Relational Markov Models and their Application to Adaptive Web Navigation

Corin R. Anderson
Dept. of Comp. Sci. & Eng.
University of Washington
Seattle, WA, USA
corin@cs.washington.edu

Pedro Domingos
Dept. of Comp. Sci. & Eng.
University of Washington
Seattle, WA, USA
pedrod@cs.washington.edu

Daniel S. Weld
Dept. of Comp. Sci. & Eng.
University of Washington
Seattle, WA, USA
weld@cs.washington.edu

ABSTRACT

Relational Markov models (RMMs) are a generalization of Markov models where states can be of different types, with each type described by a different set of variables. The domain of each variable can be hierarchically structured, and shrinkage is carried out over the cross product of these hierarchies. RMMs make effective learning possible in domains with very large and heterogeneous state spaces, given only sparse data. We apply them to modeling the behavior of web site users, improving prediction in our PROTEUS architecture for personalizing web sites. We present experiments on an e-commerce and an academic web site showing that RMMs are substantially more accurate than alternative methods, and make good predictions even when applied to previously-unvisited parts of the site.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*data mining*; I.2.6 [**Artificial Intelligence**]: Learning—*induction*; I.5.1 [**Pattern Recognition**]: Models—*statistical*

Keywords

Markov models, relational probabilistic models, Web mining, personalization, shrinkage

1. INTRODUCTION

Markov models [27] are widely used to model sequential processes, and have achieved many practical successes in areas such as web log mining, computational biology, speech recognition, natural language processing, robotics, and fault diagnosis. However, Markov models are quite limited as a representation language, because their notion of state lacks the structure that exists in most real-world domains. A first-order Markov model contains a single variable, the state, and specifies the probability of each state and of transiting from one state to another. Hidden Markov models (HMMs) contain two variables: the (hidden) state and the observation. In addition to the transition probabilities, HMMs specify the probability of making each observation in each state. Because the number of parameters of a first-order Markov model is quadratic in the number of states (and higher for higher-order models), learning Markov models is feasible only in relatively small state spaces. This requirement makes them unsuitable for many data mining applications, which are concerned with very large state spaces.

Dynamic Bayesian networks (DBNs) generalize Markov models by allowing states to have internal structure [30]. In a DBN, a state is represented by a set of variables, which can depend on each other and on variables in previous states. If the dependency structure is sufficiently sparse, it is possible to successfully learn and reason about much larger state spaces than using Markov models. However, DBNs are still limited, because they assume that all states are described by the same variables with the same dependencies. In many applications, states naturally fall into different classes, each described by a different set of variables. For example, a web site can be viewed as a state space where each page is a state and each hyperlink is a possible transition. Classes of pages for an e-commerce site include: product descriptions, shopping carts, main gateway, *etc.* Variables associated with a product description page might be the product-id, the price, the quantity on hand, *etc.* Variables associated with a shopping cart page include the customer's name, the shopping cart ID, any relevant coupons, *etc.* These variables can help predict a user's navigational patterns, but it clearly would make no sense to associate a price with the site's gateway page or a credit card number with a product description page.

Examples of multiple state classes from other areas include:

Speech and language processing. Parts of speech (*e.g.*, only verbs have tense), semantic contexts (*e.g.*, asking about flights versus asking about hotels), types of discourse, *etc.*

Mobile robotics. Types of location (*e.g.*, indoors/outdoors, offices, laboratories, bedrooms, *etc.*).

Computational biology. Components of metabolic pathways, regions of DNA, protein structures, *etc.*

Process control. Stages of a manufacturing process, machine types, intermediate products, *etc.*

Fault diagnosis. Fault states associated with different subsystems, each with a different set of sensor readings, *etc.*

This paper proposes *relational Markov models (RMMs)*, a generalization of Markov models that allows states to be of different types, with a different set of variables associated with each type. In an RMM, a set of similar states is represented by a predicate or

SIGKDD '02 Edmonton, Alberta, Canada

relation, with the state's variables corresponding to the arguments of the predicate. The domain of each argument can in turn have a hierarchical structure, over which shrinkage is carried out [19]. RMMs compute the probability of a transition as a function of the source and destination predicates and their arguments. RMMs are an example of a relational probabilistic representation, combining elements of probability and predicate calculus. Other representations of this type include probabilistic relational models [10], probabilistic logic programs [21] and stochastic logic programs [20].

We expect RMMs to be particularly useful in applications that combine low-level and high-level information, such as plan recognition from low-level actions, or speech recognition aided by natural language processing. An example of the former is inferring information-seeking goals of web site users from the sequence of links they follow. Doing this inference makes it possible to automatically adapt web sites for different users, and as a result, to minimize users' effort in reaching their goals. RMMs are able to predict user behavior even in web sites (or parts thereof) that the user has never visited before, and are thus potentially much more broadly useful than previous approaches to web log mining, including traditional Markov models. In this paper we:

- Precisely describe relational Markov models and how they extend traditional Markov models;
- Apply RMMs to predict web navigation patterns;
- Empirically compare a variety of RMMs with traditional Markov models, demonstrating that RMMs predict users' actions more accurately.

The next section describes representation, inference and learning in RMMs. The following sections describe their application to adaptive web navigation, and the experimental results obtained. We conclude with a discussion of related and future work.

2. RELATIONAL MARKOV MODELS

Consider a discrete system that evolves by randomly moving from one state to another at each time step. A *first-order Markov model* is a model of such a system that assumes the probability distribution over the next state only depends on the current state (and not on previous ones). Let S_t be the system's state at time step t. Formally, a first-order Markov model is a triple (Q, A, π), where: $Q = \{q_1, q_2, \ldots, q_n\}$ is a set of states; A is the *transition probability matrix*, where $a_{ij} = P(S_t = q_j \mid S_{t-1} = q_i)$ is the probability of transiting from state q_i to state q_j, assumed the same for all $t > 0$; and π is the *initial probability vector*, where $\pi_i = P(S_0 = q_i)$ is the probability that the initial state is q_i. Given a first-order Markov model, the probability of observing a sequence of states $(s_0, s_1, \ldots, s_T)$ is $P(S_0 = s_0, S_1 = s_1, \ldots, S_T = s_T) = P(S_0 = s_0) \prod_{t=1}^{T} P(S_t = s_t \mid S_{t-1} = s_{t-1})$. Given a set of observed sequences, the maximum-likelihood estimate of an initial probability π_i is the fraction of sequences that start in state q_i, and the maximum-likelihood estimate of a transition probability a_{ij} is the fraction of visits to q_i that are immediately followed by a transition to q_j. In an nth order Markov model, the probability of transiting to a given state depends on the n previous states, and the transition matrix is $(n+1)$th-dimensional. We refer to Markov models of any order defined in this way as *propositional Markov models (PMMs)*.

Relational Markov models (RMMs) are obtained from the propositional variety by imposing a relational structure on the set of states. For example, consider a Markov model of an e-commerce Web site, in which each page is a state. A PMM would have a unique "proposition" for each page/state: for the main entry page, for each product description page, for the checkout page, *etc.* (see Figure 1). In a PMM each state is an atomic entity, and there is no notion of types of states. In contrast, an RMM groups pages of the same type into *relations*, with each relation described by its own set of variables (see Figure 2). For example, one relation might be "product description page," with a variable "product" representing the product the page describes, and "stock_level" representing whether the product is in stock or on back order. Additionally, these variables themselves are grouped together, forming a hierarchy of values; Figure 3 shows a fragment of such a hierarchy for products at an e-commerce site. A state instance is thus uniquely described as a tuple in a relation instantiated with leaf values from each variable's domain hierarchy. For example, ProductPage(iMac, in_stock) would represent the page describing an iMac computer that is currently in stock at the site's warehouse. Moreover, a tuple using non-leaf values is possible and corresponds to an *abstraction*—a distinguished set of states that are similar to each other in the sense that they have the same type and their arguments belong to the same sub-trees in the domain hierarchies. RMMs leverage these state abstractions for much richer learning and inference than PMMs, and make useful prediction possible in very large state spaces, where many (or most) of the states are never observed in the training data. In this paper, we focus on first-order[1] RMMs, but our treatment is readily generalizable to RMMs of any order. The next subsections describe representation, learning, and inference in first-order RMMs.

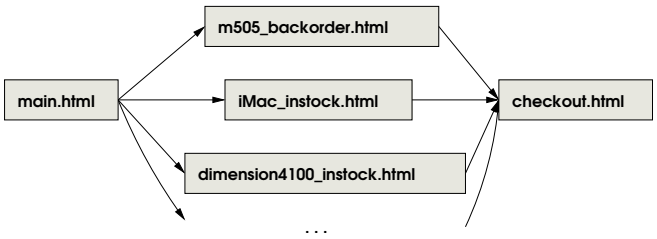

Figure 1: Propositional Markov model for an e-commerce site. Each box is a PMM state, representing a page in the site. Arrows indicate possible transitions in the PMM, and correspond to hyperlinks in the site.

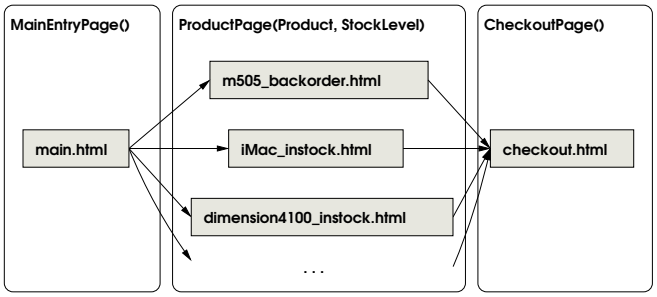

Figure 2: Corresponding relational Markov model. Each shaded box is a page/state, and states are are grouped (in rounded-corner boxes) by their relations.

[1] "First-order" is sometimes used in the literature to mean the same as "relational" or "predicate-level," in opposition to "propositional." In this paper we use it in the Markov sense, to denote the assumption that future states are independent of past states given the present state.

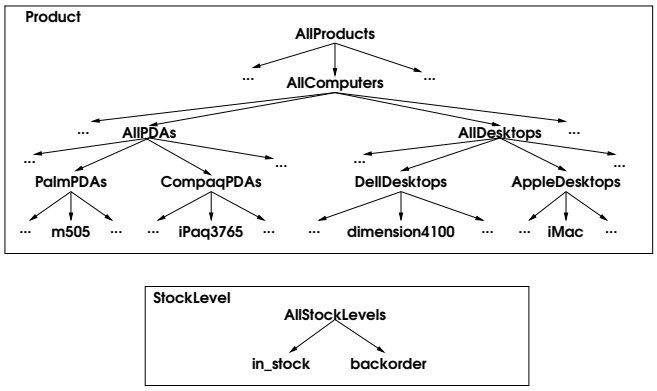

Figure 3: Abstraction hierarchy of products. Leaves in the tree represent ground values, while internal nodes denote categories of related values.

2.1 Representation

Formally, an RMM is a five-tuple $\langle \mathcal{D}, \mathcal{R}, Q, A, \pi \rangle$. $\mathcal{D}$ is a set of domains, where each domain, $D \in \mathcal{D}$, is a tree representing an abstraction hierarchy of values. Each leaf of D represents a ground value. $\mathcal{R}$ is a set of relations, such that each argument of each relation takes values from the nodes of a single domain in $\mathcal{D}$. Q is a set of states, each of which is a ground instance of one of the relations in $\mathcal{R}$, *i.e.*, where each argument is instantiated with a leaf of the corresponding domain. A (the transition probability matrix) and π (the initial probability vector) are the same as in a PMM.

To continue our simplified e-commerce example, suppose that $\mathcal{D}$ contains abstraction hierarchies for Products and StockLevels as shown in Figure 3. $\mathcal{R}$ is the set {MainEntryPage(), ProductPage(Product,StockLevel) CheckoutPage()}, where ProductPage(Product,StockLevel) specifies that the first and second arguments of the ProductPage relation must come from the Product and StockLevel domains, respectively. Q consists of several states, one of which is ProductPage(iMac, in_stock).

We now show how to use the relations and domain abstraction hierarchies to define sets of states as abstractions over Q. These abstractions are distinguished sets of states whose members are similar to each other by virtue of their relations and parameter values. That is, states whose parameter values are in common subtrees of their respective domains will appear in many abstractions together, while states with very different parameter values (or belonging to different relations) will appear together in only the most general abstractions.

We define these abstraction sets by instantiating a relation, R, with interior nodes (instead of just leaf nodes) from the domains of R's arguments. More formally, Let nodes(D) denote the nodes of a domain D. If d is a node in domain D, then let leaves(d) denote the leaves of D that are descendants of d. Let $R \in \mathcal{R}$ be a k-ary relation with domains $D_1, \ldots, D_k$. Let $d_1, \ldots, d_k$ be nodes in the corresponding domains. We define the *state abstraction corresponding to* $R(d_1, \ldots, d_k)$ to be the following subset of Q.

$$\{R(\delta_1, \ldots, \delta_k) \in Q \mid \delta_i \in \text{leaves}(d_i),\ \forall i, 1 \leq i \leq k\}$$

For example, given the domain trees shown earlier, Figure 4 shows several abstractions for the e-commerce RMM. Note that the abstraction ProductPage(AllProducts,in_stock) is the set of two ground states: {ProductPage(iMac,in_stock), ProductPage(dimension4100,in_stock) }.

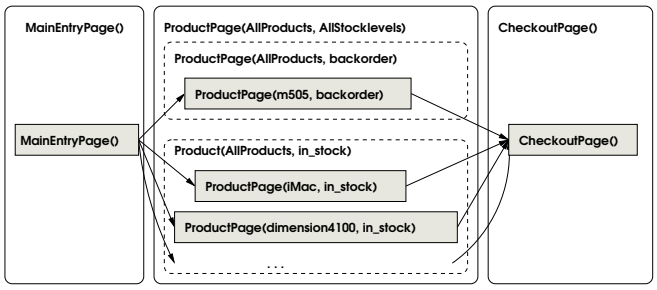

Figure 4: State abstractions for the relational Markov model. The hierarchy of Figure 3 defines abstractions over the RMM of Figure 2; the abstractions are depicted as rounded-corner boxes, labeled with their relations and arguments, and surrounding their ground states.

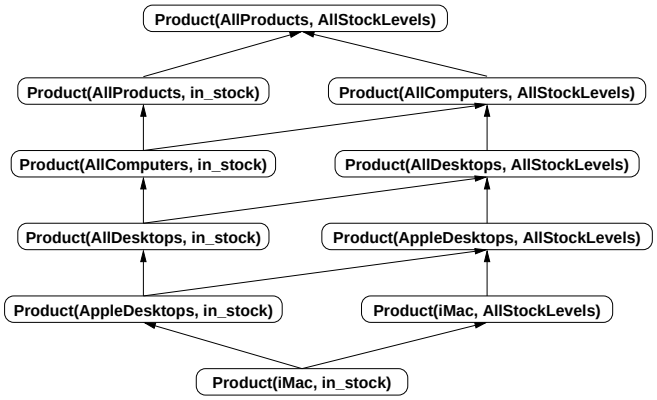

Figure 5: A lattice of abstractions. Boxes represent abstractions and arrows point in the direction of more general abstractions. These particular abstractions form the lattice for the ground state Product(iMac, in_stock).

Given a particular state $q \in Q$, it is especially interesting to know all the abstractions of which q is a member. Without loss of generality, suppose that $q = R(\delta_1, \ldots, \delta_k)$ and the domains of R's arguments are $D_1, \ldots, D_k$. We then define the *set of abstractions* of q, written $\mathcal{A}(q)$, as the following subset of the power set of Q:

$$\{R(d_1, \ldots, d_k) \subseteq Q \mid d_i \in \text{nodes}(D_i) \wedge \delta_i \in \text{leaves}(d_i), \forall i, 1 \leq i \leq k\}$$

For unary relations there is a total order on $\mathcal{A}(q)$, from the most specific ($\{q\}$) to the most general (Q). For n-ary relations, there is a partial order on $\mathcal{A}(q)$ (*i.e.*, $\mathcal{A}(q)$ forms a *lattice* of abstractions). For example, the abstractions of Product(iMac, in_stock) are shown in Figure 5, where arrows point in the direction of increasing generality. Finally, the *rank* of an abstraction $\alpha = R(d_1, \ldots, d_k)$ is defined as $1 + \sum_1^k \text{depth}(d_k)$, where depth() is defined as the depth of a node in a tree (with the root being at depth zero). The rank of Q (the most-general abstraction) is defined to be zero, and ranks increase as abstractions become more specific.

In the case of finite domains, RMMs are no more expressive than PMMs; given an RMM, an equivalent PMM can be obtained simply by creating a proposition for each tuple in Q. The advantage of RMMs lies in the additional support for learning and inference that the relational structure provides, as described in the next subsection.

2.2 Learning and Inference

In PMMs, the only possible learning consists of estimating the transition probabilities a_{ij} and initial probabilities π_i, and these estimates can be done reliably only for states that occur frequently in the training data. In many cases (*e.g.*, when modeling a user of a large Web site), most states are not observed in the training data, but it is still possible to generalize usefully from the observed behavior to unseen states. RMMs provide a formal framework for doing this generalization.

For each possible state abstraction α, we can define the corresponding initial probability π_α as the probability that the initial state is an element of α: $\pi_\alpha = \sum_{q_i \in \alpha} \pi_i$. Similarly, for each pair of state abstractions (α, β) we can define the corresponding transition probability $a_{\alpha,\beta}$ as the probability of transiting from a state in α to any state in β: $a_{\alpha,\beta} = \sum_{q_i \in \alpha} P(q_i|\alpha) \sum_{q_j \in \beta} a_{ij}$, where $P(q_i|\alpha)$ is the probability that the current state is q_i given that the current state is a member of α. The abstraction transition probabilities $a_{\alpha,\beta}$ can be estimated directly from the training data by counting. By making suitable simplifying assumptions, they can then be used to estimate the probabilities of transitions that are absent from the data. For example, if we assume that the destination state q_d is independent of the source state q_s given the destination abstraction β, then $a_{sd} = a_{\alpha,\beta} P(q_d|\beta)$, where α is the source abstraction. $P(q_d|\beta)$ can be estimated as uniform: $P(q_d|\beta) = 1/|\beta|$, where $|\beta|$ is the number of states in abstraction β. To make maximum use of all the available information, we propose to use a *mixture model* for each transition probability:

$$\begin{aligned} a_{sd} &= P(S_t = q_d \mid S_{t-1} = q_s) \\ &= \sum_{\alpha \in \mathcal{A}(q_s)} \sum_{\beta \in \mathcal{A}(q_d)} \lambda_{\alpha,\beta} a_{\alpha,\beta} P(q_d|\beta) \end{aligned} \quad (1)$$

where the sum is over all abstractions of the source and destination states, and the $\lambda_{\alpha,\beta}$'s are non-negative *mixing coefficients* that sum to 1. The generative model implicit in Equation 1 is that, to generate a transition, we first choose a pair of abstraction levels (α, β) with probability $\lambda_{\alpha,\beta}$, and then move to destination state q_d with probability $a_{\alpha,\beta} P(q_d|\beta)$. Effectively, this model performs *shrinkage* between the estimates at all levels of abstraction. Shrinkage is a statistical technique for reducing the variance of an estimate by averaging it with estimates for larger populations that include the target one [19]. Equation 1 applies shrinkage across an entire abstraction lattice, rather than over a single abstraction path (as is more usual). For example, a forecast of the number of Apple iMacs sold at a given store can be shrunk toward a more reliable forecast for the average of this quantity at all stores in the same city as the store of interest. The comparative values for the $\lambda_{\alpha,\beta}$'s effectively trade off bias and variance in the probability estimate. Terms corresponding to more general abstractions have lower variance, because they are estimated with more training data, but have a larger bias than terms from more specific abstractions. Thus, good shrinkage weights have two desirable properties: (1) they reduce the influence of abstractions with very little data; and (2) they allow increasingly specific abstractions to dominate as the training set size grows, with the RMM reducing to a PMM in the infinite-data limit. The mixing coefficients $\lambda_{\alpha,\beta}$ can be estimated in a number of ways, corresponding to different variations of our system:

- **RMM-uniform**: Uniformly (*i.e.*, all $\lambda_{\alpha,\beta}$'s are equal). This approach has the advantage of being extremely fast, but may lead to poor results.
- **RMM-EM**: Using the EM algorithm, as described in McCallum *et al.* [19]. In preliminary evaluation this option performed poorly, due to insufficient training data, so we did not evaluate it further.
- **RMM-rank**: Using a heuristic scheme based on the rank of the abstraction. In particular, we experimented with the following method:

$$\lambda_{\alpha\beta} \propto \left[\frac{n_{\alpha\beta}}{k}\right]^{\text{Rank}(\alpha)+\text{Rank}(\beta)} \quad (2)$$

where $n_{\alpha\beta}$ is the number of times that a transition from a state in α to a state in β could have occurred in the data (*i.e.*, the number of visits to a state $q_i \in \alpha$ to which a transition to a state $q_j \in \beta$ is possible), k is a design parameter, and the proportionality constant is derived from the requirement that the $\lambda_{\alpha\beta}$'s sum to 1. This heuristic meets our desiderata for the weights, although many other variations are possible. The choice of k controls how much data must be seen at a given abstraction level before that level can have a significant weight; when $n_{\alpha\beta} < k$, $\lambda_{\alpha\beta} \approx 0$. In experiments with validation data, we have found that setting $k = 10$ works well in practice.

The size of the abstraction lattices, and hence the number of terms in Equation 1, increases exponentially with the arity of the source and destination relations. Thus, when these arities are large, and/or when the abstraction hierarchies are deep, it may not be practical to compute all the terms in Equation 1. Instead, we can select the more informative ones, and set the mixture weights of the rest to zero (thus ignoring them). An efficient way of doing this culling is to learn a decision tree with the destination abstraction as the class, and the arguments of the source relation as the attributes. Each node along the path in the tree that the source state follows corresponds to an (α, β) pair that will have a non-zero weight in Equation 1. These weights can then be chosen using any of the methods suggested earlier; this approach is simply selecting *which* terms will have non-zero weight.

More precisely, we learn a *probability estimation tree* or *PET* [26], because the goal is to estimate the probability of each destination abstraction, rather than simply predicting the most likely destination. Any set of abstractions that form a partition of the destination states can in principle be used as the class. In this paper, we consider only the highest level of abstraction—the relation $R_d \in \mathcal{R}$ of the destination state. We learn a PET for each source relation separately. The candidate attributes include each argument of the source relation at each level of its domain hierarchy; thus, a k-ary relation each of whose arguments has n abstraction levels yields kn attributes. Figure 6 shows an example of such a PET.

To select the most informative terms in Equation 1 for a given source state, we consider the path the state goes down in the PET. Each node in the path has an associated probability distribution over destination abstractions (shown in Figure 6 in dashed-line boxes for a few nodes), and corresponds to a set of $a_{\alpha,\beta}$ terms in Equation 1, one for each destination abstraction. The α corresponds to the decisions made from the root to the node, and β belongs to the set of abstractions that the PET predicts. For example, the highlighted leaf node in Figure 6 corresponds to the source abstraction ProductPage(AppleDesktops, in_stock) and (like all other nodes in the tree) includes destination abstractions MainEntry(), ProductPage(AllProducts, AllStockLevels), and Checkout. The terms gathered from all the nodes along the path to the leaf are combined according to Equation 1, with the shrinkage

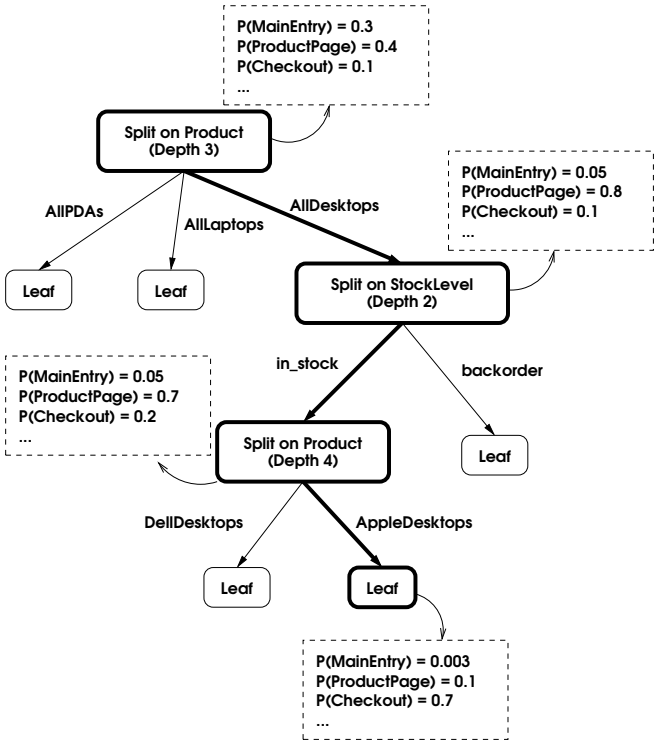

Figure 6: A PET predicting destination relation from ProductPage**.** Rounded-corner boxes represent tree nodes. The path the page ProductPage(iMac, in_stock) follows is highlighted. Each node has an associated distribution over the destination relations; these are shown for the highlighted nodes. Each node along this path selects a set of (α, β) abstraction pairs that are combined according to Equation 1. The α abstraction for a node is derived from the decisions made along the sub-path from the root to that node, and the β abstractions are the abstractions the PET is predicting (the destination relation in this example).

coefficients computed using any of the methods described above. In this paper, we choose these weights using EM; by predicting the destination relation, and not more specific abstractions, there is sufficient data to perform EM reliably. In our experiments, we call this variant **RMM-PET**.

More generally, we could build multiple PETs, each one estimating transitions to a different partition of the states; thus, the source state would follow multiple paths, one in each PET. For example, one PET would predict the destination relation, another would predict the destination at a lower level of abstraction (for instance, the relation and one variable's ground value), *etc.* The terms collected from all the PETs would then be combined according to Equation 1.

In practice, in large state spaces it is often the case that only a fraction of the states are directly reachable from a given state. For example, on a Web site only the pages that the current page links to are directly reachable from it. In this case, the $P(q_d|\beta)$ terms in Equation 1 can be replaced by terms that also condition on the knowledge of the set of states $\mathcal{C}(s)$ that are directly reachable from q_s. For states that are not reachable from q_s, $P(q_d|\beta, \mathcal{C}(s)) = 0$. For states that are reachable from q_s, under the previous assumption of uniform probability, $P(q_d|\beta, \mathcal{C}(s)) = 1/|\mathcal{C}(s)|$.

Notice that, in principle, any machine learning method could be used to predict the destination state as a function of properties of the source state. The approach proposed here implicitly encodes the learning bias that the abstraction hierarchies over the relation arguments are useful for generalization (*i.e.*, two states whose values are closer in their respective hierarchies are more likely to transition to the same state than states that are far apart).

2.3 Complexity

Inference in RMMs is slower than in PMMs, but is still quite fast. The computation of a single a_{sd} requires combining up to $|\mathcal{A}(q_s) \times \mathcal{A}(q_d)|$ estimates, but each of these estimates is obtained from the learned model by a simple lookup. Thus the overall computation is still fast, unless the abstraction hierarchy is very large. In this case, an approach such as RMM-PET can greatly reduce the number of abstractions that need to be considered, by identifying the few estimates that are most informative. Further, we can trade off time for memory by pre-computing and storing the a_{sd}'s, in which case inference becomes as fast as in PMMs.

Learning in RMMs conceptually involves estimating a transition probability for every pair of abstractions, rather than for every pair of states. However, only the transitions between abstractions that actually occur in the data need be considered (in the same way that, in PMMs, only the page transitions that actually occur need be considered). The transition probabilities for non-leaf abstractions can be computed without looking at the data, by aggregating counts for the lower-level abstractions. As a result, the dominant term in the learning time of an RMM is often the computation of the leaf-level probabilities, which is the same as for PMMs.

3. ADAPTIVE WEB NAVIGATION

Our work on RMMs is motivated by the desire to automatically personalize web sites based on a person's browsing pattern. Although individuals vary in their web navigation patterns, most web sites have a static organization that is designed for general use. In previous work we proposed the PROTEUS architecture for automatically personalizing web sites for individual visitors [3]. Adaptation in PROTEUS follows a two-step approach. First, PROTEUS mines web server logs to build models of users. Second, as users request pages at the site, PROTEUS considers all the ways in which to adapt the site (*e.g.*, add a link between two pages, rearrange list items on a page, elide content from a long page, *etc.*) and selects the adaptations that yield the greatest expected utility per the model mined in step one. PROTEUS employs heuristics and a strong bias to ensure that this search is efficient. In a study of a dozen users with a wireless web browser, PROTEUS reduced the time and navigational effort required for users to find information on small-screen, low-bandwidth devices.

In the PROTEUS framework, we found adding shortcut links to be particularly useful. A shortcut link connects two previously "distant" pages in the site, where distance is measured as the number of intermediate pages. For example, if a site contains the pages A, B, and C and the links A→B and B→C, then the shortcut A→C would shorten the path from A to C by one link. Concentrating on the shortcut creation problem, we developed the MINPATH algorithm [2], which composes many page transition predictions to predict the expected savings every possible shortcut in the site would offer. We experimented with mixtures of propositional Markov models, including first- and second-order models, and found that a mixture of first-order Markov models fared the best, saving visitors up to 40% of their navigation effort.

MINPATH's performance is limited by the quality of the underlying page navigation model, and, as we have mentioned earlier, first-order PMMs have a number of weaknesses. The most significant is that PMMs cannot offer informed guidance at pages for which there is no training data. If a web page did not exist during

the training period (or simply was not visited), the Markov model can do no better than predict a uniform distribution over the out-adjacent pages. This phenomenon is very common on large and/or dynamically-generated web sites: on a portal site the news stories change every day; customers at an e-commerce site typically view product descriptions they have not previously read; and after a semester is over, students begin viewing the course pages for a different set of courses. Instead, ideally, we would like the model to take advantage of the relational structure of the space of pages. For example, visitors prefer news stories of a particular genre and products of similar types. If a student views numerous homework pages for a particular course in a given department, then the visitor is likely to continue preferring homework pages, pages for that course, and courses in that major.

As we demonstrate in the next section, RMMs effectively address the issue of sparse training data in large sites by making use of a relational model of the web site identifying semantic correspondence between pages, both previously visited and unseen. The relational model is frequently already available, in the form of a database data model or other conceptual model that the human web site designer developed and maintains with the site content. In our evaluation we measure the predictive accuracy of RMMs for page navigation, and incorporate RMMs into our MINPATH implementation and PROTEUS system.

4. EMPIRICAL EVALUATION

In this section, we address the following questions: (1) Is our hypothesis correct that RMMs outperform propositional Markov models when data is sparse? (2) In data-rich environments when PMMs perform well, are RMMs at a disadvantage? (3) Are RMMs competitive in terms of CPU time required for learning? (4) Which of the RMM variants (uniform, rank, or PET) performs best?

To answer these questions we selected three sets of log data taken from two real web sites, `www.gazelle.com` (the e-commerce site introduced in the KDDCup 2000 competition [16]) and the instructional pages from our home institution `www.cs.washington.edu/education/courses/`. At both sites, we explicitly modeled when users ended a browsing trail, by creating a distinguished STOP page that was linked from every page in the site and which users implicitly visited at the end of a trail. We represented each page in the site as a state and the input to the models was the links users followed during the training period. The experimental task is to predict the probability a user will follow each link given the user's current page. The KDDCup data has the advantage that it represents the large class of sites dynamically-generated from database queries and page templates, but was not ideal because some domain modeling questions could not be answered without the "live" site. Our home institution's site was useful because it is operational and we have substantial amounts of data available for mining.

For both sites we collected clickstream data and the list of links on each page. Determining hyperlink connectivity was easy at our home institution—we crawled the site and parsed linkage data to create the model. However, although we had log data for `www.gazelle.com`, the site was no longer operational. Hence, we were forced to generate an approximate linkage model composed of the subset of links that were actually followed in the log data. While this solution is suboptimal (even if a link was never followed, its presence may have influenced the behavior of visitors), the alternative (attempting to randomly add spurious but untraveled links to each page) seemed questionable.

Generating good relational structure at each site was straightforward. At our home institution, for example, our model includes CourseOccurrence(Course, Term) pages for the main page of each term's offering of a course, Assignment(Course, Term, Assignment) pages for each problem set assigned, *etc.* Content on `www.gazelle.com`, like at many large web sites, was generated dynamically by combining queries over a database with HTML templates to produce pages. The challenge, however, was in inferring the schemata of pages—the set of allowable templates and the parameters that they each required—without having access to the live web site. Fortunately, the KDDCup log data encodes a comprehensive set of parameters as part of each request, and most of these parameters have an obvious intuitive meaning (page template, product identifiers, *etc.*). We removed records for all but the nine most frequently accessed page templates and for templates whose arguments are not present in the clickstream data (*e.g.*, search results pages); this set of nine templates was our initial candidate for the relation set $\mathcal{R}$. The next challenge was determining the arguments to each relation. By analyzing the frequency of non-null parameter values, it became clear that some of the templates took optional arguments. Because our framework requires relations to have constant arity, we "split" such a relation into two or more relations, one for each non-null argument pattern. This process yielded 16 distinct relations in $\mathcal{R}$. Finally, for the hierarchies over the parameter values, we used the trees defined for those parameters in the KDDCup data. Appendix A provides the detailed relational models for both sites.

In the following experiments, we compared PMMs with three RMM variants: RMM-uniform, RMM-rank, and RMM-PET. We employed Laplace smoothing [14] in the PMM and in RMM-PET. For RMM-rank we set the k parameter at 10.0, a value which had produced good results on the training data. We compute shrinkage weights in RMM-PET using EM. For each data set, we trained the models with varying numbers of examples, and we recorded the average negative log-likelihood of a test example. A negative log-likelihood score is the number of bits needed to encode an average test instance given the model; a perfect model would have a score of zero.

Our first experiment, which uses KDDCup data from `www.gazelle.com`, shows the substantial advantage that RMMs can have over PMMs (see Figure 7). With only 10 training examples, the RMMs perform significantly better than PMMs (the difference is significant at the 95% confidence level). As the amount of training data increases, all models improve their prediction, but RMM-uniform and RMM-PET consistently outperform the PMM. However, as the models are given more training data, their relative differences diminish. This result is to be expected: with suitable volumes of training data, RMM-rank and RMM-PET both converge to a PMM. RMM-uniform's consistent good performance suggests that all levels of abstraction in this site are predicting state transitions well.

Our second experiment uses log data from November 2001 at our home institution. When trained with successively more data, RMM-rank and RMM-PET showed a slight improvement over PMMs, but only when trained on up to 10,000 examples. Because the UW CSE education pages form a small site, it is a very data-rich environment, and we were pleased that RMMs were not trumped by PMMs.

Our third experiment also uses data from our home institution; it represents traffic to the pages of a single course, CSE 142 "Computer Programming I," over a full year. Here, we trained the models on data from the instances of CSE 142 in Winter, Spring, Summer, and Fall 2001, and tested the models on data from the instance in Winter 2002. Note that the instructors (and course webmasters) were different in the various instances; indeed, none of the test

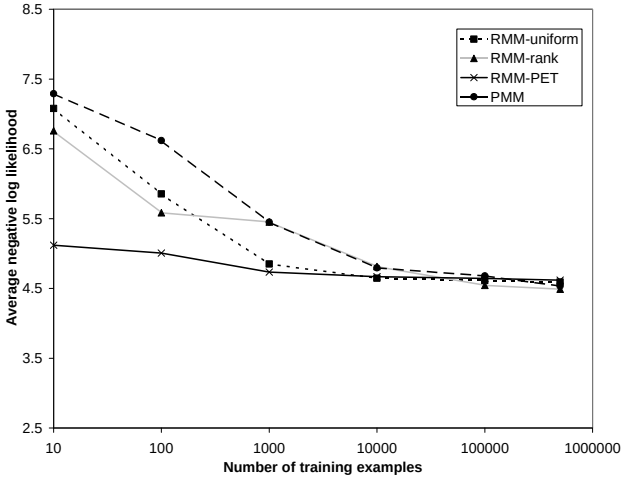

Figure 7: KDDCup 2000 data (`www.gazelle.com`). The x-axis shows the number of training instances scaled logarithmically, and the y-axis is the average negative log-likelihood of a test example. Curves are based on 2000 test instances. RMMs outperform PMMs with as few as ten training examples.

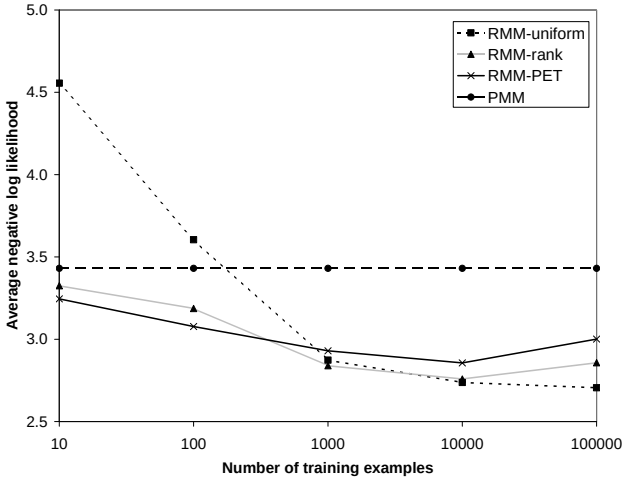

Figure 8: Winter 2002 data from UW CSE course 142. Pages in the test set (Winter 2002) did not exist during the training period (Winter 2001 - Fall 2001).

pages even existed at the time that the training data was collected. As a result, the PMM can do nothing better than predict a uniform distribution over the links on each page. In contrast, an RMM takes advantage of the related common relational structure of the training and test data to significantly improve prediction (see Figure 8).

The computation time required for the RMM variants is not substantially more than that for PMMs. The RMM variants require some preprocessing of the data, to build the abstraction sets, but this work can be done at learning time, independent of the test set. In this third experiment, for example, preprocessing the site (containing 3,749 pages) for RMM-rank and RMM-uniform took four minutes (our RMM code is implemented in C++ and the experiments were run on an 850MHz Pentium III)[2]. Inference in a PMM for a test example requires only a single ratio of counts, while a more complex set of counts must be shrunk together in the RMM variants. However, we found that, on average, RMM-rank, RMM-uniform, and the PMM method all required the same amount of time (about 430 milliseconds) per example for inference. The added cost of the RMM is hidden largely because the work for one test instance may be cached and applied to another (*e.g.*, two instances with a common source state). RMM-PET requires a different preprocessing (to learn the PET) which took 76 seconds and completed the prediction runs in 1960 milliseconds.

Finally, we have combined RMMs with our MINPATH algorithm to evaluate their use for adaptive web navigation. In preliminary testing we compared three models: RMM-uniform, RMM-rank, and a PMM. We found that the RMMs performed significantly better than the PMM, allowing MINPATH to save users more links—often 50% to 80% more links than with the PMM, particularly when training data was sparse. In future work we will more exhaustively compare the RMM variants and PMMs as used in MINPATH.

In summary, we conclude that RMMs significantly outperform PMMs when data is sparse and perform comparably when data is rich. Computation time for RMMs is competitive with PMMs, particularly when the training data can be preprocessed. The RMM-PET technique appears to be the best one for computing the mixing coefficients, with RMM-rank also performing favorably.

[2]Our implementation calculates the non-leaf abstraction transition probabilities directly from the data, and not from lower-level abstractions as we suggested in Section 2.3. Thus, our computation could be improved substantially.

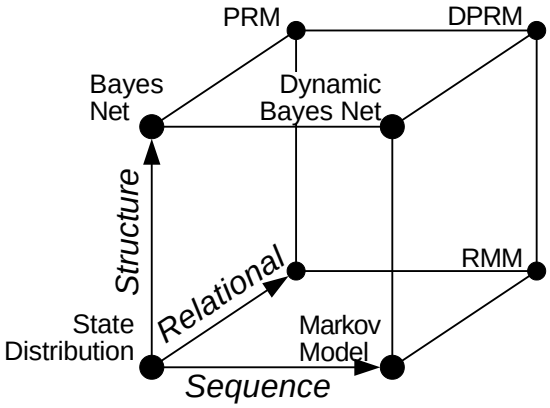

Figure 9: Probabilistic models.

5. RELATED WORK

There are two families of related work that we discuss in turn: techniques for learning probabilistic models, and algorithms for web adaptation and personalization.

5.1 Learning Probabilistic Models

Considerable work has been performed on a variety of different probabilistic models; we illustrate this space in Figure 9. The lower left corner represents a simple model containing a number of states of varying probability. Moving rightward adds *sequence* information and leads to a Markov model. Moving upwards adds *structure* by which we mean the notion of defining the states in terms of variables and representing the joint probability distribution compactly with explicit conditional independence assumptions. Moving back-

wards into the page adds *relational* information — a set of predicates and a domain of variables for each argument.

Viewed in this context, the connection between RMMs and other first-order probabilistic representations becomes clearer. Friedman *et al.* [10] extended the notion of Bayesian network to propose probabilistic relational models (PRMs). Objects in a PRM are divided into a set of classes, and a different probabilistic model is built for each class, specifying how its attributes depend on each other and on attributes of related classes. Dynamic Bayesian networks (DBNs) [6, 7, 29] form a probabilistic dependency graph for uncertain temporal reasoning. A DBN has a separate Bayesian network for each time step, in which the values of variables for time t can depend on the values of variables in previous time slices. Thus, DBNs "improve" on RMMs in their use of explicit conditional independences amongst a set of variables, but in contrast to an RMM every state in a DBN is treated the same way — it has the same variables and dependencies. To our knowledge, RMMs are the first probabilistic first-order model of sequential processes to be proposed. However, it is interesting to note that dynamic Bayesian networks can be viewed as a special form of PRM where there is only one class (the state) and the only relation is the sequential order between successive states. PRMs have been extended to allow the class to be chosen from a hierarchy [12]. RMMs allow hierarchies over the attributes in each class, and combining models at all levels using shrinkage. This approach should be useful in PRMs also. One obvious area for future work is to combine ideas from RMMs, DBNs, and PRMs to define "Dynamic probabilistic relational models" (DPRMs).

Hidden Markov models have been extended in a number of ways to accommodate richer state and observation information. For example, factorial hidden Markov models [13] decompose model states into k components, described by k state variables, that depend on each other only via the observation variable. A factorial hidden Markov model can be viewed as a form of RMM with hidden state, in which all states belong to the same k-ary relation, but which has a conditional independence assumption that state variables in subsequent states depend only on the corresponding variables in the previous state. An area of future work is in exploring how these conditional independences can be leveraged by relational Markov models. Other extensions of HMMs have been proposed (*e.g.*, Lafferty *et al.* [17]). It should be possible to subsume these within our framework; this is a matter for future research. RMMs are also related to work on abstraction in reinforcement learning (*e.g.*, Dietterich [8], Dzeroski *et al.* [9]), and may be useful in that field.

5.2 Adaptive Web Navigation

Since Perkowitz and Etzioni challenged the research community to build adaptive web sites [24], many projects have addressed components of this task. In this section we highlight the subset of work related specifically to adaptive web navigation.

Our MINPATH system [2] processes server access logs offline in order to learn a model of web navigation patterns (similar to how WebCANVAS [4] builds visitor clusters for visualization). At run-time MINPATH combines the probabilistic estimates from this model with distance information to compute *expected savings* of shortcuts, adding the links it deems most useful. In our earlier work, we evaluated a variety of visitor models, including Naïve Bayes mixture models and mixtures of Markov models, concluding that a mixture of Markov models performed best for the task. In this paper, we argue that RMMs can perform substantially better for this same task.

Perkowitz and Etzioni [25] also address the shortcut problem, but they use a simpler prediction method: for each pair of pages P, Q on the site, their system records how often Q is viewed by following some chain of links via P. When page P is requested after these statistics have been computed, the system adds the top m most-requested Q pages as shortcuts. This method doesn't make the independence assumptions of a first-order Markov model, but probably requires more user data in order to make predictions. Like the traditional Markov approaches, Perkowitz and Etzioni's system can't predict good shortcuts for pages which weren't visited in the training data.

In addition to their work on the shortcut problem, Perkowitz and Etzioni developed IndexFinder [25], which uses page meta-data to cluster web pages into conceptually similar groups, and subsequently builds coherent index or hub pages of links. The meta-data is similar in spirit to the values used to instantiate RMM relations, although IndexFinder does not segregate pages into relations, or predict navigation.

Fu *et al.*'s SurfLen [11] mines web logs for association rules, suggesting the top m pages that are most likely to co-occur with the visitor's current session; the learning method is a form of "market basket" analysis [1].

Lieberman's Letizia [18] is a client-side agent that browses the web in tandem with the user. Based on the user's actions (*e.g.*, which links were followed, whether pages were added to a bookmarks file, *etc.*), Letizia estimates the visitor's interest in as-yet-unseen pages. Information retrieval measures of page similarity and guiding queries have been quite successful at predicting navigation patterns. WebWatcher [15] and adaptive web site agents [23] use machine learning to predict the next link a user will follow — a simplified version of the shortcut problem. Sarukkai [28] uses a Markov model of web usage to suggest the most probable links a visitor may follow, and notes the need to reduce the size of the model by clustering the URLs. Space precludes discussion of all related work on sequence prediction and web usage mining.

The goal of the WebKB project [5] is to populate a relational knowledge base given the textual content and hyperlink connectivity of Web pages. This goal is different from that of RMMs — RMMs presume the existence of a relational model and predict transitions using the model. However, it would be interesting to apply the WebKB learning approach to populate a model *describing a web site* and use RMMs to predict navigation in that model. Although most e-commerce sites are dynamically generated from database queries, many other large sites (*e.g.*, corporate intranets or academic institution web sites) exist only as large collections of static web pages. The WebKB approach could prove fruitful for producing the relational information RMMs need for such static sites.

Finally, much research has been done in recent years on classifying web pages (*e.g.*, Pazzani *et al.* [22], McCallum, *et al.* [19]). Any web page classifier that yields class probabilities can in principle be used in place of RMMs for adaptive web navigation. However, many of these classifiers are based on viewing web pages as bags of words, and are unable to take advantage of the relational structure of the site. Incorporating bag-of-words information into RMMs may be useful and is a direction for future work.

6. CONCLUSIONS

This paper introduces relational Markov models (RMMs), a generalization of Markov models that represents states with relational predicates and leverages this information for better learning and inference. We believe that RMMs are applicable to a wide variety of domains besides Web navigation, such as mobile robotics, speech processing, process control, fault diagnosis, and computational bi-

ology. This paper makes the following contributions:

- We provide a precise definition of relational Markov models and describe how to estimate state transition behavior using shrinkage between abstractions of the states.
- We explain how RMMs can be used for adaptive web navigation, and present experiments demonstrating substantial advantages over traditional Markov models.
- We compare several variations of RMMs and found that using PETs to select mixture weights performed the best, followed by our RMM-rank approach.

Our experiments have shown that relational Markov models are a suitable alternative to traditional Markov models—RMMs infrequently perform worse, and can perform much better. When data about all states is available in quantity, or if the relations between states are not reflected in the distribution of the data, RMMs offer no advantage relative to traditional Markov models, and, in fact, could perform worse than PMMs. However, when data is scarce or non-existent about some states, but abundant for conceptually similar states (based on relational abstractions of the states), relational Markov models significantly outperform traditional Markov models. Intuition suggests that this latter case holds true for the vast majority of web sites, and that RMMs should prove widely useful.

In future research we plan to both extend relational Markov models and explore additional applications. An immediate area to pursue is in further developing the RMM-PET approach, by building PETs to predict finer partitions among the destination abstractions. An interesting evolution of RMMs is to relational hidden Markov models, where both the states and the observations are described by typed relations and shrinkage is carried out over their respective abstractions. Another direction is incorporating a model cluster identity into the transition probability, such as the identity of a cluster of visitors at a web site, and shrinking between many models learned for different sizes and scope of user cluster (such as a single user, a cluster of similar users, and the set of all users at the site). A third path of research is to apply RMMs to other domains, such as mobile robot localization or speech recognition.

Acknowledgments

The authors thank Cathy Anderson, AnHai Doan, Oren Etzioni, Geoff Hulten, Zack Ives, Cody Kwok for their insightful comments on this work, and to Blue Martini for providing the `gazelle.com` data. This work was funded in part by NSF grants #IIS-9872128 and #IIS-9874759, an NSF CAREER and IBM Faculty awards to the second author, and a gift from the Ford Motor Company.

7. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 207–216, 1993.

[2] C. R. Anderson, P. Domingos, and D. S. Weld. Adaptive web navigation for wireless devices. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001.

[3] C. R. Anderson, P. Domingos, and D. S. Weld. Personalizing web sites for mobile users. In *Proceedings of the Tenth International World Wide Web Conference*, 2001.

[4] I. V. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Visualization of navigation patterns on a web site using model based clustering. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining*, 2000.

[5] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence Journal*, 118(1–2):69–113, 2000.

[6] T. Dean and K. Kanazawa. Probabilistic Temporal Reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.

[7] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150, 1989.

[8] T. G. Dietterich. State abstraction in MAXQ hierarchical reinforcement learning. In S. A. Solla, T. K. Leen, and K.-R. Muller, editors, *Advances in Neural Information Processing Systems 12*, pages 994–1000. MIT Press, Cambridge, MA, 2000.

[9] S. Dzeroski and L. de Raedt. Relational reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 136–143, Madison, WI, 1998. Morgan Kaufmann.

[10] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1300–1307, Stockholm, Sweden, 1999. Morgan Kaufmann.

[11] X. Fu, J. Budzik, and K. J. Hammond. Mining navigation history for recommendation. In *Proceedings of the 2000 Conference on Intelligent User Interfaces*, 2000.

[12] L. Getoor, D. Koller, and N. Friedman. From instances to classes in probabilistic relational models. In *Proceedings of the ICML-2000 Workshop on Attribute-Value and Relational Learning*, Stanford, CA, 2000.

[13] Z. Ghahramani and M. Jordan. Factorial hidden Markov models. *Machine Learning*, 29:245–273, 1997.

[14] I. J. Good. *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press, Cambridge, MA, 1965.

[15] T. Joachims, D. Freitag, and T. Mitchell. WebWatcher: A tour guide for the World Wide Web. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[16] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. KDD-Cup 2000 organizers' report: Peeling the onion. *SIGKDD Explorations*, 2(2):86–98, 2000. http://www.ecn.purdue.edu/KDDCUP.

[17] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, Williamstown, MA, 2001. Morgan Kaufmann.

[18] H. Lieberman. Letizia: An agent that assists web browsing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995.

[19] A. McCallum, R. Rosenfeld, T. Mitchel, and A. Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[20] S. Muggleton. Stochastic logic programs. In L. de Raedt, editor, *Advances in Inductive Logic Programming*, pages

254–264. IOS Press, Amsterdam, The Netherlands, 1996.

[21] L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–177, 1997.

[22] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill & Webert: Identifying interesting Web sites. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[23] M. J. Pazzani and D. Billsus. Adaptive web site agents. In *Proceedings of the Third International Conference on Autonomous Agents*, 1999.

[24] M. Perkowitz and O. Etzioni. Adaptive web sites: an AI challenge. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[25] M. Perkowitz and O. Etzioni. Towards adaptive web sites: Conceptual framework and case study. *Artificial Intelligence Journal*, 118(1–2), 2000.

[26] F. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*. To appear.

[27] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, 1989.

[28] R. R. Sarukkai. Link prediction and path analysis using Markov chains. In *Proceedings of the Ninth International World Wide Web Conference*, 2000.

[29] P. Smyth. Clustering sequences with hidden Markov models. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing 9*, 1996.

[30] P. Smyth, D. Heckerman, and M. I. Jordan. Probabilistic independence networks for hidden Markov probability models. *Neural Computation*, 9:227–269, 1997.

APPENDIX

A. RELATIONAL SCHEMATA FOR EVALUATION SITES

A.1 www.gazelle.com

The relations for `www.gazelle.com` take up to three parameters: Assortment, Product, and Collection. The domain hierarchies for these parameters are described explicitly in the KDDCup 2000 data.

- Home()
- Boutique()
- Departments()
- Legcare_vendor()
- Lifestyles()
- Vendor()
- AssortmentDefault()
- Assortment(Assortment)
- ProductDetailLegcareDefault()
- ProductDetailLegcare(Product)
- ProductDetailLegwearDefault()
- ProductDetailLegwearProduct(Product)
- ProductDetailLegwearAssortment(Assortment)
- ProductDetailLegwearProdCollect(Product, Collection)
- ProductDetailLegwearProdAssort(Product, Assortment)
- ProductDetailLegwear(Product, Collection, Assortment)

A.2 www.cs.washington.edu

The structure for `www.cs.washington.edu/education/courses/` was derived by reverse-engineering the structure of the existing site. The Term and Course domain hierarchies each contain a root node, a level of interior nodes (grouping courses by undergraduate, graduate, *etc.* and grouping terms by the academic year), and the ground leaf values. URL variables are URLs relative to the particular CourseSite(Course) or CourseOccurence(Course, Term) to which they apply. The domain hierarchies for URL and most other variables are flat, comprising only of the root node and many leaf values.

- CourseWebs()
- CourseSite(Course)
- CourseSiteOther(Course, URL)
- CourseOccurence(Course, Term)
- CourseOccurenceOther(Course, Term, URL)
- CourseSampleCode(Course, Term, URL)
- Administrivia(Course, Term, URL)
- AllCoursework(Course, Term)
- CourseworkGeneralOther(Course, Term, URL)
- Coursework(Course, Term, Number)
- CourseworkCode(Course, Term, Number)
- CourseworkOther(Course, Term, Number, URL)
- Turnin(Course, Term, Number)
- AllExams(Course, Term)
- Exam(Course, Term, URL)
- AllLectures(Course, Term)
- LectureOtherGeneral(Course, Term, URL)
- Lecture(Course, Term, Number)
- LectureOther(Course, Term, Number, URL)
- MailIndex(Course, Term, SortBy)
- MailMessage(Course, Term, Number)
- Section(Course, Term, Section)
- SectionOther(Course, Term, Section, URL)