For want of a bit the user was lost: Cheap user modeling

by J. Orwant

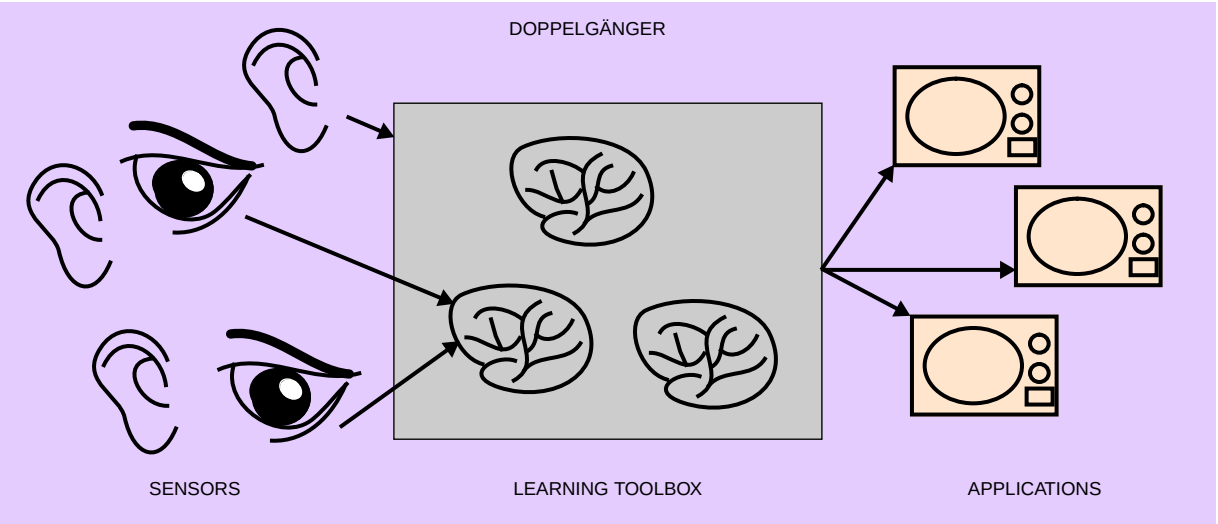
The more a computer knows about a user, the better it can serve that user. But there are different styles, and even philosophies, of how to teach our computers about us—about our habits, interests, patterns, and preferences. "Cheap" user modeling, the subject of this essay, simply means ascertaining a few bits of information about each user, processing that information quickly, and providing the results to applications, all without intruding upon the user's consciousness. In short, there are techniques for personalization that can—and should—be built into today's systems. Like most journal papers, this is a description of an existing system:* DOPPELGÄNGER. *But it is also an exhortation for readers to incorporate the described techniques and philosophy into their own systems.

User modeling is nothing more than a fancy term for automated personalization. Humans model each other all the time. I am modeling *you* as I write; my topics, presentation, and language are all aimed at a hypothetical, average reader of this journal. If I have guessed well, you will enjoy this essay. If not, you will skip to the next one. That is what user modeling systems do—they make guesses, and hopefully educated ones, about their users.

Of course, not all readers are the same, and what entices one reader may alienate another. Ideally, this essay would itself be adaptive, able to gauge the interests of individual readers and transform itself accordingly. But since that is not yet possible, I will employ some "cheap" user modeling. I will make a one-bit distinction between philosophy and engineering in this essay: after this paragraph, the left column contains the "big picture," and the right column contains details. (The separation is suspended for the section "A user modeling toolbox," then resumes until the "Reprise.") Read either column, or both, depending on your interests.

0018-8670/96/$5.00 © 1996 IBM

Figure 1 The DOPPELGÄNGER user modeling system gathers data about users from sensors, makes inferences on those data, and makes the results available to applications.



User modeling: Philosophy

You have experienced user modeling, even if you did not know it at the time. Any application that behaves differently for different users employs a user model. The models themselves can be big or small, complex or simple, rich or sparse. They often have different names: personality profiles, psychographic profiles, or consumer databases. They are all collections of information about an individual.

Any application with "novice" and "expert" modes possesses a rudimentary user model—a one-bit classification. Some applications let you specify many customizations: default printers, background images, the placement of windows, etc. These settings, taken together, constitute a slightly richer user model.

Such collections of information are, at best, embryonic precursors of an ideal user model, which would possess an intimate and thorough knowledge of its user (Table 1). In short, the user model (and any sensors, servers, and engines maintaining it) should possess all the knowledge of a lifelong friend—it should be able to recognize the user, know why the user did something, and guess what he or she wants to do next.

So at one end of the user modeling spectrum, we have a single bit dividing the entire human race into

User modeling: Engineering

A user model can be as simple as a single bit, maintained by a single application such as a word processor or a World Wide Web browser. For instance, that bit might be 1 if the user has used the application previously, and 0 otherwise.

In the DOPPELGÄNGER user modeling system,[1] each user model is a collection of files, each containing information about some domain of the user's behavior. There is one file for "vital statistics" (e.g., name, occupation, electronic or e-mail address), one for news preferences, another for locomotion information, and so on. A typical user model is approximately one megabyte in size.

Large user models such as DOPPELGÄNGER's need to be stored in a representation palatable to all applications that might potentially request information from the user modeling system—in short, as a database. Users might cringe at the notion that their personal information constitutes a database, but it is entirely accurate: there is a lot of information to be stored, and it often needs to be retrieved quickly, typically when an application is launched. Efficient storage and retrieval are essential; DOPPELGÄNGER stores its models as hash tables.

This enormous quantity of data needs to be centralized if our applications are to make informed deci-

Table 1 Components of an ideal user model

Dimension	Sample Use
What the user looks like	A security system that performs face recognition[2]
What the user sounds like	A dictaphone that can distinguish the user's voice from that of others
What the user knows	A spreadsheet that can ascertain when to teach the user a new feature
The user's interests	A VCR that records television shows without explicit instruction
The user's habits	A personalized newspaper that behaves differently on weekends

two categories: novice and expert. That single bit is cheap to store, but is not terribly useful. At the other end of the spectrum is a computational simulacrum of the user, one that knows more about the user than the user knows about him- or herself. At one end, simplistic categorization. At the other, science fiction—but it is important to realize that these two scenarios are part of the same continuum. The more bits our computers can collect about us, the better—provided they put the data to good use.

Even a few bits are better than none. Vague bits, such as the novice/expert distinction, can be used to make arbitrarily specific inferences about the user. They will be wrong much of the time, but a partially informed decision is better than an uninformed one—usually.

Breadth vs depth. The number of bits in the user model is not the only important criterion. Are the bits about one particular activity, or are they germane to all? Does a snippet of knowledge say something specific ("Bob would rather read small print on a few pages than large print on more pages") or something general ("Bob knows a great deal about telecommunications, and would not mind learning more")? A specific hypothesis about the user might help only an application or two, while a sweeping assertion about the user's behavior or knowledge might help scores of applications.

Furthermore, applications all too often operate in isolation—they do not talk to one another. But a sions about us. Each application *could* keep its own collection of information about the user, communicating with other applications only when necessary. But that would make it difficult to ascertain certain aspects of the user's behavior—some inferences can be made only when data from multiple applications are gathered. A user modeling system (strictly speaking, a "generalized user modeling shell system"[3]) reduces that overhead and makes possible broad inferences about the user's activities.

This need not be inefficient! User modeling systems such as DOPPELGÄNGER and BGP-MS (Belief, Goal, and Plan Maintenance System)[4] are structured as servers (see Figure 1) that collect, massage, and ultimately provide information about users to applications (or even users). Applications pose questions about the user ("Would the user like this article?" "What is the user's e-mail address?"); the server answers them. Most of the inferences performed by DOPPELGÄNGER can be done very quickly when "cheap" techniques, such as the ones described in this essay, are used.

The term "modeling" often implies a certain level of computational complexity. That is not always necessary—useful personalization can often be achieved by making the right data streams accessible. If your mail program can parse your electronic datebook, it can automatically generate replies on your behalf. If it *cannot* parse your datebook, you have two choices: you can explicitly type in whatever it needs to know, or use a user modeling system that can extract the required information, making it available to all your applications.

It is unreasonable to expect every calendar program to save its information in a form amenable to every mail program. And applications will not suddenly be written with user modeling in mind. Instead, user modeling should cope with the applications as they exist now. That makes acquiring user models harder, but it ensures the feasibility of user modeling in consumer applications.

But most user modeling research is predominantly theoretical, owing to its origins in knowledge representation and linguistics. Arguably, user models are merely an extension of linguistics' *discourse models*, which represent how a speaker models his or her conversational partner. Many of the theoretical results in that discipline are applicable to user modeling as well.[5]

mail program might benefit from consulting your calendar program when replying automatically on your behalf: "He's in a meeting now; he'll read your message at 4:00." User models can contain information gathered by one application to be used with another, allowing applications to communicate with one another through the user modeling system. Applications become not just consumers of information, but providers as well, helping to flesh out the user model.

Acquiring user models. For a user model to become either deep or broad, it must acquire its information about the user somehow. Where does that information come from, and what does it mean to acquire it "cheaply"?

There is a wealth of readily available information about each of us. We all continuously radiate information—every action that we take, whether we are typing, talking, or walking, reveals something about us. The problem is gathering the information—our computers might have cameras and microphones, but they cannot use them very well. Humans make better use of their eyes and ears, which is why we are so capable of subconsciously modeling our conversational partners. The rich backchannel of information (gestures, inflections, facial movements) makes user models easy, or cheap, to acquire.

- As we walk, our locomotion reveals our destinations.
- As we talk, our speech reveals our intentions.
- As we gesture, our motions reveal our thoughts.

These are all data streams: sources of information that could be used to personalize applications, if only our computers had sensors capable of gathering the data. When our computers can make as much sense of these data streams as humans can, they will be able to make the same inferences about people. Until then, our systems can focus on more accessible data streams:

- As we read, our gaze reveals our focus of attention.
- As we type, our keystrokes reveal our intentions.
- As we surf the World Wide Web, our clicks reveal our interests.

Within a decade or two, we can expect to see a dramatic increase in the connectivity of the things around us. Our computers will no longer have a

The actual inferences that can be made about users depend on the nature of the data available to the system. Many of the data streams made available by operating systems can be quite helpful:

- Our log-in patterns can help applications prepare for our arrival.
- Our computer commands can reveal the nature of our tasks.
- Our schedules can tell applications where we are, and how long we will be there.
- The articles we read indicate which topics intrigue us.

Most of DOPPELGÄNGER's sensors are of this sort. There are two exceptions: a chair sensor that can discern when a user is sitting in it, and Olivetti Active Badges**, which track users as they walk around a building.

Interactivity should never be required. The degree of interactivity preferred by users should itself be a component of the user model. DOPPELGÄNGER has a related "tolerance for intrusion" parameter. The higher that number, the more often the user is sent e-mail with questions such as "I think you're interested in football. Is that true?" and "Who would you say you most resemble: John Doe, Jane Buck, or Chris Lamb?" The user can reply if he or she wishes—DOPPELGÄNGER's e-mail is just another data stream, and its parser is just another sensor. If the user's reply is expressed in sufficiently simple English, it is incorporated into the user model.

In addition, each user receives a weekly "modeling redux," for which DOPPELGÄNGER composes a couple of paragraphs summarizing what has been learned about the user over the past week. The summary is written in predetermined (but perfect) English that makes the user modeling system seem like a conversational partner.

Unfortunately, as Brennan notes,[6] computer literacy is dangerous—when it is the computer being literate. When users interact with a program that generates complex prose, they tend to assume that the computer can understand text in kind, and reply with equally complex sentences.

This raises more general concerns about the quality of communication between the user modeling system and the user. For instance, how can the system

monopoly on computation; instead, our clothes, appliances, and furniture will all constantly relay information about us to one another. Your television set will know which shows you watch; your coffeemaker will know the settings on your alarm clock, and your chair will know when you are sitting in it.

So we have many different channels and modalities of information. Let us presume some sensors that make the data available to some centralized storage-and-retrieval facility: the user modeling system. The term "sensor" is intentionally broad: sensors might be hardware (a camera), software (a program that parses a user's command history), or even embedded into the user's environment (a file system that notifies the user modeling system when applications are launched).

An ignorant user is a happy user. User modeling should be like movie soundtracks: only mistakes are noticeable. Users should not be required to supply information to their user models; after all, not all of us want to tell our computers precisely what we want all the time. Consider a personalized newspaper application: casual users will quickly become frustrated if they have to navigate through a series of pull-down menus every time they want to read their newspaper, and revert to impersonal but familiar mass media.

Of course, some users will want to be active participants in the user modeling process. They will want to know what the system is inferring about them, and why. More power to them! User modeling systems produce the best results when users help them out. But users should never be forced to provide information to the system—they should be able to choose their own degree of interactivity.

Consider our minimalist user model, the one with the single novice/expert bit. The user can toggle the bit only if he or she is aware of it, which is by no means guaranteed with the feature-laden interfaces found in many of today's applications. It would be better for the application to perform the toggling itself. It could look at the actions taken by the user, at the user's settings for other, similar applications, or at the settings for other, similar people. Or all three.

The best source of information is often users themselves; the application could simply ask the user whether he or she is a novice or an expert. But user

Table 2 The privacy tags used in DOPPELGÄNGER

Label	Meaning
Private	Visible only to the user him- or herself
Semi-private	Visible only to particular applications or users
Semi-public	Visible to all but certain applications and users
Public	Visible to everyone and everything

describe the modeling process when the mathematical techniques involved are complex? An intuitive interface to the user modeling system is needed—one that can explain how the user is being modeled, and how the user can correct inaccuracies.[7]

In addition to the user's preferred amount of interactivity and tolerance for intrusion, user models should also provide some mechanism for allowing users to mark information as private. In DOPPELGÄNGER, every portion of the user model is categorized as shown in Table 2.

PGP (pretty good privacy)[8] cryptography software based on the RSA (Rivest-Shamir-Adleman) public-key cryptographic algorithm, is used for all network transactions containing potentially sensitive user information.

When there are user modeling systems in the home acting on behalf of the occupants, and similar systems in the office assembling demographic data about those occupants, they are competing with one another. The first wants to keep personal information private from the second. In such an environment, privacy will be inversely proportional to your information bill. Consider an application providing movies-on-demand to your computer. Your computer might request specific movies, in which case only those bits will be transmitted. Additionally, the distributor might subsidize the transmission in exchange for the rights to use any derived inferences about your movie preferences. Either way, the distributor knows what you have requested, and when you requested it, and so a little privacy is lost. Or, your computer could request the entire library of movies, keeping your subsequent selections (or its selections on your behalf) secret. The size of the

modeling systems should keep intrusion to a minimum. Forcing questionnaires on users is, in the long run, disastrous: users will take the path of least resistance, or worse, complain about the interface and switch to a less insistent application. And there is an additional downfall to relying on the user for information: even when information can be obtained, it is bound to change, or even to be wrong. People do not remain novices forever.

Unfortunately, passive sensors cannot retrieve as much information as sensors that actively query the user for information. And the information they do retrieve is less likely to be accurate.

Given these passive mechanisms for inferring information about users, mistakes will be made, especially when the system has had little exposure to the user. How, then, can user modeling systems inspire trust in their users? A user presented with a newspaper purporting to be personalized just for him or her will be skeptical, and rightfully so.

Users will be even less likely to trust a system that continuously records sensitive information about their interests, tastes, and comings and goings. All of that information must be kept private if users are to trust the user modeling system, resulting in a truly *personal* computer that knows everything about you—including your privacy boundaries—so that it can gate the flow of information between it and the outside world according to your desires.

Predicting behavior. How predictable are people? And what are some cheap methods for making predictions about them?

There is no simple answer to either question. It depends upon what is being inferred, the amount of data available, and how quickly the computation must take place. There is no single recipe for user modeling; the best one can hope for is a cookbook that lets developers choose whichever techniques best match the tasks at hand. User modeling should make use of statistics and machine learning—two domains where messy data, such as the error-prone bits gathered by user modeling systems, are expected.

Before user modeling is incorporated into an application (or operating system), designers must first realize that predictions need not be 100 percent accurate to be useful. User modeling is an inexact transmission, and therefore the expense will soar. That is the price of privacy.

The notion of a system that learns about its users, while an anathema to privacy, is necessary if our computers are to adapt to changes in our on-line behavior. Beyond the learning techniques demonstrated in this essay, two broad classes of machine learning techniques will prove relevant:

- *Distributed learning*, in which geographically or computationally remote sites gather information about the same phenomenon, but can only occasionally (or slowly) communicate with one another.
- *Cost-based learning*, in which there are quantifiable penalties for wrong (or tardy) answers, prompting the system to choose the hypothesis that maximizes the expected utility for the user, rather than the choice with the highest probability of being right.

One notion common to most machine learning paradigms is confidence: estimating the probability that an assertion is correct. In DOPPELGÄNGER, every assertion comes tagged with a confidence value, arising from either the technique's confidence in the data, or from DOPPELGÄNGER's confidence in the technique. Applications can do whatever they wish with that value. "Loose" applications might ignore it; "tight" applications might reject all assertions with a greater than 20 percent chance of being wrong.

So how much information is involved? How many bits of data are needed to describe a person? What level of mathematical complexity is needed to predict that person's actions? More importantly, given an arsenal of modeling methods, which ones should the system use, and when? Ideally, user modeling systems would themselves choose which techniques to use. Since most systems will rapidly amass huge amounts of data, they can test competing techniques for self-consistency—seeing which techniques would have predicted the already observed data. Faced with conflicting information from different techniques, how can a user modeling system choose a reasonable middle path?

In some instances, DOPPELGÄNGER lets techniques vote (or even bid) on hypotheses about the user. Suppose that three sensors each claim to know the strength of the user's preference for football in his

discipline; assertions about user preferences and cognitive state are bound to be wrong much of the time.

What is important is that systems minimize the number and severity of incorrect guesses, learn from their mistakes, and model not just users *but the techniques themselves*, so that they can bundle each guess with a numeric confidence in its accuracy.

or her newspaper: the first asserts that the user's preference is 0.6 out of 1.0 with confidence 0.35; the second says 0.3 with confidence 0.2; the third says 0.65 with confidence 0.8.

DOPPELGÄNGER first normalizes the confidences so that they sum to 1. Then the estimate of the "true" strength is straightforward: the sum of the strength-confidence products, yielding 0.585 in our example. The confidence is defined to be $1-4\sigma 2/n$, where n is the number of samples, so that very high variances yield confidences approaching 0, and very low variances yield confidences approaching 1.

This calculation assumes that the strengths range between 0 and 1, and that the distribution across all users is uniform. While that is a reasonable *a priori* assumption, additional information (such as the user's age, nationality, or occupation) might influence the distribution. In such cases, the strength is calculated in the same way, but the confidence calculation changes to reflect the additional information.

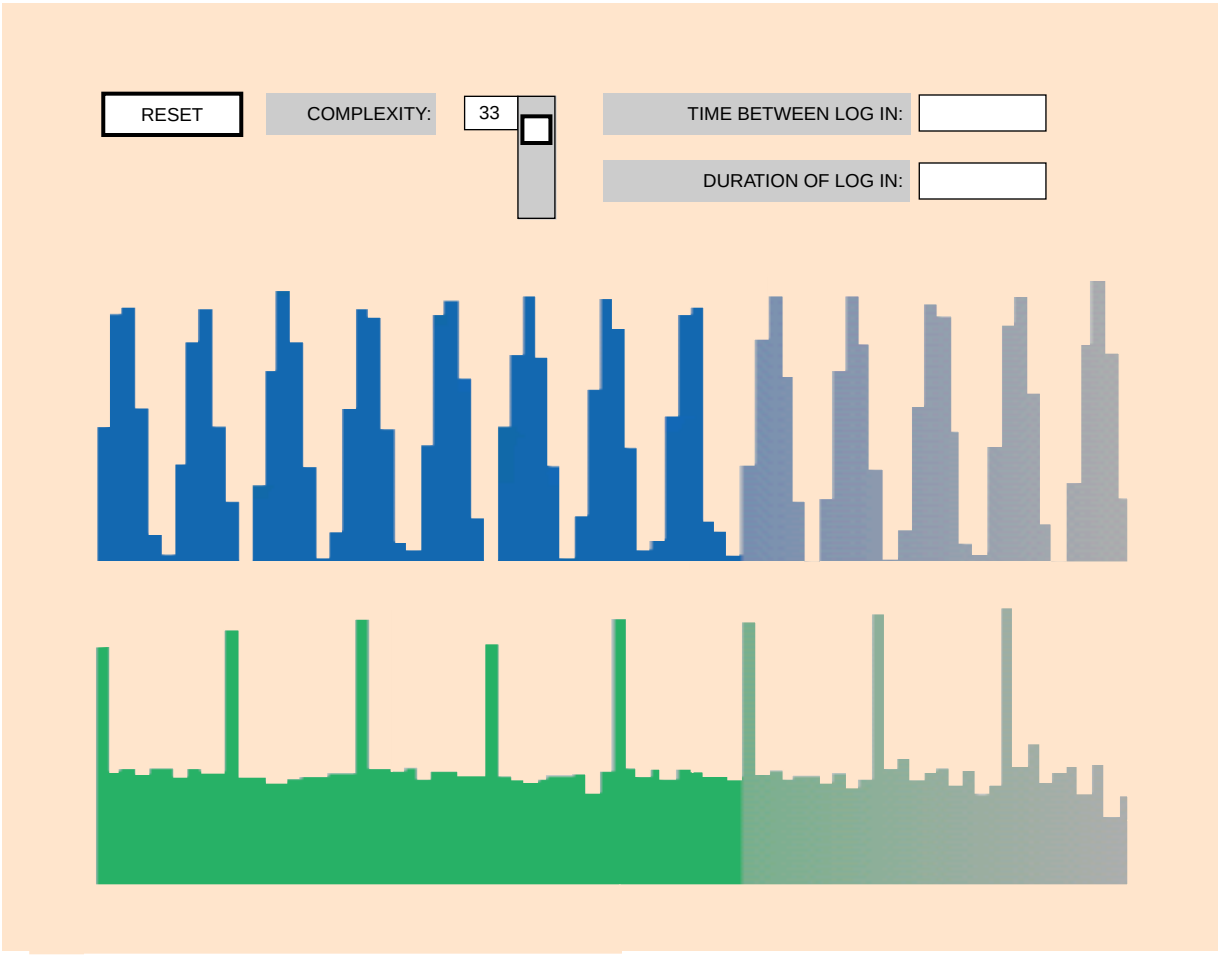
Philosophical discussions of artificial intelligence (AI) occasionally mention "AI-complete" tasks: tasks so difficult that, if they are ever accomplished, they will have paved the way for the solution to most other significant problems in AI. (The notion originated with the set of NP- [nondeterministic polynomial] complete problems in complexity theory.) Perhaps there are "UM-complete" tasks as well: an application whose user modeling needs are so great that success entails modeling the user well—so well that most other applications can be personalized too. Such a task would require recognizing a number of user patterns, such as when an event will next occur. For that, DOPPELGÄNGER uses *linear prediction*.

A user modeling toolbox

No single technique will be best for all, or even a majority, of tasks. Some data streams are temporal, others spatial; some are discrete, others continuous; some are numeric, others symbolic. It will not always be clear which technique is best, or even valid, for a particular domain. User modeling systems need to be extensible—able to use new data streams and sensors as they become available. They should be general enough to handle the seamless integration of new sensors. If a sensor suddenly becomes impaired, the system should exhibit graceful degradation—the quality of the modeling will decrease, but the modeling should not stop.

Several techniques will be described here: linear prediction, the Beta distribution, Markov models, and cluster analysis. To demonstrate how these techniques are used, one particular domain will be explored: news. DOPPELGÄNGER[1] was originally developed for use with personalized electronic newspapers;[9] customizing a newspaper spans several aspects of users' lives, both cognitive (what

Figure 2 DOPPELGÄNGER's depiction of linear prediction values. Each blue bar represents the number of seconds between the user's newspaper readings; each green bar represents the duration of the actual reading. The solid bars are previously observed values; the shaded values are DOPPELGÄNGER's predictions for the future. The higher the color saturation, the greater DOPPELGÄNGER's confidence.



they are thinking) and pragmatic (what they are doing). It is much more than merely choosing the right articles.

- *Topic selection* requires knowing user interests.
- *Timely service* requires knowing when a newspaper will be requested.
- *Breaking news* requires being able to reach the user quickly and gauging the importance of news for a particular user.
- *Tailoring presentations* requires knowing how much time the user has and how to compose effective newspaper layouts.

Suppose a user logs in every morning at 8:00 to read his or her electronic newspaper. Article selection and organization might take a few minutes: news wires have to be scanned, breaking news has to be assimilated, collated, and rendered, and the newspaper must send its data to the user's electronic mail, pager, or windowing system.

All of that takes time. It would be nice if the newspaper could know beforehand when the user is going to read the newspaper, and how much time he or she has to read it. That requires a series of discrete, linear, cyclical, and numeric observations: perfect for linear prediction.

Modeling events with linear prediction

Linear prediction (not to be confused with linear extrapolation) is particularly good for predicting cyclical behaviors. DOPPELGÄNGER uses linear prediction to guess when users will next read their newspaper, and how long they will spend reading it. It is an inexpensive technique—it does not require much computation, and the amount of computation required is scalable: computation can be sacrificed for accuracy.

Linear prediction is just one way to predict the occurrence and duration of an event. But any predictive technique used for predicting reading behavior needs to be robust enough to withstand the vagaries of daily life. Suppose a technique assumed that a user reads the newspaper (or mail) every 24 hours. And then, one day, that person comes to work at 10:00 A.M. instead of 9:00 A.M. due to unusually heavy traffic. If the system merely looked back one day, it would incorrectly conclude that the user would again read the newspaper at 10:00 A.M. the following day.

Suppose you had a slightly more clever algorithm that computed a moving average. Then a single anomalous observation would not severely impair the accuracy for our tardy user. After eighty 9:00 A.M. observations and a single 10:00 A.M. observation, the system would predict a value just slightly after 9 A.M.—not a bad estimate.

But suppose that user reads news twice a day, in the early morning and early afternoon. Then a moving average would yield an intermediate value, late morning. Thus the moving average is not good enough—something more is needed.

The best way to explain linear prediction is to consider the canonical domain for it: ocean tides. Tides possess daily cycles, seasonal cycles, and trends over the years. The cycles and trends are undeniably present, but possess innumerable small variations affecting both magnitude and phase, just like patterns of newspaper reading behavior.

DOPPELGÄNGER used linear prediction in tandem with a newspaper vending machine, the kind you might see on street corners. Instead of a stack of traditional newspapers, the machine contained a laser printer linked to a computer. Users who had stored their models on PCMCIA (Personal Computer Mem-

Modeling events with linear prediction

Linear prediction[10] uses the autocorrelation of the data sequence to predict future values. The data stream is assumed to be stationary—in other words, it is assumed that the cyclical characteristics are independent of the absolute time. That is a reasonable assumption for predicting when a user will read a newspaper, and for how long, as long as the observation span is short enough. A user might exhibit trends over the course of a year or two, but daily and weekly cycles will dominate the schedule over the short term.

Given a series of times $y_1 \ldots y_N$, linear prediction estimates the next time, y_{N+1}, by first calculating the autocorrelation vector ϕ:

$$\phi_j \equiv \langle y_i y_{i+j} \rangle \approx \frac{1}{N-j} \sum_{i=1}^{N-j} y_i y_{i+j}$$

which is then used to compute the autocorrelation coefficients d_j, via the series of linear equations satisfying:

$$\sum_{j=1}^{M} \phi_{|j-k|} d_j = \phi_k$$

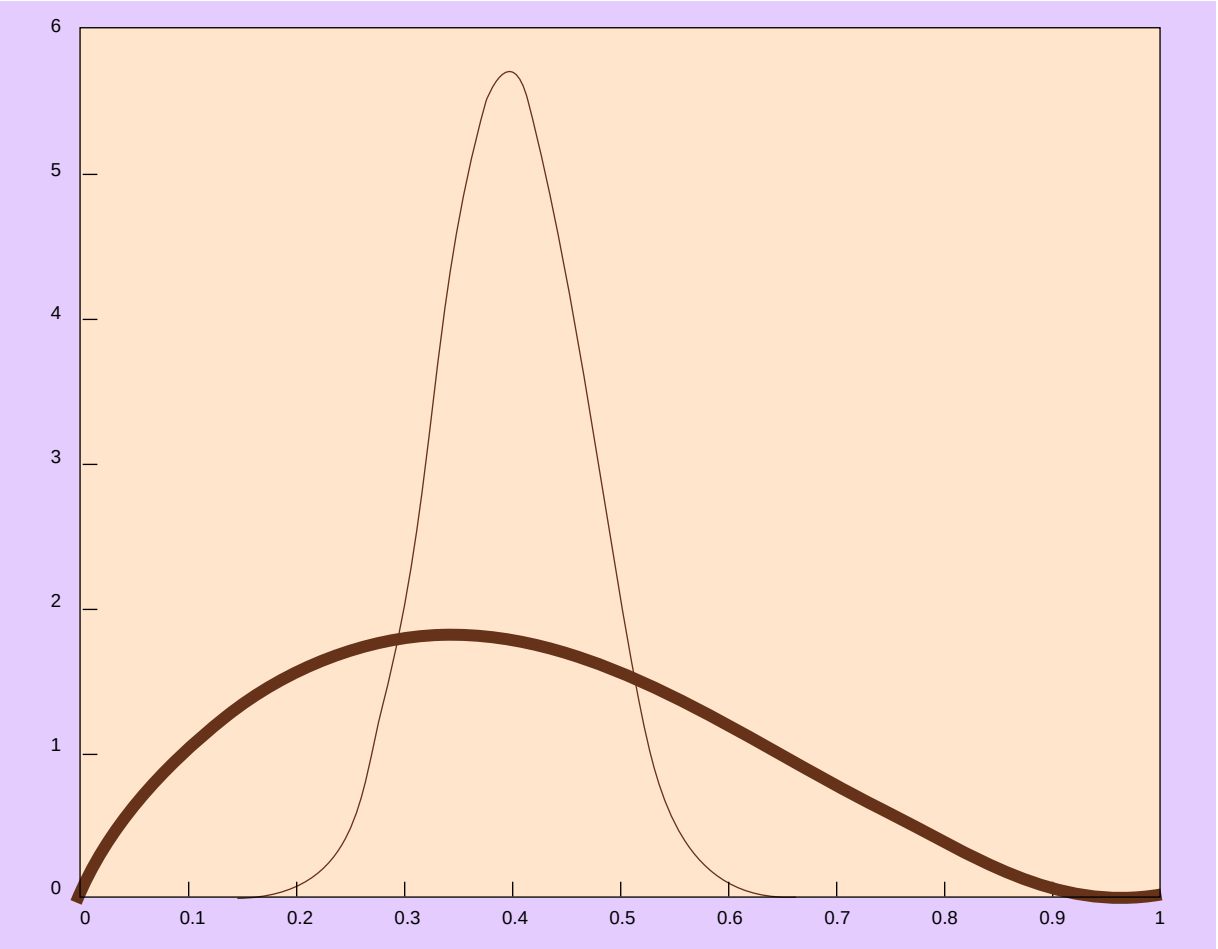
Once the d_js have been obtained, a y_n (for any n, including $N+1$ and other not-yet-observed times) is estimated as:

$$y_n = \sum_{j=1}^{M} d_j y_{n-j} + x_n$$

where x_n is an estimate of the observation error.

Figure 2 shows the differences in consistency between *when* the user reads the newspaper, and *how long* the user takes to read it. The former (depicted by the blue bars) is very predictable; the latter is not.

Figure 3 Two Beta distributions: the thick line shows the distribution generated when the user seemed to like a topic once but dislike it twice; the thin line shows the distribution generated when 19 likes and 29 dislikes have been observed. Note that with the larger number of observations, the variance decreases, and therefore DOPPELGÄNGER's confidence increases.



ory Card International Association) cards could insert them into the PCMCIA reader on top of the machine, which caused the user's personalized newspaper to be generated and printed.

Unfortunately, this whole process took about ten minutes—a long time to expect someone to wait. So linear prediction was used to predict when the user would next request a newspaper, so that it would be ready for that person beforehand.

Figure 2 is a still frame from an animation; as time elapses, the values march to the left, showing how well DOPPELGÄNGER's predictions fare against the actual observations.

Modeling interests with the Beta distribution

Assuming that our newspapers know when users will read them, how should they decide which articles to show? Selecting articles (or documents, or electronic mail messages) for a personalized newspaper relies upon two difficult tasks: extracting meaning from text and identifying the user's interests.

Extracting meaning from text. How should articles be categorized into topics? And how should the universe of topics be organized? Commercial categorization services, such as Burrelle's Information Services, provide a cut-and-dried hierarchy of topics, but users might like an article for an entirely different reason, such as the style or tone. Worse, they might want to read it specifically because they disagree with the opinions expressed.

Identifying the user's interests. Given some feedback from the user about an article, user modeling systems should attempt to extrapolate that feedback with the goal of estimating the user's true interest in the topic.

Suppose a newspaper can provide one-bit feedback (yes or no) about whether the user liked a topic. That stream of bits might come from mouse clicks: either they clicked on the article, or they did not. Or the stream of bits might come from a feedback mechanism, such as Koen's graphical "thumbs up" and "thumbs down" icons.[12] The user's true preference for the topic can be expressed as a strength (ranging from 0 to 1) and a confidence (also from 0 to 1, and proportional to the number of bits observed). The Beta distribution allows DOPPELGÄNGER to compute strength and confidence in this manner.

There are a few problems with this approach. First, all observations are treated equally: the first observation affects the estimate as much as the last. It would be better to assign higher importance to more recent responses.

Second, the article classifications are constrained to be all-or-nothing—an article either does or does not belong to a particular topic, with no compromise possible. Real life is not quite so forgiving: one article might be only a mediocre example of a topic, while another a great example of *two* topics.

Modeling interests with the Beta distribution

With linear prediction, the confidences depend on how far into the *future* the predictions are made. In the Beta distribution, they are proportional to how many *past* observations have accumulated.

The Beta distribution[11] (strictly speaking, the Beta probability density function) is cheap to calculate. It requires only two numbers, the number of hits and the number of misses, and from them it can generate an estimate and a confidence. Suppose that DOPPELGÄNGER registers h "hits" and m "misses" for a user's interest in a particular topic. The Beta distribution can then be used to pick the most likely value of the user's "true" preference, and to estimate the probability that any particular preference is the right one.

Two Beta curves are shown in Figure 3. In the distribution, x_0 ranges along all possible strengths from 0 to 1, and the height of the curve $\beta(x_0)$ indicates the probability that each possible strength is the "true" value.

$$\beta(x_0) = c(h, m)x_0^{h-1}(1 - x_0)^{m-1}$$

for $0 < x_0 < 1$, where

$$c(h, m) = \frac{(h + m - 1)!}{(h - 1)!(m - 1)!}$$

The expected value of the Beta distribution is

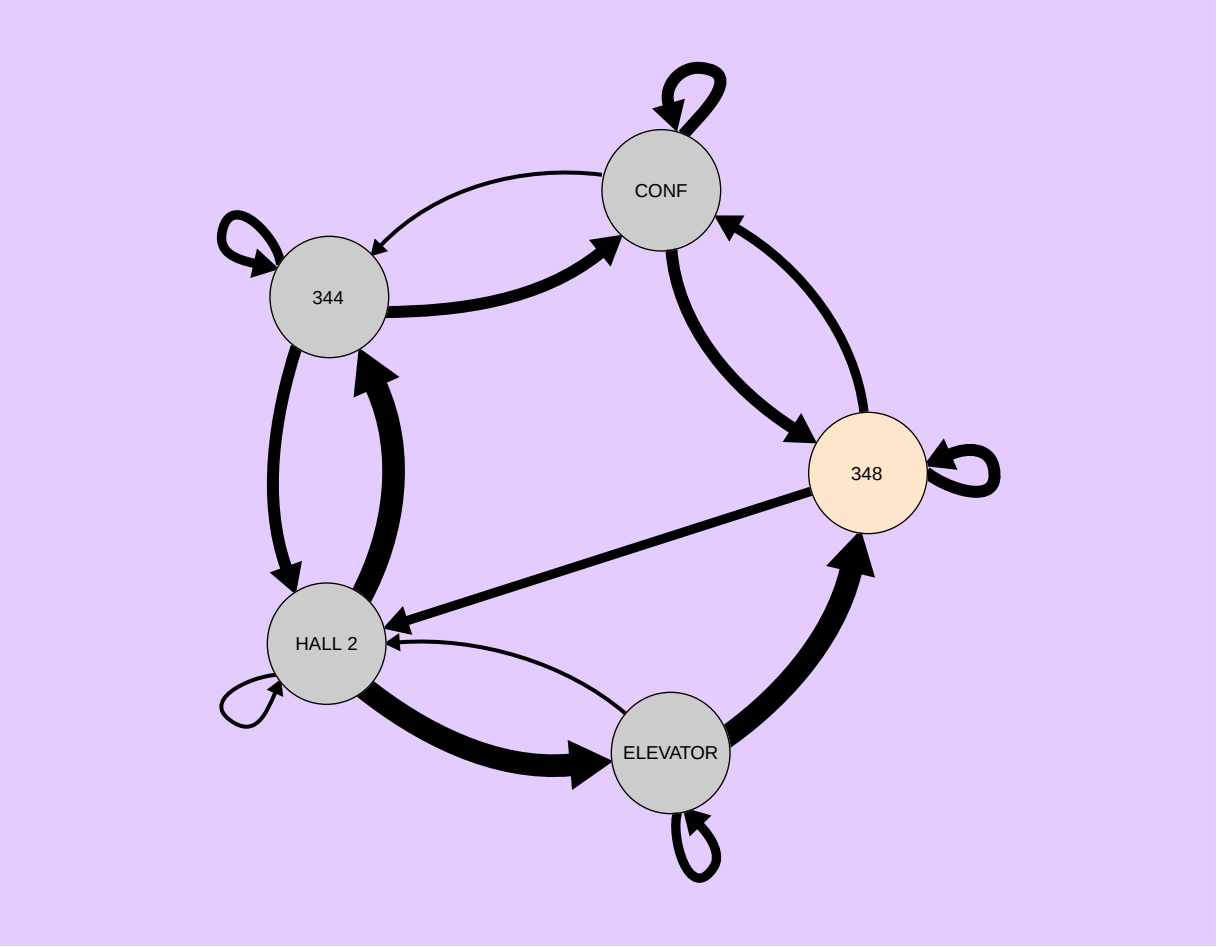
$$E(x) = \frac{h}{h + m}$$

as you might expect: the best estimate of the user's preference is the number of hits divided by the total number of trials. The variance is not quite so intuitive:

$$\sigma^2 = \frac{hm}{(h + m)^2(h + m + 1)}$$

It is just a number indicating how "wide" the curve is. The more trials, the smaller the variance, the narrower the curve, and the higher DOPPELGÄNGER's confidence in the estimate.

Figure 4 An automatically generated Markov model corresponding to a particular user's motion around the Media Lab. Arrow thickness indicates the probability of movement between areas.



Third, the user feedback is limited to "yes" or "no"—there is no distinction between "liked the article somewhat" and "liked the article very much."

Fourth, credit assignment is difficult. Many articles are reasonably good examples of more than one topic: how then do we identify which topic excited the user?

These are all serious problems. Even one of these criticisms reveals that the Beta distribution will not always be powerful enough for reliable predictions. But the immediacy of news requires a fast response from DOPPELGÄNGER, so a cheap technique that can be computed quickly, such as the Beta distribution, is necessary.

The Beta distribution is useful for estimating a single value. When the user passes through a series of discrete states, another representation is called for, such as Markov models.

Modeling location with Markov models

Assume now that we know what our user wants to read, when he or she will read, and perhaps even how much time he or she has to read. But linear prediction is no guarantee of success.

The most effective predictor would not have to predict at all—consider a sensor that parses the user's datebook. A sensor of this sort was used by DOPPELGÄNGER to provide personalized weather reports; DOPPELGÄNGER would tell the weather application where the user was, and the weather application would page the user with a weather forecast for the appropriate area.

Another mathematical technique, one that will tell us not only how long the user has to read the paper, but what the user will do next, is the Markov model. Markov models are used by DOPPELGÄNGER to identify locomotion patterns in users, with room-level granularity, using Olivetti Active Badges** as location-tracking devices. Smaller motions can be sensed with newer, passive sensors,[13] and more information can be transmitted with smarter badges personalized for each user.[14]

Every 15 seconds, each Olivetti badge emits an infrared pattern that uniquely identifies its wearer. Stationary receivers (that might be mounted on walls, desktops, or corridors) detect the badges and relay their observations to a central computer, which can then be used to query the user's location. That data stream alone, without any prediction, is useful for applications such as "smart" phone switches that route calls to the phone closest to a person.

DOPPELGÄNGER then archives that information, constructing a simplistic Markov model that can be used to predict the user's locomotion. Whenever the user moves, his or her Markov model (and hence his or her user model) changes. The more regular the user's schedule, and the longer the tracking period, the better the prediction.

Informed consent. Active badges provide one of the best solutions to the considerable privacy concerns raised by user modeling. Users have to deliberately attach the badge to their clothing to activate the tracking, and can remove the badge at any time to deactivate tracking. There is never any doubt about when data are being gathered.

Modeling location with Markov models

Every Markov model is defined by three collections of information:

- *A set of states*—In DOPPELGÄNGER, each state is a room, or corridor, or desktop. Better modeling might involve assigning states to microbehaviors: "going to the water cooler" might be one state, "leaving for the day" another.
- *A matrix of transition probabilities*—Given a particular state, a Markov model describes a particular probability that, on the next "time tick," the "system" (for our purposes, the user) will "transition" (walk) to another state (room).
- *A matrix of output probabilities*—Given a particular state, Markov models assume that some "output symbol" will be generated by the system. DOPPELGÄNGER ignores these for location modeling (but not for behavior modeling, described next).

When a user walks from Room A to Room B, that affects the transition probabilities for Room A: bolstering the value for Room B and diminishing the values for all other rooms. If the user stays in Room B for a long time, that affects the transition probabilities for that room: bolstering the value for Room B and diminishing the others. Figure 4 shows the Markov model generated for one user after ninety minutes of observations.

This method of location modeling is the cheapest technique described in this essay. Since new observations affect only the transition probabilities (unless a new room is entered, in which case a new state is grown), only a small number of multiplications and divisions are required: both updates and prediction are nearly instantaneous. And it is practical for use on a larger scale: the time required to update is proportional to the number of states and constant with the number of observations; the time required to predict is constant with both the number of states and the number of observations.

Figure 5 The Markov model for the programming state: rapid alternation between text editing, compiling, and "learning" (reading on-line manual pages)

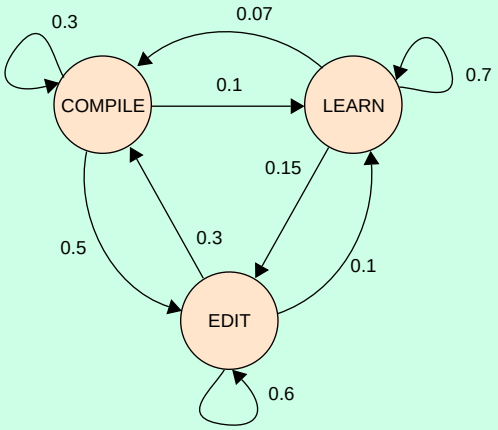
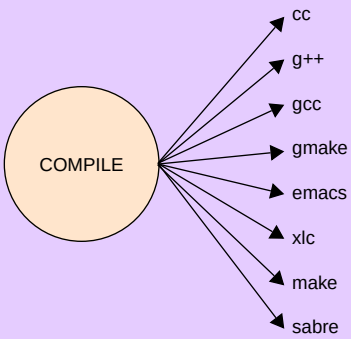


Figure 6 The "compile" state and its output symbols



AT&T Bell Laboratories developed "smart floors"[15] that sense the impact of the user's foot on the floor, and use time series to identify whose foot it is. It is a more passive sensor than the Active Badges, and therefore more convenient for users. But achieving informed consent with such sensors is quite a bit more difficult, since users might not have a choice whether or not to walk over the sensed area. Furthermore, notification of how the data are acquired and what can be done with the data requires considerably more explanation than will fit on a warning sign delimiting the area.

Categorizing behavior with hidden Markov models

DOPPELGÄNGER's locomotion modeling involves constructing a Markov model for each user. Now, let us consider a twist: given a collection of already-built Markov models, how do you select the one that best predicts a series of observations? That is how DOPPELGÄNGER classifies users into different behavioral states: frustrated, busy, concentrating, idle, and so on. The technique is quite general: it can be used whenever there is some sequence of discrete temporal observations that, taken as a whole, categorize users.

Categorizing behavior with hidden Markov models

When DOPPELGÄNGER needs to categorize the user's behavior, it uses the sequence of the user's UNIX commands to choose the Markov model (each of which corresponds to a single behavior category) that best predicts the sequence. At the broadest level:

For every Markov model (i.e., behavior):
  Calculate its likelihood of producing the observations (i.e., UNIX commands)
Then choose the Markov model with the highest probability.

Unlike the other techniques described so far, categorizing users does not begin *tabula rasa*: someone has to define the categories. I defined eight, one of which is shown in Figure 5. (The model shown, "*hacking*," is several years old and now obsolete owing to the surge in popularity of interpreted languages.[17])

The eight Markov models were determined by identifying which UNIX** commands DOPPELGÄNGER users were likely to use, and what might be inferred from their presence.

The Viterbi algorithm[16] is used to determine the most likely *a posteriori* succession of states through a particular Markov model (which is then called a hidden Markov model in the literature):

For every observation (i.e., UNIX command)
  For every state (i.e., class of commands),
    Identify the series of states most likely to produce the observation given the previous observations.

This iteration results in a continuous pruning of unlikely paths through the model states, making the algorithm operate in polynomial instead of exponential time. Additionally, calculation is done in log-space so that the probabilities and weights can be added instead of multiplied.

So how is the series of commands mapped onto the states of the Markov model? Remember that the Markov models used by DOPPELGÄNGER for location modeling ignored one of the components: the output probabilities. Here, Figure 6 shows the output symbols for the "compile" state. The associated probabilities are not shown; commands that are used for nothing but compiling (e.g., *cc*, *gcc*) have high probabilities; commands that can be used for other tasks (e.g., *emacs*, *make*) have lower probabilities.

Exactly how accurate DOPPELGÄNGER's techniques are is hard to quantify; since the efficacy of these techniques varies greatly from person to person and from sensor to sensor, rigorous evaluation is difficult. People exhibit trends; interests in news topics and expertise with applications change over time. Furthermore, completely new topics and applications become available over time; outside of lengthy testing (over the period of years, during which DOPPELGÄNGER would have to be kept static) with large test and control groups, attempts to quantify DOPPELGÄNGER's success may well raise more questions than they answer.

Furthermore, since DOPPELGÄNGER constantly recalculates the efficacy of each technique for each user, and weights them accordingly, the system learns continuously. It changes from one day to the next, and even from one minute to the next; few of the techniques permit the luxury of running once per night.

Constructing communities with cluster analysis

What can a user modeling system do when it has too little information about someone? The techniques described so far can identify patterns if patterns exist, but what if there are not enough data?

There is no reason that our systems must begin *tabula rasa*. After all, humans do not: when we hear that a graduate student spent all night at the library, we assume that the student was researching his or her thesis. We are making implicit assumptions about the student based on our knowledge about what it means to be a graduate student. It works, because there are correlations between our occupations and our actions.

When an application's query requires information missing from the user model, DOPPELGÄNGER can make an educated guess based upon the population of users. In particular, DOPPELGÄNGER bootstraps hypotheses about a user by aggregating the information from all user models. When possible, it chooses an appropriate subset: for instance, the *community* of graduate students.

How can a system know what communities exist? In two ways: the designer can name them, by identifying certain snippets of the user model as being more telling than others, or, the system can determine the communities itself, by looking for correlations between users.

A newspaper application that attempts to customize itself for a new user might present the user with a generic smattering of news. After making a few hypotheses about the user's preferences based on his or her selections (say, that the user likes the *telecommunications* and *computer* topics), DOPPELGÄNGER looks at other users with similar interests to see what else they are interested in (say, *science* or *advertising* or *telephones*) and creates hypotheses corresponding to those "once-removed" topics.

Gathering a few bits of data from many people can be more useful than many bits of data about a few topics. It is cheap artificial intelligence: a memory-based ontology.

Constructing communities with cluster analysis

A technique that is run once each night is the ISODATA unsupervised clustering algorithm,[18] which is used to group users into communities. ISODATA requires an approximate number of communities N_D, a maximum community size T, a maximum variance σ^2_S (for deciding when to split a community), a maximum distance D_m, and a maximum number of communities N_{MAX} (for deciding when to merge communities). Here is a brief overview:

For i starting at 1 and N starting at N_D,

1. Cluster the users into N communities, using K-means clustering with an n-dimensional Euclidean distance metric. Eliminate communities with fewer than T users.
2. If the communities have not changed since the last iteration, exit.
3. If $N \leq N_D/2$, or $N < 2N_D$ and i is odd, then:
   Split any communities whose users form sufficiently disjoint groups and increase N accordingly. If any communities have been split, go to step 5.
4. Merge any pair of communities whose users are sufficiently close and decrease N accordingly.
5. Increment i and return to step 1.

Unfortunately, ISODATA is not quite as unsupervised as it could be: it requires reasonable choices of the parameters T, N_D, and so on.

So DOPPELGÄNGER uses a simple program (called AUTODATA) wrapped around the ISODATA procedure. AUTODATA makes initial guesses for the parameters and adjusts them incrementally, repeating the five steps above until a sufficiently good clustering is found.

Unlike the other techniques described, the partitioning of the entire population into communities is computationally expensive. But since communities change far less often than people, the AUTODATA program can be run less often: once each night. The results are stored in a global file, one of the few DOPPELGÄNGER databases that is not part of an individual user model.

Synchronization and sharing

We have discussed hardware sensors, software sensors, the operating system itself as a sensor (via UNIX commands), and users as sensors. There is one frontier left: treating user modeling systems *themselves* as sensors. Any given user modeling system should be able to benefit from others. The techniques described in this essay are general and reusable—they are not specifically tailored for personalized newspapers. They could just as easily be employed by other user modeling systems; there should be a mechanism for eliminating redundant modeling and exploiting the multiple perspectives of users that different computers can provide.

Distributed user modeling takes place within DOPPELGÄNGER (the research project); there are several Doppelgängers (individual systems) currently in operation. Adhering to the privacy architecture described earlier, there is one Doppelgänger system in each participating computer; they share information with one another according to the user's wishes.

One application developed in tandem with DOPPELGÄNGER was a "synchronizer" application that ferries files back and forth between home and work computers. It uses two data streams: one is the pattern of log ins (gathered via the UNIX *last* command); the second is a sensor that observes the user's text editor to see which files he or she edits and how often. When the Doppelgänger system on the work computer predicts that the user is about to leave for home (using linear prediction), the synchronizer mails and installs the files on the home computer. And when the Doppelgänger system on the home computer ascertains that the user is about to leave for work, it mails and installs the files on the work computer.

That application synchronizes files, not computers. But it is the same principle: your user models should follow you around. If user models are to be used by everyday applications, they should be portable—*computationally* portable and *physically* portable. Users should either be able to carry their models around (as with DOPPELGÄNGER's PCMCIA cards) or they should be automatically shuttled between computers when they move from machine to machine. Perhaps user modeling would benefit if systems spoke a common language when expressing information about users.

Synchronization and sharing

The user models themselves are encoded in a LISP-like notation[19] that is used to transmit information to applications and to other Doppelgänger systems. (Different Doppelgänger systems, one at home and one at work, might model the same user.) Every "chunk" of the model is a list; the first element of each list is a numeric tag that indicates how to interpret the rest of the elements. There is one tag for assertions about user topic preferences, another for the actual sequence of hits and misses on that topic, and another tag for the entire user model itself. Each user model is a hierarchy; subtrees can be passed around between Doppelgänger systems. Portions of the user model can point to other portions of the same user model, or to portions of a community model.

There are approximately two hundred tags, some of which label complex data types. For instance, there is one tag that identifies a Markov model data type and consists of a name (another tag) followed by a few matrices (another tag), each of which contains probabilities (yet another tag). All Doppelgänger systems use the same representation, which facilitates communication between them.

Doppelgänger systems share information with one another through e-mail when necessary, and TCP/IP (Transmission Control Protocol/Internet Protocol) communication when possible, so that information about users can be exchanged in real time. For sites that have no users in common, community models are shared, so that inferences about groups of people (scientists, artists, graduate students, etc.) spread from system to system periodically.

This infrastructure has been used to disseminate not just user and community models, but source code as well: a Doppelgänger system that trusts TCP/IP connections from a particular IP address will automatically install encrypted DOPPELGÄNGER source code distributed from that address.

It might be argued that the best way to model users is to standardize applications so that they can communicate about users using some fixed, internal representation. "Application suites" such as Microsoft Office** would seem like reasonable development environments for such work—since all users use the same set of applications, each application can assume that all others are present and they can be developed in unison.

A user modeling protocol? Certainly, a protocol for expressing information about users is desirable. But is it possible? Consider the disparate goals of different user modeling systems: some (such as DOPPELGÄNGER) perform what the user modeling community calls "generalized" user modeling; others focus on student modeling, cognitive representations, or natural language processing. We need something that satisfies each of these domains without an excessive level of abstraction or inefficiency. It should be something that can be understood by a novice programmer in a day, and it should contain a core subset of functions that most user modeling systems can implement, earning the right to call themselves "compliant."

If that happens, application developers will have the opportunity to build provisions for user modeling into their products, querying and providing user data according to an Internet standard. All user modeling helps our applications behave better, but to maximize the impact of our work, it should percolate into Internet applications. That is where the largest user populations are, and that is where we should be gauging the success or failure of our work.

A user modeling protocol? That is a bad idea, of course—users should not be forced to use specific applications, or even specific operating systems. What we need is a protocol for encoding information about users, so that the applications and techniques developed at each site will be usable at every other site.

Should we call it a full-fledged "language," or does "protocol" suffice? Protocols are of a lower level than languages, and typically deal with matters such as the ordering of elements. Languages focus more on expressivity. There are so many things to sense about people, and so many scenarios and uses for the resulting inferences, that any communication mechanism must be open-ended, so that when new sensors are developed, or new behavior domains tracked, or new modeling techniques employed, they can be incorporated without breaking previous implementations. That suggests a language rather than a protocol, but it is intriguing to think about how an RFC (request for comments) for a user modeling protocol could be developed, and how we can begin to design a protocol that will not prohibit as-yet-undreamed-of applications. A widely adopted language would make user modeling seem more academic; a widely adopted protocol would make user modeling seem more stable.

It is almost too easy to choose a language rather than a protocol: by avoiding the constraints of a byte-by-byte representation, we can defer the hard choices and avoid making mistakes. But we risk creating something too general to be useful. The user modeling community should set its sights higher, by creating a low-level protocol for communication between applications, sensors, and user modeling systems.

What *are* the basic data types of user modeling? DOPPELGÄNGER's models contain thousands of confidence values, proper names, assertions, and keywords. Which of these merit data types? How should we standardize the building blocks that comprise user models: measurement units, time zones, if-then-else conditions, to say nothing of the more fluid and complex components: distance metrics, plans and goals, and broad assertions about likes and dislikes? There will be other, thornier issues as well: any protocol should have the capacity for letting applications specify a time limit for replies. Consider a game that needs to out-think or out-race its human opponent—a fast but mediocre answer will often be preferable to a slow but superb one.

Today, most Internet traffic is communication between users. We have a lot to say to each other, but our computers have even more to say to each other, especially when *we* are the topic. When the messages about us outnumber the messages between us, we will know user modeling has succeeded.

Reprise

Sadly, paper is not yet an interactive medium—static essays such as this one can neither anticipate reader questions nor elaborate concepts on demand. Structuring this paper as two independent yet parallel flows was meant to illustrate that documents are "applications" too, that not all readers are the same, and that information about users can be used for more than merely dividing audiences into exclusive categories. There are not really two essays here, just a single essay designed for two hypothetical readers and organized in an unusual way. By including paragraphs aimed at each reader, instead of collapsing the text into one homogeneous flow, a third channel of information is created. That channel is the *organization* of the essay, and, implicitly, the reasoning behind it: the attempt to categorize user modeling concepts while demonstrating user modeling through that very categorization.

Acknowledgments

This work was supported in part by the MIT Media Laboratory's News in the Future research consortium and International Business Machines Corporation.

**Trademark or registered trademark of Ing. C. Olivetti & C., S.p.A., X/Open Co. Ltd., or Microsoft Corp.

Cited references

1. J. Orwant, "Heterogeneous Learning in the DOPPELGÄNGER User Modeling System," *User Modeling and User-Adapted Interaction* **4**, No. 2, 107–130, Kluwer Academic Publishers, Netherlands (1995).
2. M. Turk and A. Pentland, "Eigenfaces for Recognition," *Journal of Cognitive Neuroscience* **3**, No. 1, 71–86 (1991).
3. J. Kay, "Generalised User Modelling Shells: A Taxonomy," *Proceedings of the IJCAI Workshop, Agent Modelling for Intelligent Interaction*, Sydney, Australia (August 24–30, 1991), pp. 169–185.
4. A. Kobsa and W. Pohl, "The BGP-MS User Modeling System," *User Modeling and User-Adapted Interaction* **4**, No. 2, 59–106 (1995).
5. R. Kass and T. Finin, "Modeling the User in Natural Language Systems," *Computational Linguistics* **14**, No. 3, 5–22 (1988).
6. S. Brennan, "Conversation With and Through Computers," *Second International Workshop on User Modeling*, Honolulu, Hawaii (March 30, 1990).
7. J. Orwant, "Apprising the User of User Models: DOPPELGÄNGER's Interface," *Fourth International Conference on User Modeling*, Hyannis, MA (August 15–19, 1994), pp. 151–156.
8. S. Garfinkel, *PGP: Pretty Good Privacy*, O'Reilly & Associates, Sebastopol, CA (1995).
9. J. Orwant, "Workshop in User Modeling," *News in the Future Consortium*, MIT Media Laboratory, Cambridge, MA (1993).
10. W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd edition, Cambridge University Press, Cambridge, UK (1992).
11. A. Drake, *Fundamentals of Applied Probability Theory*, McGraw-Hill, NY (1967).
12. P. Chesnais and D. Koen, "Strategies for Personal Dynamic Systems: News in the Future," *NeXTWorld Expo*, San Francisco, CA (May 25–27, 1993).
13. J. Smith, "Field Mice: Extracting Hand Geometry from Electric Field Measurements," *IBM Systems Journal* **35**, Nos. 3&4 (1996, this issue).
14. R. Borovoy, M. McDonald, F. Martin, and M. Resnick, "Things That Blink: Computationally Augmented Name Tags," *IBM Systems Journal* **35**, Nos. 3&4, 488–495 (1996, this issue).
15. T. Speeter, *Smart Floor and Smart Desktop: Concepts and Architecture*, AT&T Bell Laboratories, Holmdel, NJ (1990).
16. L. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proceedings of the IEEE* **77**, No. 2 (1989), pp. 257–285.
17. J. Orwant, *Perl 5 Interactive*, Waite Group Press, Corte Madera, CA (1996).
18. C. W. Therrien, *Decision Estimation and Classification*, John Wiley & Sons, New York (1989).
19. J. Orwant, *DOPPELGÄNGER Goes to School: Machine Learning for User Modeling*, master's thesis, MIT Media Laboratory, Cambridge, MA (1993).

Accepted for publication May 21, 1996.

Jon Orwant *MIT Media Laboratory, 20 Ames Street, Cambridge, Massachusetts 02139-4307 (electronic mail: orwant@media.mit.edu).* Mr. Orwant obtained the B.S. degree in computer science and engineering and the B.S. degree in brain and cognitive science from MIT in 1991, the M.S. degree in media arts and sciences from MIT in 1993, and is currently a doctoral candidate at the MIT Media Laboratory. He has published numerous articles about user modeling and electronic newspapers, and has appeared on television and radio and lectured internationally on those topics. He is the author of the programming book, *Perl 5 Interactive*, and publishes a quarterly magazine, *The Perl Journal*.

Reprint Order No. G321-5613