参考文章： <http://www.cnblogs.com/zhizhan/p/4110868.html>

**Hidden Markov Models （隐马尔科夫模型）**

Intuition informs much of what we do: for example, it tells us that certain words tend

to be a certain part of speech, or that if a user visits a signup page, she has a higher

probability of becoming a customer. But how would you build a model around

intuition?

我们做很多事情时常常会依赖直觉：例如，直觉会告诉我们，某些词往往是某些特定词类的一部分，或者说，当一个用户访问一个某购物网站注册页面时，其成为最终消费客户的概率会较高。但是你思考过如何围绕这些直觉来建立一个具体的模型吗？

Hidden Markov models (HMMs) are well versed in finding a hidden state of a given

system using observations and an assumption about how those states work. In this

chapter, we will first talk about how to track user states given their actions, then

explore more about what an HMM is, and finally build a part-of-speech tagger using

the Brown Corpus. The part-of-speech tagger will tag words in sentences as nouns,

pronouns, or any part of speech in the Brown Corpus.

Hidden Markov模型（HMM）擅长于寻找和预测隐藏和不能被直接观测到的指定系统的状态。其原理是通过一个可以观察到的状态和与其相关的隐藏状态序列的相关概率来进行推演和预测。在这一章中，我们将首先讨论如何跟踪用户状态，进而探讨更多关于HMM的话题，并最终通过使用布朗语料库来建立一个词性标注系统。该语音标注系统会给句中名词、代词打上标签，甚至可以标注任何演讲内容。

HMMs can be either supervised or unsupervised and also are called Markovian due

to their reliance on a Markov model. They work well where there doesn’t need to be a

lot of historical information built into the model. They also work well for adding

localized context to a classification. Unlike what we saw with Naive Bayesian Classification,

which relied on a lot of history to determine whether a user is spammy or not,

HMMs can be used to predict changes over time in a model.

HMM模型支持监督或无人监督，由于其对马尔可夫的依赖因此也被称为马尔可夫模型。

该模型不需要大量的历史数据信息也能正常工作，也能很好地为在小范围内为分类添加上下文。不像我们之前所看到的依赖于大量的历史数据信息才能确定用户是否为垃圾邮件的朴素贝叶斯模型，HMM模型可以用来预测随着时间而变化模型状态。

**Tracking User Behavior Using State Machines 使用状态机来跟踪用户行为**

Have you ever heard of the sales funnel? This is the idea that there are different levels

of customer interaction. People will start as prospects and then transition into more

engaged states (see Figure 6-1).

有人听说过销售漏斗理论吗？ 该理论认为不同层次的客户与销售间存在不同的互动。

人们通常会从观望开始，然后逐步过渡并成为最终的大客户（见图6-1）。

Prospects are “lurkers” who visit the site once or twice but usually don’t engage.

Users, on the other hand, like to browse and occasionally make purchases. Customers

are quite engaged and have bought something but usually don’t buy a lot in a short

time, and thus go back to being users temporarily.

浏览者访问购物网站的频率很低，通常只有一次或两次，即使有活动也通常不参与。

另一方面，如果是注册用户，则喜欢经常浏览该购物网站并偶尔购买商品。作为最积极的消费客户，一旦有活动就会购买商品，但消费客户通常不会在短时间内购买很多东西，因此也会暂时蜕变成普通注册用户。

Let’s say that we have an online store and determine that out of prospects that visit

the site, 15% will sign up, and 5% will become customers right away. When the visitor

is already a user, he will cancel his account 5% of the time and buy something 15% of

the time. If the visitor is a customer, he will cancel his account only 2% of the time

and go back to being a user 95% of the time instead of continually buying things.

假设我们有一个网上商店，并假定出访问该网站的浏览者中，15%将注册成用户，其中5%的用户将立刻成为消费客户。当访问者已经是一个注册用户时，他有5%概率会销户，并且有15%的概率会购买东西。如果访问者已经是一个消费客户，他注销的帐户概率只有2%，有95%的概率蜕化成普通注册用户，而不是不断的购买商品。

We could represent the information we have collected in a transition matrix, which

shows the probability of going from one state to another, or remaining in the same

state (Table 6-1).

我们可以用状态转换矩阵表来描述上述行为，该表显示了一个状态转换成另一个状态的概率，或停留在原先的状态（表6-1）。

What the transition probability defines is known as a state machine (see Figure 6-2).

It also tells us a lot about how our current customers behave. We can determine the

conversion rate, attrition rate, and other probabilities. Conversion rate is the probability

of a prospect signing up, which would be 20%—the probability of going from

prospect to user plus the probability of prospect to customer (15% + 5%). You could

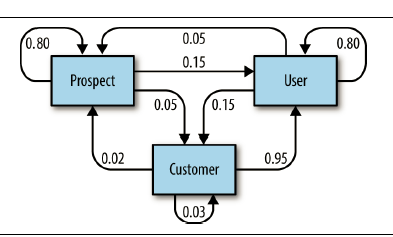
also determine the attrition rate by taking the average of 5% and 2%, which is 3.5%.

转移概率常常被称为一个状态机（见图6-2）。

它告诉了我们很多当前的客户行为。我们可以确定转换率，磨损率和其他概率。

转换率是代表一个浏览者变成注册用户的概率，此处为20%，该概率包括了从浏览者变成普通注册用户以及消费客户两部分（15% + 5%）。

你也可以通过5%和2%的平均值来确定磨损率，此处是3.5%。



This is an uncommon way of displaying user behavior in analytics, because it is too

explanatory. But it has one advantage over traditional conversion rate calculations:

the ability to look at how a user operates over time. For instance, we could determine

the probability of a user being a prospect given the last four times he was in fact a

prospect. This is the probability of being a prospect (say 80%) multiplied by the four

times they were a prospect before, which were all 80%. The probability that someone

keeps viewing the site and never signs up is low, because eventually he might sign up.

(下面一段的翻译有点不太确定)

在分析中显示用户行为的并非常见做法，因为它比较难以解释。

但与传统的转换概率计算模型相比，其优势在于该模型拥有可以查看用户如何随着时间的推移而进行角色状态转换的能力。

例如，我们可以通过概率模型计算出过去四次的时间观测点上一个普通注册 用户变成浏览者的概率，假设他目前还只是一个观望者。

根据转换概率模型(假设每次的转换概率都是80%)随着时间的推移，在每次的时间观测点上的转换概率仍然是80%。这显然不符合实际情况，

因为某人从来没有注册过该购物网站，却在每次的时间观测点上都出现了不停的查看该购物网站的行为，这样的可能性是非常低的。

But there is also one major problem with this model: there is no way for us to reliably

determine these states without asking each user individually. The state is hidden from

our observation. A user can view the site anonymously.

但是，这个模型也有缺陷：在不要求用户登录的情况下无法准确的确定用户的级别。由于每个人无须登录都查看购物网站的所有商品目录，因此其真实用户级别是我们无法观测到的。（当前匿名浏览者可能只是一个浏览者，也可能是注册并已经消费过的客户）

That is actually fine, as you will soon see. As long as we are able to observe interaction

with the site and make a judgment call about the underlying transitions from

other sources (think Google Analytics), then we can still solve this problem.

这个缺陷是可以被克服的，正如你所将看到的那样，只要我们能够观察到用户与网站的互动，并通过调用某些分析工具（如Google Analytics），那么我们仍然可以解决这个问题并鉴别用户的实际身份

We do this by introducing another level of complexity called emissions.

我们将通过引入另外一个复杂的输出模型来解决上述缺陷

Emissions/Observations of Underlying States(输出/观测隐含状态)

With our preceding example, we don’t know when someone goes from being a prospect

to a user to a customer. But we are able to observe what a user is doing and what

her behavior is. We know that for a given observation there is a probability that she is

in a given state.

如前例所述，我们并不知道何时有人会从浏览者变成普通注册用户以及消费客户。但我们能够观察用户正在做什么，她的行为是什么。我们也知道，对于一个既定的观测对象，她当前角色是浏览者还是消费客户的概率是多少。

注：隐含状态和可见状态之间的概率称为输出概率(Emission Probability)

We can determine the user’s underlying state by observing her emitted behaviors.

Let’s say, for instance, that we have five pages on our website: Home, Signup, Product,

Checkout, and Contact Us. Now, as you might imagine, some of these pages matter to

us and others do not. For instance, Signup would most likely mean the prospect

becomes a user, and Checkout means the user becomes a customer.

我们可以通过观察用户的种种行为来确定当前用户的实际潜在角色。例如，在我们的网站上有五个页面:主页，注册，产品介绍，结帐和联系我们。现在你可以想象，当中哪些页面和我们的用户角色有直接关系。例如，注册页面是最有可能将浏览者转换成为一个普通注册用户的。结帐意味着注册用户成为一个消费客户。

This information gets more interesting because we know the probabilities of states.

Let’s say we know the emission and state probabilities shown in Table 6-2.

接下来的信息将很有趣，因为我们知道每个页面当前用户的实际角色概率，所以也知道转移矩阵正如表6-2所示

We know the probability of users switching states as well as the probability of the behavior they are emitting given the underlying state. Given this info, what is the probability that a user who has viewed the Home, Signup, and Product pages becomes a customer? Namely, we want to solve the problem depicted in Figure 6-3.

我们之前已经知道用户在浏览者、注册用户以及消费客户之间的转换状态概率以及当前用户在既定状态下发生行为(如浏览指定页面)时实际角色身份的可能概率。通过这些种种信息，我们能否得出浏览者(或者是注册用户用户)在浏览主页，注册页面以及产品页面时成为一个消费客户的概率是多少？换句话说，我们要解决的问题可用图6-3来表示：

To figure this out, we need to determine the probability that a user is in the customer

state given all her previous states, or notationally, P(Customer | S1, S2), as well as the

probability of the user viewing the product page given that she was a customer multiplied

by the probability of signup given the state, or notationally, P(Product\_Page |

Customer) \* P(Signup\_Page | S2) \* P(Homepage | S1). The problem here is that there

are more unknowns than knowns.

为了解决上述问题，我们需要定义当前用户是消费者时，其之前所有可能的状态，以符号P(Customer | S1, S2)表示，因此当用户浏览产品页面时她为消费者的概率与用户浏览注册页面时为S1状态或者浏览主页时为S1状态的综合概率以符号P(Product\_Page |

Customer) \* P(Signup\_Page | S2) \* P(Homepage | S1)来表述。这里问题的关键在于我们未知参数比已知的参数更多。

This finite model is difficult to solve because it involves a lot of calculations. Calculating

a problem like P(Customer | S1, S2, ⋯, SN) is complicated. To solve this, we need to

introduce the Markov assumption.

依靠已知的有限模型很难解决这个问题，因为它需要大量的计算。逐一计算诸如像P（客户| S1，S2，⋯，SN）的概率是相当复杂的。为此，我们引入马尔可夫假设。

Emissions and observations are used interchangeably in HMM nomenclature. They

are the same thing and refer simply to what a process is emitting or what you can

observe.

输出和观测在HMM模型中经常交替使用。它们其实是同一回事，表示某个过程是输出状态或当前状态下你可以观测到什么。

Simplification Through the Markov Assumption 简化马尔可夫假设

Remember from the Naive Bayesian Classification that each attribute would independently

add to the probability of some events. So for spam, the probability would

be independently conditional on words or phrases like Prince and Buy now. In the

model that we’re building with user behavior, though, we do want dependence.

Mainly, we want the previous state to be part of the next state’s probability. In fact, we

would assert that the previous states have a relationship to the user’s current state.

依据朴素贝叶斯分类模型 ， 每个特征属性所出现的概率都相互独立。所以对于垃圾邮件的判定来说，我们假定互不关联的词或短语(就像”王子”和”现在购买”)这样词在每封邮件中出现的概率应该相互独立。但是在我们依赖用户行为所构建的模型中 ，每个用户的状态转移间有依赖关系，我们主要目的是求解在上一状态切换到下一状态的概率。事实上 ， 我们会假设上一状态与用户当前状态的存在联系。

In the case of Naive Bayesian Classification, we would make the assumption that the

probability of something was independently conditional on other events. So spam

was independently conditional on each word in the email.

在朴素贝叶斯分类中，我们假设事件之间的发生条件是相互独立的。因此垃圾邮件中每个词所出现的频率也与其他词没有关系。

We can do the same with our current system. We can state that the probability of

being in a particular state is primarily based on what happened in the previous state.

So instead of P(Customer | S1, S2, ⋯, SN), our equation would be P(Customer | SN). But

why can we get away with such a gross simplification?

我们在当前系统中也采用同样的方法来处理。我们假定出现某个特定状态的出现概率(如客户Customer)主要依赖于之前的状态。所以之前的P(Customer | S1, S2, ⋯, SN)将缩写为P(Customer | SN)。但是为什么我们可以做这样粗略的简化呢？

Given a state machine like the one we have just defined, the system infers probabilistically

and recursively where you have been in the past. For instance, if a site visitor

were in the customer state, then you could say that the most probable previous state

would be user, and that the most probable state before that would be prospect.

就像我们之前所定义的给定状态的机器，系统会根据概率并且递归的持续推导出机器之前的一系列状态。例如某个网站访问者当前是消费客户的角色，我们可以推断出她之前最有可能的状态是注册用户，在注册用户状态之前最有可能是浏览者。

This simplification also has one exciting conclusion, which leads us into our next topic: Markov chains.

根据上述的简化我们将得出一个令人兴奋的结论并引导我们进入下一主题：马尔科夫链

**Using Markov Chains Instead of a Finite State Machine使用马尔可夫链代替有限状态机**

We have been talking purely about one system, and only one outcome, thus far. But

what is powerful about the Markov assumption is that you can model a system as it

operates forever. Instead of looking locally at what the process is going to do, we can

figure out how the system will always behave. This brings us to the idea of a Markov

chain.

我们已经讨论了对单一系统且仅有一个输出结果的状况。但是马尔可夫模型的强大在于你可以假设系统的操作状态会持续进行。并非查询当前进程将要做什么 ， 我们能清晰的指出并预测系统将要发生的行为，这就是马尔可夫链。

Markov chains are exceptional at simulating systems. Queuing theory, finance,

weather modeling, and game theory all make heavy use of Markov chains. They are

powerful because they represent behaviors in a concise way. Unlike models such as

neural networks, which can become extremely complex as we add nodes, HMMs only

rely on a few probability matrices; they are extremely useful at modeling system

behaviors.

马氏链可以针对某些特定类的系统进行仿真。比如排队理论、财务、天气建模、和游戏理论都大量使用了马尔可夫链。在这些系统上基于马氏链进行预测效果非常好，因为它们表现行为的方式都很简洁。而有些模型(例如神经网络模型)一旦我们添加节点后就会变得相当复杂，马尔可夫链仅仅依赖少量的概率矩阵 因此在对此类系统的行为建模时也非常有用

Markov chains can analyze and find information within an underlying process that

will operate forever. But that still doesn’t solve our fundamental problem, which is

that we still need to determine what state a given person is in given his hidden previous

state and our own observations. For that, we will need to enhance Markov chains

with a hidden aspect.

马尔可夫链可以在一个持续运行的系统中基于一个隐藏的状态序列来分析和查找信息。但是，这仍然不能解决我们的根本问题。我们需要确定给定的人处于什么状态，通过他所隐藏的先前状态和我们自己的观察。为此，我们需要加强马尔可夫链对隐藏状态信息的观测能力。

**Hidden Markov Model 隐马尔科夫模型**

We’ve talked a lot about observation and underlying state transitions, but now we’re

almost back to where we started. We still need to figure out what a user’s state is.

To do this, we will use a Hidden Markov Model, which comprises these three

components:

我们已经谈论了很多关于状态转换和观测的基础 ， 但现在又几乎回到了起点。我们还需要了解用户的确定状态。为此 ， 我们将使用隐藏马尔可夫模型 ， 其包括三个部件 ：

Evaluation 评估

How likely is it that a sequence like Home → Signup → Product → Checkout will

come from our transition and observation of users?

在我们观测用户状态转换中有多大的概率会出现 主页->注册->产品->购买 诸如此类的可以观测到的顺序行为序列？

Decoding 解码

Given this sequence, what does the most likely underlying state sequence look like?

基于可观测到的顺序行为序列，隐藏在该序列下面的隐藏状态转移序列是什么？

Learning 学习

Given an observed sequence, what will the user most likely do next?

给的一个可观测的序列，用户下个最有可能的操作是什么？

In the following sections, we will discuss these three elements in detail. First, we’ll talk

about using the Forward-Backward algorithm to evaluate a sequence of observations.

Then we will delve into how to solve the decoding problem with the Viterbi algorithm,

which works on a conceptual level. Last, we’ll touch on the idea of learning as

an extension of decoding.

在下面的章节中，我们将详细讨论这三个部件。首先，我们要讨论如何使用前向- 后向算法来估算观察序列。然后我们会讨论如何通过维特比算法进行解码,该算法工作在概念级别。最后， 我们将学习作为解码扩展的想法。

**Evaluation: Forward-Backward Algorithm 基于前向-后向算法的评估**

Evaluation is a question of figuring out how probable a given sequence is. This is

important in determining how likely it is that your model actually created the

sequence that you are modeling. It can also be quite useful for determining, for example,

if the sequence Home→Home is more probable than Home→Signup. We perform

the evaluation step by using the Forward-Backward algorithm. This algorithm’s

goal is to figure out what the probability of a hidden state is subject to the observations.

This is effectively saying that, given some observations, what is the probability

that happened?

评价就是求解一个给定的顺序序列出现的可能概率。这一点非常重要的，必须确定你的模型实际创建序列才是你真正需要去建模的。有些顺序序列的出现概率是很容易预估的，例如，主页→主页 就比主页→注册页面更容易出现。我们执行前向-后向算法的评价步骤。该算法的目标是找出隐藏状态受观察值的影响的概率。更通俗的说，该算法就是通过添加一些观察对象来确定发生概率是多少？

**Mathematical Representation of the Forward-Backward Algorithm前向-后向算法的数学表示模型**

The Forward-Backward algorithm is the probability of an emission happening given

its underlying states—that is, P(ek | s). At first glance, this looks difficult because you

would have to compute a lot of probabilities to solve it. If we used the chain rule, this

could easily become expansive. Fortunately, we can use a simple trick to solve it

instead.

前向-后向算法是计算一个给定的模型在多大的概率下能产生某个可观察的序列(假设我们已经知道隐藏状态如隐含状态的数量以及转换概率等参数的前提下)，我们记做**P（eK | S）**。乍一看，这计算起来很困难，因为你必须计算出序列链中一系列的状态概率才能得到该值。尤其是如果使用链规则情况下，很容易膨胀并难以计算。幸运的是，我们可以用一个简单的技巧来解决这个问题。

The probability of **ek** given an observation sequence is proportional to the joint distribution

of **ek** and the observations:

EK既定观察序列的概率是EK和观察值的联合分布比例：

which we can actually split into two separate pieces using the probability chain rule:

我们可以依据概率链推导公式将其分成两个独立的部分：

This looks fruitless, but we can actually forget about x1, ... , xk in the first probability

because the probabilities are D-Separated. I won’t discuss D-Separation too much, but

because we’re asserting the Markov assumption in our model we can effectively forget

about these variables, because they precede what we care about in our probability

model:

这看起来似乎没什么用，但我们实际上可以忽略**S1，…，SK**在第一个参数概率中的部分。因为这些概率都属于有向分割的(d-separated)。在此我们不会过多讨论有向分割，因为假定在我们的马尔可夫模型中，可以忽略这些变量，因为这些变量与我们的概率模型无关：

注：1.有向分割d-separated这个概念是由Judea Pearl于1988年提出的算法的名字。这个算法是用来衡量图中的所有的条件独立关系。

令X, Y和Z是一个有向无环图 G中二个不相交节点的子集，如果在集合X和Y中所有节点间的所有路径都被集合Z所 阻塞，则称集合X和Y被Z集合d-s eparation。参考<http://www.andrew.cmu.edu/user/scheines/tutor/d-sep.html>

2. 原文是X1 X2.. Xk 但我认为是笔误

This is the Forward-Backward algorithm!这就是前向-后向算法!

Graphically, you can imagine this to be a path through this probability space (see

Figure 6-4). Given a specific emission at, say, index 2, we could calculate the probability

by looking at the forward and backward probabilities.

从图形上看，你可以把它想象成是一个通过概率空间的路径（见图6-4）。给定一个特定的输出状态(例如,S2)我们可以通过观察前向(S1->S2)和后向(S3->S2)的概率来计算其产生的概率。

The forward term is looking at the joint probability of the hidden state at point k

given all the emissions up to that point. The backward term is looking at the conditional

probability of emissions from k+1 to the end given that hidden point.

术语“前向”是指看在给定的节点K之前所有隐藏状态观测点所输出的联合概率。术语“后向”是指从节点K + 1到最后一个节点所有隐藏状态观测点所输出的联合概率

Using User Behavior 考虑用户习惯

Using our preceding example of Home→Signup→Product→Checkout, let’s calculate

the probability of that sequence happening inside our model using the Forward-

Backward algorithm. First let’s set up the problem by building a class called Forward

Backward:

以我们之前的例子序列链 “主页->注册->商品->结账” 来说，我们以前向-后向算法来计算这个例子中相关概率。我们首先建立一个类名叫ForwardBackward:

Here we are simply importing the information that we had from before—that is, the transition probability matrix and the emission probabilities. Next, we need to define our foward step, which is:

首先初始化之前所讨论的相关数据，如隐藏状态转换概率矩阵和输出概率，随后我们来定义前向(Forward)的具体步骤:

The forward algorithm will go through each state at each observation and multiply them together to get a forward probability of how the state works in this given con‐ text. Next, we need to define the backward algorithm, which is:

前向算法会依次遍历每个观察点的状态并乘以状态转换概率矩阵的概率，最后会根据给定的序列链得到前向概率。然后我们将定义后向算法：

注： 上述代码中 self.emission\_probability[k][self.observations[x\_plus]]

这一行有误，参数x\_plus不存在，应该是self.observations[i]

The backward algorithm works pretty much the same way as the forward one, except that it goes the opposite direction. Next, we need to try both forward and backward and assert that they are the same (otherwise, our algorithm is wrong):

后向算法的工作模式和前向算法类似，除了遍历时候的方向相反以外。然后我们将尝试前向后向算法是否工作正常。(否则，我们设计的算法就有问题)

The beauty of the Forward-Backward algorithm is that it’s effectively testing itself as it runs. This is quite exciting. It will also solve the problem of evaluation—remember, that means figuring out how probable a given sequence is likely to be. Next, we’ll delve into the decoding problem of figuring out the best sequence of underlying states.

前向-后向算法的魅力在于运行时候很方便被测试，这是非常让人所激动的。它很便于解决评估类的问题-但是需要记住，这意味着你需要指出最有可能出现的序列链。所以接下来，我们将深入研究如何找出隐藏状态的可见观测点的最佳序列链。（这一过程叫求解最大似然路径，在语音识别领域也称之为解码）

The Decoding Problem Through the Viterbi Algorithm通过维特比算法来求解

The decoding problem is the easiest to describe. Given a sequence of observations, we want to parse out the best path of states given what we know about them. Mathemati‐ cally, what we want to find is some specific π\* = arg max π P(x, π), where π is our state vector and x is the observations.

解码的问题其实描述起来很容易:基于给定的一系列观测点，我们想求解最有可能的序列，基于该序列路径我们所观测的结果概率最大。用数学公式来描述就是我们想找到一系列特定的π\* = arg maxπP（x，π），π代表的是我们的状态向量，x代表的是可见观测点

To achieve this, we use the Viterbi algorithm. You can think of this as a way of con‐ structing a maximum spanning tree. We are trying to figure out, given our current state, what is the best path to approach next. Similar to any sort of greedy algorithm, the Viterbi algorithm just iterates through all possible next steps and takes it.

Graphically, it would look something like Figure 6-5.

为此，我们将使用维特比算法。你可以把这想象成是一种构造最大生成树的方法。我们尝试去寻找的就是基于当前的状态的最大似然路径。和最大贪婪算法有些类似，维特比算法就是通过迭代所有可能的下一个状态概率然后计算出最有可能的路径状态序列

What we see in this figure is how a state like S1 will become less relevant over time, while a state of S3 becomes even more relevant compared to the others. The arrows are shaded to show the probability dampening.

根据上图我们可以发现状态S1是与时序无关的，状态S3则与其它的状态关系紧密。用浅色显示的箭头代表的是衰减的概率。

What we are attempting to do with this algorithm is traverse a set of states in the most optimal way. We do this by determining the probability that a state will happen given its emissions as well as the probability that it will transition from the previous state to the current. Then we multiply those two together to get the probability that the sequence will happen. Iterating through the entire sequence, we eventually find our optimal sequence.

我们将尝试用这个算法来以最佳的方式遍历状态组集合。通过概率确定的状态和下一个将发生的状态输出概率以及从之前状态变成当前状态的转移矩阵概率。然后将这两者相乘就能得到序列将要发生的概率。依次遍历整个序列，最终我们就能找到最优序列。

The Learning Problem学习问题

The learning problem is probably the simplest algorithm to implement. Given a sequence of states and observations, what is the most likely to happen next? We can do that purely by figuring out the next step in the Viterbi sequence. We figure out the next state by maximizing the next step given the fact there is no emission available yet. But you can figure out the most probable emission from there as well as the most probable state, and that is known as the next optimal state emission combo.

学习问题可能是最简单的算法实现。给定一状态序列和对应的观察点，哪一个才是最有可能发生的下一个？通过算法我们就能逐步找出下一步，最后得出维特比序列。我们计算出下一个状态是基于下一步的最大出现概率，事实上此时并没有实际输出。但是，确可以计算出最可能的输出行为，以及最可能的状态，这就是所谓的下一个最佳状态输出组合。

If this way of solving doesn’t make sense yet, don’t worry: in the next section, we will delve further into using the Viterbi algorithm.

如果这种方法并没有解决问题，不要担心：在下一节中，我们将深入探讨使用维特比算法。

Unfortunately, there isn’t any free and easily accessible data available for analyzing user behaviors over time given page views, but there is a similar problem we can solve by using a part-of-speech tagger built purely using a Hidden Markov Model.

不幸的是，我们并没有获取到基于用户某个时间段访问网页的简单数据记录，因此也没有办法以此来分析用户的行为。但这里有一个类似问题，我们可以基于一份已经标注了词性的演讲稿，以隐马尔科夫模型(Hidden Markov)来尝试解决该问题。

Part-of-Speech Tagging with the Brown Corpus

Given the phrase “the quick brown fox,” how would you tag its parts of speech? We know that English has parts of speech like determiners, adjectives, and nouns. We would probably tag the words in this phrase as determiner, adjective, adjective, noun, respectively. We could fairly easily tag this example because we have a basic under‐ standing of grammar, but how could we train an algorithm to do so?

Well, of course because this is a chapter on HMMs, we’ll use one to figure out the optimal parts of speech. Knowing what we know about them, we can use the Viterbi algorithm to figure out, for a given sequence of words, what is the best tagging sequence. For this section, we will rely on the Brown Corpus, which was the first elec‐ tronic corpus. It has over a million annotated words with parts of speech in it. The list of tags is long, but rest assured that it contains all the normal tags like adjectives, nouns, and verbs.

The Brown Corpus is set up using a specific kind of annotation. For each sequence of words, you will see something like this: