

Web site user modeling with PHP

Skill Level: Introductory

[Paul Meagher \(paul@datavore.com\)](mailto:paul@datavore.com)

CEO

Datavore Productions

30 Dec 2003

Web site user modeling, a mathematical discipline, is easier than you might expect. In this tutorial, Paul Meagher shows you how to construct a user-modeling platform with PHP and MySQL -- technologies well suited for a species of user-modeling called Web site user modeling. Even small Web-development shops can use clickstream data to build Web site user models.

Section 1. Introduction

Power to the people

User modeling is an advanced feature of software systems. Universities, research institutes, and large companies take the lead in developing advanced software systems that incorporate user modeling engines, but that shouldn't exclude the small Web-development shops that believe it is beyond their reach to incorporate user-modeling technologies into their Web sites.

In this tutorial, I will show you how to construct a user-modeling platform without esoteric technologies. PHP and MySQL technologies are up to the task and furthermore, they may be the best technologies to use for a species of user-modeling called *Web site user modeling*.

Web site user modeling is a mathematical discipline; the math involved makes many people think this technique would pose a major stumbling block for them to incorporate into their Web sites. Therefore, a second objective of this tutorial is to show you that the relevant mathematics is not necessarily as difficult as you may

expect. In this tutorial, I will discuss two mathematical concepts:

1. Transition matrices
2. Markov processes

These mathematical concepts will provide a simple and powerful perspective on how to model for Web site users.

What you need

This tutorial provides a brief review of the math used in an area of probability theory called the *Markov process theory* (or *Markov chain theory*).

Basic familiarity with matrices and probability are useful, but I will provide introductory coverage of the relevant concepts and mathematics. This tutorial can be a good starting point to familiarize yourself with Markov process theory; since it provides an interesting example application, namely Web site user modeling, it should make the mathematical concepts seem real and relevant.

Mainly, you need a willingness to learn some mathematical concepts. These concepts are presented using examples and some math notation, but primarily I focus on these concepts as reflected in their implementation as PHP classes. For example, you will map the mathematical concept of a transition matrix onto instance variables and methods supplied by a `TransitionMatrix.php` class. This approach invites you to learn more by downloading the code, playing with it, then extending it further as you learn more about these mathematical concepts.

Why should I care?

In my opinion, clickstream data is being thrown away by most Web developers, jettisoned because they are not looking at clickstream data from the point of view of its user-modeling potential. Instead, after the fact, they run report generators to figure out the number of visitors per hour, day, week, and so on.

A count of visitors is useful information, but it does not exhaust the potential of clickstream data to inform you about how sites are individually used and how they might be dynamically reconfigured based upon up-to-date user-modeling data for each individual site visitor.

The idea of Web site user modeling suggests that developers might instead use clickstream data to update dynamic models of individuals users as they click through sites. Is this possible? If so, how can it be done? These are the questions that this

tutorial will explore and propose answers to.

I will explore the Markov process theory as a way to solve a specific user-modeling problem, namely that of predicting what Web page a user will visit next. The section on Markov process theory brings with it many other applications that are only lightly touched upon in this tutorial. Should you wish to pursue Markov process theory beyond this tutorial, you will find a research area that is undergoing a renaissance pertaining to its application in many academic disciplines, from psychology to physics to compression algorithms to search engines to some of the hottest areas in statistics and mathematics. Hopefully, this tutorial will also provide you with a sense of what the Markov process buzz is about.

Tutorial outline

This tutorial has five sections:

- The first section, "[What is a Web site user model?](#)," discusses in detail the components of Web site user modeling and highlights recent research in this area. This research suggests a simple three-component Web site user model as a way to organize thinking and research about Web site user modeling.
- The second section, "[Fast and lean transition matrices](#)," discusses the concept of a transition matrix and how to implement it using PHP and MySQL.
- The third section, "[Encoding visitor clickstreams](#)," discusses how to use the `TransitionMatrix.php` class along with cookie and session variables to gather transition frequency data for each Web site user.
- The fourth section, "[Modeling users as Markov processes](#)," discusses Markov process theory, which is concerned with calculating the probability of future states based on transition matrix data. In this section, I explain what a Markov process is and how you might implement some common Markov probability calculations in PHP. A `Markov.php` class implementing these calculations is discussed.
- In the final section, "[Wrap up](#)," I discuss clickstream data issues, other applications of Markov process theory, and ethical issues involved in gathering Web site user-modeling data.

Section 2. What is a Web site user model?

What is a model?

The sciences do not try to explain, they hardly even try to interpret, they mainly make models. -- John von Neumann

To define what a Web site user model is, it is useful to first focus on one term in this composite expression, namely the word "model". One appropriate definition of *model* is to associate it with the idea of having an accurate mathematical representation of the relationship between the variables you are examining.

An example of a model in this sense would be a mathematical formula that relates distance from a fire station to the amount of fire damage a property sustains. Intuitively speaking, you would expect there to be more fire damage sustained the further a burning house is from a fire station. The formula that expresses this relationship can be captured precisely with this simple linear equation:

```
Damage =  
10.30 +  
(4.92 *  
Distance)
```

In PHP, you could use arrays to further clarify the relationship:

```
$Damage[$i]  
= 10.30 +  
4.92 *  
$Distance[$i]
```

This example can be reviewed in more detail in an article entitled "Simple Linear Regression with PHP: Part 2," found in [Resources](#) . (The table in [The fire damage study table](#) is from that article.)

The fire damage study table

The first column of this table records the distance measurement and the second column records the fire damage measurement (multiplied by \$1,000). The third column shows the amount of fire damage predicted by my simple linear model. The fourth column shows the degree of error in my prediction, which in this case is very little. The fifth and sixth columns tell me that 95 percent of the time, I can expect that

the observed fire damage cost will fall within the lower and upper limits defined in these columns.

Fire Damage Study

Table Summary					
Distance (mi)	Damage (x1000)	Predicted	Residual	Lower 95%	Upper 95%
3.4	26.6	27.03	-0.43	21.88	32.18
1.8	17.8	19.16	-1.36	13.86	24.46
4.6	31.3	32.94	-1.64	27.67	38.20
2.3	23.1	21.62	1.48	16.40	26.83
3.1	27.5	25.55	1.95	20.40	30.71
5.5	36.0	37.36	-1.36	31.88	42.85
0.7	14.1	13.74	0.36	8.15	19.34
3.0	22.3	25.06	-2.76	19.91	30.22
2.6	19.6	23.09	-3.49	17.91	28.28
4.3	31.3	31.46	-0.16	26.24	36.68
2.1	24.0	20.63	3.37	15.39	25.88
1.1	17.3	15.71	1.59	10.24	21.18
6.1	43.2	40.32	2.88	34.64	45.99
4.8	36.4	33.92	2.48	28.61	39.23
3.8	26.1	29.00	-2.90	23.83	34.17

Further goodness-of-fit statistical analysis (as reported in the article to which this table belongs) confirmed that this simple linear model was explaining most of the variability in the data and would be a useful predictive model relative to just using the average fire damage cost.

When I use the term "model" in this tutorial, I would ask you to keep this example in mind as the prototypical example of how I use the term. A model should be expressible as a mathematical relationship among variables and be useful for descriptive and predictive purposes.

Web site user model defined

A Web site user model can be precisely defined as follows:

A Web site user model is a mathematical representation of Web site users that can be used to describe and predict aspects of how they use a Web site.

A good question to ask at this point is where to find a concrete example of a Web site user model as I just defined it. Fortunately, the article "A Model of Web Site Browsing Behavior Estimated on Clickstream Data," Bucklin and Sismeiro, from the *Journal of Marketing Research* (see [Resources](#)) is an excellent example. The research in this article provides a practical example of the potential components of a Web site user model, as well as a framework for thinking about the composition of a theoretically complete Web site user model. And the article offers practical insights into data collection and preparation issues that Web site user modelers need to be aware of.

User and page-specific covariates

Before examining Bucklin and Sismeiro's Web site user model, I will briefly review the source of their data and the predictor variables they investigated in their study.

The data analyzed consisted of Apache access logs from a popular automotive site for one month, October 1999. During the month they recorded 169,910 visits and 1,462,457 requests (which I presume are page requests). The authors were particularly interested in seeing if they could develop models to predict:

1. How many pages a Web site visitor would view (the *Page-Request Model*)
2. How long a Web site visitor would view a given page (the *Page-View-Duration Model*)

Bucklin and Sismeiro investigated a number of user- and page-specific covariates that could be measured at each instant in a user's clickstream. The values of these covariates were entered into their predictive equations and used to predict:

1. Whether visitors would request another Web page or leave the site
2. How long they would view a particular page

The next section defines the user and page-specific covariates that Bucklin and Sismeiro studied.

User and page-specific covariates defined

BYTES	Sum of the client-server and server-client bytes transferred for a given page (units = 100 kilobytes).
CMAKE	Cumulative number of vehicles configured by the visitor up to the current page view.
CPAGE	Cumulative number of page views for a visitor to arrive at the current page on a given visit or session.
CSESSION	Cumulative number of site visits made by the visitor as of the current page view.
DYNAMIC	Dummy variable set equal to 1 if the current page contains dynamic content (that is, content that requires access to the company's database) and set equal to 0 otherwise.
ERROR	Dummy variable set equal to 1 when at least one error occurred during the current page transfer (assuming the visitor eventually succeeds in correctly downloading the page) and set equal to 0 otherwise.
FIRSTMAKE	Dummy variable set equal to 1 if the current page was requested at or after the page in which the visitor first configured a car, and set equal to 0 otherwise.
ORDER	Dummy variable set equal to 1 if the current page view was requested at or after the page at which the visitor ordered a car, and set equal to 0 otherwise.
PREVDUR	Duration, in minutes, of the immediately preceding page view.
RELOAD	Dummy variable set equal to 1 if the visitor reloads the current page at least once (given no errors in transmission), and set equal to 0 otherwise.
SRVRESP	Total server response time (in 10,000ths of seconds).

The next section provides the means and standard deviations for each of these variables.

Means and standard deviations for each covariate

The means and standard deviations for each of these variables are provided so that you can gain a sense of the size of the observed values that are being plugged into the Page-Request and Page-View-Duration models (which I will discuss in

Page-Request Model).

Summary statistics for covariates

<i>Variable name and abbreviation</i>	<i>Mean</i>	<i>Standard deviation</i>
Bytes transferred (BYTES)	.275	.238
Number of cars configured (CMAKE)	.201	.817
Visit depth (CPAGE)	5.886	6.186
Repeat visits (CSESSION)	1.921	2.208
Dynamic page content (DYNAMIC)	.126	.332
Downloading error (ERROR)	.081	.273
First car already configured (FIRSTMAKE)	.146	.354
Car previously ordered (ORDER)	.005	.069
Previous page-view duration (PREVDUR)	1.633	2.909
Page reloaded (RELOAD)	.233	.423
Server response time (SRVRESP)	.003	.041

Page-Request Model

This parameter estimates table tells us what model co-efficients Bucklin and Sismeiro used in their Page-Request Model and their Page-View-Duration Models. Examine it, and in the next section I will discuss in more detail how to read this table.

Model parameter estimate (posterior means and 95 percent probability intervals for between-subjects mean parameters [population-level means])

	<i>Page-request model (Model C9)</i>	<i>Page-view-duration model (Model D8)</i>
INTERCEPT	1.885 [1.834, 1.943]	3.838 [3.813, 3.867]
BYTES	.211 [.126, .308]	.558 [.503, .613]

CMAKE	.186* [.105, .269]	-.062 [-.102, -.028]
CPAGE	-.711** [-.755, -.668]	-.079** [.063, .095]
CSESSION	-.194** [-.236, -.151]	N.S.
DYNAMIC	N.S.	-.400 [-.450, -.354]
ERROR	.427 [.317, .542]	.155 [.100, .216]
PREVDUR	-.220* [-.250, -.189]	N.S.
RELOAD	-.050 [-.102, -.006]	.373 [.344, .404]
SRVRESP	-8.485 [-9.434, -7.445]	8.016 [5.855, 10.272]

N.S. Did not provide an improvement of log of PSBF; removed from final estimation.

* Enters the model in log form after adding 1 to the variable.

** Enters the model in log form.

How to read the parameter estimates table

You will note that the table consists of two columns of numbers. This is because you need different parameter estimates in:

1. The equation for predicting whether a visitor will view another page (Page-Request Model)
2. The equation for predicting how long visitors will view a given page (Page-View-Duration Model)

Recall the fire damage model. It used distance from the nearest fire station to predict the amount of fire damage that would result. I might also have looked into the existing value of the property as another factor to consider. Common sense suggests that the original value of a property should vary positively with the dollar value of the reported damage. I might have incorporated this factor into the fire damage model by adding another term to the model as follows:

$$\$Damage[\$i] = 10.35 + (\$Distance[\$i] * 2.4) + (\$PropertyValue[\$i] * 0.3)$$

The parameter estimates in this model, 2.4 and 0.3, would be obtained by statistical algorithms designed to run on sample data and pick the *best* parameter estimates to use (that is, estimates that produce the smallest prediction error). The parameter estimates reported in the table in [Page-Request Model](#) were derived in a similar manner and used to construct two multi-term predictive equations similar to this multi-term fire damage model.

The first column in the parameter estimates table can be used to construct a multi-term equation for predicting the number of pages a Web site visitor will view in a particular session. This formula doesn't attempt to predict the length of a user's session through one computation; rather, it is meant to be applied at each stage in a user's clickstream to determine whether he will view another page. The relatively large negative effect associated with the CPAGE variable means that as a user clicks further into the site, the utility of an additional page view will decrease more and more (reflecting the time-constrained behavior of surfers), eventually resulting in a page view utility below an assumed threshold. When the page view utility falls below a threshold, the Page-Request Model predicts that the user will not view another page. The number of pages viewed during a visit is determined by the number of clicks it takes for a user's page view utility to fall below a utility threshold.

What does the Page-Request Model look like?

The Page-Request model, as a PHP computation, would look like this in all its detail:

```
$UTILITY[$i][$j][$k] = 1.885 + $BYTES[$i][$j][$k] * .221
                        + log($CMAKE[$i][$j][$k] + 1) * 0.186
                        + log($CPAGE[$i][$j][$k]) * -.711
                        + log($CSESSION[$i][$j][$k]) * -1.94
                        + $ERROR[$i][$j][$k] * .427
                        + log($PREVDUR[$i][$j][$k] + 1) * -.220
                        + $RELOAD[$i][$j][$k] * -.050
                        + $SRVRESP[$i][$j][$k] * -8.485
```

The left side of the equation is a three-dimensional array called \$UTILITY. The \$i subscript is used to denote a particular visitor. The \$j subscript is used to denote a particular session. The \$k subscript is used to denote a particular page view occasion. The \$UTILITY[\$i][\$j][\$k] array holds estimates of the likelihood that visitors will request an additional page for a particular session and page view occasion.

If the value of \$UTILITY[\$i][\$j][\$k] falls below a threshold at $k=5$ then, then \$k would not be incremented for the current value of \$i and \$j. You can compare the predicted value of \$k with the observed value of \$k (actual length of session \$j for visitor \$i) to see if the Page-Request Model is making accurate predictions. You

can also look at the prediction problem from the point of view of finding individual Utility threshold values that capture the most variance in the session-length random variable.

Note that some of the terms undergo a log transformation before being entered into the equation. This is a way to incorporate nonlinear relationships into the predictive equation. Note also that 1 is added to avoid the undefined operation of taking the log of 0 (CMAKE and PREVDUR can take on the value of 0 but not when 1 is added).

A theoretically complete Web site user model

Bucklin and Sismeiro's research provides a concrete example of what a Web site user model might look like. The Page-Request and Page-View-Duration models can potentially be used to generate predictions about whether a visitor will view another page and how long they will view a page. However, a large hole in their Web site user model prevents it from being theoretically complete.

Specifically, the model does not answer questions about what particular link a user will click next. It only tells you whether a user will click another page link and how long they will spend viewing pages. You just might want to know what particular link the user would select next.

Can you design a Page-Choice Model that would allow you to predict which page a user might view next? This model would need to be sensitive to the fact that the outcome variable you are trying to predict -- what page a user will visit next -- is a categorical outcome. The Page-Request and Page-View-Duration models both produced a numerical outcome that reflected the amount of a quantitative variable (Page-View-Utility and Page-View-Duration). Instead of outputting a numerical value with interval scale properties, a Page-Choice model needs to output the name of the next page the user is likely to visit.

How difficult is it to come up with a regression equation to predict what page a person will visit next? How would the different terms in a multi-term regression model add up to the name of the predicted next Web page? One possibility is to try to devise a regression model that generates a probability value for each page given the values of the current covariates. The page with the highest probability would be the predicted next page.

I considered but abandoned this approach because I saw that there might be a simpler approach. In [Fast and lean transition matrices](#), I will discuss Markov process theory as a simpler way to get started in solving the page choice prediction problem.

To sum up, a theoretically complete Web site user model (of Web site browsing behavior) should generate three types of predictions:

1. Will a user request another page (Page-Request Model)?
2. How long will a user view a page (Page-View-Duration Model)?
3. What page will a user go to next (Page-Choice Model)?

I call a model that generates predictions about these three aspects of Web surfer behavior theoretically complete because it could be used to implement an artificial surfer exhibiting many of the same clickstream patterns as a real user.

Improving Web sites through Web site user models

Before moving on, I should point out that you can use Web site user models such as Bucklin and Sismeiro's to suggest factors to examine in order to improve the number of page views per session and the overall duration of these sessions. You can formulate a minimax design principle as follows:

If you want to maximize the number of pages requested per session or the overall session duration, look at ways to increase the values feeding into model factors having positive co-efficients and decrease the values feeding into model factors having negative co-efficients.

The minimax principle suggests that you look at all significant model factors and think about what measures might be taken to either increase or decrease their value depending on whether the factor enters as a positive or negative influence. You might also want to prioritize model factors having large coefficients.

Section 3. Fast and lean transition matrices

Demo Web site

Imagine a Web site consisting of these five pages:

1. Home page.
2. News page.
3. Products page.

4. Services page.
5. About page.

In the discussion of Web site user modeling in this section, I will focus on how you can use transition matrices to keep track of page transition frequencies and probabilities for each Web site visitor.

Frequency transition matrix

The frequency transition matrix for a particular user of the five-page Web site would look like this:

	<i>home(t)</i>	<i>news(t)</i>	<i>products(t)</i>	<i>services(t)</i>	<i>about(t)</i>
<i>home(t-1)</i>	f_{00}	f_{01}	f_{02}	f_{03}	f_{04}
<i>news(t-1)</i>	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
<i>products(t-1)</i>	f_{20}	f_{21}	f_{22}	f_{23}	f_{24}
<i>services(t-1)</i>	f_{30}	f_{31}	f_{32}	f_{33}	f_{34}
<i>about(t-1)</i>	f_{40}	f_{41}	f_{42}	f_{43}	f_{44}

You can use a frequency transition matrix like this to record the number of times a user goes from one page to another. For example, if a new user visits the home page, then goes to the news page, you would increase the frequency count in the f_{01} cell by one.

	<i>home(t)</i>	<i>news(t)</i>	<i>products(t)</i>	<i>services(t)</i>	<i>about(t)</i>
<i>home(t-1)</i>	0	1	0	0	0
<i>news(t-1)</i>	0	0	0	0	0
<i>products(t-1)</i>	0	0	0	0	0
<i>services(t-1)</i>	0	0	0	0	0
<i>about(t-1)</i>	0	0	0	0	0

You can precisely describe a frequency-transition matrix-updating algorithm as follows:

1. Select the row to update by noting what page the user is viewing at time $t-1$ (such as, home.php).
2. Select the column to update by noting what page the user is now viewing at time t (such as, news.php).

- Increment the cell frequency count by 1 where the selected row and column intersect.

Just to make sure you understand how the transition-frequency updating algorithm works, imagine that the visitor now proceeds from the news.php page to the products.php. Using the updating algorithm just described, you would increment the f_{12} by one resulting in this updated frequency transition matrix:

	<i>home(t)</i>	<i>news(t)</i>	<i>products(t)</i>	<i>services(t)</i>	<i>about(t)</i>
<i>home(t-1)</i>	0	1	0	0	0
<i>news(t-1)</i>	0	0	1	0	0
<i>products(t-1)</i>	0	0	0	0	0
<i>services(t-1)</i>	0	0	0	0	0
<i>about(t-1)</i>	0	0	0	0	0

Note that the number of transitions recorded is less than the number of pages visited because a transition is only recorded after the second page visit in a session occurs.

Probability transition matrix

Recording user clickstream data using a frequency transition matrix allows for efficient storage of information about how a user tends to travel through a site. Instead of storing raw data about what links a user visits during a particular session, you instead encode their path information in terms of the number of transitions a user makes from one page to another in the example Web site. The addition of new transition data only increases the value of a stored counter and does not require allocating new storage space to hold new transition frequency data.

The frequency transition matrix will only require more storage capacity if you add new pages to the site (and thus increase the dimensions of the matrix). A frequency transition matrix is, therefore, a compact way to store the data to be used by our Page-Choice Web site user model.

Another desirable feature of transition matrices, in addition to storage efficiency, is that they allow you to easily generate predictions about what page a user might visit next. To see how this is possible, I need to show how you to convert a frequency transition matrix to a probability transition matrix.

Converting frequency to probability

Look at the frequency transition matrix for the hypothetical user but add more transition frequency data to reflect the user's continued interaction with the Web site

across visits.

	<i>home(t)</i>	<i>news(t)</i>	<i>products(t)</i>	<i>services(t)</i>	<i>about(t)</i>
<i>home(t-1)</i>	0	12	8	4	3
<i>news(t-1)</i>	8	13	8	2	5
<i>products(t-1)</i>	5	8	8	12	3
<i>services(t-1)</i>	6	1	11	6	4
<i>about(t-1)</i>	8	7	1	3	0

To convert this frequency transition matrix to a probability transition matrix, all you need to do is sum the values in each row to obtain row marginals. The first row marginal would be 27 (12 + 8 + 4 + 3), the second row marginal 36 (8 + 13 + 8 + 2 + 5), and so on. To obtain a probability value for each cell, simply divide the cell frequency by its row marginal.

	<i>home(t)</i>	<i>news(t)</i>	<i>products(t)</i>	<i>services(t)</i>	<i>about(t)</i>
<i>home(t-1)</i>	0/27	12/27	8/27	4/27	3/27
<i>news(t-1)</i>	8/36	13/36	8/36	2/36	5/36
<i>products(t-1)</i>	5/31	8/31	8/31	12/31	3/31
<i>services(t-1)</i>	6/28	1/28	11/28	6/28	4/28
<i>about(t-1)</i>	8/19	7/19	1/19	3/19	0/19

The final form of the probability transition matrix looks like this:

	<i>home(t)</i>	<i>news(t)</i>	<i>products(t)</i>	<i>services(t)</i>	<i>about(t)</i>
<i>home(t-1)</i>	0	.444	.296	.148	.111
<i>news(t-1)</i>	.222	.361	.222	.055	.138
<i>products(t-1)</i>	.138	.222	.222	.333	.083
<i>services(t-1)</i>	.214	.036	.393	.214	.143
<i>about(t-1)</i>	.421	.368	.053	.158	0

Using the probability transition matrix to generate predictions

To see how you might use this table to generate a prediction about the next page a user will visit, you can change the time subscripts from *t-1* to *t* for the rows and from *t* to *t+1* for the columns. This is just another way of depicting the transition data, one that allows you to see how the transition matrix can be used in a predictive role.

	<i>home(t+1)</i>	<i>news(t+1)</i>	<i>products(t+1)</i>	<i>services(t+1)</i>	<i>about(t+1)</i>
--	------------------	------------------	----------------------	----------------------	-------------------

<i>home(t-1)</i>	0	.444	.296	.148	.111
<i>news(t-1)</i>	.222	.361	.222	.055	.138
<i>products(t-1)</i>	.138	.138	.222	.333	.083
<i>services(t-1)</i>	.214	.036	.393	.214	.143
<i>about(t-1)</i>	.421	.368	.053	.158	0

Begin by asking what page this user is likely to go to next after arriving at the home page first. You can find the most likely next page he will visit by looking at the probabilities appearing in the first row of the table. This row tells you that when the user visits the home page at time t , the next page they tend to look at ($t + 1$) is the news page. The news page has the highest probability of being the next page the user selects after the home page. The probability is precisely 44 percent that the user will select the news page next given that they selected the home page. It is wonderful what you can do with a table full of conditional probabilities at your disposal!

To make your prediction, you can take other information into account. This example, however, demonstrates that you can quickly generate real-time predictions about what page a user might visit next by simply consulting the relevant row of his probability transition matrix.

I will revisit the issue of combining other information into the prediction later when I discuss the issue of individual and collective transition matrices.

TransitionMatrix.php instance variables

Now that you know what a frequency and probability transition matrix is, I can get to the business of implementing a transition matrix class in PHP. I begin by defining what instance variables I might want for the `TransitionMatrix.php` class.

The discussion of transition matrices highlighted the fact that they come in two forms: frequency and probability transition matrices. When you store transition data about a user, you can store only transitional frequencies and compute transitional probabilities from this data as needed. For this tutorial, I have opted for the less storage-efficient scheme of storing both frequency and probability transition matrices for each user. The `$freq_matrix` and `$prob_matrix` instance variables are meant to hold these two types of transition matrices for each user.

```
<?php
class TransitionMatrix {
    // Container for frequency representation of transition matrix
    var $freq_matrix = array();

    // Container for probability representation of transition matrix
```

```

var $prob_matrix = array();

// Keep ongoing tally of $num_transitions
var $num_transitions = 0;

// Stores number of rows in user's transition matrices
var $num_rows = 0;

// Stores number of columns user's transition matrices
var $num_cols = 0;

// Store the current state (t) because we will need to retrieve it
// when we enter the next state (t + 1) so that we can update the
// transition matrix
var $state = null;
}
?>

```

For every Web site visitor I will instantiate a transition matrix and update all these instance variables as the visitors goes through the site.

To maintain an up-to-date transition matrix for each user you need to be able to restore, update, and store their transition matrix. In other words, you need to implement a persistent storage container for the Web site visitor's transition matrix and a way to update it with each click they make. This is what the `TransitionMatrix()` methods do.

TransitionMatrix.php storage methods

The storage approach I have taken is to store the transition matrix object for each user in a MySQL binary large object field (blob field). This means that for the code I am showing you to work, you must first create a MySQL database table to hold the transition matrix objects for each visitor. Here is the SQL you can use to create it:

```

CREATE TABLE visitors (
  vis_id varchar(32) NOT NULL,
  vis_tm_obj blob NOT NULL,
  PRIMARY KEY (vis_id)
) TYPE=MyISAM;

```

I use a cookie with the user's visitor ID as the lookup key to store and restore a Web site visitor's transition matrix. My `TransitionMatrix` API includes a `restore($visitor_id)` and a `store($visitor_id)` method that implements the lookup and storage routines for the Web site visitor's transition matrix.

```

<?php
class TransitionMatrix {

```

```
/*
 * INSERT or UPDATE the vis_tm_obj field depending whether a record
 * for $vis_id already exists.
 */
function store($vis_id, $obj) {
    include_once "connect.php";
    $vis_tm_obj = serialize($obj);
    $sql = " REPLACE INTO visitors (vis_id, vis_tm_obj) ";
    $sql .= " VALUES ('$vis_id', '$vis_tm_obj') ";
    if (mysql_query($sql)) {
        return true;
    } else {
        return false;
    }
}

/*
 * Restore the transition matrix for a particular visitor.
 */
function restore($vis_id=null) {
    include_once "connect.php";
    $sql = " SELECT vis_tm_obj FROM visitors WHERE vis_id='$vis_id' ";
    if ($result = mysql_query($sql)) {
        $this = unserialize(mysql_result($result, 0));
    } else {
        return false;
    }
}
}
?>
```

The idea of storing an object to represent a user as a binary large object in a database is a useful general approach to consider when thinking about how to maintain real-time Web site user models.

The included connect.php file establishes a database connection prior to issuing the SQL commands.

Now that you know how to store and restore the transition matrix for a given user, you can move onto figuring out how to update it as they click a new link in your site.

TransitionMatrix.php update method

The code below is the update method. As you can see, the computation involved in updating the *frequency* transition matrix is trivial. The numerical values representing the previous and current page visited is used as indices to the appropriate cell in the frequency transition matrix. The total number of transitions to date is also updated. You then update the probability transition matrix using a loop over the relevant row of the probability transition matrix. Finally, you record the value of the current page in the state variable so that you can use it to figure out what the previous page was on the next page.

```

<?php
class TransitionMatrix {
    /*
    * When a state change is added to the transition matrix,
    * all the class instance variables need to be updated
    */
    function update($prev_state, $curr_state) {

        // first update cell count in the frequency matrix
        $this->freq_matrix[$prev_state][$curr_state]++;

        // now update relevant row of the probability matrix
        $this->num_transitions++;
        $this->num_rows = count($this->freq_matrix);
        $this->num_cols = count($this->freq_matrix[0]);
        $row_sum = array_sum($this->freq_matrix[$prev_state]);
        for ($i = 0; $i < $this->num_cols; $i++) {
            $this->prob_matrix[$prev_state][$i] =
                $this->freq_matrix[$prev_state][$i] / $row_sum;
        }

        // set transition matrix state to current state
        $this->state = $curr_state;
    }
}
?>

```

The power of transitional modeling

I can think of no better demonstration of the potential power of this transitional modeling approach than to cite an example that was devised by Claude Shannon in his superb monograph "A Mathematical Theory of Communication" (in [Resources](#)). This example presents a series of approximations to English, based on using different types of sampling algorithms.

1. Zero-order approximation (symbols independent and equiprobable).

XFOML RXKHRJFFJUJ ZLPWCFWKCYJ FFJEYVKCQSGHYD QPAAMKBZAACIBZL-
HJQD.

2. First-order approximation (symbols independent but with frequencies of English text).

OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA
NAH BRL.

3. Second-order approximation (digram structure as in English).

ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TU-
COOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE.

In this example, Shannon provides a way to visualize the workings of different

sampling algorithms. Using no rhyme, reason, or rules, if you sample from a pool of letters and spaces you get this first batch of letters. If you sample according to the single-letter frequencies found in English text, you get the next batch of letters. If you sample according to "digram frequencies," you get the third batch of letters. The digram frequencies for English would be obtained by analysing a corpus of text and recording the frequencies with which different pairs of letters occur.

A good data structure to store digram frequencies is a transition matrix. Therefore, this third example demonstrates that sampling from a transition matrix is predicting some of the structure in the flow of English text and, by implication, the flow of link choices by individual users.

Enlarging the context

Interestingly, using trigram frequencies makes sampled text even more similar to English text:

4. Third-order approximation (trigram structure as in English).

IN NO IST LAT WHEY CRATIC T FROURE BIRS GROCID PONDENOME OF DEMONS-
TURES OF THE REPTAGIN IS REGOACTIONA OF CRE.

5. First-order word approximation. Rather than continue with tetragram, ..., n -gram structure it is easier and better to jump at this point to word units. Here words are chosen independently but with their appropriate frequencies.

REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NAT-
URAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES
THE LINE MESSAGE HAD BE THESE.

6. Second-order word approximation. The word transition probabilities are correct but no further structure is included.

THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHAR-
ACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT
THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED.

Note that it may be the syllabic structure of (English) language that makes using trigram frequencies produce more English-like text than digram frequencies. This improvement might not carry over to modeling Web site users when the reasons for three-state combinations are less apparent. Note also that three-state combinations are common in genetic sequence data.

The design of efficient algorithms that use larger contexts (n -gram contexts) to make predictions about what will appear next is an active area of research in the compression literature (often referred to as "Predictive Coding"). The "prediction with partial map" algorithms that you can find in this literature might offer another approach to modeling page choice patterns in clickstream data. Before you do that,

however, you will want to examine whether a simpler transition matrix approach will work because it is likely to be less of a CPU and memory drain than wider context-based algorithms. As they say in extreme programming, do the simplest thing that works.

Now I turn to the issue of how to integrate the `TransitionMatrix.php` class into a real Web site.

Section 4. Encoding visitor clickstreams

Cookies, IPs, and member IDs

Two of the main methods you can use to track *anonymous* users across sessions are:

1. A persistent visitor cookie
2. The IP of the user

If you have a site login, then you can use login member IDs to identify users' repeat visits as well. None of these methods is 100 percent reliable in identifying visitors across sessions. The most reliable approach would involve some combination of these methods.

The simplest and most feasible approach to implement is to use the persistent cookie approach. This involves depositing a visitor ID cookie on the visitor's machine and using it to do any logging, updating, or lookup of information about the visitor.

If a user does not accept persistent cookies, then your record of that individual will consist of a series of one-session encounters under different visitor IDs. You might be able to piece these together by logging the IP of each visitor ID, but this tutorial will not cover these implementation details.

I want to focus on providing a model Web site you can use to gain some intuition about how a transition matrix might be used to model the Page-Choice dynamics of Web site users.

Implementing the demo Web site

In this section of the tutorial, I discuss the implementation of the demo Web site and how the transition logging code can be integrated into it.

Recall that the demo Web site allows a user to click on a page of links with these labels:

- Home
- News
- Products
- Services
- About

In the implementation of the demo Web site, all page content is served by a single script called `page.php`. This script accepts a query string argument telling it what page the user requested and will then serve the content for the requested page.

When you include the `TransitionMatrix.php` class into the `page.php` script, what the transition matrix object actually encodes is the sequence of states that the `page.php` script is going through as the user clicks different links. The idea that a session consists of a sequence of *page states* is a useful one because for many dynamic Web sites the concept of a page breaks down.

In addition to showing you how to embed transition logging code, this section will also show you how to embed code that logs traditional information about a page hit. I include this code to reinforce the idea that you should do both types of logging. I also included it because it allows me to demonstrate another Web site user modeling technique that you might want to think about -- logging the values of cumulative session variables (such as cumulative page-view duration, cumulative number of clicks, cumulative number of bytes served, and more).

I will demonstrate a proof-of-concept example of how to log the cumulative number of links a user has clicked in a session. The idea is that this might be useful information to have available in real time when making predictions about whether a user will request another page or how long the user might spend on the current page. Recall that cumulative page-view duration, cumulative number of clicks, and cumulative number of bytes served were variables that entered as significant in Bucklin and Sismeiro's Page-View and Page-View-Duration models.

The demo Web site consists of the following five files which will be individually discussed in the next parts of this section.

- `page.php` - serves all Web site pages (or all page states).
- `header.php` - included at the top of the `page.php` script.

- navbar.php - included at the top of the page.php script.
- footer.php - included at the bottom of the page.php script
- TransitionMatrix.php

Page controller

The output of the page .php script looks like this:

```
Home | News | Products | Services | About | Model Data

Welcome

Economics is extremely useful as a form of employment for economists.
-- John Kenneth Galbraith

INSERT INTO access_log (log_vis_id, log_ip, log_sess_id, log_state, log_bytes_sent,
log_exec_time, log_serv_time, log_cum_clicks) VALUES
('bad5559b9281b8a432857ab135e2cf8e', '156.34.57.18',
'7b45930c8c0a3331e143b105042bc3ad', '0', 402, 0.018839001655579,
1068759611,73)
```

The site starts in the home state by showing the "Welcome" headline and a random Unix fortune. As you click from link to link, the headline title changes and another random Unix fortune is presented. The header.php and footer.php files include the critical code that performs the user tracking.

The code for this screen follows in [Page controller code](#).

Page controller code

```
<?php
// page.php
// Copyright 2003, Paul Meagher
// Distributed under PHP License v3.00
/*
 * Script to demonstrate storage, updating and retrieval of transition matrix
 * using visitor id embedded in client cookie.
```

```

*/
?>
<html>
  <head>
    <title>Demo Web site</title>
  </head>
  <body>

    <?php
    // if no state is set, default to 0 state which is the home state
    if (!isset($_GET["state"]) ) {
      $_GET["state"] = 0;
    }

    require_once "header.php";
    require_once "navbar.php";

    // Display page title
    switch($_GET['state']) {
      case 0:
        echo "<h3>Welcome</h3>";
        echo "<hr />";
        break;
      case 1:
        echo "<h3>Latest News</h3>";
        echo "<hr />";
        break;
      case 2:
        echo "<h3>Our Products</h3>";
        echo "<hr />";
        break;
      case 3:
        echo "<h3>Our Services</h3>";
        echo "<hr />";
        break;
      case 4:
        echo "<h3>About Us</h3>";
        echo "<hr />";
        break;
      case 5:
        echo "<h3>Model Data</h3>";
        echo "<hr />";
        // Restore transition matrix for visitor from mysql database
        if (isset($_COOKIE["vis_id"])) {
          $tm->restore($_COOKIE["vis_id"]);
        }
        // Display current value of $_COOKIE['vis_id']
        echo "vis_id: ". $_COOKIE["vis_id"] . "<br/>";
        // Display current value of $_COOKIE['sess_id']
        echo "sess_id: ". $_COOKIE["sess_id"] . "<br/>";
        // Display current value of $_GET['state']
        echo "state: ". $_GET["state"] . "<br/>";
        // Print contents of the restored transition matrix which represents
        // the most current value of the transition matrix.
        echo "<p>";
        echo "<pre>";
        print_r($tm);
        echo "</pre>";
        echo "</p>";
        echo "<hr />";
        break;
    }

    // Give the user a little humor while clicking around
    echo "<pre>";
    system("/usr/games/fortune");
    echo "</pre>";

```

```
echo "<hr />";

require_once "footer.php";
?>

<body>
</html>
```

Header script

The `header.php` script contains initialization code for starting a script execution timer, setting a visitor cookie, updating the visitor's transition matrix, and turning on output buffering so you can measure and log the number of bytes transferred. This code is intended to be included on every Web site page.

The code is liberally sprinkled with comments so you can figure out what is going on. The order of code execution in this script is critical -- you will notice this if you try to rearrange the order of execution. Note also the section of code where you register a "cum_clicks" cookie variable that is used to record the number of clicks a user has made so far in his session. I suggested previously that you might want to record, for Web site user modeling purposes, other cumulative session variables such as the cumulative duration and the cumulative bytes served. These are left as an exercise for the reader.

Another point that should be noted about this code is how easy PHP makes it to manage session-related information. The `$_SESSION` and `$_COOKIE` arrays are much easier to work with and more secure than the older session handling methods that PHP provided (such as the `session_register()` method). The security, ease of use, and programmability of PHP sessions is one reason why PHP should be considered as a platform for Web site user modeling.

The header script code follows in [Header script code](#).

Header script code

```
<?php

// header.inc

// Copyright 2003, Paul Meagher
// Distributed under PHP License v3.00

// timer function
function get_micro_time(){
    list($usec, $sec) = explode(" ",microtime());
    return ((float)$usec + (float)$sec);
}

// start our timer
```

```
$timer_start = get_micro_time();

require_once "../TransitionMatrix.php";

// Create default transition matrix for visitors
$tm = new TransitionMatrix;

// Make it easy to change the num of states
$num_states = 5;

// if visitor cookie does not exist, create one
// and then create a default transition matrix
// for the visitor
if(!isset($_COOKIE['vis_id'])) {

    // Set a client side vis_id tag that will not
    // expire in the near future
    $cookie_expiry = time()+86400*30*12; // ~12 months
    $_COOKIE["vis_id"] = md5(uniqid(rand(), true));
    setcookie("vis_id", $_COOKIE["vis_id"], $cookie_expiry);

    // Initialize visitor transition matrix
    $tm->initialize($_GET["state"], $num_states, $num_states);

    // Count the frequency of being in this state
    $tm->count_states($_GET["state"]);

    // Now store the default transition matrix for this visitor
    $tm->store($_COOKIE["vis_id"], $tm);

}

// Create a cookie-based session id if one does not exist.
if (!isset($_COOKIE["sess_id"])) {
    $_COOKIE["sess_id"] = md5(uniqid(rand(), true));
    setcookie("sess_id", $_COOKIE["sess_id"]); // session cookie
}

// Create a cookie-based number of clicks variable
if (!isset($_COOKIE["cum_clicks"])) {
    $_COOKIE["cum_clicks"] = 1;
    setcookie("cum_clicks", $_COOKIE["cum_clicks"]);
} else {
    $_COOKIE["cum_clicks"] = $_COOKIE["cum_clicks"] + 1;
    setcookie("cum_clicks", $_COOKIE["cum_clicks"]);
}

// The visitor cookie must exist before we can update the transition matrix
if (!isset($cookie_expiry)) {

    // Restore the visitors transition matrix then and assign $prev_state
    // the restored value in $tm->state
    $tm->restore($_COOKIE["vis_id"]);
    $prev_state = $tm->state;

    // Update the transition matrix
    $tm->update($prev_state, $_GET["state"]);

    // Count the frequency of being in this state
    $tm->count_states($_GET["state"]);

    // Store the update
    $tm->store($_COOKIE["vis_id"], $tm);

}

// start output buffering
ob_start();
```

```
?>
```

Web site footer

The footer provides finalization code for getting the number of bytes transferred, for flushing the buffer to the browser, recording the script execution time, and inserting a record in the `access_log` table.

```
<?php
// footer.php
// Copyright 2003, Paul Meagher
// Distributed under PHP License v3.00

// get size of file
$bytes_sent = ob_get_length();

// flush the buffer
ob_end_flush();

include_once "connect.php";

// get $time_end - $time_start was computed at beginning of script.
$timer_end = get_micro_time();

// compute script execution time
$exec_time = $timer_end - $timer_start;

// The time to insert the webstats record is not being included in
// the script execution time - this $exec_time value will be an
// underestimate unless we add some guesstimate of the insertion time.

$sql = " INSERT INTO access_log ";
$sql .= " (log_vis_id, log_ip, log_sess_id, log_state, log_bytes_sent,
        log_exec_time, log_serv_time, log_cum_clicks) ";
$sql .= " VALUES ";
$sql .= " ('".$_COOKIE["vis_id"]."', '".$_SERVER['REMOTE_ADDR']."',
        '".$_COOKIE["sess_id"]."', '".$_GET["state"]."', ";
$sql .= " $bytes_sent, $exec_time, ". time() .", ".$_COOKIE["cum_clicks"] .") ";

mysql_query($sql);

echo $sql;

?>
```

The data being logged is meant to be illustrative of how to log traditional access data to a database rather than exhaustive about what to log. Additional Webstats could and should be included in the SQL insertion code. For a dynamic Web site, you should consider populating the access log record with other data that your content-management system might provide (section of the site the page is embedded in, a page label, page codes that might describe the page, and so forth).

Making the access log code work

For this access logging code to work (from [Web site footer](#)), you will need to install an `access_log` table in your database in addition to a `visitors` table.

You can create the MySQL `access_log` table using the following SQL statement:

```
CREATE TABLE `access_log` (  
  `log_vis_id` varchar(32) NOT NULL default '',  
  `log_ip` varchar(32) NOT NULL default '',  
  `log_sess_id` varchar(32) NOT NULL default '',  
  `log_state` int(8) NOT NULL default '0',  
  `log_bytes_sent` int(8) NOT NULL default '0',  
  `log_exec_time` float(6,5) NOT NULL default '0.00000',  
  `log_serv_time` bigint(12) NOT NULL default '0',  
  `log_cum_clicks` int(4) NOT NULL default '0'  
) TYPE=MyISAM;
```

Web site navbar

For completeness, here is what the `navbar.php` file looks like.

```
<p align='center'>  
<a href='page.php?state=0'>Home</a> |  
<a href='page.php?state=1'>News</a> |  
<a href='page.php?state=2'>Products</a> |  
<a href='page.php?state=3'>Services</a> |  
<a href='page.php?state=4'>About</a> |  
<a href='page.php?state=5'>Model Data</a>  
</p>
```

As you can see, each page is identified with a number instead of a name. To understand the transition matrix that will be presented in the next section, you just need to keep in mind the one-to-one mapping between these numbers and name identifiers.

The Model Data link

The "Model Data" link has been provided as a way to see the state of your transition matrix object. The following image is what I see after clicking the "Model Data" link. I had clicked around the demo Web site for awhile before clicking the "Model Data" link. Note that this information appears in three consecutive images.

[Home](#) | [News](#) | [Products](#) | [Services](#) | [About](#) | [Model Data](#)

Model Data

```
vis_id: bad5559b9281b8a432857ab135e2cf8e  
sess_id: 7b45930c8c0a3331e143b105042bc3ad  
state: 5
```

```
transitionmatrix Object  
(  
  [freq_matrix] => Array  
  (  
    [0] => Array  
    (  
      [0] => 33  
      [1] => 8  
      [2] => 0  
      [3] => 0  
      [4] => 0  
      [5] => 2  
    )  
    [1] => Array  
    (  
      [0] => 5  
      [1] => 1  
      [2] => 5  
      [3] => 1  
      [4] => 0  
    )  
  )  
)
```

```
[2] => Array
(
    [0] => 0.14285714285714
    [1] => 0.42857142857143
    [2] => 0
    [3] => 0.28571428571429
    [4] => 0
    [5] => 0.14285714285714
)

[3] => Array
(
    [0] => 0
    [1] => 0
    [2] => 0.5
    [3] => 0
    [4] => 0.5
    [5] => 0
)

[4] => Array
(
    [0] => 0
    [1] => 0
    [2] => 0
    [3] => 0.33333333333333
    [4] => 0
    [5] => 0.66666666666667
)
```

```

        [5] => Array
            (
                [0] => 0.75
                [1] => 0
                [2] => 0
                [3] => 0
                [4] => 0.25
                [5] => 0
            )
    )

[num_transitions] => 73
[num_rows] => 6
[num_cols] => 6
[state] => 5
)

```

```

    Why are you doing this to me?
    Because knowledge is torture, and there must be awareness
before there is change.
    -- Jim Starlin, "Captain Marvel", #29

```

```

INSERT INTO access_log (log_vis_id, log_ip, log_sess_id, log_state, log_bytes_sent,
log_exec_time, log_serv_time, log_cum_clicks) VALUES
('bad5559b9281b8a432857ab135e2cf8e', '156.34.57.18',
'7b45930c8c0a3331e143b105042bc3ad', '5', 3785, 0.022086977958679,
1068760273,74)

```

The transition matrix object that you see is the result of telling PHP to dump the state of the transition matrix object. This is achieved by issuing the PHP `print_r($tm)` command which outputs a Web developer friendly version of the current state of your transition matrix. Ultimately you will get a better sense of how this code works by downloading it (see [Resources](#)) and seeing how the transition matrix is being dynamically updated as you click through the demo Web site.

Descriptive and inferential statistics

I hope this section on transition matrices gives you a better idea of what they are, how they get updated, and how they might be integrated into your overall Web site

data-collection system.

Before I move onto Markov process theory, I will take stock and note that recording user-transition data might be a good thing to do even if you don't believe users can be modeled as Markov processes. Imagine how hard it would otherwise be to compute page-transition probabilities for each user. With good transition matrix visualization tools you might achieve a much better understanding of how your Web sites are being used.

In the next section of this tutorial, [Modeling users as Markov processes](#), I will talk about Markov process theory -- an area of probability theory that uses transition matrices to reason about dynamic probabilistic systems (see Ron Howard's books in [Resources](#)). It wouldn't be too far off to suggest that the study of transition matrices is the descriptive statistics part of Markov process theory. The study of how to make inferences from this data is the inferential part of Markov process theory.

Section 5. Modeling users as Markov processes

Scope

Markov process theory comprises a large body of literature in the form of research texts, undergraduate texts, journals, and articles. A substantial research community is dedicated to furthering the reach of Markov process theory.

Fortunately, you can tackle learning this in a logical and incremental way. The approach I have taken is to develop a class called `Markov.php` that includes methods implementing basic Markov probability computations. These methods operate upon probability transition matrices and show you how the Markov probability calculus is applied to transition matrices. As you learn more about Markov process theory, you can add additional methods to the `Markov.php` class, or refactor it, to provide answers to other common questions that arise in Markov process theory.

Most of the literature on Markov process theory avoids the issue of how the probabilities in their probability transition matrices were arrived at. They often simply state that the transition probabilities are this or that value and proceed to show you how to derive calculations based upon those arbitrarily selected probability values. In the real world, however, you are fundamentally concerned with how these probability estimates come about, and in particular, the amount and quality of the data used to estimate the true transition probabilities.

It should be acknowledged now that predicting what a user might do next can only take place in the context of having sufficient transition data from the visitor's clickstream to provide accurate estimates of the transition probabilities. But how much data is enough? This is an important issue, but one which I unfortunately cannot include in the scope of this tutorial. For now it is enough to note that the `Markov.php` class assumes that the probability values you are working with are not problematic. They just crank out probabilities based on the probabilities fed into it.

The Markov property

In the grand scheme of Markov process theory, the probability transition matrix I have implemented would be called a one-step transition probability matrix. For each Web site user, the probability transition matrix specifies the one-step probabilities of going from page state s_i (at time t) to page state s_j (at time $t+1$).

Using a first-order probability transition matrix to model a Web site user implies the belief that longer causal chains are not playing a role in determining the next page a user will select. The only causal influence on the user's choice of what page to go to next is presumed to arise from factors related to the current page they are on. In Markov process theory, this idea is expressed mathematically as follows:

$$P(X_{t+1} = s_{t+1} | X_t = s_t, X_{t-1} = s_{t-1}, \dots, X_0 = s_0) = P(X_{t+1} = s_{t+1} | X_t = s_t)$$

The equation equates two probabilities. It says the probability computed by the first expression is the same as the probability computed by the second expression. The vertical bar tells you that you need to read the expression in parenthesis -- $P(\dots | \dots)$ -- as a conditional probability, the probability that a user will visit a particular page next given that the same user has visited one or more pages in the past.

This equation tells you that the probability of a future page state is the same whether you take into account the complete history of page visits or just take into account the current page the visitor is looking at. Because these probabilities are equivalent, it means you can forget about all prior history of the process when trying to predict its future state. All you need to examine is the current state of process in order to predict its future state (the probabilities associated with all possible next-page states).

Systems that satisfy this formula are said to exhibit the Markov property and when they do, Markov process theory is an appropriate tool to use to analyze system dynamics. Even if undetected historical influences determine the user's next page choice (beyond just the current page), Markov process theory is not rendered useless. It is still likely that variables having the strongest effect on a surfer's choice of next Web page are those issuing from the Web surfer's immediate milieu. In such cases, the Markov Property is technically violated, but may still be useful to assume because it has some predictive utility.

In reality, it may be difficult to even test to see if the Markov property holds. You might just assume it to be true and see if your Markov calculations seem to be coming up with useful predictions.

Ronald Howard, who has written extensively about Markov process theory, had this to say about when to use it:

No experiment can ever show the ultimate validity of the Markovian assumption; hence, no physical system can ever be classified absolutely as either Markovian or non-Markovian -- the important question is whether the Markov model is useful. If the Markovian assumption can be justified, then the investigator can enjoy analytical and computational convenience not often found in complex models.

Simulating a Markov process

A good way to get a better feel for a Markov process is to simulate a Markov process. It is actually quite easy to develop code to simulate a Markov process. All you need to do is define an initial state for your process (such the "home page" state). You then use this initial state to look up the probabilities associated with the next state.

	<i>home(t+1)</i>	<i>news(t+1)</i>	<i>products(t+1)</i>	<i>services(t+1)</i>	<i>about(t+1)</i>
<i>home(t-1)</i>	0	.444	.296	.148	.111
<i>news(t-1)</i>	.222	.361	.222	.055	.138
<i>products(t-1)</i>	.138	.222	.222	.333	.083
<i>services(t-1)</i>	.214	.036	.393	.214	.143
<i>about(t-1)</i>	.421	.368	.053	.158	0

If the user's initial state is the "home page" state, then you would look at the probabilities in the first row of the table to determine the probability associated with each possible next outcome. It is important to realize that the first row of this table is a conditional probability distribution (which is being estimated by the frequency transition data collected on that user to date). It is the probability distribution of next page states *given that* the current page state is the "home page" state. The successive rows in the table tell you that the probability of observing a next state varies depending upon the page state the user is currently in.

To simulate the behavior of a Markov process, you need to add an element of randomness to the determination of the next state of the process. This randomness can be achieved by selecting a random number between 0 and 1 and then using this value and the conditional probabilities to jointly determine what the next state should be. The following PHP code shows you how to select the next state based upon both the random value that is selected and the conditional probabilities associated with the current state.

```
<?php
// set current state to 0 which is the "home" state
$state = 0;

// the number of states is 5
$num_states = 5;

// generate a random number between 0 and 1
$rnd_val = rand() / getrandmax();

for ($j = 0; $j < $num_states; $j++) {
    $rnd_val = $rnd_val - $this->prob_matrix[$state][$j];
    if ($rnd_val < 0) {
        $state = $j;
        echo "Next page state in markov process is $state <br />";
    }
}
?>
```

The source for the above algorithm is Laurie Snell's book *An Introduction to Probability*. It is a simple and elegant way to select the next state in a Markov process simulation. The weather example to be discussed was also covered in this text and was used to confirm the Markov methods are computing the correct values.

Web surfers are like the weather, I

To see how simulating a Markov process might be useful, you can examine how you might use a Markov process simulation to estimate long term patterns in a weather system. Imagine a weather system that can be characterized as consisting of three states:

- Rain, denoted by the letter "R."
- Nice, denoted by the letter "N."
- Snow, denoted by the letter "S."

Furthermore, imagine that you have been observing the weather system over a long period of time and are able to estimate the one-step transition probabilities quite accurately. The one-step probability transition matrix for the weather system might look like this:

	$R(t+1)$	$N(t+1)$	$S(t+1)$
--	----------	----------	----------

$R(t)$.5	.25	.25
$N(t)$.5	0	.5
$S(t)$.25	.25	.5

To use the `Markov.php` class to simulate the unfolding of the three-state weather system, you first initialize the Markov class with the weather system transition probability matrix. You then invoke the `simulate` method with arguments consisting of the starting state for the simulation, the number of desired iterations, and the names of the states (so that states can be output by name rather than a numerical value).

Web surfers are like the weather, II

Following is the code to simulate the weather system.

```
<?php
// simulate.php

$prob_matrix[0][0] = .5;
$prob_matrix[0][1] = .25;
$prob_matrix[0][2] = .25;

$prob_matrix[1][0] = .5;
$prob_matrix[1][1] = .0;
$prob_matrix[1][2] = .5;

$prob_matrix[2][0] = .25;
$prob_matrix[2][1] = .25;
$prob_matrix[2][2] = .50;

require_once "Markov.php";

$markov = new Markov($prob_matrix);

$start = 0;
$steps = 500;
$names = array("R", "N", "S");

$markov->simulate($start, $steps, $names);

?>
```

When this script is executed it generates an output that looks like this:

States

```

SSSRRRSNRSRSNRRRRRRRNSNRSRRRRRNRRRRRRSS
RSSNRNRNRSSSSRRNRNRNSSRSSNRRSNRSNSSSRSSNR
RRRRRRSNNSNSSSSNRRRRNRNRSSRRSNRNSNSSSRRRSNS
NRRRRNRNRNRNRNRNRNSSNRNRSSSRSSSRRRRNRRNR
RSNSRRSSSSNSNSRSRSSSSSNSSSRSSNSRNSNSSSSNSN
SRNSRNRRRSRSNRRNRNRNRNRNRNSSRNRRNRSSSRNSN
SSNSRRRSSSRNRNSSRNRRRRRRNRNRNRSSSRNRRRRR
SNRNRSSSRSSSRRRRRSNRNRSSSRSRNRNRNSSRSSNS
NSSSSSRNRSSSRNRNRSSSRNRSSSNRRRRSSSNSSNR
SSRNSNRSSSNRRSSNSNSSSSSNRRRRRRSRRRRSSNS
SSSSNSSSRSSRRNRNSSSSSRNRNRNRSSSSNSNSRNR
SRRRRRRRSRSRRNRNSSSSSSNRRNRSSSSSSSRNRRRR
RRRRSNRNRSNSSSSSNSSSNR

```

Summary

State	Times	Fraction
R	210	0.42
N	93	0.186
S	197	0.394

Web surfers are like the weather, III

As you can see, the script in [Web surfers are like the weather, II](#) generates a sequence of states that is jointly determined by the probability transition matrix and the random number generated for that trial.

The script output also shows you that running this simulation for 500 trials can be used to estimate the proportion of time the weather system will occupy each weather state. A similar simulation could be run on a Web site user transition matrix to estimate the long-term proportion of time a user will occupy each page state.

Before discussing the simulation code, I should note that the probability transition matrix for a weather system is not likely to stay the same across seasons of the year. Indeed, it is likely to change quite a bit. In the language of Markov process theory, I would say that the probability transition matrix is not *stationary* for a weather system.

In the examples that follow, I assume that the probability transition matrix for a particular Web site user is stationary. This is a useful simplifying assumption to make and possibly true. The possibility that the probability transition matrix might change over time, however, should be kept in mind because it does allow for some additional options in how you model a Web site user.

Initialization and simulation methods

The Markov process simulation called two methods provided by the `Markov.php` class:

1. A `Markov()` constructor method to load a user-supplied probability transition matrix
2. A `simulate()` method to specify the starting state, number of iterations, and name of the states

Following is the PHP code implementing these methods:

```
<?php
// Markov.php
// Copyright 2003, Paul Meagher
// Released under PHP License v3.00

class Markov {
    // Matrix container
    var $prob_matrix = array();

    // Record dimensions of prob matrix
    var $num_rows = 0;
    var $num_cols = 0;

    /*
    * Constructor method sets the instance variables based
    * on a user supplied probability transition matrix.
    */
    function Markov($prob_matrix) {

        if ( ! is_null($prob_matrix) ) {
            $this->num_rows = count($prob_matrix);
            $this->num_cols = count($prob_matrix[0]);
        } else {
            die ("Must supply a probability matrix.");
        }

        for ($row=0; $row < $this->num_rows; $row++ ) {
            for ($col=0; $col < $this->num_cols; $col++ ) {
                $this->prob_matrix[$row][$col] = $prob_matrix[$row][$col];
            }
        }
    }
}
```

```

/*
 * Generate states of a markov process. Must specify the starting
 * state, the number of iterations and the names of the states.
 */
function simulate($start, $steps, $names) {

    $num_states = count($this->prob_matrix[0]);

    $state = $start;

    // 1. Code to simulate a markov process. Also outputs and
    // records the unfolding states of the process.
    echo "<h2>States</h2> ";
    for ($i = 0; $i < $steps; $i++) {
        $rnd_val = rand() / getrandmax();
        for ($j = 0; $j < $num_states; $j++) {
            $rnd_val = $rnd_val - $this->prob_matrix[$state][$j];
            if ($rnd_val < 0) {
                $state = $j;
                echo $names[$state]. " ";
                $j = $num_states; // end the loop
            }
        }
        $freq_dist[$state] = $freq_dist[$state] + 1;
    }
    echo "<br>";

    // 2. Output a summary table
    echo "<h2>Summary</h2>";
    echo "<table border='1'>";
    echo "<tr><th>State</th><th>Times</th><th>Fraction</th></tr>";
    for ($state = 0; $state < $num_states; $state++) {
        $fraction = $freq_dist[$state] / $steps;
        echo "<tr>";
        echo "<td>". $names[$state] . "</td>";
        echo "<td>". (int) $freq_dist[$state] . "</td>";
        echo "<td>". $fraction . "</td>";
        echo "</tr>";
    }
    echo "</table>";

    return true;
} // end simulate
} // end Markov
?>

```

Feeding the probability matrix into simulation code

Note that you can feed a user's probability matrix into this Markov simulation code as follows:

```

<?php
// initialize the Markov simulation
$start = 0;
$steps = 500;
$names = array("Home", "News", "Products", "Services", "About", "Model Data");

// load and instantiate the transition matrix

```

```
require_once "./TransitionMatrix.php";
$tmp = new TransitionMatrix;

// use a visitor id from visitors table
$visitor_id = "558f70bb1f04a03f15260c08678e5832";

// get probability transition matrix for visitor
$tmp->restore($visitor_id);
$prob_matrix = $tmp->prob_matrix;

// run the Markov simulation and see what you get
require_once "./Markov.php";
$markov = new Markov($prob_matrix);
$markov->simulate($start, $steps, $names);

?>
```

K-step transition probability

Another common calculation that is carried out in order to understand the dynamics of a Markov process is matrix multiplication. One type of multiplication that is commonly performed is to multiply the probability transition matrix by itself. This operation is also called raising the Matrix to a power because it is similar to the ordinary operation of raising a number to a power ($6^3 = 6 \times 6 \times 6$).

Raising the matrix to a power has the effect of calculating the k-step transition probabilities for all cells of the matrix. A k-step transition probability specifies the probability of going from state i to state j in k steps.

$$\Phi(k, i, j) = P(S(t + k) = j \mid S(t) = i)$$

To understand why raising a matrix to the power k has the effect of computing the k -step transition probability, it will help to recall the three-state weather system. As you go from one day to the next, the possible states you can enter into are defined by the relevant row of the transition matrix. You can visualize the unfolding state of the weather system as a branching process in which you start off in some state, then you can branch into the other states reachable from that state, and then from each of these states you branch again to represent the states reachable from these states, and so on. After k branches, you sum the probabilities of all the paths leading to state you are interested in.

The process of multiplying a matrix of numbers by itself k times is equivalent to performing this branching process k times and summing the probabilities of all paths leading to the state you are interested in.

The Markov class offers methods that perform matrix multiplication.

Matrix multiplication algorithm

You can learn matrix multiplication by studying this algorithm for performing it. You can also do a Google search on matrix multiplication to find other resources. The Markov class uses the `matrix_multiplication()` method as the basis for implementing a `matrix_power()` method. There are other approaches to deriving the power of a matrix. This approach has the advantage of being simple to understand.

```
<?php
class Markov {
    /*
    * Return array consisting of num rows and num cols
    */
    function matrix_dimensions($A) {
        return array( count($A), count($A[0]) );
    }

    /*
    * Returns product of two matrices
    *
    * Matrix multiplication approach ported from the first edition
    * of Christianson & Torkington's Perl Cookbook, p. 60-61
    */
    function matrix_multiply($m1, $m2) {

        // get the dimensions of the supplied matrices
        list($m1_rows, $m1_cols) = $this->matrix_dimensions($m1);
        list($m2_rows, $m2_cols) = $this->matrix_dimensions($m2);

        // num of columns in matrix 1 must equal the number of
        // rows in matrix 2
        if ($m1_cols != $m2_rows) {
            return PEAR::raiseError("IndexError: matrix dimensions don't match:
                $m1_cols != $m2_rows");
        }

        // perform the matrix multiplication
        $result = array();
        for ($row_i=0; $row_i < $m1_rows; $row_i++) {
            for ($col_j=0; $col_j < $m2_cols; $col_j++) {
                for ($k=0; $k < $m1_cols; $k++) {
                    $result[$row_i][$col_j] += $m1[$row_i][$k] * $m2[$k][$col_j];
                }
            }
        }

        // return the matrix multiplication result
        return $result;
    }

    /*
    * Get the value of a matrix raised to a power
    */
    function matrix_power($power) {
        $Q = $this->prob_matrix;
        for($i = 1; $i <= $power; $i++) {
            $Q = $this->matrix_multiply($this->prob_matrix, $Q);
        }
        return $Q;
    }
}
```

```
?>
```

To see how these class methods might be utilized you will create and discuss a script that demonstrates usage.

Into the future

The following script called `snell.php` demonstrates usage. It uses the weather example again. The first part of the script computes successive powers of the weather transition matrix and prints the k-step probability matrix after each power (so that, k-step) is computed. The second part of the script shows you that you might have obtained the same final result using the more convenient `matrix_power()` method instead.

```
<?php
// snell.php

// Construct the probability transition matrix for
// the weather system

$prob_matrix[0][0] = .5;
$prob_matrix[0][1] = .25;
$prob_matrix[0][2] = .25;

$prob_matrix[1][0] = .5;
$prob_matrix[1][1] = .0;
$prob_matrix[1][2] = .5;

$prob_matrix[2][0] = .25;
$prob_matrix[2][1] = .25;
$prob_matrix[2][2] = .50;

// Set the power we want to raise the matrix to
$power = 6;

require_once "Markov.php";
$markov = new Markov($prob_matrix);

// Print out the first power of the matrix
$Q = $markov->prob_matrix;
echo "<b>P^1</b> :";
echo "<br>";
echo "<pre>";
print_r($Q);
echo "</pre>";

// Print out the remaining powers of the matrix
for($i=2; $i <= $power; $i++) {
    $Q = $markov->matrix_multiply($markov->prob_matrix, $Q);
    echo "<b>P^$i</b> :";
    echo "<br>";
    echo "<pre>";
    print_r($Q);
    echo "</pre>";
}
```

```
// Show that we could compute the final matrix power
// more easily by using the matrix_power method
echo "<hr><br>";
$Q = $markov->matrix_power($power);
echo "<b>P^$power</b> :";
echo "<br>";
echo "<pre>";
print_r($Q);
echo "</pre>";

?>
```

The script generates a large amount of output.

Script output to the sixth power

I will list the last part of the generated output, the sixth power of the matrix computed using the `matrix_power()` method and output using PHP's `print_r` function:

P⁶ :

```
Array
(
    [0] => Array
        (
            [0] => 0.4000244140625
            [1] => 0.20001220703125
            [2] => 0.39996337890625
        )
    [1] => Array
        (
            [0] => 0.4000244140625
            [1] => 0.199951171875
            [2] => 0.4000244140625
        )
    [2] => Array
        (
            [0] => 0.39996337890625
            [1] => 0.20001220703125
            [2] => 0.4000244140625
        )
)
```

Ronald Howard provides a very clear and authoritative interpretation of what the values in this k -step probability transition matrix mean:

Thus the proper way to interpret the multistep transition probabilities is that these are the probabilities we must assign to the process's occupying each state after n transitions. If we observe its initial state, if we do not observe it in the interim, and if the operation of the process is to be consistent with its transition probability matrix.

Notice that when you raise the probability transition matrix for the weather system to

the sixth power all the rows in the resulting matrix are identical ($p(0) = .4$, $p(1) = .2$, $p(2) = .4$). This is how Markov processes tend to behave in the long run: eventually the initial state of the process no longer exerts an influence on what the next state probabilities will be, as indicated by the identical rows that start to appear as the probability transition matrix is raised to higher powers. Once the row probabilities start to become the same they will not change as you raise the matrix to higher powers. When this occurs, the probability transition matrix is said to have entered an *absorbing state*.

Note also that these row probabilities are similar to the fraction of time that the simulated weather system spent in the different weather states. The `simulate()` method can be used to estimate the absorbing state values that will appear when the probability transition matrix for the weather system is raised to a high power using the `matrix_power()` method.

This ends your sampling of basic ideas and calculations to be found in Markov process theory. The probability calculus of Markov process theory can be extended much further than this and related to many other areas of probability and statistics.

The main purpose of this discussion of Markov process theory was to show that regarding a user as a Markov process allows you to use probability transition matrices to make other inferences about the possible future page states of a Web site user.

Section 6. Wrap up

Data issues

When Bucklin and Sismeiro prepared their Apache access logs for analysis, they removed all visits involving a single page request. They defined a site visit as including two or more page views. Their rationale was that they were interested in understanding browsing behavior and that such data was not informative on that score. They also pointed out that "If we were to retain visits with single-page requests, they would dominate the entire sample."

The fact that many visitors only provide one data point of information should be kept in mind when thinking about the results one might observe under real-world conditions. Only a small subset of the visitors are likely to provide enough user transition data that you can begin to use it to try to predict the next pages they will view.

Another data issue that Bucklin and Sismeiro needed to address was an operational definition of a site visit. In particular, they asked how you should count sessions when a person is idle on your site for 50 minutes, then comes back and starts surfing again. They decided that this was to count as a new session because a 30-minute period of inactivity had transpired. Other researchers also used this 30-minute inactivity criterion to define when a new session should be declared.

My access logging code does not currently implement the idea of starting a new session if a visitor's last access was 30 minutes ago. The solution would involve recording the time stamp of the last access in the transition matrix. When you retrieve the visitor's transition matrix at the start of each page, you could check the time stamp value for the last access and if it was greater than 30 minutes, you could create a new session ID for the visitor.

Bucklin and Sismeiro also needed to deal with the issue of what method they would use to track users across sessions: Cookies, IPs, or member IDs? They decided to remove records from their analysis that did not have a cookie identifier. They retained about 90 percent of their records after doing this. Basing user tracking on cookie identifiers is probably one of the most reliable ways to identify users across sessions; however, it is not without limitations as well (such as, users can delete their cookies). As indicated previously, I believe that the most reliable method to track users across sessions is probably a redundant approach that uses both cookie and IP data, and member ID data as a third backup if you have it.

The individual and the collective

It would be possible to use transition data to update both an individual and a collective transition matrix.

The collective transition matrix would aggregate the transition frequencies for all users in the site. Each user would contribute new data to the collective transition matrix every time they clicked from one page to another. It might also be possible to derive the group transition matrix as the sum of individual frequency transition matrices.

What would be the benefit of doing this? From a predictive point of view, having a collective transition matrix might provide additional information that your prediction algorithms can use to forecast what page a user is likely to visit next. The observed clickstream of a user can be regarded as a function of both Web site variables and user variables. The collective transition matrix might be a useful way to represent the influence of Web site variables.

The collective transition matrix represents the averaging out of all idiosyncratic tendencies of users. Because the collective transition matrix doesn't represent how any individual in particular uses the site, it might be regarded as estimating the

general flowpath that the site tends to produce.

Predicting the next page a visitor will view might be improved if you combine information about the idiosyncratic tendencies of the user (measured by their individual transition matrix) and the general flowpath the site tends to produce (measured by the collective transition matrix). The relative weight of these factors as well as their method of combination are matters for future research.

Intelligent flowpath management

My feeling is that Markov theory might be used, in various ways, for intelligent flowpath management purposes. In chapter 4 of John Lenker's book, *Train of Thought*, he defines Intelligent Flowpath Management Systems as follows:

Because Web enterprises seem to struggle so much to provide information in a "structure" that's effective for people, my assertion is that "architecture", both as metaphor and as a process, has outlived its usefulness. The time has come to enlist a more powerful vision that reaches farther. Instead of information architecture, I believe we need to begin developing Intelligent Flowpath Management Systems that procedurally analyze the needs of individuals and then release appropriate sequences of notions. These sequences must be programmed to guide people's minds naturally and in a manner that helps them to build anticipation of that which will succeed in fulfilling their expectations.

Web site user transition matrices offer a way to procedurally analyze the needs of individual Web site users. The transition matrix of an individual person might be interpreted as a measure of the expected value they assign to each next page state. If you have enough visitor interaction data, you can, using a cookie, recover his user model and deliver "creative" for the home page that utilizes his transition matrix to anticipate modular content that will fulfill the user's next-page expectations. The home page would be dynamically generated so it conforms to the user's value expectancies about where to go next. In this way, you might increase the *stickiness* of your site which has been shown to increase the likelihood of buying products or services.

If you are an advertising-driven site, then stickiness is also critically important. Eytan Adar and Bernardo A. Huberman expound further upon the economic importance of stickiness and the need for versioned information services in "The Economics of Surfing."

From the marketing side, James Hering discusses a similar notion in his recent discussion of behavioral targeting:

...most people think behavioral targeting involves tracking user behavior, then dynamically leveraging the knowledge to serve relevant messaging. I think of it from

the customer perspective; how can I reward a potential or current customer with more relevant information, as indicated by their implied or expressed behavior? ... We use expressed behavior to trigger the ad server to queue your creative on the next page. Think of it as "sequentially relevant ad serving".

One correction is that you can in fact use expressed behavior to trigger the ad server for "this page" (instead of just the "next page") because predictive information about where a user will go next is available (in the `header.php` script) before a page is ever generated.

Markov process theory provides a framework for understanding how intelligent flowpath management and sequentially relevant ad serving might be implemented in practical terms. I see individual and collective transition matrices as providing the individual and collaborative filters needed to implement an intelligent flowpath management system.

Other applications

Markov process theory has many other areas of application beside Web site user modeling. I would be remiss not to mention a small sampling of some of these application areas:

- **Bioinformatics.** In modern terms, Gregor Mendel's lasting legacy is the idea that you can use a transition matrix to model the transmission of features from parents to offspring. Indeed, Markov process theory and bioinformatics both have as their central concern the understanding of dependent sequences. Markov process theory offers a framework for organizing and thinking about patterns in sequential data which is the hallmark of much biological data. Those interested in the emerging area of bioinformatics would do well to study Markov process theory further.
- **Queuing problems.** A queue is a waiting line. Queues are ubiquitous not only in business contexts, but also in work-scheduling or transportation problems, for example. These problems share the feature that what happens to the queue at one moment in time determines the state of the queue at the next moment of time. Queueing theory relies extensively upon simulation to model the many factors that might affect queue dynamics. Markov process theory has been extensively applied to the simulation of queue dynamics. Indeed, simulation books in general often devote considerable space to Markov process models because they are a primary tool that simulation-based analysts use to gain insight into the temporal structure of the systems they study.
- **Human behavior.** Markov process theory has been used to understand the dynamics of turn taking in meetings, the behavior of rats in mazes, the

allocation of attention in the processing of faces, and other behaviors. You could, for example, replace the names of links in my demo Web site with the names of people participating in a meeting. As each person talks, you could record that fact by clicking on a link corresponding to the name of the person. If a person speaks, do they continue speaking? If they fall silent, do they remain silent? The answer is often yes, indicating conversational dynamics exhibit the Markov property. The distribution of sounds in a word, words in speech, or words in text can also be approached through Markov process theory. The possible applications of Markov process theory to human behavior are only limited by one's imagination.

Markov process theory was originated by Andrei Andreivich Markov (1856-1922) in a 1907 paper called "Extension of the Limit Theorems of Probability Theory to a Sum of Variables Connected in a Chain". Semi-Markov Processes, Hidden Markov Chains, and Monte Carlo Markov Chains (MCMC) are comparatively recent additions to Markov process theory that are being used to extend the power and reach of Markov theory to new domains. These theoretical additions, and the increase in computing power for Markov-based simulations, are accounting for a renaissance of Markov process theory in many domains of inquiry.

Ethical deployment

I haven't talked much about ethical issues that might arise as a result of increased tracking and monitoring of individual Web site users. Some users might be offended at the idea that their individual actions are being monitored in such minute detail. Transition matrices, however, don't provide data that can't already be reconstructed from access log data. Transition matrices don't create new data, they just provide a way of structuring existing data that makes it more readily available for real-time use in determining what content to display.

What most users probably object to more is the possibility that their actions might be linked to their identity. In other words, as long as user tracking is carried out in such a way that the identity of the user is kept anonymous, users are less likely to raise ethical objections with increasing the level of user surveillance if some user benefit might be realized.

I see few real ethical objections to anonymous Web site user modeling. I recognize, however, the real urge to associate a visitor cookie or IP with an actual identity so that you can leverage the additional information for usability, marketing, research, or commercial purposes.

If you decide you are going to incorporate identity information into your Web site user models, then in my opinion, you should consider obtaining the permission of your Web site users before doing so. This is easier said than done, as Web site user

modeling often works best if it kicks in right away. At the very least, you should provide a document that indicates such tracking is occurring so that the user can elect not to visit the site in the future if this is a concern.

Some maintain that such user tracking is no different than a video camera mounted in a store and requires about as much permission, namely none. Even with this analogy, you can distinguish between the ethics of mounting hidden video cameras versus making them clearly visible to the customer so they know they are being monitored.

The next generation

In this tutorial, I defined what a Web site user model is and used Bucklin and Sismeiro's research to provide you with a concrete example of what a Web site user model might look like. Bucklin and Sismeiro identified two critical components of a Web site user model:

- A Page-Request Model
- A Page-View Duration Model

To construct a complete Web site user model, I suggested that a third component was required:

- A Page-Choice Model

This tutorial was largely dedicated to elaborating upon how one might specify and implement the Page-Choice component of a Web site user model. Towards this end, I discussed and implemented transition matrices and parts of Markov process theory. The Markov process approach appears promising but I have no results to report as yet. I only recently developed the code as proof-of-concept that this real-time approach to Page-Choice modeling might work.

Others have looked into applying Markov process theory to clickstream data; however, it is generally in the context of mining Web logs to estimate Web site user models rather than implementing a real-time system for retrieving and updating Web site user models. This difference is critical in terms of what you can ultimately do with your Web site user-models. It was suggested that such real-time user models might be used to construct Intelligent Flowpath Management Systems.

Another difference between the current research and other research on Web site user modeling is that this research clearly addresses user behavior at the individual level. Many Web site user models are not really user models in this sense. Instead, they are more accurately described as consensus user models that presume that general parameter estimates apply to the individual. Web site user models based

upon collaborative filtering, for example, presume that a user will want to see what other users in the same situation elected to see. This is quite different than tracking individual visitor clickstreams across sessions and using their individual transition matrices to predict what content they might want to see next. In the end, I recommend using both consensus models and individual Web site user models to arrive at useful predictions about what content a user might want to see next.

I hope this tutorial has at least convinced you that clickstream data can be used for purposes other than determining the volume and distribution of visitor traffic on your Web site. It can also be used to estimate Web site user models. In the next generation of Web sites, this latter role will become increasingly important.

Acknowledgements

I would like to thank Raymond Klein for giving me the opportunity to present an earlier version of this research to faculty and students at Dalhousie University.

Resources

Learn

- Find Bucklin and Sismeiro's Web site user modeling research in this article from the *Journal of Marketing Research*, "[A Model of Web Site Browsing Behavior Estimated on Clickstream Data](#)".
- Read J. Laurie Snell's *Introduction to Probability* (1988), still an excellent textbook for learning about probability theory and Markov process theory in particular.
- In my explorations of probability theory I continually refer to James Higgins's and Sallie Keller-McNulty's *Concepts in Probability and Stochastic Modeling* (Duxbury, 1995).
- Explore John Lenker's [Train of Thoughts](http://www.trainofthoughts.com) (<http://www.trainofthoughts.com>). This provocative book, especially "Chapter 4: Patterns of Anticipation -- The art of Flowstream Development," postulates replacing the relatively static concept of Information Architecture with the more dynamic concept of Flowstream Development. (You need a Flash player version 6 for this resource.)
- Visit the [IBM Research, Math Sciences division](#) for information on applied mathematics, algorithms, data mining, and statistics.
- In "[The Economics of Surfing](#)" by Eytan Adar and Bernardo A. Huberman, read a discussion of "mechanisms for implementing temporal discrimination in surfing by dynamically configuring sites and versioning information services".
- Read the author's more detailed discussion of the Fire Damage Model in the *developerWorks* article, "[Simple linear regression with PHP: Part 2](#)" (April 2003).
- Find a clear and detailed account of Predictive and Partial Match (PPM) algorithms in Khalid Sayood's book *Introduction to Data Compression*.
- At the [Aries Lab](#) at the University of Saskatchewan, discover a plethora of cutting-edge research on intelligent tutoring systems and adaptive learning environments.
- In developing this tutorial, the author ran into the issue of preparing mathematical equations and matrices for presentation on the Web and in PDF documents. Read the IBM *developerWorks* article "[Building dynamic Web sites with mathematical content](#)" for useful advice to developers about the tools to use and how to use them (February 2002).
- Find updates to this code at www.phpmath.com.

Get products and technologies

- Download the sample code in this compressed file, [usermodel.tar.gz](#).

About the author

Paul Meagher

Paul Meagher is a freelance Web developer, writer, and data analyst. Paul has a graduate degree in Cognitive Science and has spent the last six years developing Web applications. His current projects and interests center around e-learning, content management, and math-enabled Web applications. Paul resides in Truro, Nova Scotia.