Naive Bayes Models for Probability Estimation

Daniel Lowd LOWD@CS.WASHINGTON.EDU
Pedro Domingos PEDROD@CS.WASHINGTON.EDU
Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195-2350, USA

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. Most significantly, naive Bayes inference is orders of magnitude faster than Bayesian network inference using Gibbs sampling and belief propagation. This makes naive Bayes models a very attractive alternative to Bayesian networks for general probability estimation, particularly in large or real-time domains.

1. Introduction

Bayesian networks can efficiently represent complex probability distributions, and have received much attention in recent years (Pearl, 1988). Unfortunately, this efficiency does not extend to inference, which is #P-complete (Roth, 1996). In practice, methods for exact inference in Bayesian networks are often too costly, and we have to resort to approximate methods like Markov chain Monte Carlo (Gilks et al., 1996) and loopy belief propagation (Yedidia et al., 2001). However, the application of these methods is fraught with difficulties: the inference time is unpredictable, convergence is difficult to diagnose, in some cases it does not occur, and sometimes incorrect values are output. As a result, and despite much research, inference in Bayesian networks remains to a large extent a black art, and this significantly limits their applicability.

Ideally, we would like to have a representation for joint probability distributions that is as powerful as Bayesian networks and no harder to learn, but lends itself to exact inference in linear time. In this paper, we observe that (perhaps surprisingly) such a representation is readily available, in the form of naive Bayes models. Naive Bayes is a special form of Bayesian network that is widely used for classification (Domingos & Pazzani, 1997) and clustering (Cheeseman & Stutz, 1996), but its potential for general probabilistic modeling (i.e., to answer joint, conditional and marginal queries over arbitrary distributions) remains largely unexploited. Naive Bayes represents a distribution as a mixture of components, where within each component all variables are assumed independent of each other. Given enough components, it can approximate an arbitrary distribution arbitrarily closely. Inference time is linear in the number of components and the number of query variables. When learned from data, naive Bayes models never contain more components than there are examples in the data, guaranteeing that inference will be tractable. Whether this approach leads to models that are as accurate as Bayesian networks of arbitrary structure is an empirical question, and in this paper we answer it affirmatively by conducting extensive experiments in 50 domains. Notice that our main goal in this paper is to learn accurate models that allow for fast inference, as opposed to gaining insight into the structure of probabilistic dependencies in the domain.

We begin by briefly reviewing the necessary background on Bayesian networks (Section 2). We then describe naive Bayes models for general probability estimation and our NBE algorithm for learning them (Section 3). Finally, we report our experiments and discuss their results (Section 4).

2. Bayesian Networks

A Bayesian network encodes the joint probability distribution of a set of n variables, $\{X_1, \ldots, X_n\}$, as a directed acyclic graph and a set of conditional probability distributions (CPDs). Each node corresponds to a variable, and the CPD associated with it gives the probability of each state of the variable given every possible combination of states of its parents. The set of parents of X_i, denoted π_i, is the set of nodes with an arc to X_i in the graph. The structure of the network encodes the assertion that each node is conditionally independent of its non-descendants given its parents. The joint distribution of the variables is thus given by

Appearing in *Proceedings of the 22^{nd} International Conference on Machine Learning*, Bonn, Germany, 2005.

$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\pi_i)$. For discrete domains, the simplest form of CPD is a conditional probability table, but this requires space exponential in the number of parents of the variable. A more scalable approach is to use decision trees as CPDs, taking advantage of context-specific independencies (Friedman & Goldszmidt, 1996).

When the structure of the network is known, learning reduces to estimating CPD parameters. When some variables are unobserved in some or all examples, this can be done using the EM algorithm, which alternates between computing expectations for the unobserved values using the current parameters and computing the maximum likelihood or MAP parameters using the current expectations (Dempster et al., 1977). When the structure is unknown, it can be learned by starting with an empty or prior network and greedily adding, deleting and reversing arcs to optimize some score function (Heckerman et al., 1995). Learning structure given incomplete data requires a computationally expensive combination of EM and structure search (Friedman, 1998).

The goal of inference in Bayesian networks is to answer arbitrary marginal and conditional queries (i.e., to compute the marginal distribution of a set of query variables, possibly conditioned on the values of a set of evidence variables). Because exact inference is intractable, approximate methods are often used, of which the most popular is Gibbs sampling, a form of Markov chain Monte Carlo (Gilks et al., 1996). A Gibbs sampler proceeds by sampling each non-evidence variable in turn conditioned on its Markov blanket (parents, children and parents of children). The distribution of the query variables is then approximated by computing, for each possible state of the variables, the fraction of samples in which it occurs. Gibbs sampling can be very slow to converge, and many variations have been developed, but choosing and tuning an MCMC variant for a given application remains a difficult, labor-intensive task. Diagnosing convergence is also difficult. Another popular inference method is loopy belief propagation (Yedidia et al., 2001). In this approach, each node iteratively sends messages to its neighbors about their expected distribution, and updates its own. While belief propagation can be faster than MCMC, it sometimes does not converge, and sometimes converges to incorrect values.

3. Naive Bayes Probability Estimation

Naive Bayes models are so named for their "naive" assumption that all variables X_i are mutually independent given a "special" variable C. The joint distribution is then given compactly by $P(C, X_1, \ldots, X_n) = P(C) \prod_{i=1}^{n} P(X_i|C)$. The univariate conditional distributions $P(X_i|C)$ can take any form (e.g., multinomial for discrete variables, Gaussian for continuous ones). When the variable C is observed in the training data, naive Bayes can be used for classification, by assigning test example $(X_1, \ldots, X_n)$ to the class C with highest $P(C|X_1, \ldots, X_n)$ (Domingos & Pazzani, 1997). When C is unobserved, data points $(X_1, \ldots, X_n)$ can be clustered by applying the EM algorithm with C as the missing information; each value of C corresponds to a different cluster, and $P(C|X_1, \ldots, X_n)$ is the point's probability of membership in cluster C (Cheeseman & Stutz, 1996).

Naive Bayes models can be viewed as Bayesian networks in which each X_i has C as the sole parent and C has no parents. A naive Bayes model with Gaussian $P(X_i|C)$ is equivalent to a mixture of Gaussians with diagonal covariance matrices (Dempster et al., 1977). While mixtures of Gaussians are widely used for density estimation in continuous domains, naive Bayes models have seen very little similar use in discrete and mixed domains. (See Breese et al. (1998) for one exception in the domain of collaborative filtering.) However, they have some notable advantages for this purpose. In particular, they allow for very efficient inference of marginal and conditional distributions. To see this, let X be the set of query variables, Z be the remaining variables, and k be the number of mixture components (i.e., the number of values of C). We can compute the marginal distribution of X by summing out C and Z:

$$\begin{aligned}
P(X = x) &= \sum_{c=1}^{k} \sum_{z} P(C = c, X = x, Z = z) \\
&= \sum_{c=1}^{k} \sum_{z} P(c) \prod_{i=1}^{|X|} P(x_i|c) \prod_{j=1}^{|Z|} P(z_j|c) \\
&= \sum_{c=1}^{k} P(c) \prod_{i=1}^{|X|} P(x_i|c) \prod_{j=1}^{|Z|} \left(\sum_{z_j} P(z_j|c) \right) \\
&= \sum_{c=1}^{k} P(c) \prod_{i=1}^{|X|} P(x_i|c)
\end{aligned}$$

where the last equality holds because, for all j, $\sum_{z_j} P(z_j|c) = 1$. Thus the non-query variables Z can simply be ignored when computing $P(X = x)$, and the time required to compute $P(X = x)$ is $O(|X|k)$, independent of $|Z|$. This contrasts with Bayesian network inference, which is worst-case exponential in $|Z|$. Similar considerations apply to conditional probabilities, which can be computed efficiently as ratios of marginal probabilities: $P(X = x \mid Y = y) = P(X = x, Y = y)/P(Y = y)$.

A slightly richer model than naive Bayes which still allows for efficient inference is the mixture of trees, where, in each cluster, each variable can have one other parent in addition to C (Meila & Jordan, 2000). However, the time required to learn mixtures of trees is quadratic instead of linear in

Table 1. The NBE learning algorithm.

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: Compute maximum likelihood parameters for M, using the filled-in data.
    If $\log P(H|M)$ is best so far, save M in M_{best}.
    Every 5 cycles, prune low-weight components of M.
  until $\log P(H|M)$ fails to improve by ratio δ_{EM}.
  $M \leftarrow M_{best}$
  Prune low weight components of M.
  $k \leftarrow 2k$
until $\log P(H|M)$ fails to improve by ratio δ_{Add}.
Execute E-step and M-step twice more on M_{best}, using examples from both H and T.
Return M_{best}.

the number of variables per iteration of EM, and they have not been extensively studied empirically. Comparing them with naive Bayes is an item for future research.

In principle, a clustering algorithm like AutoClass (Cheeseman & Stutz, 1996) could be used to learn naive Bayes models. However, in our experiments we found AutoClass to be very slow and inaccurate. This is probably due to the fact that it was designed for clustering, where the desired number of mixture components is typically very low. In our case, the goal is accurate joint probability estimation, and this may require a large number of components. We have thus developed a new algorithm for this purpose, called NBE (Naive Bayes Estimation). Table 1 shows its pseudo-code.

NBE is essentially EM wrapped in an outer loop that progressively adds and prunes mixture components. We model discrete variables with multinomials and continuous ones with Gaussians, although other distributions could easily be added. We first divide the training data into a training set and a hold-out set. We begin with a single component consisting of each variable's marginal distribution. In each cycle, we add k new components, using a random training example to initialize each component as follows. If the ith variable is discrete, we assign a probability of $(1.0 + 0.1P(x_i))/1.1$ to its value x_i in the example, and a probability of $0.1P(x'_i)/1.1$ to all other values, where $P(x'_i)$ is the value's relative frequency in the data. If the ith variable is continuous, we use its value x_i as the mean of the Gaussian, and 0.4 of its variance in the data as the variance. The k seed examples are removed from the data to avoid overfitting. In each cycle, the number of components added doubles. If there are m components before the cycle starts and n new ones are added, the weight $P(c)$ of each pre-existing component is rescaled by $m/(m+n)$, and each new component receives an initial weight of $1/(m+n)$.

Within each cycle, we run EM to fit the expanded set of components until the log likelihood of the held-out data fails to increase by at least a fraction δ_{EM}. At each iteration, we save the current model if it yields the best hold-out log likelihood so far. After every five steps of EM and after it ends we prune low-weight components. We accomplish this by sorting all components by weight, and keeping the first components whose weights 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 has more than 1,000 components, a higher pruning threshold should be used.

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.

4. Empirical Evaluation

We now describe our empirical evaluation of NBE's learning time, inference time and accuracy. Full details of the experiments are given in an online appendix at http://www.cs.washington.edu/ai/nbe. This appendix includes source code for our implementations of NBE and Bayesian network inference, the NBE models and Bayesian networks learned in each domain, a detailed description of the parameter settings and experimental methodology used, and a more complete set of experimental results.

4.1. Datasets

We used 47 datasets from the UCI repository (Blake & Merz, 1998), ranging in number of variables from five to 618, and in size from 57 examples to 67,000. The datasets are listed in Table 2. Although many of these datasets were designed for classification, learning the joint distribution over all variables remains an interesting problem. (This is confirmed by the fact that the number of mixture components found by NBE was typically much greater than the number of classes in the original data. For example, Letter Recognition contains 26 classes, but NBE learned a model with over 600 components.) In addition, we used two collaborative filtering datasets (Jester (Goldberg et al., 2001)

and EachMovie[1]) and the KDD Cup 2000 clickstream prediction dataset (Kohavi et al., 2000). Collaborative filtering and clickstream prediction are examples of tasks where probabilistic modeling is potentially very useful, but its applicability is severely limited by the need for short, predictable inference times. (For example, an e-commerce site like Amazon.com has millions of users, and needs to generate recommendations and click predictions in fractions of a second.)

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 ten-fold cross-validation instead of using a single train-test split. From each training set we selected a standard hold-out set, to be used by all algorithms for overfitting avoidance. In the case of cross-validated datasets, one hold-out set was generated for each of the ten 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.

4.2. Training

We used NBE with 50 initial components for training sets of over 1,500 examples, and 10 initial components for smaller training sets. We used $\delta_{EM} = 0.0001$ and $\delta_{Add} = 0.001$, based on performance in preliminary experiments.

We compared our naive Bayes models to Bayesian networks with context-specific independence, learned using Microsoft Research's WinMine Toolkit, which is arguably the best available software for learning Bayesian networks (Chickering, 2002). WinMine effectively learns a probabilistic decision tree for each node as a function of the others, interleaving the induction of all trees, and ensuring at each step that no cycles are created. WinMine finds a local maximum of the Bayesian score function it uses. Unlike NBE, it does not use hidden variables. Rather than attempt to infer missing values, WinMine treats "missing" as a distinct value, and we configured NBE to do the same. We used all default settings for WinMine,[2] except for κ, which is used to control overfitting and is effectively the penalty for adding an arc to the network. The default value for κ is 0.01, but the best value can vary greatly from one dataset to another. Since we use a hold-out set to avoid overfitting in NBE, we felt it was appropriate to use the same hold-out set to automatically tune κ. We began with a conservative value of 0.0001, which leads to 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 hold-out.

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.

Both NBE and WinMine can readily handle continuous variables. However, in our experiments Gibbs sampling with continuous variables was so slow as to make a large-scale study infeasible. We therefore discretized all continuous variables into five equal-frequency bins.

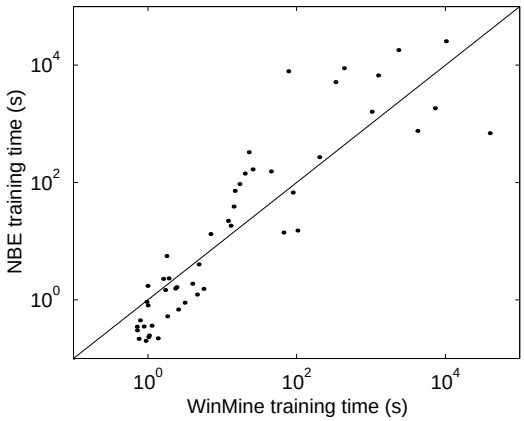

Figure 1. Learning times for NBE and WinMine. Each point represents one of the 50 datasets.

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, with neither method performing consistently better. Learning times for each model on each dataset are shown in Figure 1.

NBE is not faster than WinMine because, while it does no structure search, it uses EM, which can take substantial time to converge. If both models were required to learn with missing data, rather than making "missing" a distinct value, these results might look very different. Because NBE already incorporates EM and allows for fast inference, it can readily handle missing data. In contrast, combining EM with structure learning for Bayesian networks

[1]Provided by Compaq, http://research.compaq.com/SRC/eachmovie/. We used a 10% subset to speed up learning, and decreased sparsity by converting the 0–5 movie rating scale into Boolean "like" (4–5) and "dislike" (0–3) values.

[2]–acyclic flag; min=10, bdu=−1, ksqmarg=−1, maxin=−1, maxout=−1; all variables are "input-output," and have tree-multinomial distributions (continuous variables were pre-discretized; see below).

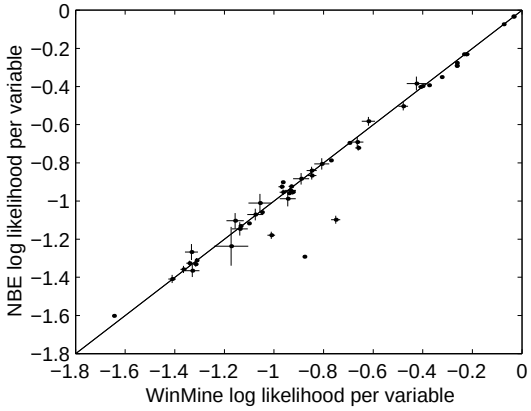

Figure 2. Comparison of log likelihoods. Error bars are four standard deviations.

can be extremely slow (Friedman, 1998).

4.4. Modeling Accuracy

For each dataset, we compared the log likelihoods of the test data given each model. For cross-validated datasets, we averaged log likelihoods over the ten splits. The results are shown in Table 2. All log likelihoods were computed using natural logarithms.

NBE's models were more accurate than WinMine's on 23 of the 50 datasets. Using a two-tailed paired t-test ($p = 0.05$), NBE performed significantly better on 15 datasets and WinMine performed better on 22. Though many differences were 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.

4.5.1. Inference Methods

For NBE, we used linear-time exact inference, as described in Section 3. Because exact inference in Bayesian networks can take exponential time, we used the most popular approximation algorithms: Gibbs sampling and belief propagation. WinMine does not include any inference routines, so we implemented them ourselves. All algorithms (including NBE inference) were implemented in C++, profiled, and optimized for speed. (We spent considerably more time optimizing Gibbs sampling and belief propagation than optimizing NBE inference.) The Gibbs sampler is initialized by sampling each variable given its parents. This is followed by burn-in and mixing. During the latter, counts are updated every time a query variable is sampled. Source

Table 2. Average log likelihoods on benchmark datasets.

Name	NBE	WM	Marg.
1985 Auto Imp.	−25.68	−24.55	−36.16
Abalone	−7.25	−7.27	−13.91
Adult	−13.00	−12.72	−15.96
Annealing	−10.78	−10.24	−15.08
Anon. MSWeb	−9.91	−9.69	−11.36
Audiology	−16.15	−15.65	−18.98
Auto MPG	−9.17	−9.10	−12.72
Breast Cancer Wisc.	−36.57	−31.33	−48.95
BUPA	−9.86	−9.87	−10.16
Car	−7.82	−7.70	−8.30
Census	−11.03	−10.79	−15.16
Chess Endgames	−10.79	−9.72	−15.11
Connect-4	−15.08	−13.90	−20.63
Contracept. Choice	−9.24	−9.30	−10.13
Credit Screening	−15.26	−14.78	−17.34
Forest Cover Type	−16.03	−14.46	−22.91
Glass Identification	−11.04	−11.57	−13.20
Hepatitis	−17.68	−17.81	−18.78
House Votes	−9.90	−10.52	−14.04
Housing	−13.22	−13.08	−19.50
Image Segmentation	−11.74	−11.29	−24.49
Ionosphere	−37.49	−37.64	−52.29
Iris Types	−5.06	−5.28	−7.38
Iso. Letter Speech	−798.70	−542.06	−912.89
King Rook vs. King	−11.22	−11.52	−13.14
Labor Negotiations	−21.04	−19.93	−19.79
Landsat	−26.70	−24.42	−59.54
Letter Recognition	−15.73	−16.48	−26.93
Monks Problem #1	−6.72	−6.57	−6.78
Musk	−183.41	−125.44	−261.68
New Thyroid	−7.61	−8.00	−8.83
Nursery	−9.53	−9.44	−10.60
Page Blocks	−9.24	−9.33	−16.46
Pima Diabetes	−11.98	−11.84	−12.59
Pois. Mushrooms	−9.15	−9.22	−22.66
Promoter	−78.81	−79.18	−79.37
Servo	−6.83	−6.65	−7.70
Shuttle	−6.96	−6.96	−11.98
Solar Flare	−5.22	−5.32	−7.01
Soybean Large	−18.12	−17.25	−37.32
Spambase	−13.38	−13.53	−16.85
Splice Junction	−79.98	−80.01	−83.28
Thyroid (comb.)	−12.97	−12.37	−16.50
Tic-Tac-Toe	−9.02	−9.64	−10.26
Waveform	−29.17	−29.49	−34.90
Yeast	−10.18	−10.21	−10.87
Zoo	−6.53	−7.23	−11.77
EachMovie	−121.63	−120.94	−173.47
Jester	−95.44	−96.29	−130.20
KDD Cup 2000	−2.10	−2.23	−2.41

code for Gibbs sampling and belief propagation is provided in the online appendix.

It was very difficult to select a Gibbs sampling scheme that would be fair in terms of both speed and accuracy: running too few iterations would reduce accuracy, and running too many would reduce speed. 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 free variable. In preliminary experiments, further increasing the number of iterations did not help significantly.

For single-variable queries, we used Rao-Blackwellisation: instead of giving a unit count to the sampled state of a variable, we gave each state a partial count equal to its probability conditioned on the Markov blanket. This yields increased accuracy with fewer samples. To avoid configurations with zero probability, we gave each configuration a fractional prior count, with all prior counts summing 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., 2001) over a Bayesian network by first converting it into a pairwise Markov network. Unfortunately, when the local decision-tree models are converted to potential function matrices, we may see an exponential blow-up in size. Belief propagation thus failed on many of the larger datasets, where it would have required gigabytes of memory just to 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. Because belief propagation only computes conditional probabilities of single variables, we omitted it from the experiments with multiple variables. 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.

4.5.2. Experimental Methods

To assess the relative performance of NBE and Bayesian networks, we generated marginal, single-variable conditional, and multiple-variable conditional queries for each of the datasets. In each case, we started with up to 1,000 examples from the test set and created random queries by removing the values of one or more randomly-chosen variables, termed the free variables. Each inference method computed the joint distribution over a subset of these called the query variables. We refer to other removed variables as hidden, and the remaining variables as the evidence.

We estimate inference accuracy on a particular query as the log likelihood of the configuration in the original test example. This is a very rough approximation of the Kullback-Leibler (K-L) divergence between the inferred distribution and the true one, estimated using a single sample. As we average over many queries, our estimate asymptotically approaches the average K-L divergence, making this a reasonable accuracy estimate.

The differences among the three query scenarios are the number of query, hidden, and evidence variables. In the marginal experiments, we compute the marginal distribution of one to five variables, conditioning on no evidence variables. This represents the greatest challenge to Gibbs sampling, due to the large number of hidden variables that must be summed out. In the conditional experiments, we used one to five free variables, leaving the rest as evidence. For the single-variable case, the distribution of each free variable was computed independently, treating the other variables as hidden. This is the only scenario of the three for which we could include belief propagation, since the others include multivariate queries. In the multiple variable case, we computed the joint distribution of all one to five free variables.

4.5.3. Query Speed

Query speed varied drastically depending on dataset and scenario, but NBE was almost always fastest, usually by several orders of magnitude. Speed results for two of the three scenarios are summarized in Figure 3.

On marginal queries, Gibbs sampling with 10 chains of 10,000 sampling iterations took between one second and half an hour per marginal query. For this reason, we only had time for 100 queries per dataset using this Gibbs sampling configuration. In contrast, NBE never averaged more than 3 milliseconds per query. Even the fastest configuration of Gibbs sampling was 100 to 150,000 times as slow; the slowest was 6,000 to 188 million times as slow.

Gibbs sampling ran much faster on the conditional experiments, since it had at most five free variables to sample. Even so, its running time remained orders of magnitude slower than NBE in the average case. On the Letter Recognition dataset, 10 chains of 10,000 iterations took Gibbs sampling 23 seconds per query with five free variables,

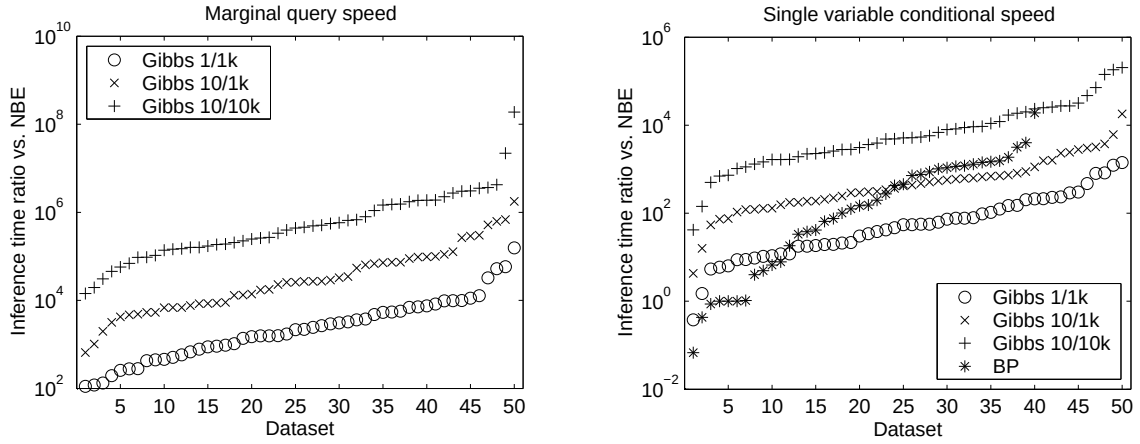

Figure 3. Query speed for marginal and conditional queries, relative to NBE. Of the different number of query or free variables, we chose the number yielding the lowest ratios. Time ratios are plotted on a logarithmic scale. Datasets are ordered by increasing time ratio; indices refer to the nth smallest ratio, not to particular datasets. Note that belief propagation only produced results for 40 datasets; on the remaining 10, it required excessive memory ($>$ 1 GB).

while NBE never averaged more than 30 milliseconds per query. Multiple and single variable cases had very similar running times for Gibbs sampling, since the number of free variables was unchanged.

Belief propagation was fastest on three datasets, but its overall performance remained mediocre due to the overhead of large state spaces in the generated Markov network. At its worst, it averaged 45 seconds per query with five free variables on the Shuttle dataset.

4.5.4. Query Accuracy

For each scenario and each number of query or free variables, we tallied the number of datasets on which NBE was more accurate than each Bayesian network inference method used. This summary is displayed in Table 3. Recall that for marginal and multiple-variable conditional queries we varied the number of query variables, while for single-variable conditional queries we varied the number of free variables.

For marginal queries, NBE was consistently more accurate than Gibbs sampling on most datasets. We were surprised to find 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 in log likelihood. This suggests that Gibbs sampling may not have converged on these datasets. They were also the slowest for Gibbs sampling, requiring 10 minutes and half an hour per query, respectively. This illustrates the fact that NBE may be preferable to Bayesian networks even in some domains that are better modeled by the latter, because their inference time may be unacceptable.

NBE did worse on the single-variable conditional queries,

Table 3. Number of datasets (out of 50) for which NBE inference was more accurate than Gibbs sampling and belief propagation

#Query/free vars.	1	2	3	4	5
Marginal queries					
– 1 chain, 1k samples	38	40	41	47	47
– 10 chains, 1k samples	28	36	39	39	41
– 10 chains, 10k samples	23	29	31	30	29
Single-var. conditional					
– 1 chain, 1k samples	18	17	20	18	23
– 10 chains, 1k samples	18	15	20	16	21
– 10 chains, 10k samples	18	15	20	15	20
– Belief propagation	31	36	30	34	30
Multiple-var. conditional					
– 1 chain, 1k samples	18	19	25	26	26
– 10 chains, 1k samples	18	19	23	22	24
– 10 chains, 10k samples	18	18	23	21	22

always losing to Gibbs sampling on a majority of the datasets. However, the losses were not overwhelming. Figure 4 compares the log likelihoods of NBE and Gibbs sampling with five free variables. (Graphs with one to four free variables 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 4 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.

Belief propagation was about as accurate as NBE when it could be run, but its failure to run on 10 of the 50 datasets was a critical weakness. We counted these failures as wins for NBE.

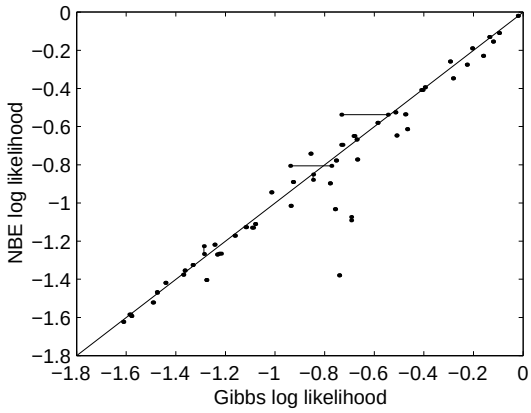

Figure 4. Comparison of the log likelihoods of NBE and Gibbs sampling on single-variable conditional queries with five free variables. For each dataset, we plot a pair of points corresponding to two Gibbs sampling configurations: one chain with 1,000 sampling iterations and ten chains with 10,000 sampling iterations each. Each pair is connected by a line, but these lines are only visible when the results of the two experiments are different. Error bars are omitted for clarity.

In multiple-variable conditional queries, the log likelihoods of NBE and Gibbs sampling were close, with a few outliers, similar to the single-variable results depicted in Figure 4. Gibbs sampling's accuracy is somewhat worse in the multiple-variable case, although that is partially remedied by running Gibbs sampling for longer. This is also expected: since the multivariate distributions have more states than the single-variable ones, each state will tend to have fewer counts and therefore a less accurate probability estimate. As we increase the number of inferred variables, the situation only becomes worse for Gibbs sampling, while NBE is unaffected.

Accuracy apart, it could be argued that Bayesian network structure gives more insight into the domain than a naive Bayes model. However, the learned structure's high sensitivity to variations in the training set makes its significance questionable. On the NBE side, valuable insight can be obtained from the clusters formed, their relative sizes, and the variables with the most unbalanced distributions in each.

5. Conclusion

This paper proposes naive Bayes models as an alternative to Bayesian networks for general probability estimation tasks. Experiments on a large number of datasets show that the two take similar time to learn and are similarly accurate, but naive Bayes inference is orders of magnitude faster. Directions for future work include refining our NBE algorithm, extending it to relational domains, and applying it to new real-world problems. The NBE source code is available at http://www.cs.washington.edu/ai/nbe.

Acknowledgements

We thank all those who provided the datasets we used, and Marina Meila for useful discussions. This work was partly funded by an NSF Graduate Research Fellowship (first author), an NSF CAREER Award and a Sloan Fellowship (second author), and ONR grant N00014-02-1-0408.

References

Blake, C. L., & Merz, C. J. (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/-∼mlearn/MLRepository.html.

Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. *Proc. UAI-98* (pp. 43–52).

Cheeseman, P., & Stutz, J. (1996). Bayesian classification (AutoClass): Theory and results. In *Advances in knowledge discovery and data mining*, 153–180. Menlo Park, CA: AAAI Press.

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

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

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

Friedman, N. (1998). The Bayesian structural EM algorithm. *Proc. UAI-98* (pp. 129–138).

Friedman, N., & Goldszmidt, M. (1996). Learning Bayesian networks with local structure. *Proc. UAI-96* (pp. 252–262).

Gilks, W. R., Richardson, S., & Spiegelhalter, D. J. (Eds.). (1996). *Markov chain Monte Carlo in practice*. London, UK: Chapman and Hall.

Goldberg, K., Roeder, T., Gupta, D., & Perkins, C. (2001). Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, *4(2)*, 133–151.

Heckerman, D., Geiger, D., & Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statist. data. *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.

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

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

Roth, D. (1996). On the hardness of approximate reasoning. *Artificial Intelligence*, *82*, 273–302.

Yedidia, J. S., Freeman, W. T., & Weiss, Y. (2001). Generalized belief propagation. In *Adv. NIPS 13*, 689–695.