q(\theta|\theta(t)) = E\{L_c(\theta; Z)|X; \theta(t)\}; \quad (3)$$

M-步：通过最大化 $Q(\theta|\theta(t))$ 来获得新的 $\theta$ 。

通过交替使用这两个步骤，EM算法逐步改进模型的参数，使参数和训练样本的似然概率逐渐增大，最后终止于一个极

一套参数或者事先粗略地给定某个初始参数 $\lambda_0$ ，确定出对应于这组参数的最可能的状态，计算每个训练样本的可能结果的概率，在当前的状态下再由样本对参数修正，重新估计参数 $\lambda$ ，并在新的参数下重新确定模型的状态，这样，通过多次的迭代，循环直至某个收敛条件满足为止，就可以使得模型的参数逐渐逼近真实参数。

EM算法的主要目的是提供一个简单的迭代算法计算后验密度函数，它的最大优点是简单和稳定，但容易陷入局部最优。



原文见：<http://49805085.spaces.live.com/Blog/cns!145C40DDDB4C6E5!670.entry>

10

注意上面那段粗体字，读者如果觉得EM算法不好理解的话，就记住这段粗体字的意思吧！



后向算法和EM算法的预备知识，下一节我们就正式的谈一谈前向-后向算法。



马尔科夫模型（HMM）的三个基本问题中，第三个HMM参数学习的问题是最难的，因为对于给定的观察序列 $O$ ，没有任何一种方法可以精确地找到一组最优的隐马尔科夫模型参数（ $A$ 、 $B$ 、 $\pi$ ）使 $P(O)$ 最大。因而，学者们退而求其次，不能使 $P(O)$ 全局最优，就寻求使其局部最优（最大化）的解决方法，而前向-后向算法（又称之为Baum-Welch算法）就成了隐马尔科夫模型学习问题的一种替代（近似）解决方法。

我们首先定义两个变量。**给定观察序列 $O$ 及隐马尔科夫模型，定义 $t$ 时刻位于隐藏状态 $S_i$ 的概率变量为：**

回顾一下**第二节**中关于前向变量 $\alpha_t(i)$ 及后向变量 $\beta_t(i)$ 的定义，我们可以很容易地将上式用前向、后向变量表示为：

其中分母的作用是确保：

**给定观察序列 $O$ 及隐马尔科夫模型，定义 $t$ 时刻位于隐藏状态 $S_i$ 及 $t+1$ 时刻位于隐藏状态 $S_j$ 的概率变量为：**

该变量在网格中所代表的关系如下图所示：

同样，该变量也可以由前向、后向变量表示：

而上述定义的两个变量间也存在着如下关系：

如果对于时间轴上的所有相加，我们可以得到一个总和，它可以用微分头几项隐藏状态 $S_i$ 的期望值，网格中的

期望值。相似地，如果对时间轴 $t$ 上求和（从 $t=1$ 到 $t=T-1$ ），那么该和可以被解释为从状态 $S_i$ 到状态 $S_j$ 的状态转移期望值。即：

上一节我们定义了两个变量及相应的期望值，本节我们利用这两个变量及其期望值来重新估计隐马尔科夫模型（HMM）参数， $A$ 及 $B$ ：



10

我们定义当前的HMM模型为，那么可以利用该模型计算上面三个式子的右端；我们再定义重新估计的HMM模型为，上面三个式子的左端就是重估的HMM模型参数。Baum及他的同事在70年代证明了因此如果我们迭代地的计算上面三个式子，由此不断地重新估计HMM的参数，那么在多次迭代后可以得到的HMM模型的一个最大似然估计。不过需要注意，前向-后向算法所得的这个结果（最大似然估计）是一个局部最优解。

前向-后向算法和EM算法的具体关系的解释，大家可以参考HMM经典论文《A tutorial on Hidden Markov Models and selected applications in speech recognition》，这里就不详述了。下面我们给出UMDHMM中的前向-后向算法示例，这个算法比较复杂，这里只取主函数，其中所引用的函数大家如果感兴趣的话可以自行参考UMDHMM。

前向-后向算法程序示例如下（在baum.c中）：

```
void BaumWelch(HMM *phmm, int T, int *O, double **alpha, double **beta, double **gamma, int *pniter, double *plogprobinit, double *plogprobfinal)
{
    int i, j, k;
    int t, l = 0;

    double logprobf, logprobb, threshold;
    double numeratorA, denominatorA;
    double numeratorB, denominatorB;
```

```

deltaprev = 10e-70;

xi = AllocXi(T, phmm->N);

scale = dvector(1, T);

ForwardWithScale(phmm, T, O, alpha, scale, &logprobf);

/* probinit = logprobf; /* log P(O | initial model) */
ForwardWithScale(phmm, T, O, beta, scale, &logprobb);
ComputeGamma(phmm, T, alpha, beta, gamma);

/* routeXi(phmm, T, O, alpha, beta, xi);
logprobprev = logprobf;

/* reestimate frequency of state i in time t=1 */
for (i = 1; i <= phmm->N; i++)
    phmm->pi[i] = .001 + .999*gamma[1][i];

/* reestimate transition matrix and symbol prob in
    each state */
for (i = 1; i <= phmm->N; i++)
{
    denominatorA = 0.0;
    for (t = 1; t <= T - 1; t++)
        denominatorA += gamma[t][i];

    for (j = 1; j <= phmm->N; j++)
    {
        numeratorA = 0.0;

```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



```

        numeratorA += xi[t][i][j];
    phmm->A[i][j] = .001 +
        .999*numeratorA/denominatorA;
}

denominatorB = denominatorA + gamma[T][i];
for (k = 1; k <= phmm->M; k++)
{
    numeratorB = 0.0;
    for (t = 1; t <= T; t++)
    {
        if (O[t] == k)
            numeratorB += gamma[t][i];
    }

    phmm->B[i][k] = .001 +
        .999*numeratorB/denominatorB;
}
}

ForwardWithScale(phmm, T, O, alpha, scale, &logprobf);
BackwardWithScale(phmm, T, O, beta, scale, &logprobb);
ComputeGamma(phmm, T, alpha, beta, gamma);
ComputeXi(phmm, T, O, alpha, beta, xi);

/* compute difference between log probability of
    two iterations */
delta = logprobf - logprobbprev;
logprobbprev = logprobf;

```



```

    }

    while (delta > DELTA); /* if log probability does not
                           change much, exit */

    *pniter = l;
    *plogprobfinal = logprob; /* log P(O|estimated model) */
    xi(xi, T, phmm->N);
    free_dvector(scale, 1, T);
}

```



.后向算法就到此为止了。

## 八、 [Summary)

通常，模式并不是单独的出现，而是作为时间序列中的一个部分——这个过程有时候可以被辅助用来对它们进行识别。在基于时间的进程中，通常都会使用一些假设——一个最常用的假设是进程的状态只依赖于前面N个状态——这样我们就有了一个N阶马尔科夫模型。最简单的例子是 $N = 1$ 。

存在很多例子，在这些例子中进程的状态（模式）是不能够被直接观察的，但是可以非直接地，或者概率地被观察为模式的另外一种集合——这样我们就可以定义一类隐马尔科夫模型——这些模型已被证明在当前许多研究领域，尤其是语音识别领域具有非常大的价值。

在实际的过程中这些模型提出了三个问题都可以得到立即有效的解决，分别是：

\* 评估：对于一个给定的隐马尔科夫模型其生成一个给定的观察序列的概率是多少。前向算法可以有效的解决此问题。

\* 解码：什么样的隐藏（底层）状态序列最有可能生成一个给定的观察序列。维特比算法可以有效的解决此问题。

\* 学习：对于一个给定的观察序列样本，什么样的模型最可能生成该序列——也就是说，该模型的参数是什么。这个问题可以通过使用前向-后向算法解决。

隐马尔科夫模型（HMM）在分析实际系统中已被证明有很大的价值；它们通常的缺点是过于简化的假设，这与马尔可夫假设不同，即一个状态只依赖于前一个状态，并且这种依赖关系是独立于时间、初始、与时间无关、

关于隐马尔科夫模型的完整论述，可参阅：

*L R Rabiner and B H Juang, 'An introduction to HMMs', IEEE ASSP Magazine, 3, 4-16.*

**全文完！**

后记：这个翻译系列终于可以告一段落了，从6月2日起至今，历史四个多月，期间断断续续翻译并夹杂些自己个人的理解。希望这个系列对于HMM的学习者能有些用处，我个人也就很满足了。接下来，我会结合HMM在自然语言处理中的一些应用，譬如词性标注、中文分词等，从实践的角度讲讲自己的理解，欢迎大家继续关注52nlp。



10







### HMM在自然语言处理中的应用一：词性标注1

词性标注 ( Part-of-Speech tagging 或 POS tagging)是指对于句子中的每个词都指派一个合适的词性，也就是要确定每个词是名词、动词、形容词或其他词性的过程，又称词类标注或者简称标注。词性标注是自然语言处理中的一项基础任务，在语音识别、信息检索及自然语言处理的许多领域都发挥着重要的作用。因此，在关于自然语言处理的书籍中，都会将词性标注单列一章重点讲解，对此有兴趣的读者可参考《自然语言处理综论》第一版第8章或《统计自然语言处理基础》第10

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



The Fulton County Grand Jury said Friday an investigation of Atlanta's recent primary election produced "no evidence" that any irregularities took place.

需要为句子中的每个单词标上一个合适的词性标记，既输出含有词性标记句子：

The/at-tl Fulton/np-tl County/nn-tl Grand/jj-tl Jury/nn-tl said/vbd Friday/nr an/at investigation/nn of/in Atlanta's/np\$ recent/jj primary/jj election/nn produced/vbn "no/at evidence/nn" that/cs any/dti irregularities/nns took/vbd place/nn ./.

在进行词性标注时，前提条件之一便是选择什么样的标记集？Brown语料库标记集有87个，而英语中其他标记集多数是从小语料库中的标记集发展而来的，如最常用的Penn Treebank标记集，包含45个标记，是小标记集。汉语标记集中常用的有北大《人民日报》语料库词性标记集、计算所汉语词性标记集等。

Brwon语料库标记集的详细信息可参考：

<http://www.comp.leeds.ac.uk/amalgam/tagsets/brown.html>

计算所汉语词性标记集的详细信息可参考：

[http://www.ictclas.org/ictclas\\_docs\\_003.html](http://www.ictclas.org/ictclas_docs_003.html)

定使用某个标记集之后，下一步便是如何进行词性标注了！如果每个单词仅仅对应一个词性标记，那么词性标注就非常容易了。但是语言本身的复杂性导致了并非每一个单词只有一个词性标记，而存在一部分单词有多个词性标记可以选择，如book这个单词，既可以是动词（book that flight），也可以是名词(hand me that book)，因此，词性标注的关键问题就是消解这样的歧义，也就是对于句子中的每一个单词在一定的上下文中选择恰如其分的标记。

关于词性标注歧义问题，对Brown语料库进行统计，按歧义程度排列的词型数目（The number of word types in Brown corpus by degree of ambiguity）DeRose(1988)给出了如下的标记歧义表：

无歧义（Unambiguous）只有1个标记：35,340

歧义（Ambiguous）有2-7个标记：4,100

2个标记：3,764

3个标记：264

4个标记：61

5个标记：12

6个标记：2

7个标记：1

可见英语中的大多数单词都是没有歧义的，也就是这些单词只有一个单独的标记。但是，英语中的最常用单词很多都

tagger), 最后一类是混合型的标注算法。基于规则的标注算法一般都包括一个手工制作的歧义消解规则库; 随机标注算法一般会使用一个训练语料库来计算在给定的上下文中某一给定单词具有某一给定标记的概率, 如基于HMM的标注算法; 而混合型标注算法具有上述两种算法的特点, 如TBL标注算法。

关于词性标注的基本问题本节就到此为止了, 也算是一点准备工作, 下一节我们将进入与HMM相关的词性标注问题的正题!

## HM 自然语言处理中的应用一: 词性标注2

10

本节我们对自然语言处理中词性标注的基本问题进行了描述, 从本节开始我们将详细介绍HMM与词性标注的关系以及如何使用HMM进行词性标注。首先回顾一下隐马尔科夫模型(HMM)的定义和三大基本问题, 并由此与词性标注的基本问题做一个对比。

隐马尔科夫模型(HMM)是什么? 说白了, 就是一个数学模型, 用一堆数学符号和参数表示而已, 包括隐藏状态集合、观测集合、初始概率向量, 状态转移矩阵A, 混淆矩阵B。

隐马尔科夫模型(HMM)的三大基本问题与解决方案包括:

1. 对于一个观察序列匹配最可能的系统——评估, 使用前向算法 (forward algorithm) 解决;
2. 对于已生成的一个观察序列, 确定最可能的隐藏状态序列——解码, 使用维特比算法 (Viterbi algorithm) 解决;
3. 对于已生成的观察序列, 决定最可能的模型参数——学习, 使用前向-后向算法 (forward-backward algorithm) 解决。

回顾完HMM, 这里暂且先放下词性标注, 瞎扯一下数学建模。

记得以前在大学里参加数学建模竞赛, 本着拿奖的目的, 稀里糊涂的就和几个同学一起组队参加, 并没有仔细考虑过数学建模的本质到底是什么。反正感觉和平常作数学题不同, 数学题都是定义好的, 只需给出一个解答就行, 而数学建模给的问题都很实际, 并没有按数学题的形式出题, 不仅要把这个实际问题转化为一个合理的数学问题, 还要给出一个解答, 由于自己概括问题的能力有限, 在数学建模竞赛上也基本毫无建树。

我在Google上搜索了一下数学建模的定义, 有好几种解释, 觉得下面这个最符合本质:

把现实世界中的实际问题加以提炼, 抽象为数学模型, 求出模型的解, 验证模型的合理性, 并用该数学模型所提供的解答来解释现实问题, 我们把数学知识的这一应用过程称为数学建模。


好了, 这就是数学建模, 如果把词性标注问题作为一个数学建模的题目来出, 该如何作答? 套用上面的定义, 可以解


别T，也就是词性标记，不过词性标注考虑的是整体标记的好坏，既整个句子的序列标记问题；



2、抽象为数学模型：对于分类问题，有很多现成的数学模型和框架可以套用，譬如HMM、最大熵模型、条件随机场、SVM等等；

3、求出模型的解：上述模型和框架一旦可以套用，如何求解就基本确定好了，就像HMM中不仅描述了三大基本问题，并相应的给出了求解方案一样；

4 验证模型的合理性：就是词性标注的准确率等评测指标了，在自然语言处理中属于必不可少的评测环节；

 解释现实问题：如果词性标注的各项指标够好，就可以利用该数学模型构造一个词性标注器来解决某种语言的标注问题！<sup>10</sup>

 标注的数学建模就这样了，自然语言处理中的多数分类问题与此相似。这里讲得是HMM的应用，所以其他模型暂且不谈，以后有机会有条件了我们再说。

 建立一个与词性标注问题相关联的HMM模型？首先必须确定HMM模型中的隐藏状态和观察符号，也可以说成观察状态，由于我们是根据输入句子输出词性序列，因此可以将词性标记序列作为隐藏状态，而把句子中的单词作为观察符号。  
 对于Brown语料库来说，就有87个隐藏状态（标记集）和将近4万多个观察符号（词型）。

确定了隐藏状态和观察符号，我们就可以根据训练语料库的性质来学习HMM的各项参数了。如果训练语料已经做好了标注，那么学习这个HMM模型的问题就比较简单，只需要计数就可以完成HMM各个模型参数的统计，如标记间的状态转移概率可以通过如下公式求出：

$$P(T_i|T_j) = C(T_j, T_k) / C(T_j)$$

而每个状态（标记）随对应的符号（单词）的发射概率可由下式求出：

$$P(W_m|T_j) = C(W_m, T_j) / C(T_j)$$

其中符号C代表的是其括号内因子在语料库中的计数。

如果训练语料库没有标注，那么HMM的第三大基本问题“学习”就可以派上用场了，通过一些辅助资源，如词典等，利用前向 - 后向算法也可以学习一个HMM模型，不过这个模型比之有标注语料库训练出来的模型要差一些。

总之，我们已经训练了一个与语料库对应的HMM词性标注模型，那么如何利用这个模型来解决词性标注问题呢？当然是采用维特比算法解码了，HMM模型第二大基本问题就是专门来解决这个问题的。

说完了如何建模，下一节我们将利用UMDHMM这个HMM工具包来实现一个toy版本的HMM词性标注器。

### HMM在自然语言处理中的应用一：词性标注3

原计划这一节讲解如何利用UMDHMM这个HMM工具包来实现一个toy版本的HMM词性标注器，自己也写了几个相关的小脚本，不过由于处理过程中需要借用Philip Resnik教授写的另外几个小脚本，所以这里先介绍一下他的工作。

Resnik教授利用UMDHMM写了一个关于如何使用隐马尔科夫模型的介绍和练习，主要目标包括以下四个方面：

- 1、 在一个“似英语”的文本上训练一个HMM模型（Train an HMM on a sample of English-like text）；
- 2、 对于训练好的模型进行检测（Inspect the resulting model）；
- 3、 根据训练好的模型随机生成句子（Generate sentences at random from the model）；

4、 对于测试句子寻找最可能隐藏状态序列（Create test sentences and find the most likely hidden state sequence）。

我的工作和Resnik教授的主要区别在于，他的训练集没有进行词性标注，利用了前向 - 后向算法生成HMM模型，并且需要有一定想象能力来作虚拟词性标注；而我所用的训练集是有标注的，主要是通过统计的方法生成HMM模型，并且对于训练集标注是直观的。但是，殊途同归，都是为了建立一个HMM模型，都是为了能利用UMDHMM。

如何下载和使用这个工具包具体请参考“[Exercise: Using a Hidden Markov Model](#)”，这里我主要讲解一些要点和一个例子。

这个工具包主要是在命令行方式下利用ftp命令，估计有的读者不太习惯这种方式，所以我在网络硬盘上上传了一个已下载的版本，有需要的读者可以去这个地址下载：[solaris.tar.gz](http://solaris.tar.gz)。

首先对这个工具包解压：tar -zxvf solaris.tar.gz

主要包括几个perl脚本，UMDHMM编译后生成的几个可执行程序以及两个样例文件夹，需要注意的是，几个perl脚本需要根据你所使用的环境修改perl的执行路径，另外UMDHMM的几个可执行程序是Resnik教授在Solaris 5.5系统下编译的，并不适用于其他操作系统，因此最好自己单独编译一下UMDHMM，关于UMDHMM的编译和使用，不太清楚的读者可以先看一下我之前写得一点介绍：[UMDHMM](#)。

对于这几个perl脚本，需要作一点预处理，第一使其可执行：chmod u+x \*.pl 或 chmod 755 \*.pl；第二，修改每个脚本的perl解释器目录，由于我用的是ubuntu9.04，所以处理的方法是，注释掉第一行，将第二行“/usr/local/bin/perl5”修改为“/usr/bin/perl”。


修改完毕perl脚本使其可执行之后，就可以进入example0目录进行练习了：cd example0

example0目录下有一个example0.train文件，只有一行，但是包含了一百句英语句子，这一百句英语句子只用了11个单词和两个标点符号“.”和“?”生成，是一个“似英语”句子生成器生成的，主目录下有这个程序，是lisp程序写的，我不明白怎么使用。如下所示部分句子：

the plane can fly the typical plane can see the plane a typical fly can see who might see? the large can might see a can the can can

对于这个训练集，Resnik教授建议读者写一个简单的词性列表，并尝试为每一个单词分配一个词性标记，并且同一个单词可以有不同的标记。注意这个练习并不是要在这个文件中进行，可以在别的地方，譬如纸上或者心里都可以，不做也行的。我就偷懒了，因为不知道如何标记，并且手工标记的工作量较大，我用了一个基于Brown语料库训练的词性标注器标注了一下，这个之后再详细说明。

由于UMDHMM这个工具包处理的都是数字而非符号，所以需要先将这个训练集转换为数字序列，由create\_key.pl这个脚本完成：

 `create_key.pl example0.key < example0.train > example0.seq`

<sup>10</sup> 这一步生成两个文件：example0.key及example0.seq，前者主要将训练集中出现的单词符号映射为数字编号,如：

1 th  
2 p  
8 a  
4 fl  
3 c  
7 see  
12 large  
11 ?  
10 might  
9 who  
6 typical  
5 .  
13 destroy

后者主要根据example0.key中的编号对训练集进行转换，并且形式为UMDHH中的训练集输入形式，如：

T= 590  
1 2 3 4 5 1 6 2 3 7 1 2 5 8 6 4 3 7 5 9 10 7 11 1 12 3 10 7 8 3 5 1 3 3 13 8 12 3 5 9 10 7 11 9 10 4 11 9 3 4 11 1 3 10 7 5 1 2 3 4 8 6 4  
5 9 3 4 11 1 12 4 3 4 5 9 3 7 11 9 3 7 8 3 11...

## HMM在自然语言处理中的应用一：词性标注4

分类 [标注](#), [自然语言处理](#), [隐马尔科夫模型](#)

在继续昨晚的工作之前，先聊两句Philip Resnik教授。作为美国马里兰大学的教授，他的主要研究领域是自然语言处理，不过最近他被美国某个网站评为“当代卫生保健领域最具创新性和最有影响力的百位革新者之一(the most creative and influential innovators working in healthcare today)”，Resnik教授也非常吃惊（Much to my surprise），之所以入选，再于他利用自然语言处理来提高医用编码（medical coding）的水平，具体什么是医用编码我不太清楚，不过这项工作至少说明自然语言处理是有相当的应用前景的。

2005年的时候，他的一个学生开发了一套统计机器翻译系统，取名为“Hiero”，在当年NIST机器翻译评测中表现出色，虽然没有拿到第一，但是其提出的“层次短语模型”的论文获得了当年ACL的最佳论文，此人名叫David Chiang，中文名叫David Chiang。

之前有一段时间我对Web [平行语料库自动采集](#)比较感兴趣，就找了很多这方面的paper,其中最有名的当属Resnik教授的Juncos和LDC的BITS了，只是当时没有仔细考虑过他是何方神圣。今天仔细读了一下他的个人主页，觉得他在自然语言处理领域也是一个比较神奇的人物，有兴趣的读者不妨看看他的这个[主页](#)，对于扩展研究思路和把握当前的研究动态还是非常有好处的。好了，下面我们转入HMM词性标注的正题。

在将训练集转换成UMDHMM需要的形式后，就可以利用UMDHMM中编译好的可执行程序esthmm来训练HMM模型了。esthmm的作用是，对于给定的观察符号序列，利用BaumWelch算法（前向-后向算法）学习隐马尔科夫模型HMM。这里采用如下的命令训练HMM模型：

```
../esthmm -N 7 -M 13 example0.seq > example0.hmm
```

其中 N指示的隐藏状态数目，这里代表词性标记，这个例子中可以随便选，我选的是7，下一节会用到。注意Resnik教授给出的命令：

```
esthmm 6 13 example0.seq > example0.hmm
```

是错误的，需要加上“-N”和“-M”。example0.hmm的部分形式如下：

M= 13

N= 7

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



...

B:

0.001000 0.366300 0.420021 0.215676 0.001000 0.001001 0.001001 0.001000 0.001001 0.001000 0.001000 0.001001 0.001000

...

pi:

0.001000 0.001000 0.001005 0.001000 0.001000 0.999995 0.001000



这个HMM模型的效果如何，这里不得不感叹前向 - 后向算法或者EM算法的神奇。当然这里只是一个练习，实际处理中需要加上一些辅助手段，譬如词典之类的，这种无监督的学习是非常有难度的。



这个HMM模型，就可以作些练习了。首先我们利用genseq来随机生成句子：

```
seq -T 10 example0.hmm > example0.sen.seq
```



T指示的是输出序列的长度，如下所示：

T=



8 12 4 5 9 3 7 5 9 3

注意 Resink教授给出的命令仍然是错的，上面的输出结果可读性不好，读者可以对照着example0.key将这个句子写出来，不过Resnik教授写了一个ints2words.pl的脚本，帮助我们完成了这件事：

```
../ints2words.pl example0.key < example0.sen.seq > example0.sen
```

example0.sen中包含的就是这个HMM模型随机生成的句子：

a large fly . who can see . who can

虽然不是一句整句，但是局部还是可读的，注意这两步可以利用管道命令合并在一起：

```
../genseq -T 10 example0.hmm | ../ints2words.pl example0.key
```

注意每次的结果并不相同。

最后一个练习也是最重要的一个：对于一个测试句子寻找其最可能的隐藏状态序列（Finding the Hidden State Sequence for a Test Sentence），对于本文来说，也就是词性序列了。我们使用testvit来完成这个任务，当然，前提是先准备好测试句子。可以根据example0.key中的单词和标点自己组织句子，也可以利用上一个练习随机生成一个句子，不过我选择了训练集



the can can destroy the typical fly .

虽然违背了自然语言处理中实验的训练集与测试集分离的原则，不过考虑到这只是一个练习，另外也是为下一节做个小准备，我们就以此句话为例建立一个文件example0.test.words。不过UMDHMM还是只认数字，所以Resnik教授有为我们写了一个words2seq.pl处理此事：

```
../words2seq.pl example0.key < example0.test.words > example0.test
```



10

ple0.test就是UMDHMM可以使用的测试集了，如下所示：

T= 8

1 3 6 4 5



就可以使用testvit，这次Resnik教授没有写错：

```
vit example0.hmm example0.test
```



结果了吗？我们得到了一个隐藏状态序列：

...

Optimal state sequence:

T= 8

6 1 5 2 6 3 1 7

...

如果之前你已经建立好了隐藏状态与词性标记的一一映射，那么就可以把它们所对应的词性标记一个一个写出来了！  
这个词性标注结果是否与你的期望一样？

如果你还没有建立这个映射，那么就可以好好发挥一下想象力了！无论如何，恭喜你 and 52nlp 一起完成了Philip Resnik教授布置的这个练习。

## HMM在自然语言处理中的应用一：词性标注5

分类 [标注](#), [自然语言处理](#), [隐马尔科夫模型](#)

低，在实际应用中多是那些建立在有词性标注训练集基础上的机器学习算法，如最大熵模型、决策树等，所学习的词性标注器能获得较高的标注准确率。本节我们就以一个标注好的训练集为基础，来学习一个最简单的HMM词性标注器。

首先就是准备训练集，作为一个练习，52nlp也本着尽量简单的原则，所以这里仍然选用Resnik教授所使用的[example0.train](#)，这个训练集虽然包含了一百句“似英语”的句子，但是只有一行，所以我们首先做一个断句处理，不过这些句子只采用了“.”和“?”作为句尾标志，因此断句相对简单。不过实际处理中英文断句问题比较麻烦，也有很多学者这方面做了很多研究工作。这里52nlp写了一个简单的[sentsplit.pl](#)脚本来处理这个训练集：



```
split.pl example0.train example0.sentences
```

10

[example0.sentences](#)就成了每一句为一行的训练集，如下所示：

```
the can fly .
the plane can see the plane .
a typical fly can see .
who can see ?
the can might see a can .
the can can destroy a large can .
...
```

但是，这个训练集只包含纯粹的单词句子，因此需要做一下词性标注，当然人工标注并检查是最好的了，但是我不懂，于是找了一个开源的词性标注工具对这些句子进行了标注，关于这个词性标注器的细节，下一节我会具体介绍，先来看标注后得到的包含词性标记的训练集[example0.all](#)，部分示例如下：

```
the/at plane/nn can/md fly/vb ./
the/at typical/jj plane/nn can/md see/vb the/at plane/nn ./
a/at typical/jj fly/nn can/md see/vb ./
who/wps might/md see/vb ?/.
the/at large/jj can/nn might/md see/vb a/at can/nn ./
...
```

无论什么方法，建立HMM词性标注器的关键就是根据这个训练集来学习一个合适的HMM模型了。我们先来确定HMM

```
../create_key.pl words.key < example0.sentences > example0.seq
```

所得到的`words.key`就包含了训练集中的词型及其数字编号：

```
1 the
2 plane
8 a
4 fl
3 can 10
7 see
12 I
11 ?
10 I
9 w
6 ty
5 .
13 destroy
```

注意另一个副产品`example0.seq`在这一节里并不需要。同样我们也需要利用`create_key.pl`来确定训练集中的词性标记及其编号，不过这里我们需要先将`example0.all`中的词性标记序列抽取出来。这里52nlp写了一个简单的脚本`extractpos.pl`来处理此事：

```
./extractpos.pl example0.all example0.pos
```

所得到的`example0.pos`文件部分示例如下：

```
at nn md vb .
at jj nn md vb at nn .
at jj nn md vb .
wps md vb .
at jj nn md vb at nn .
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



有了这个文件，就可以再次利用create\_key.pl了：

```
../create_key.pl pos.key < example0.pos > example0.posseq
```

所得到的pos.key就包含了训练集中的词性标记及其数字编号：

```
4 vb
6 jj
3 in
2 nn 10
7 wnc
5 .
1 at
```

同样，另一个副产品example0.posseq这里也不需要。

好了该HMM模型中的隐藏状态（词性标记）和观察符号（词型）后，下一步便是要计算HMM模型中其他三个基本要素了，包括初始概率向量，状态转移矩阵A，混淆矩阵B。

我们先预处理一下语料库，主要的目标是对一元词性、二元词性及词型与词性的组合进行计数，这里52nlp写了一个脚本pretrain.pl来处理此事：

```
./pretrain.pl example0.all lex ngram
```

所得到的lex文件主要是统计词型及其词性标记的组合在训练集中出现的次数：

```
typical jj 25
large jj 22
might md 42
fly nn 20
a at 58
? . 57
plane nn 34
the at 35
```

see vb 45  
 destroy vb 9  
 fly vb 46  
 . . 43  
 can md 58



n文件主要包含的是一元词性及二元词性在训练集中的出现次数：

vb 100  
 jj 4  
 md 100  
 nn 1  
 wps 57  
 . 10  
 at 93  
 vb . 50  
 md vb 100  
 vb at 50  
 at jj 47  
 wps md 57  
 nn . 50  
 at nn 46  
 jj nn 47  
 nn md 43

有了这几个预处理文件，我们就可以训练一个简单的HMM词性标注模型了,这里52nlp写了一个约100行的脚本

```
./hmmtrain.pl words.key pos.key ngram lex example.hmm
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



M= 13

N= 7

A:

0.0100 0.4700 0.0100 0.0100 0.0100 0.4800 0.0100

...

B:

0.3: 0.0094 0.0094 0.0094 0.0094 0.0094 0.0094 0.5566 0.0094 0.0094 0.0094 0.0094 0.0094

...

pi:

0.1695 0.1695 0.1695 0.1695 0.1695 0.0797 0.0966

感兴趣的读者，可以对比一下上一节利用BaumWelch算法（前向-后向算法）所学习的HMM词性标注模型example0.hmm。

这个脚本，其中对于状态转移矩阵A，混淆矩阵B的计算采用了最简单的加一平滑来处理那些在训练集中的未出现事件，关于加一平滑，不清楚读者可以在“[MIT自然语言处理第三讲：概率语言模型（第四部分）](#)”中找到参考，或者任何一本自然语言处理书中关于ngram语言模型的章节都会介绍的。

现在我们可以作上一节最后一个词性标注的练习了，仍然选择训练集中的第91句：

the can can destroy the typical fly .

可以利用Resnik教授的words2seq.pl来对此句进行转换，或者利用上一节已经处理好的UMDHMM可读的example0.test：

T= 8

1 3 3 13 1 6 4 5

现在就可以使用testvit及刚刚训练好的example.hmm来作词性标注了：

```
../testvit example.hmm example0.test
```

同样得到了一个隐藏状态序列：

T= 8

1 2 3 4 1 6 2 5

...

不过这次我们已经有了词性标记序列及其数字编号，可以对应着把它们写出来：

at n        b at jj nn .



与测试句子合在一起即是：

the        nn can/md destroy/vb the/at typical/jj fly/nn ./.



example.all里的第91句：

the        nn can/md destroy/vb the/at typical/jj fly/nn ./.



二者是一样的，不过这个绝不能说明此HMM词性标注器是100%正确的。

好了，本节就到此为止了，这一节的相关例子及小脚本可以单独按链接下载，也可以打包在这里供下载：[52nlpexample.zip](#)

不过这套小工具还不足以处理实际问题中的词性标注问题，下一节我将介绍一个更加健壮的HMM词性标注开源工具。

## HMM在自然语言处理中的应用一：词性标注6

分类 [标注](#), [自然语言处理](#), [隐马尔科夫模型](#)

有一段时间没有谈HMM和词性标注了，今天我们继续这个系列的最后一个部分：介绍一个开源的HMM词性标注工具并且利用Brown语料库构造一个英文词性标注器。

[上一节](#)借用umdhmm构造的HMM词性标注工具是二元语法(bigram)标注器，因为我们只考虑了前一个词性标记和当前词性标记，算的上是最基本的马尔科夫模型标注器。这个HMM词性标注器可以通过好几种方式进行扩展，一种方式就是考虑更多的上下文，不只考虑前面一个词性标记，而是考虑前面两个词性标记，这样的标注器称之为三元语法（trigram）标注器。是非常经典的一种词性标注方法。在《自然语言处理综论》及《统计自然语言处理基础》中被拿来介绍。

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



horsten/tnt/), 可谓“知行合一”。但是要获得这个工具必须填一个表, 并且传真给对方, 比较麻烦。不过幸好在英文维基百科关于词性标注的介绍页面上有替代品: [Part-of-speech\\_tagging](#).

在这个页面的“External links (外部链接)”的最后一行, 提到了一个名叫Citar的利用C++开发的隐马尔科夫模型(HMM)三元语法词性标注器:

“Citar LGPL C++ Hidden Markov Model trigram POS tagger, a Java port named Jitar is also available”

同时, 它也提供Java版本的Jitar。不过可惜, 这个页面目前无法直接访问到。如果读者对这个词性标注工具感兴趣的话, 提供一个Citar的下载链接: [citar-0.0.2.zip](#)

以下是citar的简要介绍:

Citar is a simple part-of-speech tagger, based on a trigram Hidden Markov Model (HMM). It (partly) implements the ideas set forth in [1]. Citar is written in C++. There is also a Java/JDK counterpart named Jitar,

which is available at: <http://code.google.com/p/jitar/>

[1]指的是“[TnT — Statistical Part-of-Speech Tagging](#)”, 其具体的实现思想在这篇文章里描述的很细致, 我觉得主要需要的几个地方是trigram的平滑算法, 未登录词的处理方法(主要是针对英文的), 以及柱搜索(beam search)解码算法。

编译citar直接make就可以了, 生成三个可执行文件: train, tag, evaluate。顾名思义, “train”是用来训练一些必要的文件的, tag则是进行标注的, 而evaluate则是用来评价标注结果的。下面我们以Brown语料库为例来演示如何使用这三个可执行程序。

关于Brown语料库, 我是从NLTK的包中得到的, NLTK提供了两个版本的语料库, 一个是纯文本格式, 另外一个XML格式, 都进行了词性标注, 如果你对NLTK不熟悉, 可以从下面两个链接直接下载这两个语料库:

- 1、[XML格式的brown语料库](#), 带词性标注;
- 2、[普通文本格式的brown语料库](#), 带词性标注;

至于Brown语料库的具体介绍, 大家可以参考这个页面: [BROWN CORPUS MANUAL](#)。在这个练习中, 我采用的是纯文本格式的brown语料库, 但是这些文件是按照类别分布在很多小文件里, 并且包含很多空行, 所以我处理了一下这个文件, 把它们合并到一个大的文件中, 并且去除了行首的空格及空行, 共得到57340个带词性标注的句子(brown.corpus)。我们首先对这个语料库进行划分, 从中选取前55340句作为训练集(brown.train), 选取剩余的2000句作为测试集(brown.test), 现在我们就来运行这三个命令。

首先利用train来训练:

```
../train brown.train brown.lex brown.ngram
```



得，只是结果文件里的格式稍有不同而已。

有了上述两个文件，就可以利用tag进行词性标注了，我们拿citar里的一个示例句子来实验一下：

```
echo "The cat is on the mat ." | ../tag brown.lex brown.ngram
```

得到如下的结果：

```
The/at cat/nn is/bez on/in the/at mat/nn ./.
```

如果对一个没有标注的文件进行标注，可以利用如下的命令：

```
 brown.lex brown.ngram < input > output
```

<sup>10</sup>最后，我利用evaluate来验证一下基于brown.train训练出来的词性标注器的准确率,在测试集brown.test上进行测试：

```
 luate brown.lex brown.ngram brown.test
```

得到如下的结果：

```
 racy (known): 0.964621
```


```
Accuracy (unknown): 0.740937
```

```
 racy (overall): 0.956389
```

说明这个词性标注器对于语料库中已存在的词的标注准确率是96.46%，对于未登录词的标注准确率是74.09%，而整体标注准确虑是95.63%。

好了，关于Citar我们就到此为止，有兴趣的读者可以找一些标注好的语料库来试试，包括中文的词性标注语料库，只不过它用于英文的未登录词处理方法对于中文并不合适而已。上面所提到的几个文件，包括处理好的brown.corpus,训练集brown.train,测试集brown.test及中间生成的brown.lex,brown.ngram我已经打包放在了网络硬盘里，可以在如下地址下载：[brown test.zip](#)

关于HMM在词性标注中的应用就说完了，再次回头说词性标注时，我会基于其他的模型来作相关的词性标注练习。下一个关于HMM在自然语言处理的应用，将会谈谈中文分词的相关问题，欢迎继续关注52nlp。

 目前您尚未登录，请 [登录](#) 或 [注册](#) 后进行评论



jkka 2017-09-06 13:44

[回复](#)

8楼

10

好多公式都不见了，乱七八糟，太不走心了吧前辈



eyu\_93 2017-09-04 11:03

[回复](#)

7楼



前辈您好，文章的公式和图片没有显示出来，有木有文档类型的可以发一份给我。我想很好地理解HMM模型，您的  
一份写的很详细，能否发份电子版的给我，1961507734@qq.com，谢谢您



jiebei2009 2014-12-26 21:53

[回复](#)

6楼

太罗嗦，看不下去了

[查看 8 条热评](#) ∨

## HMM——维特比算法(Viterbi algorithm)



zb1165048017 2015年09月19日 19:39  5881

本文修正了原创里面的一些错误。

## HMM的应用与Forward算法、Viterbi算法



baimafujinji 2016年05月06日 15:04  10466

隐马尔科夫模型（HMM）是机器学习中的一种重要技术，也是一种PGM。HMM在自然语言处理、图像处理和计算机视觉等领域都有重要应用。而要深入了解HMM，就不得不谈到用于评估、预测和 Decoding 的...

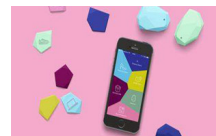
加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

[登录](#)

[注册](#)



你认为做前端想拿到高年薪难吗？




## 隐马尔可夫模型 HMM 原理及实现




xiaotianlan

2014年08月29日 14:20

2378

简介  马尔可夫模型（Hidden Markov Model，HMM）创立于20世纪70年代。主要用于行为识别，语音识别，文字识别等。

原理简述  10 隐马尔可夫模型由五个...


## 史上最详细最容易理解的HMM文章




GarfieldEr007

2015年11月18日 13:00

1280

http:// 52nlp.cn/hmm-learn-best-practices-four-hidden-markov-models wiki上一个比较好的HMM例子 HMM(隐马尔可夫模型)



## HMM原理简述和使用说明



u010076558

2015年03月02日 22:20

2817


1、原理简述 为了对GMM-HMM在语音识别上的应用有个宏观认识，花了些时间读了下HTK（用htk完成简单的孤立词识别）的部分源码，对该算法总算有了点大概认识，达到了预期我想要...

## 程序员不会英语怎么行？

老司机教你一个数学公式秒懂天下英语



## HMM的(五个基本要素，三个假设，三个解决的问题)

了解HMM的人们，都知道HMM有五个基本要素，三个假设和解决的  u013378306 2017年02月13日 15:56 1841

三个问题：首先看下HMM的五个基本要素：HMM是个五元组  $\lambda = (S, O, \pi, A, B)$  S:状态值集...

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



## 隐马尔科夫模型 (HMM) 中的三个主要问题及相关算法（待续）

本文针对HMM中的三个主要问题：评价(Evaluation or Likelihood Computation)、解码(Decoding) 和训练 (Training) 以及其对应的算法给以详细的介绍...

## 隐马尔可夫模型三个问题的求解(一)

u012116229 2015年02月04日 15:15 2665

上一篇文章《马尔可夫模型介绍》中讲解了马尔可夫假设和隐马尔可夫模型 HMM，并提到了 HMM 中的三个基本问题，但没有展开讨论其求解。本篇就此做出解答。本文主要参考《HMM 学习最佳范例》。...

## HMM topology and transition modeling

u010384318 2014年08月17日 21:15 1929

HMM topology and transition modeling 介绍 在这里我们将介绍在kaldi中如何表示HMM topologies和我们如何让建模和训练HMM 转移概率的。我们将简...

## HMM经典介绍论文【Rabiner 1989】翻译（一）——介绍

A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition Rabiner, 1989.尽管...

## HMM经典介绍论文【Rabiner 1989】翻译（五）——HMM的三个基本问题

2.3 HMM的三个问题在上一小节给出了HMM的形式，在实际应用中还有三个基本问题需要解决，分别是：问题1：给定观测序列 $O=O_1O_2\cdots O_T$ 和模型 $\lambda=(A,B,\pi)$ ...

VictoriaW 2017年11月30日 22:21 194

## 学习大数据需要哪些基本知识

大数据要学什么



加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



## 训练中文分词HMM模型，得到A（状态转移矩阵）、B（混淆矩阵）、Pi（初始状态...

#!F://python # page coding=utf-8 # 本代码用来训练中文分词HMM模型，得到A矩阵（状态转移矩阵）、B矩阵（混淆矩阵）、Pi向量（初始概率向量）并且用pickle 将他...



u010378878 2016年06月08日 08:31 2567

## 【源码剖析HMM（附Python代码）】2.隐马尔科夫链HMM的EM训练过程

隐马尔科夫链HMM的参数 $\theta$ 的EM训练过程 现在回到前一节最后提出的参数 $\theta$ 的最大似然函数上来，先对其做个对数变换，做对数变换考虑到序列X的概率计算公式中包含了连乘，为了方便计算同时避免序列X的概率过...



2017年04月27日 13:28 2169

## 隐马尔可夫模型（HMM）攻略



likelet 2011年12月09日 10:44 170951

隐马尔可夫模型 (Hidden Markov Model，HMM) 最初由 L. E. Baum 和其它一些学者发表在一系列的统计学论文中，随后在语言识别，自然语言处理以及生物信息等领域体现了很大的价...

## 中文分词的python实现----HMM、FMM



Together\_CZ 2017年06月27日 18:50 490

转自：<http://blog.csdn.net/orlandowww/article/details/52706135> 隐马尔科夫模型（HMM）模型介绍 HMM模型是由一个“五元组...

## 使用python实现HMM



u014365862 2015年12月27日 16:50 3836

一直想用隐马尔可夫模型做图像识别，但是python的scikit-learn组件包的hmm module已经不再支持了，需要安装hmmlearn的组件，不过hmmlearn的多项式hmm每次出来的结果都...

## 精品课程：从优秀到卓越产品经理晋升之路

北风网全力打造的精品课程，涵盖了所有产品经理需要了解的方向！



## 机器学习中的隐马尔科夫模型（HMM）详解



baimafujinji

2016年05月01日 16:03

32048

本文介绍机器学习中非常重要的隐马尔可夫模型（HMM，Hidden Markov Model），它也是一种PGM。更准确地说，HMM是一种特殊的贝叶斯网络。HMM在自然语言处理、计算机视觉，以及语言识别...

## EM与HMM参数估计代码

2015年01月08日 10:05

298KB

下载



## 随机森林、EM、HMM、LDA



qq\_23617681

2016年05月23日 16:55

693

本篇简单介绍这些概念。随机森林（Random Forest）是一种分类和回归算法，它包含了多个决策树，形成一个森林，随机森林的类别是所有决策树分类回归结果的众数决定。由于它优点很多，适...

## GMM-HMM学习笔记



davidie

2015年07月17日 16:31

10997

最近几天钻研了语音处理中的GMM-HMM模型，阅读了一些技术博客和学术论文，总算是对这个框架模型和其中的算法摸清了皮毛。在这里梳理一下思路，总结一下这几天学习的成果，也是为以后回顾时提高效率。本文主...