Probabilistic Abstraction Lattices: A Computationally Efficient Model for Conditional Probability Estimation

Daniel Lowd

January 14, 2004

1 Introduction

Probabilistic models have shown increasing popularity for machine learning, data mining, and artificial intelligence in general. Probabilistic approaches have been successfully applied to collaborative filtering, robot navigation, text analysis, medical diagnosis, and many other areas. In general, we wish to efficiently represent the joint probability distribution among a set of variables in order to solve these and other problems. Most of the current approaches do so using a graph with a single node for each modeled variable, hidden or observed.

While these models have proven effective in many domains, they also have certain limitations. We focus on Bayesian networks since they are the most popular and one of the most general types of probabilistic graphical models. In Bayesian networks, complete inference may take exponential time in the number of variables modeled. Approximation algorithms such as Gibbs sampling can be effective, but this is only a partial solution: iterative sampling is a very slow process as well. While this is fine for problems with limited numbers of variables, special structure, or no time constraints, inference time is an issue for other domains such as collaborative filtering, in which a typical problem may have thousands of variables and require a split-second response.

Even ignoring time constraints, a graphical model may not be the most effective representation for some problems. As an alternate representation, association rules remain popular for studying market basket data. An association rule specifies the probability that a user purchasing a certain set of items will also purchase another specific item (Agrawal et al., 1993). For example, an association rule might specify that customers who buy milk are likely to buy bread, with 70% probability, or that customers who buy pens and pencils are likely to buy paper, with 60% percent probability. These association rules may be efficiently mined from data using an algorithm such as APRIORI (Agrawal and Srikant, 1994). The hope is that some association rules will provide valuable insights into customer behavior and aid in the development of business strategies.

Of course, association rules alone do not provide a full joint probability distribution over all variables. Association rules are also somewhat limited, in that they are generally limited to conjunctions of true boolean variables. If these limitations could be overcome, we could construct a new type of probabilistic model where each model component is not a variable but an *abstraction* that applies to a certain portion of the data as an association rule applies to a certain subset of the observations. In addition to providing rapid inference and perhaps better accuracy in certain domains, an abstraction-based probabilistic model could be easier to understand: while traditional graphical methods depend on an understanding of conditional probability, association rules only require an understanding of cooccurrence.

2 Modeling with Abstractions

Every day, we use abstraction as a tool for dealing with the complexities of the world around us. Based on prior experience, learning, and cultural influence, we label certain observations as "trees" or "houses" or "conference papers." Each such label is an abstract way of considering a collection of related observations in order to effectively learn, reason, and communicate with each other about the world around us. The underlying observation may actually be the effect of neurons responding to photons at certain wavelengths reflected from organic molecules composing a tree-like structure, but this is rarely a useful way to see things. By calling a tree a tree, we can infer properties based on prior experiences with trees.

Data mining algorithms are also faced with many observations, each typically represented by a collection of variables with particular values: $\{x_1, x_2, \ldots, x_n\}$. In this context, an abstraction would be any predicate on those values. For example, if all x_i were boolean variables, possible abstractions could include $(x_1 \wedge x_2)$, $(x_2 \vee x_3)$, (x_1), $(x_2 \wedge \neg x_1)$, or even `true`, the abstraction that applies to all observations. As before, multiple abstractions may apply to any given observation, and some abstractions are strictly more specific than others. For example, the abstraction $(x_1 \wedge x_2)$ is strictly more specific than (x_1), which in turn is strictly more specific than `true`. This concept of an abstraction is trivial to extend to multi-valued or continuous data as well. Note that, by this definition, an abstraction does not specify anything else *a priori* about the values of the observations to which it applies.

We can model any joint probability distribution as a mixture of abstractions given three sets of information: the abstractions to use (A), the probability that each abstraction will generate the next observation ($P(A_i)$), and a probability distribution for each variable given each abstraction ($P(x_j|A_i)$).

The overall probability for any observation in our generative model is as follows:

$$P(o) = \sum_{A_i \in A} \Pi_{j=1\ldots n} P(x_j = o_j|A_i)P(A_i)$$

In the discrete case, we can perfectly represent any probability distribution by simply having one abstraction per possible observation and letting the probability of that abstraction be the frequency with which that observation occurs. Since this would require an exponential number of abstractions, we hope that the structure inherent in most problems will allow them to be effectively modeled with a relatively small number of abstractions. Note that Bayesian networks face a similar worst-case scenario: if all variables depend directly on all others, then the network will have an exponential number of edges and require conditional probability tables of an exponential size.

3 Learning with Abstractions

We have argued that abstractions can be effectively applied to the problem of modeling a joint probability distribution. In this section, we present the details of our algorithm to learn a Probabilistic Abstraction Lattice (PAL) from data based on these ideas.

For this first exploration and set of experiments, we have focused solely on positive, conjunctive abstractions of boolean variables. That is, all abstractions are of the form $(x_i \wedge x_j \wedge x_k \wedge ...)$. This subclass of abstractions is both easier to deal with and easier to find, and such abstractions have already proven their usefulness as the basis for association rule mining.

Given a positive, conjunctive abstraction A_i, $P(x_j|A_i) = 1$ if x_j appears in A_i's conjunctive definition. Similarly, $P(\neg x_j|A_i) = 0$ if x_j appears in A_i's conjunct. But what if A_i does not

reference x_j at all?

3.1 Model One

The simplest approach is to assume a uniform distribution over x_j. In the boolean case, $P(x_j|A_i) = P(\neg x_j|A_i) = 0.5$. We can generalize this to assume any constant distribution for all unspecified variables. Again, in the boolean case, $P(x_j|A_i) = q$ and $P(\neg x_j|A_i) = 1 - q$. We refer to this simple distribution as model one.

3.2 Model Two

Model two allows each variable x_j to have a different default distribution. That is, when A_i does not reference x_j, $P(x_j|A_i) = P(x_j)$. This model introduces the additional flexibility that some variables may have very different distributions, though this additional flexibility could also permit greater overfitting.

3.3 Model Three

In model three, an unspecified variable's distribution depends on the assumed abstraction as well. To accomplish this, each $P(x_j|A_i)$ is a model parameter that may be learned from data. Again, the greater flexibility allows for more finely tuned abstractions, but also could permit more overfitting. Model three, in addition to being the most general, can actually be seen as a version of the naïve Bayes classifier, where the class node is never observed and corresponds to the index of the abstraction chosen.

3.4 Selecting Abstractions

We took advantage of existing software implementations of the APRIORI algorithm to find frequent item sets, positive conjunctions occurring with at least a certain frequency (Agrawal and Srikant, 1994). Each frequent item set became a candidate abstraction. Since association rules and frequent item sets describe certain aspects of a dataset's structure, we would expect them to be useful abstractions as well. We also included a catch-all abstraction that will apply to all abstractions. This ensures that every observation matches at least one abstraction.

3.5 Learning Abstraction Weights

We train all model parameters using the Expectation Maximization (EM) algorithm (Dempster et al., 1977). The EM algorithm guarantees that we will converge to a locally optimal model with respect to the likelihood of the training data. The EM algorithm consists of two steps: an E-step in which the values of all hidden data are estimated using the current model, and an M-step in which new model parameters are chosen to maximize the likelihood of the complete data, both observed and hidden.

3.5.1 E-Step

For a Probabilistic Abstraction Lattice, the hidden variables we are trying to estimate in the E-step are the abstractions from which each training example was generated. We can compute a probability distribution over this hidden variable A_i by applying Bayes' rule and the total probability rule as follows:

$$P(A_i|o) = \frac{P(o|A_i)P(A_i)}{P(o)} \quad (1)$$

$$= \frac{P(o|A_i)P(A_i)}{\sum_j P(o|A_j)P(A_j)} \quad (2)$$

$$= \frac{P(A_i)\Pi_k P(o_k|A_i)}{\sum_j P(o|A_j)P(A_j)} \quad (3)$$

Though each observation is assumed to be generated by a single abstraction, in this step we fractionally assign each observation to any number of abstractions, based on its probability distribution over abstractions.

3.5.2 M-Step

For all models, we will need to recompute all $P(A_i)$. For model one, we must compute q; for model two, we must compute all $P(x_j)$; for model three, we must compute all $P(x_j|A_i)$. All of these can be easily computed by looking at the fraction of examples assigned to each abstraction.

For example, in model three, we can compute the new value of $P(x_j|A_i)$ as the sum of all fractional observations assigned to A_i where x_j was true divided by the sum of all fractional observations assigned to A_i. Specifically,

$$P'(x_j|A_i) = \frac{\sum_{o \in O, o_j = \texttt{true}} P(A_i|o)}{\sum_{o \in O} P(A_i|o)}$$

Models one and two are similar, taking care to only count variable occurrences when unspecified in the assigned abstraction.

4 Inference with Abstractions

Given a Probabilistic Abstraction Lattice, we can readily compute any conditional probability in time that is linear in the number of variables and the number of abstractions. Since the conditional probability $P(X|Y)$ is simply the fraction $P(X \wedge Y)/P(Y)$, it is sufficient to compute marginal probabilities $P(Z)$ in linear time. This is done as follows:

$$P(Z) = \sum_i P(Z|A_i)P(A_i)$$

For the case where some variable x_j is unspecified in Z, we assume that $P(Z_j|A_i)$ is 1, since an unspecified variable covers all possible values. If there are n variables and m abstractions in the model, then computing $P(Z|A_i)$ is $O(n)$ and so computing $P(Z)$ is $O(m \times n)$. This is an amazing improvement over Bayesian networks.

5 Empirical Evaluation

In order to assess the effectiveness of PALs relative to existing methods, we compared their performance to that of a Bayesian network toolkit on two versions of a web session dataset, MSWeb. We chose to use the WinMine Toolkit from Microsoft Research (Chickering, 2002) because its algorithms have been applied to the MSWeb dataset before before (Breese et al., 1998).

Name	Variables	Density	Training	Test
MSWeb	293	0.01	32,711	5,000
Dense MSWeb	11	0.52	1,782	267

Table 1: Statistics for web session datasets used.

Model	MSWeb	Dense MSWeb
WinMine	**-9.707**	-6.046
PAL Model3	-9.984	**-5.923**
PAL Model2	-10.299	-6.167
PAL Model1	-11.0845	-6.358
Marginal	-11.358	-6.759

Table 2: Log likelihood results for each model on both datasets. The best result for each dataset is shown in bold.

Unfortunately, while performing the experiments it became clear that MSWeb's sparsity was weakening the experiments. If we used only a few query variables and only a few conditional variables, then most test examples only listed web sites not visited. Neither model could predict very well , simply because there wasn't very much information to work with. For this reason, we selected the 11 most visited web sites and a subset of the examples, so that the density was artificially increased to about 50%. Statistics about both datasets are summarized in table 1.

5.1 Likelihood Experiments

For both the MSWeb and the Dense MSWeb dataset, we computed the average log likelihood over all test examples using all three PAL models, one Bayesian network, and the marginal distribution. We found that while the Bayesian network produced with WinMine did best on MSWeb, the third PAL model did best on the Dense MSWeb data. For both datasets, more flexible PAL models outperformed less flexible ones, suggesting that overfitting is not yet of primary concern. The results are summarized in table 2.

To further investigate the relationship between the most effective PAL model and WinMine's Bayesian network, we varied the number of training examples used for each and found that WinMine did even better with fewer training examples, as shown in figure 1. This suggests that a PAL might match WinMine's performance given enough training examples.

5.2 Conditional Likelihood Experiments

While Bayesian networks can efficiently compute the likelihood of a complete observation, it is with considerably more difficulty that they compute a conditional probability or likelihood of a partial observation. Since inference requires exponential time, in many cases a lengthy, iterative approximation is the best method available.

We compared the performance of Bayesian networks to that of PAL model three for conditional likelihood using the Dense MSWeb dataset. We would have liked to use the original MSWeb dataset as well, but ran into trouble with running Gibbs sampling on the Bayesian network: in order to compute accurate likelihood estimates, we had to sample for 30 seconds per test example. Since we are using 5,000 test examples, a single experiment would take over 40 hours. Due to time constraints, these experiments had to be omitted. Nevertheless, this example serves to show that

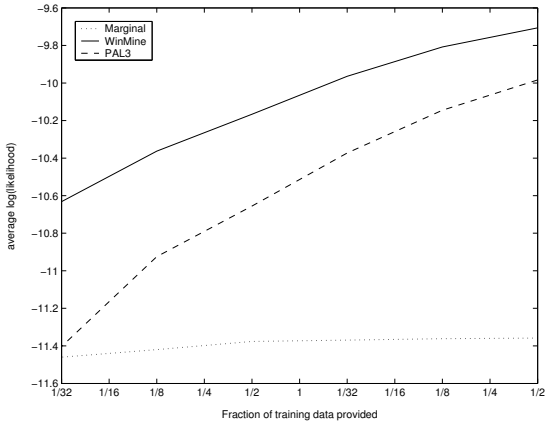

Figure 1: Dependency of third PAL model and WinMine model on number of training examples used, relative to the marginal model.

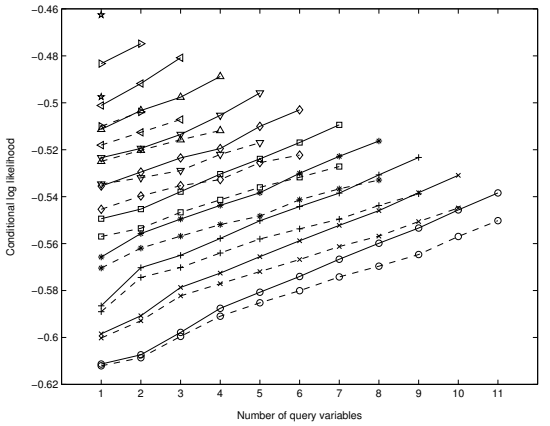

Figure 2: Conditional likelihood performance of third PAL model (solid lines) and WinMine Bayesian networks with 1000 iterations of Gibbs sampling (dashed lines). Each line holds the number of conditioning variables constant at some number 0 through 10. For either model, lines with higher conditional likelihood are those that had more conditioning variables.

the running time of Gibbs sampling on a Bayesian network is significant, even problematic. The PAL model was able to the same task in less than a minute.

On the Dense MSWeb dataset, we ran one experiment for each possible number of conditioning and query variables. In a first test run, we simply hid portions of the original 267 test examples for the conditioning and query variables. However, with only 267 examples, the sampling variance from randomly choosing which variables to hide resulted in a jagged graph. By duplicating the original examples 100 times before doing the hiding, we were able to decrease the variance and smooth the graph. Our results for Dense MSWeb are shown in figure 2. The PAL model three consistently outperforms the WinMine model, and it does so by an increasingly large margin as the number of query or conditioning variables is increased.

6 Conclusion

We have presented a novel alternative to traditional graphical probabilistic models, based on a mixture of abstractions rather than a graph of conditional dependencies. In one of the datasets we used, our new approach outperformed one of the leading Bayesian network algorithms. For conditional likelihood estimation, it was faster by several orders of magnitude while achieving greater accuracy.

For future work, we plan to continue exploring the application of abstraction-based models to other domains in order to determine where their true strengths and weaknesses lie. One of the next steps will be to learn abstractions inductively rather than relying on a frequent set algorithm, then to experiment with multi-valued and continuous data.

References

Agrawal, R., Imielinski, T., and Swami, A. N. (1993). Mining association rules between sets of items in large databases. In Buneman, P. and Jajodia, S., editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C.

Agrawal, R. and Srikant, R. (1994). Fast Algorithms for Mining Association Rules. In Bocca, J. B., Jarke, M., and Zaniolo, C., editors, *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann.

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

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

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