Naive Bayes Models for Probability Estimation

Daniel Lowd
lowd@cs.washington.edu
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350 USA

January 17, 2005

Abstract

Naive Bayes models have been widely used for clustering and classification. However, they are seldom used for general probabilistic learning and inference (i.e., for estimating and computing arbitrary joint, conditional and marginal distributions). In this paper we show that, for a wide range of benchmark datasets, naive Bayes models learned using EM have accuracy and learning time comparable to Bayesian networks with context-specific independence. Coupled with their linear inference time, this makes them a very attractive alternative, particularly in large domains.

1 Introduction

Naive Bayes models have become quite popular for classification and clustering due to their simplicity, efficiency, and accuracy. The naive Bayes model is simple due to its "naive" assumption that each input variable, $X_1, X_2, \ldots, X_n$, is conditionally independent given the class or cluster variable, C. This assumption also permits inference to be done in linear time, with respect to the number of classes and the number of input variables.

Although this conditional independence rarely holds in practice, naive Bayes models tend to perform well, even optimally, in many class-prediction scenarios (Domingos & Pazzani, 1997). Naive Bayes models have successfully been applied to the classification task of spam filtering (Sahami et al., 1998), the clustering task of grouping stars by their spectra (Cheeseman & Stutz, 1995), and many other interesting problems. What these tasks have in common is an important class or cluster variable: in spam filtering, we wish the know the grouping of email into spam and non-spam; in clustering stars, we may not know the number of clusters ahead of time, but we wish to discover the clusters themselves and assign new stars to them.

In other machine learning problems, we wish to estimate the full joint probability distribution over n input variables, $X_1, X_2, \ldots, X_n$. We may then ask for arbitrary marginal and conditional distributions over any of these variables. For this type of problem, the most popular approach does not use naive Bayes models, but Bayesian networks.

Bayesian networks (Pearl, 1988) represent each variable with a node in a directed acyclic graph. Rather than assuming that each variable is independent given some class variable, they assume that each variable is conditionally independent given its parents in the graph. Bayesian networks utilizing context-specific independence offer additional flexibility, since they let independencies depend on specific values of the parents. While selecting the optimal factorization is NP-hard (Chickering et al., 1994), good models can be learned relatively efficiently.

While Bayesian networks seem more flexible and less simplistic, they suffer one critical weakness: exact inference is #P-complete. In other words, although good models may be learned quickly, using them may take exponential time. To compensate for this weakness, approximate inference is often used instead. The two most popular methods are Gibbs sampling and belief propagation.

Gibbs sampling is easy to implement and guaranteed to converge to the correct distribution. Unfortunately, it could take a great many iterations before it converges. Furthermore, detecting convergence is very difficult: convergence criteria exist, but they come with no guarantees.

Belief propagation tends to converge much more quickly, after only a few iterations. Unfortunately, convergence is not guaranteed, and even if it does converge, it may not converge to the correct distribution. Finally, belief propagation in its simplest form does not compute arbitrary joint distributions, only the marginal distribution of each unspecified variable. Generalized belief propagation can answer more complex queries, but is much more computationally expensive (Yedidia et al., 2003).

Each of these techniques is applicable to some problems: Gibbs sampling where robustness is a priority, belief propagation where speed is a priority. Unfortunately, there will always be applications for which neither is satisfactory. Consider the task of inferring which products or web pages would interest different web site visitors. Gibbs sampling would be too slow to accomodate visitors in real-time, especially on a busy site that could produce hundreds of queries per second. Belief propagation might be fast enough, but with no guarantee of accuracy.

Clearly, Bayesian networks have their limitations. Even if Gibbs sampling or belief propagation could apply to a particular application, the modeler would still have to choose when to halt Gibbs sampling, or implement belief propagation and verify its accuracy.

The solution to all of this is not another inference method, but another model. In this paper, we present a naive Bayes mixture model where exact inference is linear in model size, rather than exponential in the number of variables. We further demonstrate that learning time and accuracy are both comparable to state-of-the-art Bayesian network methods. A final advantage of naive Bayes is simple implementation.

1.1 Related Work

While we demonstrate empirically that naive Bayes models can be effective at general probabilistic learning and inference, the knowledge that they could be used in this manner is not new. The following related work illustrates this.

The AutoClass system (Cheeseman et al., 1988) is a naive Bayes model used for clustering, but clustering can be viewed as a special case of joint probability estimation. Both AutoClass and our proposed model learn joint distributions from data using expectation maximization (EM) (Dempster et al., 1977). The main differences are in stated goal (cluster discovery versus general estimation and inference) and in details of the learning algorithm.

(Breese et al., 1998) applied a naive Bayes cluster model similar to AutoClass to collaborative filtering. Unlike many other applications of clustering, the goal was to infer user preferences, not to discover clusters. This therefore represents an application of naive Bayes to general probabilistic learning and inference.

The Mixture of Trees (MT) model (Meila & Jordan, 2000) is a mixture model in which each component is a tree-structured Bayesian network. This allows exact inference to be done in time $O(kn^2)$, where n is the number of variables and k is the number of mixture components. Our work is also a mixture model for joint inference, but we employ the simpler naive Bayes model, assuming that variables are wholly independent given the component.

1.2 Outline

The remainder of our paper is organized as follows. In Section 2, we give additional background on Bayesian networks. Section 3 covers the naive Bayes estimator (NBE) model for learning arbitrary joint distributions. In Section 4, we describe our experiments evaluating the accuracy and inference time of naive Bayes models relative to other Bayesian networks. Sections 5 and 6 contain discussion, conclusions, and future work.

2 Bayesian Networks

A Bayesian network utilizes the notion of conditional independence to compactly represent a joint probability distribution over a set of variables, $X_1, X_2, \ldots, X_n$ (Pearl, 1988). For this paper, we will assume that all variables are discrete or have been discretized. A Bayesian network is typically represented as a directed, acyclic graph with one node per variable. The graph encodes the assertion that each variable is conditionally independent of its non-descendents given its parents in the graph. For each variable, there is also a conditional probability distribution (CPD) that describes the distribution of that variable for each configuration of its parents. Together, the CPDs completely describe the joint probability of all variables:

$$P(X_1, X_2, \ldots, X_n) = \prod_i P(X_i | Parents(X_i))$$

The most popular type of CPD is the conditional probability table. A conditional probability table provides one distribution for the target variable for each possible configuration of its parent variables. A table therefore takes exponential space in the number of parent variables of the target node. Decision trees reduce the space required by exploiting context-specific independence, allowing distributions to be reused for multiple configurations of the parent variables. (Friedman & Goldszmidt, 1996; Chickering et al., 1997) have found that decision trees allow more complex models to be stored in less space.

The Bayesian network representation has several advantages. First, it represents a joint probability distribution among many variables in relatively little space. To store a probability table describing the likelihood of each configuration of all variables would require exponential space. A Bayesian network, in contrast, can deal with n much smaller distributions consisting of each variable and its parents.

Second, the means of factorization may be a good fit for many problems. By thinking of a variable's parents as "causes", we can come up with many independencies: consider a medical application, in which a set of risk factors cause a set of diseases, which in turn cause a set of symptoms. A corresponding Bayesian network would have arcs from each risk factor to each related disease, and from each disease to each related symptom. If each disease depends on only a few risk factors, and each symptom depends on only a few diseases, then the local probability distributions will be relatively small. This model nonetheless encodes much of our knowledge about the domain in a fairly intuitive manner.

2.1 Learning

When the graph structure of a Bayesian network is known and there is no missing data, each CPD may be learned independently from data. When the structure is unknown, we must search for a good structure. Although finding the optimal structure is NP-hard (Chickering et al., 1994), good structures can be found using greedy algorithms.

Heckerman et al. (1995) describe a widely used hill-climbing algorithm for learning Bayesian network structure and parameters when the CPDs are tables. The algorithm starts with some initial graph structure and considers, at each step, all individual arc additions, deletions, or reversals that maintain an acyclic graph. These modifications are evaluated using the Bayesian Dirichlet score, which integrates the likelihood of the training data over all possible parameters, given the structure. Structures are weighted by a structure prior to incorporate prior knowledge or prevent overfitting to excessively complex structures.

The process is similar when decision trees are used as CPDs, but with some modifications as described by (Chickering et al., 1997). A related algorithm is implemented in Microsoft Research's WinMine toolkit, which we used for our experiments (Chickering, 2002).

2.2 Inference

Exact inference in a Bayesian network is #P-complete. Because of this, any exact inference algorithms will only be tractable on small or specially structured Bayesian networks. We therefore focus on approximate inference algorithms, of which the most popular are Gibbs sampling and belief propagation.

Gibbs sampling repeatedly samples each variable in the network conditioned on its Markov blanket (Gilks et al., 1995). If the distribution is positive, i.e. no configuration has a probability of zero, then the set of samples generated is guaranteed to converge to the true joint distribution of the network. In other words, although two consecutive samples may be highly correlated, the unordered set of all samples eventually approximates a set of i.i.d. samples from the joint distribution described by the Bayesian network. Conditional and marginal probabilities may therefore be computed by counting the frequency of specific configurations among the many samples. By setting evidence variables to fixed values, we may compute conditional probabilities as well.

Belief propagation is a message-passing algorithm originally used to perform exact inference on tree-structured Bayesian networks. Each node in the network receives messages from its neighbors about its expected distribution,

and each node broadcasts messages about the expected distributions of its neighbors (Yedidia et al., 2003). In this manner, local information eventually propagates through the entire network. Unfortunately, inference is no longer exact and convergence is no longer guaranteed in Bayesian networks with loops. Even so, belief propagation has been found to be useful in practice, even in very "loopy" graphs.

Generalized belief propagation groups nodes into clusters to increase the accuracy and flexibility of belief propagation. While ordinary belief propagation only provides the marginal distribution of each variable, generalized belief propagation gives the joint distribution in each cluster. See (Yedidia et al., 2003) for more details on belief propagation and generalized belief propagation.

3 Naive Bayes Probability Estimation

As a simpler, faster alternative to Bayesian networks, we propose Naive Bayes Estimation (NBE), a naive Bayes mixture model for general purpose probabilistic learning and inference. We begin with additional background on naive Bayes in general, since our probabilistic model follows the naive Bayes schema. What makes NBE different is the application to joint probability estimation, rather than clustering or classification. We then present our learning algorithm and briefly discuss inference.

3.1 Naive Bayes

Naive Bayes models are so named for their "naive" assumption that every variable X_i is conditionally independent, given the class or cluster variable C. By making this assumption, the probability distribution may be efficiently represented as the product of many simpler distributions:

$$P(X_1, X_2, \ldots, X_n, C) = P(C) \prod_{i=1}^{n} P(X_i|C)$$

A naive Bayes model is a special case of a Bayesian network over the variables $X_1, \ldots, X_n$ and C, in which the parent of each X_i is C and C has no parents. The efficiency gains over general Bayesian networks come from using this restricted structure.

One of the most common uses of naive Bayes is classification, in which C represents the class to be predicted and the X_i variables represent the observable attributes. For example, in a naive Bayes spam filter, C could represent the classes "spam" and "non-spam", and X_i could represent whether or not the ith word is present in the candidate email. The component distributions can be learned from training data by simply counting word occurrences in labeled spam and non-spam. In this example, as in many applications of naive Bayes, the "naive" independence assumption is clearly false: the words present in an email do depend on each other, both in spam and in legitimate email. Nevertheless, such models tend to perform quite well in practice, and theoretical analysis has shown naive Bayes to be optimal under 0-1 loss in many cases (Domingos & Pazzani, 1997).

Naive Bayes is also used for clustering, in which the goal is to learn the underlying structure of the data. Unlike the classification task, C represents a cluster variable which is never directly observed. In most cases, even the number of clusters present is unknown and must be inferred. The AutoClass system is one example of a naive Bayes clustering system, which has been used to automatically discover classes among star spectra, DNA introns and exons, and LandSat data (Cheeseman & Stutz, 1995).

Clustering models are typically learned using expectation maximization (EM), an iterative process consisting of two alternating steps (Dempster et al., 1977). In the expectation step (E-step), data points are fractionally assigned to clusters using the partially learned model. In the maximization step (M-step), the cluster distributions are updated to maximize the likelihood of the training data. By iterating these two steps, the model is guaranteed to converge to a local maximum in likelihood.

NBE is the application of naive Bayes models to the problem of general probabilistic learning and inference. As in the clustering problem, we have a cluster variable that is never observed and must be learned using EM. In fact, the clustering problem is really a special case of general learning, in which the primary goal is to discover and interpret the structure of the data. In NBE, the classes or clusters indexed by C are typically referred to as *components*, since their mixture composes the entire probability distribution.

Procedure TrainNBE
Input: training set T, hold-out set H, initial number of components k_0, and convergence thresholds δ_{EM} and δ_{Add}.

Initialize M with one component.
$k \leftarrow k_0$
repeat
  Add k new mixture components to M, initialized using k random examples from T.
  Remove the k initialization examples from T.
  repeat
    E-step: Fractionally assign examples in T to mixture components, using M.
    M-step: Adjust parameters of M to maximize the likelihood of this fractional assignment.
    If $\log P(H|M)$ is the highest we've seen so far, save M in M_{best}.
  until $\log P(H|M)$ fails to improve by ratio δ_{EM} over the last iteration.
  $k \leftarrow 2 \times k$
until $\log P(H|M)$ fails to improve by ratio δ_{Add} over the last iteration.
Execute E-step and M-step twice more on M_{best}, using examples from both H and T.
Return M_{best}.

Table 1: NBE iterative learning algorithm.

3.2 Learning

In this section, we present our algorithm for training an NBE model from data. The NBE model is learned by iterating two steps: extension (adding mixture components) and refinement (training with them until convergence). The extension step is important both for initializing the model and for adding more mixture components to a partially learned model. By adding components as we learn the model, we avoid needing to train the model multiple times with different numbers of initial components, the approach taken by AutoClass (Cheeseman et al., 1988). The refinement step uses EM to find the best parameters for those components and prunes low-weight components to speed up the search. In each iteration, we double the number of components added to the model so that our expected running time is within a constant factor of the optimal running time, the running time were we to know the optimal number of components before starting.

We test for convergence and avoid overfitting by using a hold-out set, both in the refinement step and the overall process. These examples are only processed by the learning algorithm at the very end, when two final iterations of EM are run with all of the training data, held-out and regular.

Pseudocode is listed in Table 1. In the following sections we describe the extension and refinement steps in greater detail.

3.2.1 Model Extension

Model extension begins by randomly selecting k points from the training set. These points are analogous to initial cluster centers. The algorithm generates components from the initial points by inducing a probability distribution for each variable, given the component. We combine the single observation of a random point with a low-weight Dirichlet prior of the variable's overall marginal distribution, as estimated from the entire training set. In our tests, we set the weight of the Dirichlet prior to be 0.1 equivalent counts. This yields a probability of $(1.0 + 0.1 \times P(x_i))/(1.0 + 0.1)$ for the random point's original variable value, x_i, and a distribution of $0.1 \times P(x_j)/(1.0 + 0.1)$ for all other variable values x_j, where $P(x_m)$ is the relative frequence of the variable value x_m in all training data. Though not covered in our experiments, we can handle continuous variable distributions as well. These are initialized as Gaussians with the same mean as the center and a variance equal to a fixed fraction of the variable's global variance.

We believe that this technique makes sense in general because a random sampling of points is more likely to include points from regions of high probability, where there ought to be clusters. The Dirichlet prior and fractional variance allow other slightly different points to be probabilistically included in the component.

Points used for initializing components are removed from the training data considered by the refinement step in order to avoid overfitting to those points.

3.2.2 Model Refinement

In the refinement step, we iterate the EM algorithm until these initial component distributions converge to a local likelihood maximum. After each step of EM, the model is evaluated using the hold-out set. If the log likelihood of the hold-out set given the model is the highest so far, then we cache the current model. If the log likelihood of the hold-out set decreases, or increases by less than the fraction δ_{EM}, the refinement step ends. If the log likelihood decreased, then we revert to the cached model. This protects us from ending up with a worse model by running too many steps of EM.

After every five steps of EM, as well as at the end of the refinement step, we prune low-weight components. We accomplish this by sorting all mixture components, c_i, by their relatively frequencies, $P(c_i)$, and keeping the first components whose total probabilities sum to at least 99.9%. Since each step of EM takes time linear in the number of components, pruning can speed up learning significantly. However, it also imposes an indirect limit on model complexity: any model with 1,000 or more components will lose at least one component in this step. If the best model for the data uses more than 1,000 components, the pruning step would yield a worse model. This can be avoided by increasing the pruning threshold appropriately for the dataset.

When an entire refinement step passes with little or no improvement on the hold-out set, we run two final steps of EM on the best model with the held-out data included and terminate the entire learning procedure.

3.2.3 Other Learning Methods

In addition to the procedure described above, we experimented with an alternate method for selecting the number of components to use. The main difference is that instead of adding components as the model is learned, we pick the optimal number of components to start with using the hold-out data.

For any given dataset, we partially trained a model with 10 random components, then another with 20, then 40, etc. Mixture components were initialized using points in the training set, as before. For each of these models, we performed early stopping, running only six iterations of EM rather than training to convergence. Early stopping can perform very effective model selection in much less time, as discussed in (Meek et al., 2001). When the log likelihood of the hold-out data given the model with $2k$ components is worse than given the model with k components, we pick k as the optimal number of components. Having selected the number of components to use, we finally train a model to convergence using that k, once again using the hold-out data to test for overfitting and running two final iterations of EM.

We found that this alternate method yields models of equivalent accuracy, but takes significantly longer. Our hypothesis is that adding components iteratively allows the "easy" components to be learned faster, since there are fewer components to start out with. Additional subtleties may be captured by additional components, but these will require less training given a partially trained model. While the final number of components may be the same, most of the EM iterations can be run when there are fewer components.

In general, we believe that any effective method for training mixture models could be applicable here. We simply chose a procedure that we found to be fast and effective.

3.2.4 Time Complexity

The time complexity for either of the above procedures is $O(ikd|T|)$, where i is the number of EM iterations run, k is the total number of components considered, d is the number of variables, and $|T|$ is the numbers of training examples (including the hold-out set). The number of EM iterations required will depend on the dataset being modeled and the convergence criteria being used. Unfortunately, there are no analytical upper bounds on the number of iterations that may be required.

A number of methods exist to speed up EM under certain circumstances. We observed faster learning by adding components incrementally, thus reducing k for the first iterations of EM. Pruning components with low weight also sped up the algorithm. Partial E-steps and lazy EM may provide additional speed-ups for large datasets, as described

in (Thiesson et al., 2001). Alternately, an optimization technique such as conjugate gradient ascent or quasi-Newton methods could require substantially fewer iterations than EM (Press et al., 1993).

3.3 Inference

Since NBE is still a naive Bayes model, the inference methods we present are not new and apply to any naive Bayes model.

For marginal queries, let X be the set of query variables, C be the k-state component variable, and let Z be the set of all other hidden variables. (We assume that the component variable is never part of a query, since it is never observed in the training data.) Let x refer to a fixed state of the query variables. We can compute the marginal probability directly by summing out over all possible states of C and Z:

$$P(X = x) = \sum_{c=1}^{k} \sum_{z} P(X = x, Z = z, C = c)$$

By the naive Bayes assumption, every other variable is independent, given C. Using this fact, we may now express the probability in terms of the learned distributions $P(C)$ and $P(X_i|C)$:

$$P(X = x) = \sum_{c=1}^{k} \sum_{z} P(c) \prod_{i=1}^{|Z|} P(z_i|c) \prod_{j=1}^{|X|} P(x_j|c)$$

Because $\sum_z P(z_i|c)$ is always 1, the variables in Z do not affect the probability computation and may safely be ignored:

$$P(X = x) = \sum_{c=1}^{k} P(c) \prod_{j=1}^{|X|} P(x_j|c)$$

Since this expression is the product of k sums, each sum containing $|X| + 1$ terms, the total time required to compute $P(X = x)$ is $O(|X|k)$. Note that naive Bayes inference time is constant in the number of hidden variables, while exact Bayesian network inference is worst-case exponential in the number of hidden variables.

Conditional probabilities can be computed efficiently as ratios of marginal probabilities. Let Y refer to the set of conditioning variables in a conditional query, and let y refer to a fixed state of Y.

$$P(X = x|Y = y) = \frac{P(X = x, Y = y)}{P(Y = y)}$$

The marginal probabilities in the numerator and denominator can be computed in times $O(k(|X| + |Y|))$ and $O(k|Y|)$, respectively. Therefore, the conditional probability $P(X = x|Y = y)$ can be computed in time $O(k(|X| + |Y|))$. Note that our running time remains independent in the number of hidden variables.

Since a full probability distribution over multiple variables specifies a probability for each of an exponential number of states, storing these probabilities would take exponential space and time. We can, however, efficiently generate a reduced naive Bayes model that incorporates all of the evidence. In the reduced model, all variables from Y have been removed and the evidence in y has been incorporated by adjusting the component weights, $P(C)$. This is the approach we adopt in our experiments with multiple query variables.

4 Empirical Evaluation

In this section, we describe an empirical evaluation of naive Bayes models in terms of learning time, the time required to learn a probabilistic model from data; modeling accuracy, the extent to which a learned model predicts a test set; and inference time, the rate at which a model can answer queries.

4.1 Datasets

In order to gauge the overall effectiveness of our model on real data, we selected 50 datasets, including 47 from the UCI machine learning repository (Blake & Merz, 1998), a subset of the EachMovie dataset, the Jester collaborative filtering dataset, and a subset of the KDD Cup 2000 data. Basic statistics for each dataset are given in Table 2.

The UCI datasets range in theme from census data to chess endgames, in variables from five to 618, and in size from 57 to 67,000 examples. Many of the UCI datasets were designed for classification rather than joint probability estimation, but they nonetheless provide a variety of real-world multivariate datasets. When present, we did not treat the class variable any differently from the other variables, and so learned joint distributions over all variables. Furthermore, the NBE models learned were significantly different from naive Bayes classification models, since we typically found many more mixture components than classes. For example, Letter Recognition features 26 classes but NBE learned a model with over 600 components.

The EachMovie dataset is a set of movie ratings used for testing recommender systems, collected by Compaq corporation over an 18-month period. We used a 10% subset to speed up learning. We additionally decreased sparsity by transforming the zero to five movie rating scale into boolean "like"/"dislike" ratings, where a rating of four or five implied "like". The Jester dataset is also used for testing recommender systems, but consists of ratings for 100 jokes rather than 1,648 movies. KDD Cup 2000 data consists of clickstream and purchase data for a now-defunct online retailer (Kohavi et al., 2000).

These three datasets are examples of real-world problems where joint probability estimation is the primary goal. With movie ratings, for example, the task is to predict other, unrated movies that each user is likely to enjoy. This can easily be inferred from a probability distribution but is a harder classification task, since each movie would have to be treated separately.

Each dataset was partitioned into a training set and a test set. If a standard partition already existed, we used it. Otherwise, we assigned examples to the test set with probability 0.1. On datasets with fewer than 2,000 examples we performed 10-way cross-validation rather than using a single pair of training and test sets.

For each training set, we also selected a standard hold-out set. This data was used by all algorithms to prevent overfitting. In the case of cross-validated datasets, one hold-out set was generated for each of the 10 splits. Training examples were randomly selected for inclusion in the hold-out set with a probability that depended on the overall size of the training set. For training sets with fewer than 3,000 examples, we used one third of the training data; for more than 10,000 examples, we used one tenth; for in-between sizes, we used 1,000 examples.

All continuous variables were discretized. Our discretization method used five states and chose value cut-offs that equally partitioned the non-missing training values among those five states, whenever possible. In the case of continuous variables that were predominantly of one value (e.g., 90% of the values were 0.0), this actually resulted in fewer useful states (e.g., two states – zero vs. all values greater than zero). As with other discrete variables, missing values were assigned to a separate state.

Although continuous variables were discretized for our experiments, NBE can represent them directly by replacing the multinomial models with Gaussians. Since NBE is a mixture model, the effective distribution of any continuous variable becomes a mixture of Gaussians. Except for component initialization, learning is identical to the discrete case. WinMine can also handle continuous variables, using Gaussian or log Gaussian models.

Unfortunately, Gibbs sampling and belief propagation do not readily accomodate continuous variables. In Gibbs sampling, continuous samples must be discretized or smoothed to handle the inifinite number of continuous configurations. Belief propagation assumes compact representations of probability distributions, which are difficult to do in the continuous case. We therefore avoid continuous variables due to inference complications with Bayesian networks, not due to any limitations inherent in NBE.

4.2 Training

NBE models were trained as described in Section 3, with the following additional details. In our experiments, we used 50 initial components for training sets of over 1,500 points and 10 initial components for smaller training sets.

For our Bayesian network implementation, we chose Microsoft Research's WinMine Toolkit (Chickering, 2002) because it is freely available, easy to use, and represents the state-of-the-art in Bayesian network learning algorithms. One of its greatest strengths is that it uses decision trees, rather than conditional probability tables (CPTs), as its local

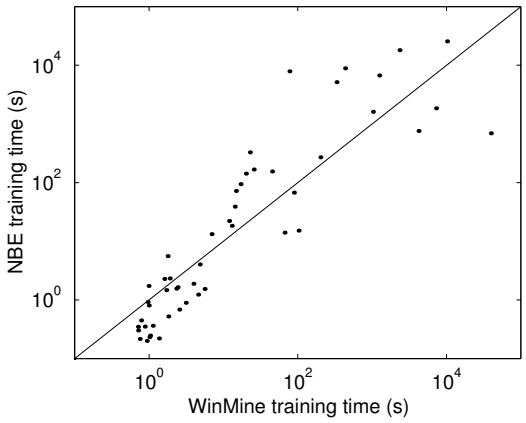

Figure 1: Learning times for Bayesian networks (using the WinMine toolkit) and NBE. Each point represents one of the 50 benchmark datasets considered in our experiments.

probability models at each node. Using decision trees permits context-specific independence as well as conditional independence: the independence of two variables may be contingent on the observed value of a third variable. The practical advantage is that more relationships can be learned with less space, since decision trees don't need to store a separate probability for every combination of the parent variables.

In its learning, WinMine does not attempt to infer missing data values; instead, it models the probability of a variable value being missing as part of its distribution. We configured NBE to handle missing values the same way.

WinMine avoids overfitting via the parameter κ, the penalty for each free parameter in the model. The default value for κ is 0.01, but different datasets do better with larger or smaller values. Since we use a hold-out set to avoid overfitting in NBE, it seemed appropriate to use the same hold-out set for automatically tuning κ. We began with a conservative value of 0.0001, which learns very simple models, and iteratively increased it by factors of 10. In each iteration, we trained a model using the training set (with the hold-out set removed), and tested it on the hold-out set. When the log likelihood of the hold-out set began to decrease, the process was halted. We then trained a model with the best-known value of κ, using all training data including the hold-out set.

As a simple baseline algorithm, we used the marginal distribution of each variable. This is equivalent to a naive Bayes model with only one component.

We also tried running the AutoClass system (Cheeseman et al., 1988), but found its accuracy to be worse than the marginal distribution in many cases, probably because it was designed and tuned for class discovery rather than general inference. Furthermore, the process of repeatedly learning and relearning models with different numbers of components resulted in very long training times.

4.3 Learning Time

We measured the learning time for both methods on all datasets, running under Windows XP on a 2.6 GHz P4 with 1 GB of RAM. For most datasets, learning times were within an order of magnitude of each other; sometimes WinMine was faster, sometimes NBE was faster. Overall, neither model had a clear advantage in learning times. Learning times for each model on each dataset are displayed graphically in Figure 1.

If both models were required to learn with missing data, rather than making "missing" a distinct value, these results might look very different. Because its inference is fast, NBE can readily handle missing data. In contrast, Bayesian networks take a substantial performance hit from doing the same, due to comparatively slow approximate inference. We further investigate differences in inference time in the following section.

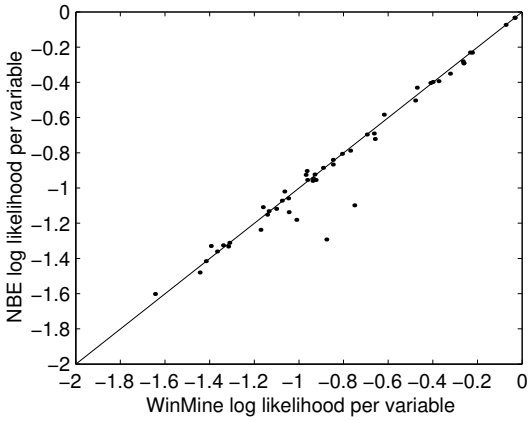

Figure 2: Comparative average log likelihoods for all benchmark 50 datasets.

4.4 Modeling Accuracy

To evaluate the accuracy of NBE models relative to Bayesian networks with context-specific independence, we trained models for each dataset using each method and compared the log likelihoods of the test data given each model. For cross-validated datasets, we averaged log likelihoods over the 10 train/test splits. Results for each algorithm on each dataset are given in the last three columns of Table 2. All log likelihoods were computed using the natural logarithm.

NBE models demonstrated greater accuracy than those generated by WinMine for 23 of the 50 datasets. Using a two-tailed paired t-test ($p = 0.05$), NBE performed significantly significantly better on 15 datasets and WinMine performed better on 22. Though many of the differences appear statistically significant, the actual differences in accuracy were typically very small. Figure 2 compares the average log likelihoods for NBE and WinMine on all 50 datasets, after dividing each log likelihood by the number of variables in its dataset. WinMine does much better on the Isolated Letter Speech and Musk datasets, but the others are very close. While NBE showed no clear advantage here, neither did WinMine: both remained competitive.

4.5 Query Speed and Accuracy

In many applications, we are interested in inferring the distribution of one or more variables, possibly conditioned on some evidence. For example, we may wish to determine the probability of a set of diseases, given a set of symptoms as evidence. For such applications, we are interested in the speed and accuracy of marginal and conditional queries.

It is in this area that we expect NBE to shine, since it can perform exact inference in linear time, with respect to model size. Exact inference in Bayesian networks, however, can take exponential time, with respect to the number of variables in the model. Therefore, we implemented the most popular approximation algorithms for Bayesian networks, Gibbs sampling and belief propagation. All algorithms (including NBE inference) were implemented in C++, profiled, and optimized for speed.

For Gibbs sampling, we tried many different configurations before settling on a few that offered the best speed / accuracy tradeoffs. First, we explored increasing the iterations until accuracy ceased to improve; of course, this was just a preliminary hack, since it involved monitoring our actual score on the test data. We tried to improve on this by running 10 randomly initialized chains, testing for mixing using the Gelman diagnostic (Gilks et al., 1995), and sampling until we had 95% confidence that each probability was within 5% of its true value, as described in (DeGroot & Schervish, 2002). Oddly enough, these methods ran only the minimum number of iterations in many cases, suggesting that these datasets may be easy for Gibbs sampling, or that the convergence diagnostics may imperfectly detect convergence.

It was very difficult to select a scheme that would be fair in terms of both speed and accuracy – if we ran too few iterations, Gibbs sampling would do unnecessarily poorly, but if we ran too many, Gibbs sampling would take unnecessarily long. In the end, we decided to use fixed numbers of iterations because the method is simple, interpretable,

and not subject to subtle interactions between datasets and convergence diagnostics. We ran three Gibbs sampling scenarios: a single chain with 100 burn-in iterations and 1,000 sampling iterations per chain; 10 separate chains, again with 100 burn-in iterations and 1,000 sampling iterations; and 10 chains with 1,000 burn-in iterations and 10,000 sampling iterations per chain. Each iteration consisted of sampling every hidden variable. Further increasing the number of iterations did not help significantly.

We Rao-Blackwellized our Gibbs sampler, so each sample gave partial counts to several states, rather than a single count to the chosen state. Each time we sampled a k-state variable x, we computed the probability that x equaled each of the k states, using Markov-blanket inference. In ordinary Gibbs sampling, one of these states would be chosen at random, and one count would be given to the new configuration. In our Rao-Blackwellized version, we gave fractional counts to each of the k possible configurations that could result from sampling this variable, corresponding to their probabilities. This yields increased accuracy with fewer samples.

In the single-variable case, Rao-Blackwellization gives fractional counts to all states in each iteration. When there are multiple query variables, it is still possible for a state to receive zero counts. To avoid zero probabilities, we gave each configuration a fractional prior count, chosen so that all prior counts summed to one. For example, in a query over 100 possible configurations, we gave each state an initial count of 0.01.

We implemented the standard belief propagation algorithm (Yedidia et al., 2003) over a Bayesian network by first converting it into a pairwise Markov network. Unfortunately, when the decision tree local models are converted to potential function matrices, this can yield an exponential blow-up in size. Belief propagation thus failed on many of the larger datasets, where it would have required gigabytes of memory to even store the pairwise Markov network. In some cases, we were able to compensate by applying belief propagation to a simpler model. We generated simpler models by training alternate networks using smaller values of κ, down to 0.0001. If belief propagation would not run on the original Bayesian network in under 1 GB of RAM, we used the most complex network where it could. In this manner, we were able to apply belief propagation to 40 of the 50 datasets.

Belief propagation offers no guarantee of convergence or correctness, and in its standard form, only computes the conditional probabilities of single variables. For computing joint probabilities among several variables, Gibbs sampling or some form of generalized belief propagation is required. Therefore, we omit belief propagation from those experiments. As a convergence criterion, we stopped when no probability at any node changed by more than 1% from one iteration to the next. Decreasing this threshold to 0.1% did not significantly change our results.

4.5.1 Marginal Queries

We first compared inference speed and accuracy for marginal queries of one to five variables. Since belief propagation only computes distributions of single variables, we omit it from this experiment.

For each dataset, we queried the marginal distributions of random sets of one to five variables. For each marginal query, we used the log likelihood of one test example to assess the accuracy of each inference method. By averaging over all queries, we computed the expected log likelihood of a random marginal query. In hindsight, it would be more accurate to test each marginal query using all test examples, but we made this realization too late to rerun the experiments. Even so, the methods are asymptotically equivalent: given infinite test examples and infinite queries, both methods would eventually converge to the true expected value. For inference time, we compute the average time required to compute a single marginal distribution.

Because of the slowness of Gibbs sampling, we never used a test set of more than 1,000 examples per dataset. When running Gibbs sampling with 10 chains of 10,000 sampling iterations, we only had time for 100 examples per dataset. In this setup, Gibbs sampling took between 1 second and half an hour per marginal query. In contrast, NBE never averaged more than 3 milliseconds per query.

Results for both speed and accuracy are displayed at the top of Figure 5. NBE is more accurate than the most accurate version of Gibbs sampling, and the least accurate version of Gibbs sampling is less accurate than NBE on almost every dataset. While additional iterations of Gibbs sampling might increase accuracy, Gibbs sampling is already vastly slower than NBE: the fastest version is 100 to 150,000 times as slow, and the slowest version is 6,000 to 188 million times as slow.

We were also surprised to discover that NBE was always more accurate on Musk and sometimes more accurate on Isolated Letter Speech, the two datasets for which WinMine had the clearest advantage. Furthermore, these two datasets were the slowest for Gibbs sampling, requiring half an hour per query for Isolated Letter Speech and 10

minutes per query for Musk. This suggests that even if a Bayesian network models a particular dataset much better, inference time may be prohibitive and still yield worse results than NBE.

4.5.2 Single Variable Conditional Queries

We then compared inference speed and accuracy for single variable queries conditioned on varying amounts of evidence. This is easier for Gibbs sampling in two ways. First, the evidence given to conditional queries reduces the number of variables that Gibbs sampling must sample in each iteration, and greatly reduces the size of the space for Gibbs sampling to explore. This will tend to reduce Gibbs sampling's running time while increasing its accuracy. Second, by only querying single variables, Gibbs sampling has fewer configurations to keep track of than in the multivariate case. This increases the expected counts per configuration and avoids configurations with no counts at all. Querying single variables also allows us to try the alternate inference method of belief propagation.

We generated the queries for each dataset by hiding one to five random variable values from examples in the corresponding test set. Each inference method was then used to determine the distribution of each hidden variable. Since we know the true value of each hidden variable, we were able to estimate each inference method's accuracy as the average log likelihood of the true values. The average log likelihood in this case is a crude approximation of the Kullback-Leibler divergence. (We also ran experiments with more hidden variables, but the trend remained the same.)

Figure 5 shows our results for both timing and accuracy, in the center graphs. Our analysis follows.

Belief propagation was faster than NBE inference on three or four datasets (four datasets when three variables were hidden; three otherwise). In spite of several fast datasets, its overall performance remains mediocre due to the overhead incurred by the large state spaces of the generated Markov network. At its worst, it averaged 45 seconds per query with five hidden variables on the Shuttle dataset. A more efficient representation might yield good speed-ups, but this is beyond the scope of our investigation. In terms of accuracy, belief propagation did well enough on the datasets where it could be run. Its failure to run on 10 of the 50 datasets, however, is a critical weakness. Overall, NBE was more accurate and much faster most of the time.

Gibbs sampling is much faster with conditional queries than with marginal queries, but remained much slower than NBE: to take a mere 1,000 samples of five variables took up to 1,400 times as long as NBE's exact inference. Running any more iterations of Gibbs sampling takes even longer. On the Letter Recognition dataset, 10 chains of 10,000 iterations took Gibbs sampling 23 seconds per query with five hidden variables, while NBE never averaged more than 30 milliseconds per query.

Gibbs sampling had better accuracy than belief propagation, always winning on a majority of the datasets and failing on none. We were surprised to find that this held true even with a single chain of 1,000 sampling iterations. The wins were not overwhelming. Figure 3 compares NBE and Gibbs sampling in terms of log likelihood with five variables hidden. (Graphs with one to four variables hidden are very similar.) As in overall accuracy, most log likelihoods are very close, with a few outliers (Isolated Letter Speech, Forest Cover Type, Musk, and Breast Cancer Wisc.) that favor Bayesian networks. Figure 3 also shows that increasing the number of chains from one to 10 and the sampling iterations from 1,000 to 10,000 only affected two of the 50 datasets. This suggests that Gibbs sampling for conditional queries on most of the datasets converges quickly.

4.5.3 Multiple Variable Conditional Queries

We then tested the speed and accuracy of Gibbs sampling and naive Bayes inference for inferring conditional distributions over several variables. For each dataset, we hid one to five random variables from the test examples and asked for the joint distribution of these hidden variables. All hidden variables were query variables. This represents a greater challenge for Gibbs sampling than the single variable case, since it must collect counts for all joint configurations.

Our implementation of NBE inference was as described in Section 3: from the evidence, we built reweighted mixture models to model the conditional joint distributions of all unseen variables. Since we could do this in linear time (with respect to model size), there was no need to do any approximation at all.

Our speed and accuracy results are displayed at the bottom of Figure 5. As in the single variable experiments, Gibbs sampling typically took several orders of magnitude longer than NBE. At its worst, Gibbs took 28 seconds per five-variable query on Jester, while NBE never averaged more than 12 milliseconds per query. This is not surprising: NBE inference is still linear, and Gibbs sampling still scales with the number of samples taken and the number of

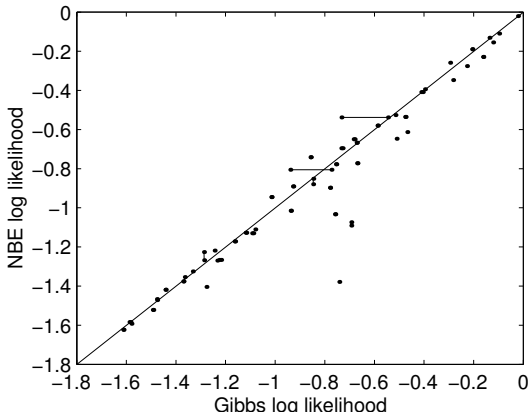

Figure 3: Comparison of NBE and Gibbs sampling log likelihoods for single variable conditional queries with five hidden variables. For each dataset, we plot a pair of points corresponding to two Gibbs sampling configurations: one chain with 1,000 sampling iterations and ten chains with 10,000 sampling iterations each. Each pair is connected by a line, but these lines are only visible when the results of the two experiments are different. This suggests that most single-variable conditional queries converge very quickly.

sampled variables. We further observe that the relative accuracy of Gibbs sampling is somewhat worse, though that is partially remedied by running Gibbs sampling for longer. This is also expected: since multivariate distributions have more states than the single variable distributions, each state will tend to have fewer counts and thus be a less accurate probability estimate. As we increase the number of inferred variables, the situation only gets worse for Gibbs sampling while NBE is unaffected.

We were surprised, however, to discover that Gibbs sampling did even this well. Figure 4 shows relative log likelihoods of Gibbs and NBE with one and five variables hidden. When only one variable is hidden, Rao-Blackwellized Gibbs sampling is equivalent to exact inference. With five variables hidden, the number of possible configurations is huge, so we expect Gibbs sampling to be much less accurate than NBE, which uses exact inference both times. Instead, we find that Gibbs sampling remains competitive in all datasets. This suggests that the conditional inference problems posed by our experiments are not especially challenging ones for Gibbs sampling.

Although Gibbs sampling maintained its accuracy in these scenarios, it remained much slower than NBE. Furthermore, joint queries of greater complexity could be intractable for Gibbs sampling, as the size of the state space increases exponentially with the number of hidden variables. While NBE can readily compute the joint distribution of 1,000 variables, given another 1,000 variables as evidence, Gibbs sampling is simply not equipped to handle this.

5 Discussion

Our experiments show that simple naive Bayes models achieve comparable accuracy to Bayesian networks over a wide variety of benchmark datasets. Furthermore, naive Bayes inference can compute exact probabilities in linear time, with respect to model size. While model size could potentially be exponential, such a model could only be learned from an exponential number of training examples.

For marginal queries, Gibbs sampling's accuracy was much worse while taking many orders of magnitude longer to run. While Gibbs sampling achieved comparable accuracy in the conditional experiments, it remained very slow. Belief propagation, while usually faster than Gibbs sampling, was less accurate and could not be applied to 10 of the datasets.

Neither method is theoretically stronger; either can converge to any distribution, given enough data. Our experiments suggest that naive Bayes models are a good fit for many real-world datasets. Even when a Bayesian network provides greater modeling accuracy, the faster inference time of naive Bayes may more than offset this cost.

It could be argued that Bayesian networks give more insight into the domain, but this is dubious because of their

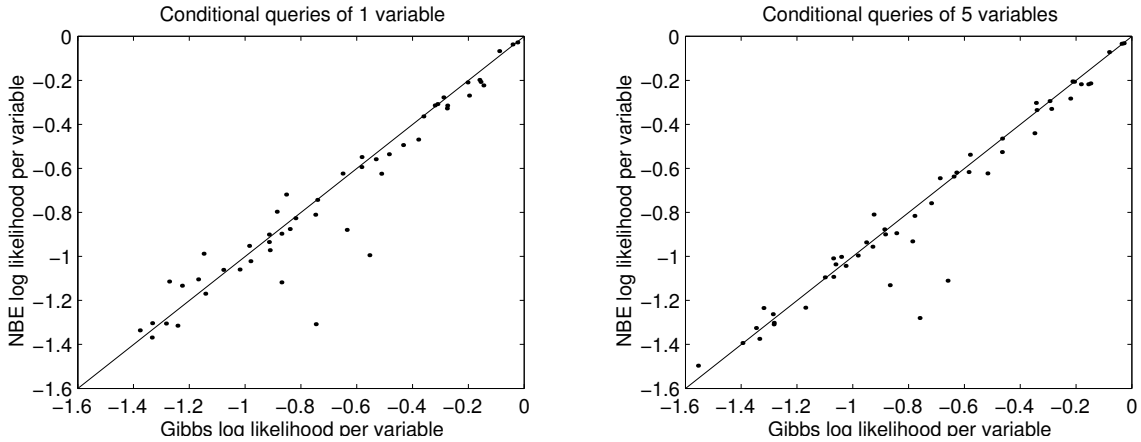

Figure 4: Comparison of NBE and Gibbs sampling log likelihoods for conditional queries of one and five query variables. Gibbs sampling results are from running 10 chains with 1,000 sampling iterations each. Log likelihoods for five-variable queries were divided by five, in order to be more directly comparable to the single variable queries. This demonstrates that NBE and Gibbs sampling achieve comparable accuracies on most datasets for queries of varied complexity.

instability. Furthermore, using a Bayesian network for visualization does not preclude using a separate naive Bayes model for fast, accurate inference.

One final advantage of naive Bayes models is that they are easier to implement than the leading algorithms for learning Bayesian network structure and parameters. Our entire implementation required only 2,500 lines of C++, including whitespace and comments.

6 Conclusion and Future Work

Though usually overlooked, naive Bayes models are an effective model for general probabilistic learning and inference. Our experiments with NBE and WinMine demonstrate that using a simple model does not necessarily yield significantly worse accuracy. Furthermore, NBE offers efficient exact inference. Not only is NBE orders of magnitude faster than the best Bayesian network approximate inference algorithms, it also sidesteps the headache of selecting which inference algorithm to use, along with suitable convergence criteria. NBE gives the flexibility of general probabilistic inference without the headaches that plague Bayesian network implementations.

We see three main directions for future work based on NBE. First, modifications to the model itself might improve accuracy. For example, the mixture-of-trees (MT) model can be seen as an extension of naive Bayes that allows dependencies among some of the input variables, in addition to the component variable (Meila & Jordan, 2000). Unfortunately, no study has yet compared its performance to Bayesian networks on real-world datasets.

A second direction is the application of similar mixture models to relational domains. Since probabilistic relational models feature interactions among many objects, inference tends to be particularly difficult. If an NBE-like model could be applied to relational domains, then its efficient inference could make these problems more tractable.

Finally, NBEs should be applied to specific, real-world domains. While we used real-world data, the types of queries we generated may not reflect any real-world modeling goals. Real-world case studies can reveal domains for which NBE is particularly well suited. Since NBE can efficiently handle queries that would be infeasible for Bayesian networks, NBE can increase the number of tractable domains and the types of queries that may be answered in them.

7 Acknowledgements

The author would like to thank Compaq for providing the EachMovie dataset, Ken Goldberg for providing the Jester dataset, and Blue Martini for providing the KDD Cup 2000 data, as well as all contributors to the UCI data repository. This material is based on work supported under a National Science Foundation Graduate Research Fellowship.

References

Blake, C. L., & Merz, C. J. (1998). UCI repository of machine learning databases.

Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence* (pp. 43–52).

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., & Freeman, D. (1988). AutoClass: a Bayesian classification system. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 54–64). San Mateo, California.

Cheeseman, P., & Stutz, J. (1995). Bayesian classification (AutoClass): Theory and results. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining*, 153–180. AAAI Press/MIT Press.

Chickering, D., Geiger, D., & Heckerman, D. (1994). *Learning Bayesian networks is NP-hard* (Technical Report MSR-TR-94-17). Microsoft Research Technical Report.

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*, 80–89. Providence, RI: Morgan Kaufmann.

Chickering, D. M. (2002). *The WinMine Toolkit* (Technical Report MSR-TR-2002-103). Microsoft, Redmond, WA.

DeGroot, M. H., & Schervish, M. J. (2002). *Probability and statistics, third edition*. Addison Wesley.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, *39*, 1–38.

Domingos, P., & Pazzani, M. J. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, *29*, 103–130.

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

Gilks, W., Richardson, S., & Spiegelhalter, D. (1995). *Markov chain Monte Carlo in practice*. Chapman & Hall/CRC.

Heckerman, D., Geiger, D., & Chickering, D. (1995). Learning Bayesian networks. *Machine Learning*, *20*, 197–243.

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

Meek, C., Thiesson, B., & Heckerman, D. (2001). The learning-curve sampling method applied to model-based clustering. *Journal of Machine Learning Research*, *2*, 397–418. Also in AI and Statistics 2001.

Meila, M., & Jordan, M. I. (2000). Learning with mixtures of trees. *Journal of Machine Learning Research*, *1*, 1–48.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Francisco, CA: Morgan Kaufmann.

Press, W., Teukolsky, S., Vetterling, W., & Flannery, B. (1993). *Numerical Recipes in C*. Cambridge, UK: Cambridge University Press.

Sahami, M., Dumais, S., Heckerman, D., & Horvitz, E. (1998). A Bayesian approach to filtering junk E-mail. *Learning for Text Categorization: Papers from the 1998 Workshop*. Madison, Wisconsin: AAAI Technical Report WS-98-05.

Thiesson, B., Meek, C., & Heckerman, D. (2001). Accelerating EM for large databases. *Machine Learning*, *45*, 279–299.

Yedidia, J. S., Freeman, W. T., & Weiss, Y. (2003). *Exploring artificial intelligence in the new millennium*, chapter Understanding belief propagation and its generalizations, 239–269. Morgan Kaufmann Publishers Inc.

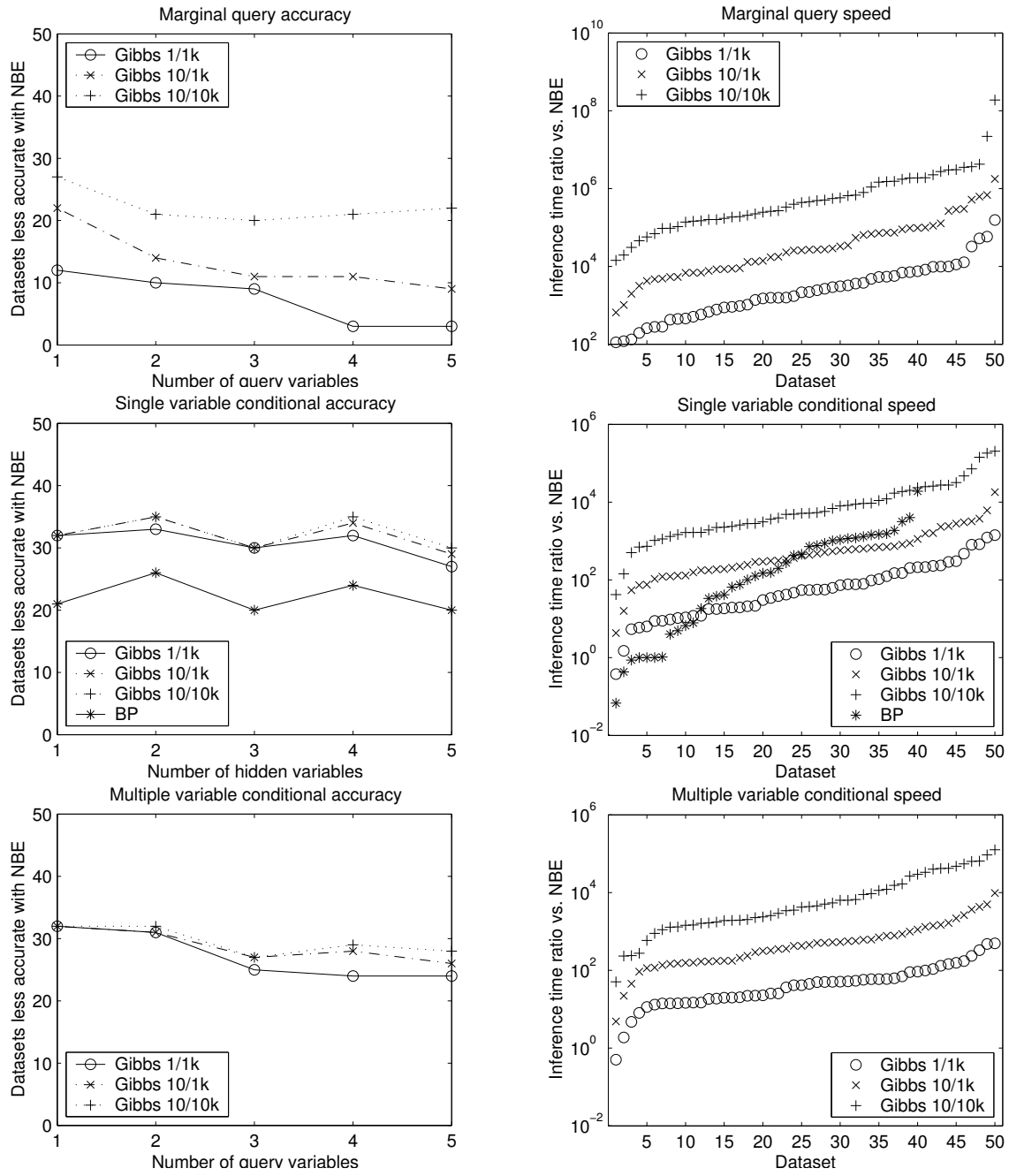

Figure 5: Query speed and accuracy for marginal and conditional queries. For the accuracy graphs on the left, we plot the number of datasets on which NBE achieves a worse log likelihood than each other inference method. For the inference speed graphs on the right, we show the time ratio of each inference method on each dataset, relative to NBE. Of the different number of query or hidden variables, we picked the number yielding the lowest ratios. Time ratios are plotted on a logarithmic scale. Datasets are ordered by increasing time ratio; indices refer to the nth smallest ratio, not to a single dataset in particular. Note that belief propagation only produced results for 40 datasets; on the remaining 10, it required excessive memory (> 1 GB). For additional details, see Sections 4.5.1, 4.5.2, and 4.5.3.

Name	# Attr.	# Train ex.	# Test ex.	NB Acc.	WM Acc.	Marg. Acc.
1985 Auto Imports	26	184.5	20.5	-29.579	-27.195	-40.616
Abalone	9	3758	419	-7.251	-7.265	-13.908
Adult	15	4888	535	-13.002	-12.715	-15.960
Annealing	39	718.2	79.8	-10.936	-10.389	-15.293
Anonymous MSWeb	294	32711	5000	-9.908	-9.687	-11.357
Audiology	70	203.4	22.6	-16.195	-15.689	-19.045
Auto MPG	8	358.2	39.8	-9.210	-9.140	-12.767
Breast Cancer Wisc.	31	512.1	56.9	-36.590	-31.348	-48.964
BUPA	7	310.5	34.5	-9.907	-9.920	-10.234
Car	7	1555.2	172.8	-7.831	-7.711	-8.309
Census	14	40766	4456	-11.027	-10.787	-15.160
Chess Endgames	37	2869	327	-10.791	-9.723	-15.107
Connect-4	43	60849	6708	-15.078	-13.901	-20.628
Contraceptive Choice	10	1325.7	147.3	-9.236	-9.304	-10.125
Credit Screening	16	621.0	69.0	-15.268	-14.797	-17.358
Forest Cover Type	55	25916	2946	-16.029	-14.454	-22.914
Glass Identification	10	192.6	21.4	-11.086	-11.614	-13.251
Hepatitis	20	139.5	15.5	-17.708	-17.847	-18.820
Housing	14	455.4	50.6	-13.254	-13.111	-19.570
House Votes	17	391.5	43.5	-9.912	-10.535	-14.057
Image Segmentation	17	2047	263	-11.743	-11.287	-24.484
Ionosphere	35	315.9	35.1	-37.522	-37.669	-52.329
Iris Types	5	135.0	15.0	-5.099	-5.325	-7.420
Isolated Letter Speech	618	6238	1559	-798.702	-542.064	-912.885
King Rook vs. King	7	25188	2868	-11.216	-11.516	-13.140
Labor Negotiations	17	40	17	-21.036	-19.932	-19.788
Landsat	37	4435	2000	-26.702	-24.419	-59.539
Letter Recognition	17	18012	1988	-15.733	-16.480	-26.926
Monks Problem #1	7	124	432	-6.717	-6.572	-6.779
Musk	167	428.4	47.6	-183.493	-125.523	-261.794
New Thyroid	6	193.5	21.5	-7.976	-8.368	-9.218
Nursery	9	9953	1072	-9.527	-9.434	-10.597
Page Blocks	11	4899	574	-9.244	-9.327	-16.464
Pima Indians Diabetes	9	691.2	76.8	-11.991	-11.851	-12.601
Poisonous Mushrooms	23	7337	787	-9.146	-9.221	-22.658
Promoter	58	95.4	10.6	-78.906	-79.274	-79.462
Servo	5	150.3	16.7	-7.398	-7.224	-8.474
Shuttle	10	43500	13293	-6.957	-6.955	-11.982
Solare Flare	13	959.4	106.6	-5.251	-5.343	-7.051
Soybean Large	36	307	376	-18.124	-17.244	-37.316
Spambase	58	4166	435	-13.375	-13.532	-16.844
Splice-Junction	61	2093	1097	-79.982	-80.007	-83.280
Thyroid Disease (comb.)	33	2800	972	-12.970	-12.364	-16.503
Tic-Tac-Toe	10	862.2	95.8	-9.032	-9.656	-10.274
Waveform	22	4484	516	-29.171	-29.492	-34.895
Yeast	9	1335.6	148.4	-10.188	-10.219	-10.879
Zoo	17	90.9	10.1	-7.315	-8.016	-12.558
EachMovie (subset)	1648	5526	591	-121.632	-120.934	-173.472
Jester	100	16302	1696	-95.443	-96.289	-130.202
KDD Cup 2000 (subset)	65	10000	3552	-2.102	-2.233	-2.414

Table 2: Summary of datasets and results. For cross-validated datasets, train and test set sizes are averaged across the 10 train/test splits.