Journal of Machine Learning Research 1 (2000) 49-75 Submitted 5/00; Published 10/00

Dependency Networks for Inference, Collaborative Filtering, and Data Visualization

David Heckerman HECKERMA@MICROSOFT.COM
David Maxwell Chickering DMAX@MICROSOFT.COM
Christopher Meek MEEK@MICROSOFT.COM
Robert Rounthwaite ROBERTRO@MICROSOFT.COM
Carl Kadie CARLK@MICROSOFT.COM
Microsoft Research, One Microsoft Way, Redmond, WA 98052 USA

Editor: Leslie Pack Kaelbling

Abstract

We describe a graphical model for probabilistic relationships—an alternative to the Bayesian network—called a dependency network. The graph of a dependency network, unlike a Bayesian network, is potentially cyclic. The probability component of a dependency network, like a Bayesian network, is a set of conditional distributions, one for each node given its parents. We identify several basic properties of this representation and describe a computationally efficient procedure for learning the graph and probability components from data. We describe the application of this representation to probabilistic inference, collaborative filtering (the task of predicting preferences), and the visualization of acausal predictive relationships.

Keywords: Dependency networks, Bayesian networks, graphical models, probabilistic inference, data visualization, exploratory data analysis, collaborative filtering, Gibbs sampling

1. Introduction

The Bayesian network has proven to be a valuable tool for encoding, learning, and reasoning about probabilistic relationships. In this paper, we introduce another graphical representation of such relationships called a *dependency network*. The representation can be thought of as a collection of regressions or classifications among variables in a domain that can be combined using the machinery of Gibbs sampling to define a joint distribution for that domain. The dependency network has several advantages and disadvantages with respect to the Bayesian network. For example, a dependency network is not useful for encoding causal relationships and is difficult to construct using a knowledge-based approach. Nonetheless, there are straightforward and computationally efficient algorithms for learning both the structure and probabilities of a dependency network from data; and the learned model is quite useful for encoding and displaying predictive (i.e., dependence and independence) relationships. In addition, dependency networks are well suited to the task of predicting preferences—a task often referred to as *collaborative filtering*—and are generally useful for *probabilistic inference*, the task of answering probabilistic queries.

In Section 2, we motivate dependency networks from the perspective of data visualization and introduce a special case of the graphical representation called a consistent dependency network. We show, roughly speaking, that such a network is equivalent to a Markov network, and describe how Gibbs sampling is used to answer probabilistic queries given a consistent dependency network. In Section 3, we introduce the dependency network in its general form and describe an algorithm for learning its structure and probabilities from data. Essentially, the algorithm consists of independently performing a probabilistic classification or regression for each variable in the domain. We then show how procedures closely resembling Gibbs sampling can be applied to the dependency network to define a joint distribution for the domain and to answer probabilistic queries. In addition, we provide experimental results on real data that illustrate the utility of this approach, and discuss related work. In Section 4, we describe the task of collaborative filtering and present an empirical study showing that dependency networks are almost as accurate as and computationally more attractive than Bayesian networks on this task. Finally, in Section 5, we describe a data visualization tool based on dependency networks.

2. Consistent Dependency Networks

For several years, we used Bayesian networks to help individuals visualize predictive relationships learned from data. When using this representation in problem domains ranging from web-traffic analysis to collaborative filtering, these individuals expressed a single, common criticism. We developed dependency networks in response to this criticism. In this section, we introduce a special case of the dependency-network representation and show how it addresses this complaint.

Consider Figure 1a, which contains a portion of a Bayesian-network structure describing the demographic characteristics of visitors to a web site. We have found that, when shown graphs like this one and told they represent causal relationships, an untrained person often gains an accurate impression of the relationships. In many situations, however, a causal interpretation of the graph is suspect—for example, when one uses a computationally efficient learning procedure that excludes the possibility of hidden variables. In these situations, the person only can be told that the relationships are "predictive" or "correlational." In these cases, we have found that the Bayesian network becomes confusing. For example, untrained individuals who look at Figure 1a will correctly conclude that Age and Gender are predictive of Income, but will wonder why there are no arcs from Income to Age and to Gender—after all, Income is predictive of Age and Gender. Furthermore, these individuals will typically be surprised to learn that Age and Gender are dependent given Income.

Of course, people can be trained to appreciate the (in)dependence semantics of a Bayesian network, but often they lose interest in the problem before gaining an adequate understanding; and, in almost all cases, the mental activity of computing the dependencies interferes with the process of gaining insights from the data.

To avoid this difficulty, we can replace the Bayesian-network structure with one where the parents of each variable correspond to its Markov blanket—that is, a structure where the parents of each variable render that variable independent of all other variables. For example, the Bayesian-network structure of Figure 1a becomes that of Figure 1b. Equally important, we do not change the feature of Bayesian networks wherein the conditional probability of a variable given its parents is used to quantify the dependencies. In our experience,

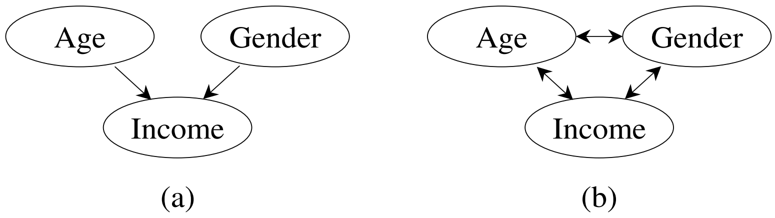

Figure 1: (a) A portion of a Bayesian-network structure describing the demographic characteristics of users of a web site. (b) The corresponding consistent dependency-network structure.

individuals are quite comfortable with this feature. Roughly speaking, the resulting model is a dependency network.

2.1 Definition and Basic Properties

We now describe dependency networks more formally. To do so, we begin with some notation. We denote a variable by a capitalized token (e.g., X, X_i, Θ, Age), and the state or value of a corresponding variable by that same token in lower case (e.g., x, x_i, θ, age). We denote a set of variables by a bold-face capitalized token (e.g., $\mathbf{X}, \mathbf{X}_i, \mathbf{Pa}_i$). We use a corresponding bold-face lower-case token (e.g., $\mathbf{x}, \mathbf{x}_i, \mathbf{pa}_i$) to denote an assignment of state or value to each variable in a given set. We use $p(x|y)$ to denote the probability that $X = x$ given $Y = y$. We also use $p(x|y)$ to denote the probability distribution for X given Y. Whether $p(x|y)$ refers to a probability or a probability distribution will be clear from context. In this paper, we shall limit our discussion to domains where all variables are discrete and finite valued and where the joint distribution is positive—that is, where all assignments of the domain variables have a non-zero probability. Although much of what we develop can be extended to more general circumstances, the extensions are tedious and we omit them.

Given a domain of interest having a set of finite variables $\mathbf{X} = (X_1, \ldots, X_n)$ with a positive joint distribution $p(\mathbf{x})$, a *consistent dependency network for* $\mathbf{X}$ is a pair $(\mathcal{G}, \mathcal{P})$ where $\mathcal{G}$ is a (cyclic) directed graph and $\mathcal{P}$ is a set of conditional probability distributions. Each node in $\mathcal{G}$ corresponds to a variable in $\mathbf{X}$. We use X_i to refer to both the variable and its corresponding node. The parents of node X_i, denoted $\mathbf{Pa}_i$, correspond to those variables $\mathbf{Pa}_i \subseteq (X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n)$ that satisfy

$$p(x_i|\mathbf{pa}_i) = p(x_i|x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = p(x_i|\mathbf{x} \setminus x_i). \tag{1}$$

The distributions in $\mathcal{P}$ are the *local probability distributions* $p(x_i|\mathbf{pa}_i), i = 1, \ldots, n$. The dependency network is consistent in the sense that each local distribution can be obtained (via inference) from the joint distribution $p(\mathbf{x})$. In the next section, we relax this condition.

The independencies in a dependency network are precisely those of a Markov network with the same adjacencies. A Markov network for $\mathbf{X}$, also known as an undirected graphical model or Markov random field for $\mathbf{X}$, is a pair $(\mathcal{U}, \Phi)$ where $\mathcal{U}$ is an undirected graph and $\Phi = (\phi_1, \ldots, \phi_c)$ is a set of potential functions, one for each of the c maximal cliques in $\mathcal{U}$, such that joint distribution has the form

$$p(\mathbf{x}) = \prod_{i=1}^{c} \phi_i(\mathbf{x}^i), \tag{2}$$

where $\mathbf{X}^i$ are the variables in clique i, $i = 1, \ldots, c$ (e.g., see Lauritzen, 1996). The following theorem shows that consistent dependency networks and Markov networks have the same representational power.

Theorem 1: The set of positive distributions that can be encoded by a consistent dependency network with graph $\mathcal{G}$ is equal to the set of positive distributions that can be encoded by a Markov network whose structure has the same adjacencies as $\mathcal{G}$.

The two graphical representations are different in that Markov networks quantify dependencies with potential functions, whereas dependency networks use conditional probabilities. We have found the latter to be significantly easier to interpret.

The proof of Theorem 1 appears in the Appendix, but it is essentially a restatement of the Hammersley-Clifford theorem (e.g., Besag, 1974). This correspondence is no coincidence. As is discussed in Besag (1974), several researchers who developed the Markov-network representation did so by initially investigating a graphical representation that fits our definition of consistent dependency network. In particular, several researchers including Lévy (1948), Bartlett (1955, Section 2.2), and Brook (1964) considered lattice systems where each variable X_i depended only on its nearest neighbors $\mathbf{Pa}_i$, and quantified the dependencies within these systems using the conditional probability distributions $p(x_i|\mathbf{pa}_i)$. They then showed, to various levels of generality, that the only joint distributions mutually consistent with each of the conditional distributions also satisfy Equation 2. Hammersley and Clifford, in a never published manuscript, and Besag (1974) considered the more general case where each variable could have an arbitrary set of parents. They showed that, provided the joint distribution for $\mathbf{X}$ is positive, any graphical model specifying the independencies in Equation 1 must also satisfy Equation 2. One interesting point is that these researchers argued for the use of conditional distributions to quantify the dependencies. They considered the resulting potential form in Equation 2 to be a mathematical necessity rather than a natural expression of dependency. As we have just discussed, we share this view.

The equivalence of consistent dependency networks and Markov networks suggests a straightforward approach for learning a consistent dependency network from exchangeable (i.i.d.) data. Namely, one learns the structure and potentials of a Markov network (e.g., Whittaker, 1990), and then computes (via probabilistic inference) the conditional distributions required by the dependency network. Alternatively, one can learn a related model such as a Bayesian network, decomposable model, or hierarchical log-linear model (see, e.g., Lauritzen, 1996) and convert it to a consistent dependency network. Unfortunately, the conversion process can be computationally expensive in many situations. In the next section, we extend the definition of dependency network to include inconsistent dependency networks and provide algorithms for learning such networks that are more computationally efficient than those just described. In the remainder of this section, we apply well known results about probabilistic inference to consistent dependency networks. This discussion will be useful for our further development of (general) dependency networks.

2.2 Probabilistic Inference

Given a graphical model for $\mathbf{X}$, we often wish to answer probabilistic queries of the form $p(\mathbf{y}|\mathbf{z})$, where $\mathbf{Y}$ (the "target" variables) and $\mathbf{Z}$ (the "input" variables) are disjoint subsets of $\mathbf{X}$. This task is known in the graphical modeling community as *probabilistic inference.* An important special case of probabilistic inference is the determination of $p(\mathbf{x})$ for given instances $\mathbf{x}$ of $\mathbf{X}$—a key component of *density estimation*, which has numerous applications.

Given a consistent dependency network for $\mathbf{X}$, we can perform probabilistic inference by converting it to a Markov network, triangulating that network (forming a decomposable graphical model), and then applying one of the standard algorithms for probabilistic inference in the latter representation—for example, the junction tree algorithm of Jensen, Lauritzen, and Olesen (1990). Alternatively, we can use Gibbs sampling (e.g., Geman and Geman, 1984; Neal, 1993; Besag, Green, Higdon, and Mengersen, 1995; Gilks, Richardson, and Spiegelhalter, 1996), which we examine in some detail.

First, let us consider the use of Gibbs sampling for recovering the joint distribution $p(\mathbf{x})$ of a consistent dependency network for $\mathbf{X}$. In one simple version of Gibbs sampling, we initialize each variable to some arbitrary value. We then repeatedly cycle through each variable $X_1, \ldots, X_n$, in this order, and resample each X_i according to $p(x_i|\mathbf{x}\backslash x_i) = p(x_i|\mathbf{pa}_i)$. We call this procedure an *ordered Gibbs sampler.* As described by the following theorem, this ordered Gibbs sampler recovers the joint distribution for $\mathbf{X}$.

Theorem 2: An ordered Gibbs sampler applied to a consistent dependency network for $\mathbf{X}$, where each X_i is finite (and hence discrete) and each local distribution $p(x_i|\mathbf{pa}_i)$ is positive, defines a Markov chain with a unique stationary joint distribution for $\mathbf{X}$ equal to $p(\mathbf{X})$ that can be reached from any initial state of the chain.

Proof: Let $\mathbf{x}^t$ be the sample of $\mathbf{x}$ after the t^{th} iteration of the ordered Gibbs sampler. The sequence $\mathbf{x}^1, \mathbf{x}^2, \ldots$ can be viewed as samples drawn from a homogenous Markov chain with transition matrix $\mathbf{P}$ having elements $\mathbf{P}_{ij} = p(\mathbf{x}^{t+1} = j|\mathbf{x}^t = i)$. The matrix $\mathbf{P}$ is the product $\mathbf{P}^1 \cdot \mathbf{P}^2 \cdot \ldots \cdot \mathbf{P}^n$, where $\mathbf{P}^k$ is the "local" transition matrix describing the resampling of X_k according to the local distribution $p(x_k|\mathbf{pa}_k)$. The positivity of local distributions guarantees the positivity of $\mathbf{P}$, which in turn guarantees the irreducibility of the Markov chain. Consequently, there exists a unique joint distribution that is stationary with respect to $\mathbf{P}$. The positivity of $\mathbf{P}$ also guarantees that this stationary distribution can be reached from any starting point. That this stationary distribution is equal to $p(\mathbf{x})$ is proved in the Appendix. $\Box$

After a sufficient number of iterations, the samples in the chain will be drawn from the stationary distribution for $\mathbf{X}$. We use these samples to estimate $p(\mathbf{X})$. There is much written about how long to run the chain before keeping samples (the "burn in") and how many samples to use to obtain a good estimate (e.g., Gilks, Richardson, and Spiegelhalter, 1996). We do not discuss the details here.

Next, let us consider the use of Gibbs sampling to compute $p(\mathbf{y}|\mathbf{z})$ for particular instances $\mathbf{y}$ and $\mathbf{z}$ where $\mathbf{Y}$ and $\mathbf{Z}$ are arbitrary disjoint subsets of $\mathbf{X}$. A naive approach uses an ordered

Gibbs sampler directly. During the iterations, only samples of $\mathbf{X}$ where $\mathbf{Z} = \mathbf{z}$ are used to compute the conditional probability. An important difficulty with this approach is that if either $p(\mathbf{y}|\mathbf{z})$ or $p(\mathbf{z})$ is small (a common occurrence when $\mathbf{Y}$ or $\mathbf{Z}$ contain many variables), many iterations are required for an accurate probability estimate.

A well-known approach for estimating $p(\mathbf{y}|\mathbf{z})$ when $p(\mathbf{z})$ is small is to fix $\mathbf{Z} = \mathbf{z}$ during ordered Gibbs sampling. It is not difficult to generalize the proof of Theorem 2 to show that this *modified ordered Gibbs sampler* has a stationary distribution and yields the correct conditional probability given a consistent dependency network.

When $\mathbf{y}$ is rare because $\mathbf{Y}$ contains many variables, we can use the independencies encoded in a dependency network along with the law of total probability to decompose the inference task into a set of inference tasks on single variables. For example, consider the dependency network $[X_1 \quad X_2 \leftrightarrow X_3]$. Given the independencies specified by this network, we have

$$p(x_1, x_2, x_3) = p(x_1)\ p(x_2)\ p(x_3|x_2),$$

and can obtain $p(x_1, x_2, x_3)$ by computing each term separately. The determination of the first term requires no Gibbs sampling—the distribution can be read directly from the local distribution for X_1 in the dependency network. The second two terms can be determined using modified ordered Gibbs samplers each with a singleton target (x_2 and x_3, respectively). Note that this approach has the additional advantage that some terms may be obtained by direct lookup, thereby avoiding some Gibbs sampling.

In general, given a consistent dependency network for $\mathbf{X}$ and disjoint sets of variables $\mathbf{Y} \subseteq \mathbf{X}$ and $\mathbf{Z} \subseteq \mathbf{X}$, we can obtain $p(\mathbf{y}|\mathbf{z})$ for a particular instance of $\mathbf{y}$ and $\mathbf{z}$ as follows:

Algorithm 1:

```
1   U := Y   (* the unprocessed variables *)
2   P := Z   (* the processed and conditioning variables *)
3   p := z   (* the values for P *)
4   While U ≠ ∅
5       Choose X_i ∈ U such that X_i has no more parents in U than any variable in U
6       If all the parents of X are in P
7           p(x_i|p) := p(x_i|pa_i)
8       Else
9           Use a modified ordered Gibbs sampler to determine p(x_i|p)
10      U := U − X_i
11      P := P + X_i
12      p := p + x_i
13  Return the product of the conditionals p(x_i|p)
```

The key step in this algorithm is step 7, which bypasses Gibbs sampling when it is not needed. This step is justified by Equation 1.

3. General Dependency Networks

As we have discussed, consistent dependency networks and Markov networks are interchangeable representations: given one, we can compute the other. In this section, we consider a more general class of dependency networks and part company with Markov networks.

Our extension of consistent dependency networks is motivated by computational concerns. For domains having a large number of variables and many dependencies among those variables, it is computationally expensive to learn a Markov network from data and then convert that network to a consistent dependency network. As an alternative, we start with the observation that the local distribution for variable X_i in a dependency network is the conditional distribution $p(x_i|\mathbf{x} \setminus x_i)$, which can be estimated by any number of probabilistic classification techniques (or regression techniques, if we were to consider continuous variables) such as methods using a probabilistic decision tree (e.g., Buntine, 1991), a generalized linear model (e.g., McCullagh and Nelder, 1989), a neural network (e.g., Bishop, 1995), a probabilistic support-vector machine (e.g., Platt, 1999), or an embedded regression/classification model (Heckerman and Meek, 1997). This observation suggests a simple, heuristic approach for learning the structure and probabilities of a dependency network from exchangeable (i.i.d.) data. Namely, for each variable X_i in domain $\mathbf{X}$, we *independently* estimate its local distribution from data using a classification algorithm. Once we have all estimates for the local distributions, we then construct the structure of the dependency network from the (in)dependencies encoded in these estimates.

For example, suppose we wish to construct a dependency network for the domain $\mathbf{X} = (X_1, X_2, X_3)$. To do so, we need to estimate three conditional probability distributions: $p(x_1|x_2, x_3)$, $p(x_2|x_1, x_3)$, and $p(x_3|x_1, x_2)$. First, we use prior knowledge to decide how each distribution is to be modeled. We may decide, for example, that a logistic regression, a multilayer neural net, and a probabilistic decision tree are appropriate models for $p(x_1|x_2, x_3)$, $p(x_2|x_1, x_3)$, and $p(x_3|x_1, x_2)$, respectively. (There is no requirement that each distribution be chosen from the same model class.) In addition, for each estimation, we may choose to use a feature-selection algorithm that can discard some of the inputs. Next, we apply the estimation/learning procedures. For the sake of illustration, suppose we discover that X_1 is not a significant predictor of X_3 and that X_3 is not a significant predictor of X_1. Finally, we construct the dependency network structure—in this case, $X_1 \leftrightarrow X_2 \leftrightarrow X_3$—and populate the local distributions with the conditional-probability estimates. Note that feature selection in the classification/regression process governs the structure of the dependency network.

This algorithm will be computationally efficient in many situations where learning a Markov network and converting it to a dependency network is not. Nonetheless, this procedure has the drawback that, due to—for example—heuristic search and finite-data effects, the resulting local distributions are likely to be *inconsistent* in that there is no joint distribution $p(\mathbf{x})$ from which each of the local distributions may be obtained via the rules of probability. For example, in the simple domain $\mathbf{X} = (X_1, X_2)$ it is possible that the estimator of $p(x_1|x_2)$ discards X_2 as an input whereas the estimator of $p(x_2|x_1)$ retains X_1 as an input. The result is the structural inconsistency that X_1 helps to predict X_2, but X_2 does not help to predict X_1. Numeric inconsistencies are also likely to result. Nonetheless, in situations where the data set contains many samples, strong inconsistencies will be rare because each local distribution is learned from the same data set, which we assume is generated from a single underlying joint distribution. In other words, although dependency networks learned using our procedure will be inconsistent, they will be "almost" consistent when learned from data sets with large sample sizes. This observation assumes that the model classes used for the local distributions can closely approximate the conditional distributions consistent with the underlying joint distribution. For example, probabilistic decision trees, which we shall use, satisfy this assumption for variables with finite domains.

Because a dependency network learned in this manner is almost consistent, we can imagine—as a heuristic—using procedures resembling Gibbs sampling to extract a joint distribution and to answer probabilistic queries. In the remainder of this section, we formalize these ideas and evaluate them with experiments on real data.

3.1 Definition and Basic Properties

Given a domain of interest having a set of finite variables $\mathbf{X} = (X_1, \ldots, X_n)$, let $\mathcal{P} = (p_1(x_1|\mathbf{x} \setminus x_1), p_2(x_2|\mathbf{x} \setminus x_2), \ldots, p_n(x_n|\mathbf{x} \setminus x_m))$ be a set of conditional distributions, one for each variable in $\mathbf{X}$. In normal use, these distributions are intended to correspond to classifications/regressions learned from a single data set but, formally, they can be any set of conditional distributions. We do not require that these distributions be consistent—that is, we do not require that they can be obtained via inference from a single joint distribution $p(\mathbf{x})$. A *dependency network for* $\mathbf{X}$ *and* $\mathcal{P}$ is a pair $(\mathcal{G}, \mathcal{P}')$ where $\mathcal{G}$ is a (usually cyclic) directed graph and $\mathcal{P}'$ is a set of conditional probability distributions satisfying

$$p_i(x_i|\mathbf{pa}_i) = p_i(x_i|\mathbf{x} \setminus x_i)$$

for every p_i in $\mathcal{P}$. Again, we call the set of conditional distributions $p_i(x_i|\mathbf{pa}_i), i = 1, \ldots, n$ the local probability distributions for the dependency network.

In most graphical modeling research, a graphical model is used to define a joint distribution for its variables. We do the same using a dependency network. In particular, the procedure of the ordered Gibbs sampler described in the previous section, when applied to a dependency network, yields a joint distribution for the domain. The result holds whether or not the local distributions in the dependency network are consistent. More formally, we have the following theorem, which is established by tracing the proof of Theorem 2 given in the previous section and noting that it does not rely on the consistency of the distributions.

Theorem 3: The procedure of an ordered Gibbs sampler applied to a dependency network for $\mathbf{X}$ and $\mathcal{P}$, where each X_i is finite (and hence discrete) and each local distribution in $\mathcal{P}$ is positive, defines a Markov chain with a unique stationary joint distribution for $\mathbf{X}$ that can be reached from any initial state of the chain.

Technically speaking, the procedure of Gibbs sampling applied to an inconsistent dependency network is not itself a Gibbs sampler, because there is no joint distribution consistent with all the local distributions. Consequently, we call this procedure an *ordered pseudo-Gibbs sampler.*

One rather disturbing property of this procedure is that it produces a joint distribution that is likely to be inconsistent with many of the conditional distributions used to produce it. Nonetheless, as we have suggested and shall examine further, when each of the local distributions of a dependency network are learned from the same data, these distributions should be almost consistent with the joint distribution.

Another disturbing observation is that the joint distribution obtained will depend on the order in which the pseudo-Gibbs sampler visits the variables. For example, consider the dependency network with the structure $X_1 \to X_2$, saying that X_2 depends on X_1 but X_1 does not depend on X_2. If we draw sample-pairs (x_1, x_2)—that is, x_1 and then x_2—then the resulting stationary distribution will have X_1 and X_2 dependent. In contrast, if we draw sample-pairs (x_2, x_1), then the resulting stationary distribution will have X_1 and X_2 independent. Due to the near consistency of the local distributions, however, the joint distributions obtained from different orderings will be close. If we indeed discover the dependency-network structure $X_1 \to X_2$ in practice, then X_1 and X_2 must be "almost" independent.

Given a graphical model for a domain and an ordered Gibbs sampler that extracts a joint distribution from that model, we can apply the rules of probability to this joint distribution to answer probabilistic queries. Alternatively, we can apply Algorithm 1 directly to a given dependency network. Such an application may yield different answers than those computed from the joint distribution obtained from the dependency network, but can be justified heuristically due to near consistency. In Sections 3.3 and 3.4, we examine the issue of near consistency more carefully. In the remainder of this section, we mention two basic properties of dependency networks.

First, let us consider the distributions that can be represented by a general dependency-network structure. Unlike the situation for consistent dependency networks, a general dependency-network structure and a Markov network structure with the same adjacencies do not represent the same distributions. In fact, a dependency network with a given structure defines a larger set of distributions than a Markov network with the same adjacencies. As an example, consider the dependency-network structure $X_1 \leftrightarrow X_2 \leftrightarrow X_3$. In a simple experiment, we sampled local distributions for this structure from a uniform distribution. We then computed the stationary distribution of the Markov chain defined by a pseudo-Gibbs sampler with variable order (X_1, X_3, X_2). In all runs, we found that X_1 and X_3 were conditionally *dependent* given X_2 in the stationary distribution. In a Markov network with the same adjacencies, X_1 and X_3 must be conditionally independent given X_2.

Second, let us consider a simple necessary condition for consistency. We say that a dependency network for $\mathbf{X}$ is *bi-directional* if X_i is a parent of X_j if and only if X_j is a parent of X_i, for all X_i and X_j in $\mathbf{X}$. In addition, let $\mathbf{pa}_i^j$ be the j^{th} parent of node X_i. We say that a consistent dependency network is *minimal* if and only if, for every node X_i and for every parent $\mathbf{pa}_i^j$, X_i is not independent of $\mathbf{pa}_i^j$ given the remaining parents of X_i. With these definitions, we have the following theorem, proved in the Appendix.

Theorem 4: A minimal consistent dependency network for a positive distribution $p(\mathbf{x})$ must be bi-directional.

3.2 Probabilistic Decision Trees for Local Distributions

When learning a dependency network from data, a variety of classification/regression techniques may be used to estimate the local distributions. We have used methods based on probabilistic decision trees (e.g., Buntine, 1991) and probabilistic support vector machines (e.g., Platt, 1999). For simplicity, we limit our discussion in this paper to the use of probabilistic decision trees.

In this approach, for each variable X_i in $\mathbf{X}$, we learn a probabilistic decision tree where X_i is the target variable and $\mathbf{X} \setminus X_i$ are the input variables. Each leaf is modeled as a multinomial distribution. To learn the decision-tree structure, we use a simple hill-climbing approach in conjunction with a Bayesian score (posterior probability of model structure) as described by Friedman and Goldszmdit (1996) and Chickering, Heckerman, and Meek (1997). To learn a decision-tree structure for X_i, we initialize the search algorithm with a singleton root node having no children. Then, we replace each leaf node in the tree with a binary split on some variable X_j in $\mathbf{X} \setminus X_i$, until no such replacement increases the score of the tree. Our *binary split* on X_j is a decision-tree node with two children: one of the children corresponds to a particular value of X_j, and the other child corresponds to *all other* values of X_j. Our Bayesian scoring function uses a uniform prior distribution for the parameters of all multinomial distributions, and a structure prior proportional to κ^f, where $\kappa > 0$ is a tunable parameter and f is the number of free parameters in the decision tree. In studies that predated those described in this paper, we have found that the setting $\kappa = 0.1$ yields accurate predictions over a wide variety of datasets. We use this same setting in the experiments described in this paper.

3.3 Probabilistic Inference With Real Data

We have suggested that, because the local distributions learned from data sets of adequate size will be close to the true underlying distribution and hence almost consistent, the procedures for extracting a joint distribution from a dependency network and for answering probabilistic queries should yield fairly accurate results. Nonetheless, there is a concern. It could be that the application of pseudo-Gibbs sampling amplifies the inconsistencies. That is, it could be that small perturbations from the true conditional distributions $p(x_i|\mathbf{x} \setminus x_i)$ could lead to large perturbations from the true joint distribution $p(\mathbf{x})$. If this phenomenon did occur, it is likely that predictions of new data rendered by a dependency network would be inaccurate. In this section, we compare the predictions of dependency networks and Bayesian networks on real data sets as a first examination of this concern.

We use four datasets: (1) *Sewall/Shah*, data from Sewall and Shah (1968) regarding the college plans of high-school seniors, (2) *Women and Mathematics (WAM)*, data regarding women's preferences for a career in Mathematics (Fowlkes, Freeny, and Landwehr, 1988), (3) *Digits*, images of handwritten digits made available by the US Postal Service Office for Advanced Technology (Frey, Hinton, and Dayan, 1995), and (4) *Nielsen*, data about whether or not users watched five or more minutes of network TV shows aired during a two-week period in 1995 (made available by Nielsen Media Research). In each of these datasets, all variables are finite. For the digits data, we report results for only two (randomly chosen) digits, "2" and "6". Additional details about the datasets are given in Table 1.

To evaluate the accuracy of dependency networks on these datasets, we randomly partition each dataset into a training set and a test set used to learn models and evaluate them, respectively. We measure the accuracy of each learned model on the test set $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$

using the log score:

$$\mathrm{Score}(\mathbf{x}_1, \ldots, \mathbf{x}_N|\mathrm{model}) = -\frac{\sum_{i=1}^{N} \log_2 \ p(\mathbf{x}_i|\mathrm{model})}{nN}, \tag{3}$$

where n is the number of variables in $\mathbf{X}$. This score can be thought of as the average number of bits needed to encode the observation of a variable in the test set. Note that we measure how well a dependency network predicts an entire case. We could look at predictions of particular conditional probabilities, but because Algorithm 1 uses products of queries to produce a prediction on a full case, we expect comparisons on individual queries to be similar.

In our experiments, we determine the probabilities $p(\mathbf{x}|\mathrm{model})$ in Equation 3 from a learned dependency network using Algorithm 1. For each pseudo-Gibbs sampler invoked in Algorithm 1, we average 5000 iterations after a 10-iteration burn-in. For each data set, these Gibbs-sampling parameters yield scores with a range of variation of less than 0.1% across 10 runs starting with different (random) initial states.

For comparison, we measure the accuracy of two additional model classes: (1) a Bayesian network, and (2) a Bayesian network with no arcs—a *baseline model*. When learning the non-baseline Bayesian network, we use the algorithm described in Chickering, Heckerman, and Meek (1997) wherein each local distribution consists of a decision tree with binary splits. We use the same parameter and structure priors as used in the learning of dependency networks. We determine $p(\mathbf{x}|\mathrm{model})$ from a Bayesian network using the law of total probability—pseudo-Gibbs sampling is not needed. Probability estimates obtained from both Bayesian networks and dependency networks correspond to the *a posteriori* mean of the (multinomial) parameters.

The results are shown in Table 1. The Bayesian networks produce density estimates that are better than those of dependency networks, but only slightly so. In particular, consider the summary score in the second row from the bottom of the table, $2^{\mathrm{Score(DN)}-\mathrm{Score(BN)}}$, which is the geometric mean of $p(\mathbf{x}|\mathrm{BN})/p(\mathbf{x}|\mathrm{DN})$ averaged over all cases in a dataset. For Digit2, the data set having the worst dependency-network performance, the dependency network assigns a probability to a case, on (geometric) average, that is only 3% lower than that assigned by the Bayesian network.

That dependency networks are (slightly) less accurate than Bayesian networks is not surprising. In each domain, the number of parameters in the Bayesian network are fewer than the number of parameters in the corresponding dependency network. Consequently, the dependency-network estimates should have higher variance. This explanation is consistent with the observation that, roughly, differences in accuracy are larger for the data sets with smaller sample size.

Because dependency networks produce joint probabilities via pseudo-Gibbs sampling whereas Bayesian networks produce joint probabilities via multiplication, non-convergence of sampling is another possible explanation for the greater accuracy of Bayesian networks. The small variances of the pseudo-Gibbs-sampler estimates across multiple runs, however, makes this explanation unlikely.

Overall, our experiments suggest that Algorithm 1 applied to inconsistent dependency networks learned from data yields accurate joint probabilities.

Table 1: Details for datasets and Score (bits per observation) for a Bayesian network (BN), dependency network (DN), and baseline model (BL) applied to these datasets. The lower the Score, the higher the accuracy of the learned model.

	Dataset				
	Sewall/Shah	WAM	Digit2	Digit6	Nielsen
Number of variables	5	6	64	64	203
Training cases	9286	790	700	700	1637
Test cases	1032	400	399	400	1637
Score(BN)	1.274	0.907	0.542	0.422	0.188
Score(DN)	1.277	0.911	0.584	0.454	0.189
Score(BL)	1.382	0.930	0.823	0.752	0.231
Score(DN)-Score(BN)	0.002	0.004	0.042	0.033	0.001
Score(BL)-Score(BN)	0.107	0.022	0.281	0.330	0.044
$2^{\text{Score(DN)}-\text{Score(BN)}}$	1.00	1.00	1.03	1.02	1.00
$\frac{\text{Score(DN)}-\text{Score(BN)}}{\text{Score(BL)}-\text{Score(BN)}}$	0.02	0.16	0.15	0.10	0.02

Finally, let us consider issues of computation. Joint estimates produced by dependency networks require far more computation than do those produced by Bayesian networks. For example, on a 600 MHz Pentium III with 128 MB of memory running the Windows 2000 operating system, the determination of $p(\mathbf{x})$ for a case in the Nielsen dataset takes on average 2.0 seconds for the dependency network and 0.0006 seconds for the Bayesian network. Consequently, one should use a Bayesian network in those situations where it is known in advance that joint probabilities $p(\mathbf{x})$ are needed. Nonetheless, in situations where general probabilistic inference is needed, exact inference in a Bayesian network is often intractable; and practitioners often turn to Gibbs sampling for inference. In such situations, Bayesian networks afford little computational advantage.

3.4 Near Consistency: Theoretical Considerations

Our concerns of the previous section were motivated by the possibility that small deviations in the local distributions of a dependency network could be amplified by the process of pseudo-Gibbs sampling. In this section, we examine this concern more directly and from a theoretical perspective.

Consider two dependency networks: one learned from data and another that encodes the true distribution from which the data was sampled. Note that it is always possible to find a dependency network that encodes the true distribution—for example, a fully-connected dependency network can encode any joint distribution. Let $\tilde{\mathbf{P}}$ and $\mathbf{P}$ denote the transition matrices of the Markov chains defined by the learned and truth-encoding dependency networks, respectively. In addition, let $\tilde{\pi} = (\tilde{\pi}_1, \ldots, \tilde{\pi}_k)$ and $\pi = (\pi_1, \ldots, \pi_k)$ denote the stationary distributions corresponding to $\tilde{\mathbf{P}}$ and $\mathbf{P}$, respectively. Our concern about sensitivity to deviations in local distributions can now be phrased as follows: If $\tilde{\mathbf{P}}$ is close to $\mathbf{P}$, will $\tilde{\pi}$ be close π?

There is a large literature in an area generally known as perturbation theory of stochastic matrices that provides answers to this question for various notions of "close". Answers usually are of the form

$$\|\tilde{\pi} - \pi\| \leq \kappa \ \|\mathbf{E}\|, \quad \text{or} \tag{4}$$

$$|\tilde{\pi}_j - \pi_j| \leq \kappa_j \ \|\mathbf{E}\|, \quad \text{or} \tag{5}$$

$$\left|\frac{\tilde{\pi}_j - \pi_j}{\pi_j}\right| \leq \kappa_j \ \|\mathbf{E}\|, \tag{6}$$

where $\mathbf{E} = \tilde{\mathbf{P}} - \mathbf{P}$, $\|\cdot\|$ is some norm, and κ or κ_j are measures of sensitivity called *condition numbers*. When working with probabilities, where relative as opposed to absolute errors are often important, bounds of the form shown in Equation 6 are particularly useful. Here, we cite a potentially useful bound of this form given by Cho and Meyer (1999). These authors provide references to many other bounds as well.

Theorem (Cho and Meyer, 1999): Let $\mathbf{P}$ and $\tilde{\mathbf{P}} = \mathbf{P} + \mathbf{E}$ be transition matrices for two homogenous, irreducible k-state Markov chains with respective stationary distributions π and $\tilde{\pi}$. Let $\|E\|_\infty$ denote the infinity-norm of $\mathbf{E}$, the maximum over $j = 1, \ldots, k$ of the column sums $\sum_{i=1}^{k} |\mathbf{E}_{ij}|$. Let M_{ij} denote the mean first passage time from the i^{th} state to the j^{th} state in the chain corresponding to $\mathbf{P}$—that is, the expected number of transitions to move from state i to state j. Then, the relative change in the j^{th} stationary probability π_j is given by

$$\frac{|\tilde{\pi}_j - \pi_j|}{\pi_j} \leq \frac{1}{2} \ \|E\|_\infty \ \max_{i \neq j} M_{ij}. \tag{7}$$

Their bound is tight in the sense that, for any $\mathbf{P}$ satisfying the conditions of the theorem, there exists a perturbation $\mathbf{E} \neq \mathbf{0}$ such that the inequality in Equation 7 can be replaced with an equality. Nonetheless, when applied to dependency networks, the bound is typically not tight. To illustrate this point, we computed each term in Equation 7 for a transition matrix $\mathbf{P}$ corresponding to the Bayesian network learned from 790 cases from the WAM data set, and a $\tilde{\mathbf{P}}$ corresponding to a dependency network learned from a random sample of 790 cases generated from that Bayesian network. For $j = 0$, the state corresponding to $X_1 = \cdots = X_6 = 0$ in the WAM domain, we obtain the inequality

$$\begin{array}{ccccccccc} \frac{|\pi_j - \tilde{\pi}_j|}{\pi_j} & \leq & \frac{1}{2} & \times & \|E\|_\infty & \times & \max_{i \neq j} M_{ij} & & \\ 0.32 & \leq & 0.5 & \times & 0.84 & \times & 40 & = & 16.8 \end{array}$$

which is far from tight.

Nonetheless, the appearance of mixing time in Equation 7 is interesting. It suggests that that chains with good convergence properties will be insensitive to perturbations in the transition matrix. In particular, it suggests that the joint distribution defined by a dependency network is more likely to be insensitive to errors in the local distributions of the network precisely when Gibbs sampling is effective. Hopefully, additional research will produce tighter bounds that better characterize those situations in which dependency networks can be used safely.

3.5 Related Work

Before we consider new applications of dependency networks, we review related work on the basic concepts. As we have already mentioned, several researchers who developed Markov networks began with an examination of what we call consistent dependency networks. For an excellent discussion of this development as well as original contributions in this area, see Besag (1974). Besag (1975) also described an approach called *pseudo-likelihood estimation*, in which the conditionals are learned directly—as in our approach—without respecting the consistency constraints. We use the name *pseudo*-Gibbs sampling to make a connection to his work. Tresp and Hofmann (1998) describe (general) dependency networks, calling them *Markov blanket networks*. They stated and proved Theorem 3, and evaluated the predictive accuracy of the representation on several data sets using local distributions consisting of conditional Parzen windows.

4. Collaborative Filtering

We now turn our attention to collaborative filtering (CF), the task of predicting preferences. Examples of this task include predicting what movies a person will like based on his or her ratings of movies seen, predicting what news stories a person is interested in based on other stories he or she has read, and predicting what web pages a person will go to next based on his or her history on the site. Another important application in the burgeoning area of e-commerce is predicting what products a person will buy based on products he or she has already purchased and/or dropped into his or her shopping basket.

Collaborative filtering was introduced by Resnick, Iacovou, Suchak, Bergstrom, and Riedl (1994) as both the task of predicting preferences and a class of algorithms for this task. The class of algorithms they described was based on the informal mechanisms people use to understand their own preferences. For example, when we want to find a good movie, we talk to other people that have similar tastes and ask them what they like that we haven't seen. The type of algorithm introduced by Resnik et al. (1994), sometimes called a *memory-based algorithm*, does something similar. Given a user's preferences on a series of items, the algorithm finds similar users in a database of stored preferences. It then returns some weighted average of preferences among these users on items not yet rated by the original user.

As done in Breese, Heckerman, and Kadie (1998), let us concentrate on the *application* of collaborative filtering—that is, preference prediction. In their paper, Breese et al. (1998) describe several CF scenarios, including binary versus non-binary preferences and implicit versus explicit voting. An example of explicit voting would be movie ratings provided by a user. An example of implicit voting would be knowing only whether a person has or has not purchased a product. Here, we concentrate on one scenario important for e-commerce: implicit voting with binary preferences—for example, the task of predicting what products a person will buy, knowing only what other products they have purchased.

A simple approach to this task, described in Breese et al. (1998), is as follows. For each item (e.g., product), define a variable with two states corresponding to whether or not that item was preferred (e.g., purchased). We shall use "0" and "1" to denote not preferred and preferred, respectively. Next, use the dataset of ratings to learn a Bayesian network for the joint distribution of these variables $\mathbf{X} = (X_1, \ldots, X_n)$. The preferences of each user constitutes a case in the learning procedure. Once the Bayesian network is constructed, make predictions as follows. Given a new user's preferences $\mathbf{x}$, use the Bayesian network to estimate $p(x_i = 1|\mathbf{x} \setminus x_i = 0)$ for each product X_i not purchased. That is, estimate the probability that the user would have purchased the item had we not known he did not. Then, return a list of recommended products—among those that the user did not purchase—ranked by these estimates.

Breese et al. (1998) show that this approach outperforms memory-based and cluster-based methods on several implicit rating datasets. Specifically, the Bayesian-network approach was more accurate and yielded faster predictions than did the other methods.

What is most interesting about this algorithm in the context of this paper is that only estimates of $p(x_i = 1|\mathbf{x} \setminus x_i = 0)$ are needed to produce the recommendations. In particular, these estimates may be obtained by a direct lookup in a dependency network:

$$p(x_i = 1|\mathbf{x} \setminus x_i = 0) \approx p_i(x_i = 1|\mathbf{pa}_i), \tag{8}$$

where $\mathbf{pa}_i$ is the instance of $\mathbf{Pa}_i$ consistent with $\mathbf{X}$. Thus, dependency networks are a natural model class on which to base CF predictions. In the remainder of this section, we compare this approach with that based on Bayesian networks for datasets containing binary implicit ratings.

4.1 Datasets

We evaluated dependency networks and Bayesian networks on three datasets: (1) *Nielsen*, the dataset described in Section 3.3, (2) *MS.COM*, which records whether or not users of microsoft.com on one day in 1996 visited areas ("vroots") of the site (available on the Irvine Data Mining Repository), and (3) *MSNBC*, which records whether or not visitors to MSNBC on one day in 1998 read stories among the most popular 1001 stories on the site. In each of these datasets, users correspond to cases and items possibly viewed correspond to variables. The MSNBC dataset contains 20,000 users sampled at random from the approximate 600,000 users that visited the site that day. In a separate analysis on this dataset, we found that the inclusion of additional users did not produce a substantial increase in accuracy. Table 2 provides additional information about each dataset. All datasets were partitioned into training and test sets at random. The train/test split for Nielsen was the same as for the density-estimation experiment described in Section 3.3. The learning algorithms for dependency networks and Bayesian networks and their parameters described in Section 3 were used here.

4.2 Evaluation Criteria and Experimental Procedure

We have found the following three criteria for collaborative filtering to be important: (1) the accuracy of the recommendations, (2) prediction time—the time it takes to create a recommendation list given what is known about a user, and (3) the computational resources needed to build the prediction models. We measure each of these criteria in our empirical comparison. In the remainder of this section, we describe our evaluation criterion for accuracy.

Table 2: Number of users, items, and items per user for the datasets used in evaluating the algorithms.

	Dataset		
	MS.COM	Nielsen	MSNBC
Training cases	32711	1637	10000
Test cases	5000	1637	10000
Total items	294	203	1001
Mean items per case in training set	3.02	8.64	2.67

Our criterion attempts to measure a user's expected utility for a list of recommendations. Of course, different users will have different utility functions. The measure we introduce provides what we believe to be a good approximation across many users.

The scenario we imagine is one where a user is shown a ranked list of items and then scans that list for preferred items starting from the top. At some point, the user will stop looking at more items. Let $p(k)$ denote the probability that a user will examine the kth item on a recommendation list before stopping his or her scan, where the first position is given by $k = 0$. Then, a reasonable criterion is

$$\text{cfaccuracy}_1(\text{list}) = \sum_k p(k)\ \delta_k,$$

where δ_k is 1 if the item at position k is preferred and 0 otherwise. To make this measure concrete, we assume that $p(k)$ is an exponentially decaying function:

$$p(k) = 2^{-k/a}, \tag{9}$$

where a is the "half-life" position—the position at which an item will be seen with probability 0.5. In our experiments, we use $a = 5$.

In one possible implementation of this approach, we could show recommendations to a series of users and ask them to rate them as "preferred" or "not preferred". We could then use the average of $\text{cfaccuarcy}_1(\text{list})$ over all users as our criterion. Because this method is extremely costly, we instead use an approach that uses only the data we have. In particular, as already described, we randomly partition a dataset into a training set and a test set. Each case in the test set is then processed as follows. First, we randomly partition the user's preferred items into *input* and *measurement* sets. The input set is fed to the CF model, which in turn outputs a list of recommendations. Finally, we compute our criterion as

$$\text{cfaccuracy}(\text{list}) = \frac{100}{N} \sum_{i=1}^{N} \frac{\sum_{k=0}^{R_i-1} \delta_{ik}\ p(k)}{\sum_{k=0}^{M_i-1} p(k)}, \tag{10}$$

where N is the number of users in the test set, R_i is the number of items on the recommendation list for user i, M_i is the number of preferred items in the measurement set for user

i, and δ_{ik} is 1 if the kth item in the recommendation list for user i is preferred in the measurement set and 0 otherwise. The denominator in Equation 10 is a per-user normalization factor. It is the utility of a list where all preferred items are at the top. This normalization allows us to more sensibly combine scores across measurement sets of different size.

We performed several experiments reflecting differing numbers of ratings available to the CF engines. In the first protocol, we included all but one of the preferred items in the input set. We term this protocol *all but 1*. In additional experiments, we placed 2, 5, and 10 preferred items in the input sets. We call these protocols *given 2*, *given 5*, and *given 10*.

The *all but 1* experiments measure the algorithms' performance when given as much data as possible from each test user. The various *given* experiments look at users with less data available, and examine the performance of the algorithms when there is relatively little known about an active user. When running the *given m* protocols, if an input set for a given user had less than m preferred items, the case was eliminated from the evaluation. Thus the number of trials evaluated under each protocol varied.

All experiments were performed on a 300 MHz Pentium II with 128 MB of memory running the Windows NT 4.0 operating system.

4.3 Results

Table 3 shows the accuracy of recommendations for dependency networks and Bayesian networks across the various protocols and three datasets. For a comparison, we also measured the accuracy of recommendation lists produced by a Bayesian network with no arcs (baseline model). This model recommends items based on their overall popularity, $p(x_i = 1)$. A score in boldface corresponds to a statistically significant winner. We use ANOVA (e.g., McClave and Dietrich, 1988) with $\alpha = 0.1$ to test for statistical significance. When the difference between two scores in the same column exceed the value of RD (required difference), the difference is significant.

As in the case of density estimation, we see from the table that Bayesian networks are more accurate than dependency networks, but only slightly so. In particular, the ratio of (cfaccuracy(BN) − cfaccuracy(DN)) to (cfaccuracy(BN) − cfaccuracy(Baseline)) averages 4 ± 5 percent across the datasets and protocols. As before, the differences are probably due to the fact that dependency networks are less statistically efficient than Bayesian networks.

Tables 4 and 5 compare the two methods with the remaining criteria. Here, dependency networks are a clear winner. They are significantly faster at prediction—sometimes by almost an order of magnitude—and require substantially less time and memory to learn. Overall, Bayesian networks are slightly more accurate but much less attractive from a computational perspective.

5. Data Visualization

Our initial motivation for developing dependency networks concerned the visualization of predictive relationships. In this section, we examine this application in more detail and describe a tool developed at Microsoft Research, called DNetViewer, that employs dependency networks for data visualization. For illustration, we use a real data set, provided by Media Metrix, that contains demographic and internet-use data for about 5,000 individuals during the month of January 1997.

Table 3: CF accuracy for the MS.COM, Nielsen, and MSNBC datasets. Higher scores indicate better performance. Statistically significant winners are shown in boldface.

	MS.COM			
Algorithm	Given2	Given5	Given10	AllBut1
BN	**53.18**	52.48	51.64	66.54
DN	52.68	52.54	51.48	66.60
RD	*0.30*	*0.73*	*1.62*	*0.34*
Baseline	43.37	39.34	39.32	49.77

	Nielsen			
Algorithm	Given2	Given5	Given10	AllBut1
BN	**24.99**	30.03	33.84	**45.55**
DN	24.20	29.71	33.80	44.30
RD	*0.32*	*0.40*	*0.65*	*0.72*
Baseline	12.65	12.72	12.92	13.59

	MSNBC			
Algorithm	Given2	Given5	Given10	AllBut1
BN	**40.34**	**34.20**	30.39	**49.58**
DN	38.84	32.53	30.03	48.05
RD	*0.35*	*0.77*	*1.54*	*0.39*
Baseline	28.73	20.58	14.93	32.94

Table 4: Number of predictions per second for the MS.COM, Nielsen, and MSNBC datasets.

	MS.COM			
Algorithm	Given2	Given5	Given10	AllBut1
BN	3.94	3.84	3.29	3.93
DN	23.29	19.91	10.20	23.48

	Nielsen			
Algorithm	Given2	Given5	Given10	AllBut1
BN	22.84	21.86	20.83	23.53
DN	36.17	36.72	34.21	37.41

	MSNBC			
Algorithm	Given2	Given5	Given10	AllBut1
BN	7.21	6.96	6.09	7.07
DN	11.88	11.03	8.52	11.80

Table 5: Computational resources for model learning.

	MS.COM	
Algorithm	Memory (Meg)	Learn Time (sec)
BN	42.4	144.65
DN	5.3	98.31

	Nielsen	
Algorithm	Memory (Meg)	Learn Time (sec)
BN	3.3	7.66
DN	2.1	6.47

	MSNBC	
Algorithm	Memory (Meg)	Learn Time (sec)
BN	43.0	105.76
DN	3.7	96.89

Figure 2 shows DNetViewer's display of a dependency-network structure learned from this data. After only a short inspection, an interesting relationship becomes apparent: there are many dependencies among demographics, and many dependencies among frequency-of-use, but there are few dependencies between demographics and frequency-of-use. We have found numerous interesting dependency relationships such as this one across a wide variety of datasets using dependency networks for visualization. In fact, we have given dependency networks this name because they have been so useful in this regard.

DNetViewer allows a user to display both the dependency-network structure and the probabilistic decision tree associated with each variable. Navigation between the views is straightforward. To view a decision tree for a variable, a user double clicks on the corresponding node in the dependency network. Figure 3 shows the tree for Shopping.Freq. Note that there is an interesting relationship between the dependency-network structure and the individual decision-tree structures. Namely, there will be a split on variable X in the decision tree for Y if and only if there is an arc from X to Y in the dependency network. We have found that this correspondence facilitates the process of data visualization.

Besides avoiding the sometimes confusing semantics of Bayesian networks, a dependency network—in particular, an inconsistent dependency network—learned from data offers an additional advantage for visualization over Bayesian networks. If there is an arc from X to Y in such a network, we know that X is a *significant* predictor of Y—significant in whatever sense was used to learn the network with finite data. Under this interpretation, a uni-directional link from X to Y is not confusing, but rather informative. For example, in Figure 2, we see that Socioeconomic status is a significant predictor of Sex, but not vice versa—an interesting observation. Of course, when making such interpretations, one must always be careful to recognize that statements of the form "X helps to predict Y" are made in the context of the other variables in the network.

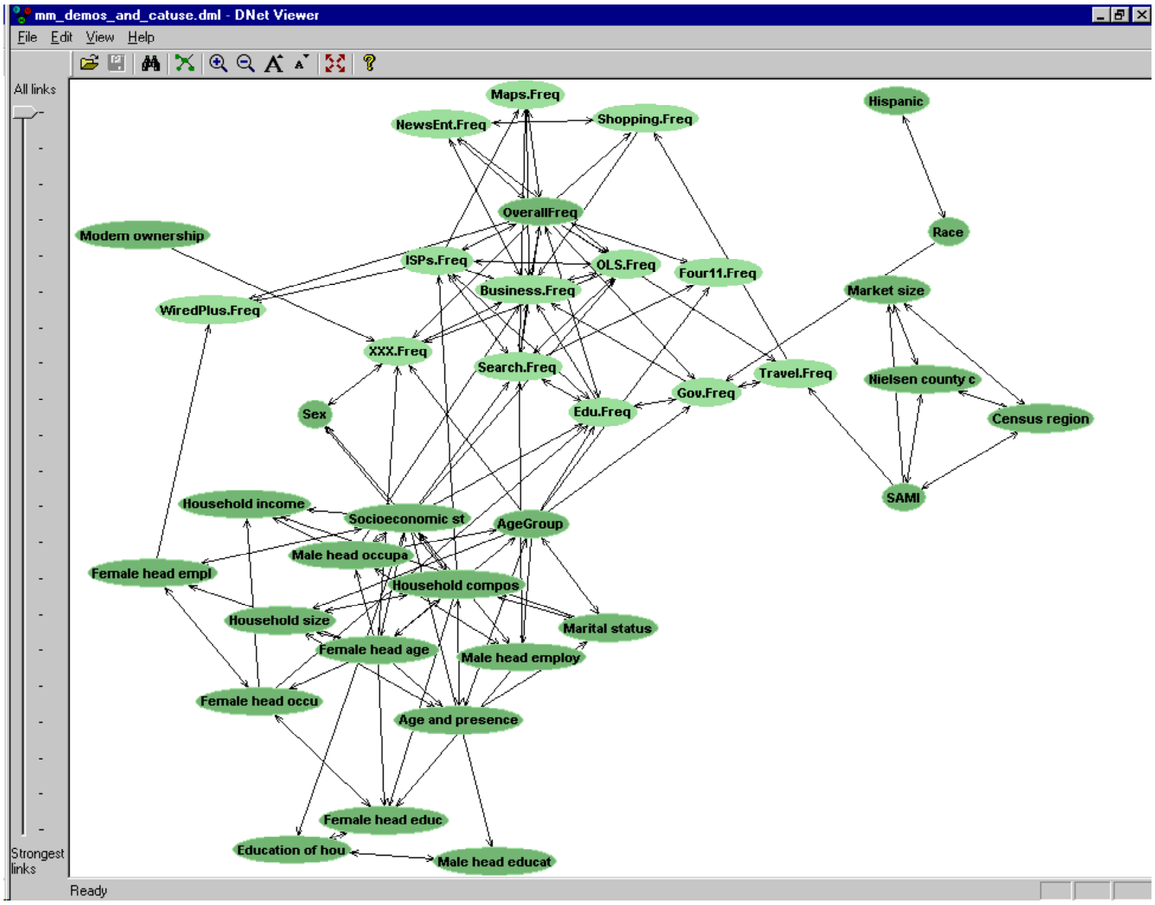

Figure 2: A dependency network for Media Metrix data. The dataset contains demographic and internet-use data for about 5,000 individuals during the month of January 1997. The node labeled Overall.Freq represents the overall frequency-of-use of the internet during this period. The nodes Search.Freq, Edu.Freq, and so on represent frequency-of-use for various subsets of the internet.

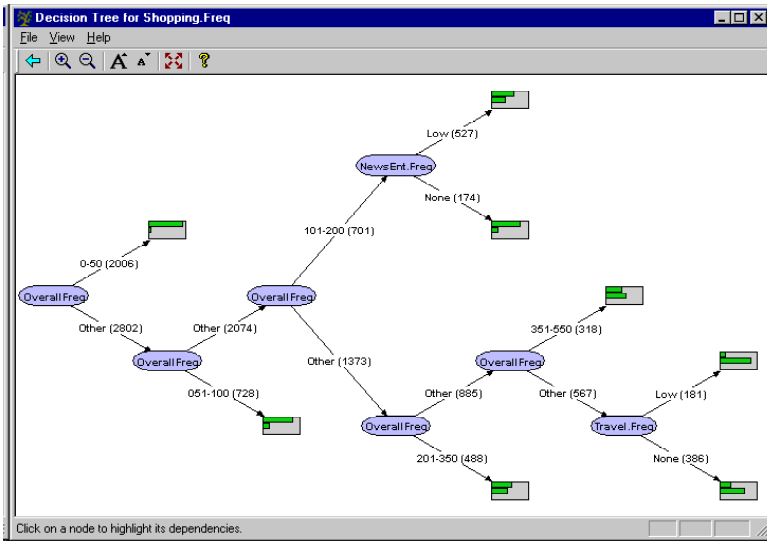

Figure 3: The probabilistic decision tree for Shopping.Freq obtained by double-clicking the corresponding node in the dependency-network graph. The histograms at the leaves correspond to probabilities of Shopping.Freq use being zero, one, and greater than one visit per month, respectively.

In DNetViewer, we enhance the ability of dependency networks to reflect strength of dependency by including a slider (on the left). As a user moves the slider from bottom to top, arcs of decreasing strength are added to the graph. When the slider is in its upper-most position, all arcs (i.e., all significant dependencies) are shown. There are several reasonable methods for ranking arc strength. The one we use determines the order in which arcs would be added during a (greedy) structure search that grows all decision trees in parallel. (In practice, we construct the trees one after the other, but we can imagine a parallel procedure.) At each step of this imagined construction process, we compute the increase in score (log posterior probability) of each tree for every possible new split. We accept the split with the largest increase in score and iterate. As the slider is moved up, we add arcs in the order in which this procedure accepts corresponding splits.

Figure 4 shows the dependency network for the Media Metrix data with the slider at half position. At this setting, we find the interesting observation that the dependence between Sex and XXX.Freq (frequency of hits to pornographic pages) is the strongest among all dependencies between demographics and internet use.

6. Summary and Future Work

We have described a graphical representation for probabilistic dependencies similar to the Bayesian network called a dependency network. Like a Bayesian network, a dependency network has a graph and a probability component. In its consistent form, the graph component is a cyclic directed graph such that a node's parents render that node independent of all other nodes in the network. As in a Bayesian network, the probability component consists of the probability of a node given its parents for each node—the local distributions.

In practice, for computational reasons, we learn the structure and parameters of a dependency network for a given domain by independently performing a classification/regression for each variable in the domain with inputs consisting of all variables except the target variable. The parameterized model for each variable is the local distribution for that variable; and the structure of the network reflects any independencies discovered in the classification/regression process (via feature selection). As a result of this learning procedure, the dependency network is usually inconsistent—that is, it is not the case that the local distributions can be obtained via inference from a single joint distribution for the domain. Nonetheless, because each local distribution is learned from the same data, the local distributions are "almost" consistent when there is adequate data. Consequently, as a useful heuristic, we can apply the machinery of Gibbs sampling to this network to extract a joint distribution for the domain and to answer probabilistic queries. Experiments on real data show this approach to yield accurate predictions.

In addition to their application to probabilistic inference, we have shown that dependency networks are useful for collaborative filtering (the task of predicting preferences) and for the visualization of acausal predictive relationships. In fact, Microsoft has included dependency networks in two of its products—SQL Server 2000 and Commerce Server 2000—for both the collaborative filtering and data visualization tasks.

The intent of our paper has been to introduce the basic concepts and applications of dependency networks. Consequently, there is significant additional work to be done. For example, many of the results described in this paper can be extended to domains

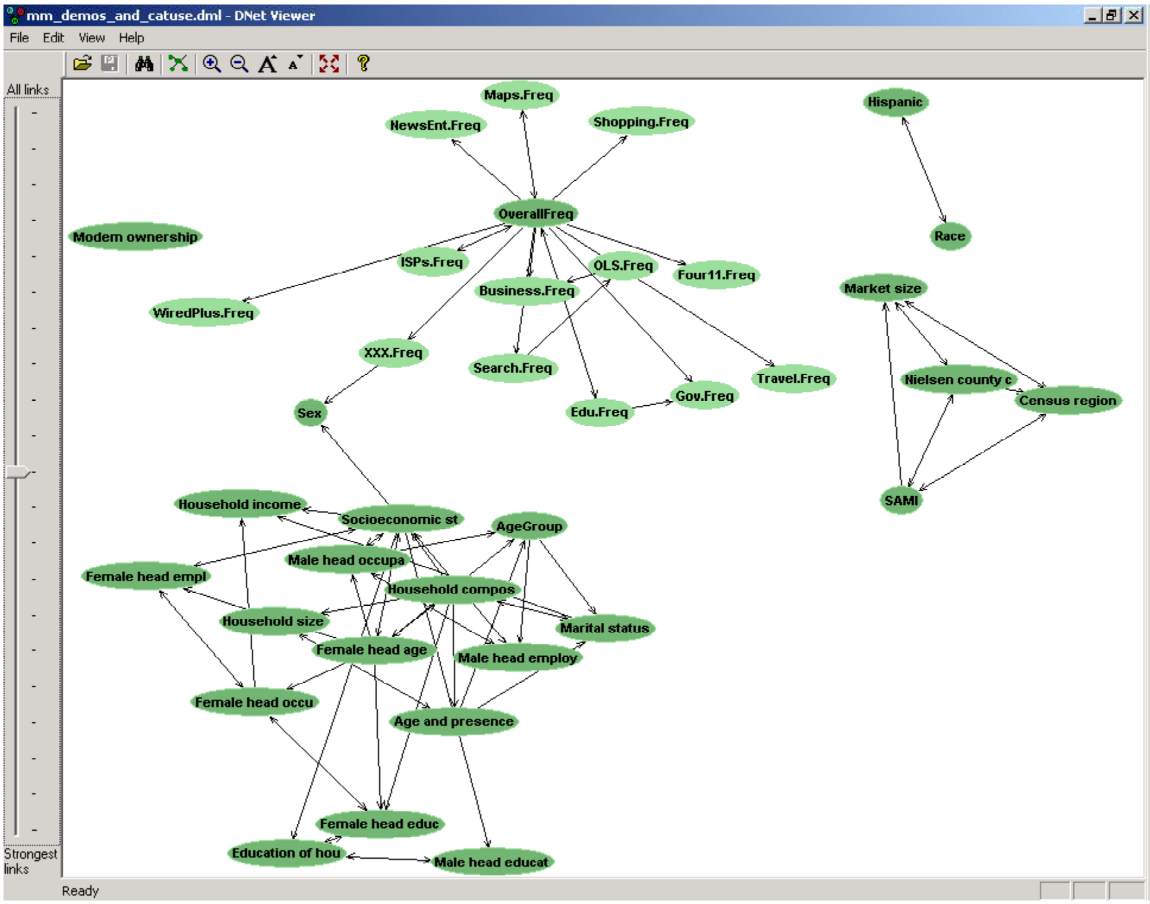

Figure 4: The dependency network in Figure 2 with the slider set at half position.

that include continuous variables. In addition, more work is needed to characterize those situations in which the joint distribution defined by an (inconsistent) dependency network is insensitive to errors in the learned local distributions. As another example, experimental work is needed to examine the predictive accuracy of dependency networks across a variety of domains using alternative methods for classification and regression. It may also be useful to consider pseudo-Gibbs sampling methods that resample variables in random rather than fixed order.

Finally, we note that the representation itself can be generalized. Recall that a dependency network is useful for collaborative filtering primarily because the network stores in its local distributions precisely the probabilistic quantities needed by the ranking algorithm. In general, we can construct a "query network" that directly learns probabilities corresponding to a set of given queries. As an illustration, suppose we have a domain consisting of variables W, X, Y, and Z, and we know we will be answering the query $p(w, x|y, z)$. We can learn this distribution directly from data by performing a series of (independent) classifications/regressions. We can construct the classifications/regressions $p(w|x, y, z)$ and $p(x|y, z)$ and use multiplication to answer the query. Alternatively, we may construct the classifications/regressions $p(w|x, y, z)$ and $p(x|w, y, z)$ and use a pseudo-Gibbs sampler to answer the query. In either case, with sufficient data, the conditional probabilities learned will be "almost" consistent with the true distribution, and are likely to produce accurate answers to the query.

Acknowledgments

We thank Julian Besag, Oliver Downs, Reimar Hofmann, Steffen Lauritzen, Adrian Raftery, Volker Tresp, and the anonymous referees for comments on earlier drafts of this manuscript. We thank David Steinkraus for his implementation of Gibbs sampling for dependency networks. Datasets for this paper were generously provided by Media Metrix, Nielsen Media Research (Nielsen), Microsoft Corporation (MS.COM), and Microsoft Corporation and Steven White (MSNBC).

Appendix: Proofs of Theorems

Theorem 1: The set of positive distributions that can be encoded by consistent dependency networks with graph $\mathcal{G}$ is equal to the set of positive distributions that can be encoded by Markov networks whose structure have the same adjacencies as those in $\mathcal{G}$.

Proof: Let p be a positive distribution defined by a Markov network where $\mathbf{A}_i$ are the adjacencies of X_i, $i = 1, \ldots, n$. Construct a consistent dependency network from p by extracting the conditional distributions $p(x_i|\mathbf{x} \setminus x_i), i = 1, \ldots, n$. Because these probabilities came from the Markov network, we know that $p(x_i|\mathbf{x} \setminus x_i) = p(x_i|\mathbf{A}_i), i = 1, \ldots, n$ so that the adjacencies in the dependency-network structure are the same as those in the Markov network.

Now let p be a positive distribution encoded by a consistent dependency network. By definition, we have that X_i is independent of $\mathbf{X} \setminus X_i$ given $\mathbf{Pa}_i$, $i = 1, \ldots, n$. Because p is positive and these independencies comprise the global Markov property of a Markov network with $\mathbf{A}_i = \mathbf{Pa}_i$, $i = 1, \ldots, n$, the Hammersley–Clifford theorem (Besag, 1974;

Lauritzen, Dawid, Larsen, and Leimer, 1990) implies that p can be represented by this Markov network. $\square$

Theorem 2: An ordered Gibbs sampler applied to a consistent dependency network for $\mathbf{X}$, where each X_i is finite (and hence discrete) and each local distribution $p(x_i|\mathbf{pa}_i)$ is positive, defines a Markov chain with a unique stationary joint distribution for $\mathbf{X}$ equal to $p(\mathbf{X})$ that can be reached from any initial state of the chain.

Proof: In the body of the paper, we showed that the Markov chain can be described by the transition matrix $\mathbf{P} = \mathbf{P}^1 \cdot \ldots \cdot \mathbf{P}^n$, where $\mathbf{P}^k$ is the "local" transition matrix describing the resampling of X_k according to the local distribution $p(x_k|\mathbf{pa}_k)$. We also showed that this Markov chain has a unique joint distribution that can be reached from any starting point. Here, we show that $p(\mathbf{x})$ is that stationary distribution—that is, $p(\mathbf{x}) = \sum_{\mathbf{x}'} p(\mathbf{x}')\ \mathbf{P}_{\mathbf{x}|\mathbf{x}'}$, where $\mathbf{P}_{\mathbf{x}|\mathbf{x}'} = p(\mathbf{x}^{t+1} = \mathbf{x}'|\mathbf{x}^t = \mathbf{x})$. To do so, we show that for each $\mathbf{P}^i$, $p(\mathbf{x}) = \sum_{\mathbf{x}'} p(\mathbf{x}')\ \mathbf{P}^i_{\mathbf{x}|\mathbf{x}'}$.

$$\begin{aligned}
\sum_{\mathbf{x}'} p(\mathbf{x}')\ \mathbf{P}^i_{\mathbf{x}|\mathbf{x}'} &= \sum_{\mathbf{x}'} p(x_i'|\mathbf{x}' \setminus x_i')\ p(\mathbf{x}' \setminus x_i')\ \mathbf{P}^i_{\mathbf{x}|\mathbf{x}'} & (11) \\
&= \sum_{\mathbf{x}'} p(x_i'|\mathbf{x}' \setminus x_i')\ p(\mathbf{x}' \setminus x_i')\ p(x_i|\mathbf{pa}_i') & (12) \\
&= \sum_{\mathbf{x}'} p(x_i'|\mathbf{x}' \setminus x_i')\ p(\mathbf{x}' \setminus x_i')\ p(x_i|\mathbf{x}' \setminus x_i') & (13) \\
&= \sum_{\mathbf{x}'} p(x_i'|\mathbf{x}' \setminus x_i')\ p(\mathbf{x} \setminus x_i)\ p(x_i|\mathbf{x} \setminus x_i) & (14) \\
&= p(\mathbf{x} \setminus x_i)\ p(x_i|\mathbf{x} \setminus x_i)\ \sum_{\mathbf{x}'} p(x_i'|\mathbf{x}' \setminus x_i') & (15) \\
&= p(\mathbf{x}) & (16)
\end{aligned}$$

Equation 12 follows from Equation 11 using the definition of $\mathbf{P}^i$. Equation 13 follows from Equation 12 using the definition of a consistent dependency network. Equation 14 follows from Equation 13 by observing that the value of $\mathbf{x}$ and $\mathbf{x}'$ can differ only in variable x_i. The other steps are straightforward.$\square$

Theorem 4: A minimal consistent dependency network for a positive distribution $p(\mathbf{x})$ must be bi-directional.

Proof: We use the graphoid axioms of Pearl (1988). Suppose the theorem is false. Then, there exists nodes X_i and X_j such that X_j is a parent of X_i and X_i is not a parent of X_j. Let $\mathbf{Z} = \mathbf{Pa}_i \cap \mathbf{Pa}_j$, $\mathbf{W} = \mathbf{Pa}_i \setminus (\mathbf{Pa}_j \cup \{X_j\})$, and $\mathbf{Y} = \mathbf{Pa}_j \setminus \mathbf{Pa}_i$. From minimality, we know that $X_i \perp\!\!\!\perp X_j|\mathbf{W}, \mathbf{Z}$ does not hold. By decomposition, $X_i \perp\!\!\!\perp X_j, \mathbf{Y}|\mathbf{W}, \mathbf{Z}$ does not hold. Given positivity, the intersection property holds. By the intersection property, at least one of the following conditions does not hold: (1) $X_i \perp\!\!\!\perp X_j|\mathbf{W}, \mathbf{Y}, \mathbf{Z}$, (2) $X_i \perp\!\!\!\perp \mathbf{Y}|X_j, \mathbf{W}, \mathbf{Z}$. (If $\mathbf{Y} = \emptyset$, then condition 2 holds vacuously.)

Now, from Theorem 2, we know that $X_i, \mathbf{W} \perp\!\!\!\perp X_j|\mathbf{Y}, \mathbf{Z}$. Using weak union, we have that $X_i \perp\!\!\!\perp X_j|\mathbf{W}, \mathbf{Y}, \mathbf{Z}$—that is, condition 1 holds. Also, from Theorem 2, we know that $X_i \perp\!\!\!\perp \mathbf{Y}|X_j, \mathbf{W}, \mathbf{Z}$—that is, condition 2 holds, yielding a contradiction. $\square$

References

Bartlett, M. (1955). *An Introduction to Stochastic Processes.* University Press, Cambridge.

Besag, J. (1974). Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society, B*, *36*, 192–236.

Besag, J. (1975). Statistical analysis of non-lattice data. *The Statistician*, *24*, 179–195.

Besag, J., Green, P., Higdon, D., & Mengersen, K. (1995). Bayesian computation and stochastic systems. *Statistical Science*, *10*, 3–66.

Bishop, C. (1995). *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford.

Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of Fourteenth Conference on Uncertainty in Artificial Intelligence,* Madison, Wisconsin. Morgan Kaufmann.

Brook, D. (1964). On the distinction between the conditional probability and the joint probability approaches in the specification of nearest-neighbor systems. *Biometrika*, *51*, 481–483.

Buntine, W. (1991). Theory refinement on Bayesian networks. In *Proceedings of Seventh Conference on Uncertainty in Artificial Intelligence,* Los Angeles, CA, pp. 52–60. Morgan Kaufmann.

Chickering, D., Heckerman, D., & Meek, C. (1997). A Bayesian approach to learning Bayesian networks with local structure. In *Proceedings of Thirteenth Conference on Uncertainty in Artificial Intelligence,* Providence, RI. Morgan Kaufmann.

Cho, G., & Meyer, C. (1999). Markov chain sensitivity by mean first passage times. Tech. rep. 112242-0199, North Carolina State University.

Fowlkes, E., Freeny, A., & Landwehr, J. (1988). Evaluating logistic models for large contingency tables. *Journal of the American Statistical Association*, *83*, 611–622.

Frey, B., Hinton, G., & Dayan, P. (1996). Does the wake-sleep algorithm produce good density estimators?. In Touretsky, D., Mozer, M., & Hasselmo, M. (Eds.), *Neural Information Processing Systems*, Vol. 8, pp. 661–667. MIT Press.

Friedman, N., & Goldszmidt, M. (1996). Learning Bayesian networks with local structure. In *Proceedings of Twelfth Conference on Uncertainty in Artificial Intelligence,* Portland, OR, pp. 252–262. Morgan Kaufmann.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *6*, 721–742.

Gilks, W., Richardson, S., & Spiegelhalter, D. (1996). *Markov Chain Monte Carlo in Practice.* Chapman and Hall.

Heckerman, D., & Meek, C. (1997). Models and selection criteria for regression and classification. In *Proceedings of Thirteenth Conference on Uncertainty in Artificial Intelligence,* Providence, RI. Morgan Kaufmann.

Jensen, F., Lauritzen, S., & Olesen, K. (1990). Bayesian updating in recursive graphical models by local computations. *Computational Statisticals Quarterly*, *4*, 269–282.

Lauritzen, S. (1996). *Graphical Models.* Claredon Press.

Lauritzen, S., Dawid, A., Larsen, B., & Leimer, H. (1990). Independence properties of directed Markov fields. *Networks*, *20*, 491–505.

Lévy, P. (1948). Chaines doubles de Markoff et fonctions aleatories de deux variables. *Academcy of Science, Paris*, *226*, 53–55.

McClave, J., & Dieterich, F. (1988). *Statistics.* Dellen Publishing Company.

McCullagh, P., & Nelder, J. (1989). *Generalized Linear Models, Second Edition.* Chapman and Hall, New York.

Neal, R. (1993). Probabilistic inference using Markov chain Monte Carlo methods. Tech. rep. CRG-TR-93-1, Department of Computer Science, University of Toronto.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, CA.

Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods—Support Vector Learning.* MIT Press.

Resnik, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the ACM 1994 Conference on Computer Supported Cooperative Work*, pp. 175–186. ACM.

Sewell, W., & Shah, V. (1968). Social class, parental encouragement, and educational aspirations. *American Journal of Sociology*, *73*, 559–572.

Tresp, V., & Hofmann, R. (1998). Nonlinear Markov networks for continuous variables. In *Advances in Neural Information Processing Systems 10*, pp. 521–527. MIT Press.

Whittaker, J. (1990). *Graphical Models in Applied Multivariate Statistics.* John Wiley and Sons.