A Brief Intro to Bayesian Belief Networks

Elliot Pickens

November 19, 2019

Abstract

Frequentist statistical learning algorithms are currently the dominant form of "machine learning" used today, but they are far from the only form of "machine learning" in use. Possibly the greatest competitor to that class of algorithms lies in Bayesian statistical learning. Bayesian methods offer nearly all the capabilities of their frequentist counterparts, while allowing for a greater degree of probabilistic reasoning. In this paper I introduce the a few of the fundamental tools used in Bayesian statistical learning and show how they relate to more commonly used Bayesian algorithms.

1 Overview

Key to all Bayesian methods are joint probability distributions. These joint distributions are used to describe how sets of random variables interact with one another. Each random variable has an associated probability distribution, so in the context of building a predictive model we can imagine each feature being given a random variable.

Joint distributions have a habit of getting quite complicated, so probabilistic graphical models (PGMs) are frequently used to describe the ways in which variables affect one another by graphically representing a joint probability distribution. In addition to benefiting interpretability, PGMs open new opportunities for computational efficiency by combining Graph theory with Probability theory. [1]

In this paper I focus on unravelling one sub-class of probabilistic graphical models: *belief networks*. Belief networks are a type of directed graphical model (DGM) and are sometimes referred to as Bayes Nets.

2 Bayes Rule

Before we can create Bayesian Networks we must first briefly discuss their namesake: Bayes Theorem. With Bayes Theorem we can succinctly relate the conditional probability of two (or more) random variables and find the probability of Y (which is yet to be observed) given some X (which can be loosely considered to be some form of evidence). [3]

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)} \tag{1}$$

It should also be noted that Bayes Theorem shares a direct relationship to marginal probability. The $p(X)$ in equation 1 can be re-written as a marginal probability that acts as a normalizing constant in Bayes Theorem. [3]

$$p(X) = \sum_{Y} p(X|Y)p(Y) \tag{2}$$

The standard form of Bayes Theorem can be seen in equation 1, but for many applications the $p(X)$ in the denominator is ignored to reduce the computation required to calculate posterior probabilities leaving us with the proportional form: [5]

$$p(Y|X) \propto p(X|Y)p(Y) \tag{3}$$

3 Belief Networks

Belief networks or Bayesian Networks are DGMs used to represent the independence assumptions made in the creation of a probability distribution. By illustrating the independence assumptions present in a given distribution, Bayes nets can be easily (and are generally) used to show casual relationships between random variables. [1]

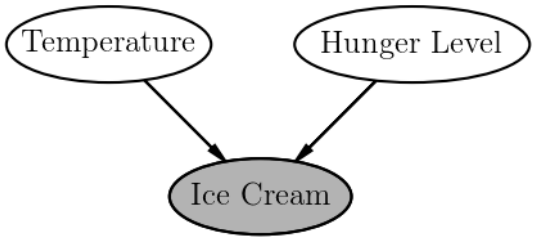

Figure 1: PGM Approximating Bat's Choice to Eat Ice Cream

For example, consider the network shown above. Imagine that on a given day we make an observation as to whether or not my friend Bat ate ice cream. Every time we observe Bat we record whether or not he ate some ice cream, but we also want to know why we observed what we did. One way to peak at what might be causing our observation would be to create a simple PGM where Bat's perceived hunger level and the temperature outside both influence Bat's decision. Using such a model we can then use hunger and temperature observations to infer about what Bat's choice will be.

More formally, we can define a belief network can be defined as a directed acrylic graph (DAG) $G = \langle V, E \rangle$. Since this graph is representing a belief network, each vertex $v \in V$ is a random variable X_v and each edge $e = (u, v) \in E$ is a dependency between two random variables X_u and X_v. This structure leaves us with the somewhat odd looking probability distribution function: [1]

$$p(x_1, ..., p_V) = \prod_{i=1}^{V} p(x_i | pa(x_i)) \tag{4}$$

In equation 4, $pa(x_i)$ is used to represent the parents of x_i, and each term $p(x_i|pa(x_i))$ is a conditional probability table (or CPD)[1]. [1] [7] Equation 4 should also clarify why we require that all belief networks are DAGs. Without the assumption that our network is a DAG (and assurance that there are no internal cycles) we cannot calculate the joint distribution given by equation 4, because $pa(x_i)$ becomes meaningless. Each CPD denotes the distribution of the random variable at some vertex X_v that has been conditioned over the values of the random variables it is dependent upon in the graph: $D(v) = \{u|(u.v) \in E\}$. [1]

Going back to our example with Bat, we have a PGM representing the relationships between our variables, but we lack the CPDs required to calculate probabilities. In tables 1, 2, and 3 I have laid out a set of example CPDs for Bat's behavior. In this case, I simply chose values I thought made sense (a loosely-informed prior), but I cannot say with any certainly that these values are correct. Even with expert knowledge it can be difficult to select valid values for CPDs, so it is common to learn them from the data being worked with[2].

Table 1: Temperature (Hot)

T	F
0.7	0.3

Table 2: Hunger Level (Hungry)

T	F
0.6	0.4

[1] In this paper I use the notation $pa(x_i)$, but the notations $x_pa(t)$ and pa_k are also common [3] [7] [6]

[2] That learning process is known as parameter learning and is covered in section 4

Table 3: Ice Cream (Consumed)

		Ice Cream	
Hungry	Hot	T	F
F	F	0.2	0.8
F	T	0.7	0.3
T	F	0.7	0.3
T	T	0.9	0.1

The above CPDs can be combined with the equations in section 1 to solve for unknown probability values. For example, imagine we want to find the probability that it was hot outside on a day that Bat chose to eat ice cream. We can solve for that value using Bayes Theorem and some simple marginalisation. [5]

$$\begin{aligned} p(Hot = T|Consumed = T) &= \frac{p(Consumed = T|Hot = T)p(Hot = T)}{p(Consumed = T)} \\ &= \frac{p(Consumed = T, Hot = T)}{p(Consumed = T, Hot = T) + p(Consumed = T, Hot = F)} \end{aligned} \tag{5}$$

Which when marginalised becomes:

$$p(Consumed = T, Hot = T) = \sum_{Hungry} Consumed = T, Hot = T, Hungry) \tag{6}$$

$$p(Consumed = T, Hot = F) = \sum_{Hungry} Consumed = T, Hot = F, Hungry) \tag{7}$$

Where:

$$p(Consumed, Hot, Hungry) = p(Consumed|Hot, Hungry)p(Hot)p(Hungry) \tag{8}$$

This is setup and the calculations associated with it are somewhat tedious, but not difficult. That simplicity makes the applications of belief networks computationally tractable even if the tedium prevents many algorithms that operate on belief networks from being truly fast.

4 Discovering Belief Networks

The previous sections described the basic structure of a belief network and a basic example as to how they can be applied. In this section we will discuss how we can build (or learn) a belief network from data via two key problems: structure learning and parameter learning.[1]

Structure learning is used to find the DAG that best represents the relationships and dependencies between features in a given data set. This portion of learning is used to find the edges (and directions of edges) best suited to the data.[1]

Parameter learning takes in a data set and a DAG and then tries to find the parameter values that best suit it. During parameter learning the parameter values being learned are the conditional probability distribution values.[1]

4.1 Structure Learning

Structure learning is a very difficult and by no means fully solved problem. The problem is commonly mired in computational complexity issues that can often result in very exponential run times. [1]

Put formally, the task of structure learning is finding the DAG ($G = \langle V, E \rangle$) which best maximizes $\mathbb{P}(G|D)$ (where D is our data). $\mathbb{P}(G|D)$ is sometimes also expressed as: [1]

$$\mathbb{P}(G|D) \propto \mathbb{P}(D|G)\mathbb{P}(G) \tag{9}$$

When searching for G it is possible to set the prior $\mathbb{P}(G)$ that appears in equation 9. In the graphical context of belief networks, a prior describes the expected relationship between any two nodes. For example, a uniform prior would give every pair of nodes $(i, j) \in V$ a probability of $\frac{1}{3}$ of having an edge from $i \to j$, an edge $j \to i$, and no edge. Normally a basic uniform prior is chosen given that it is difficult to know the correct graph structure in advance, but it is possible to input an informed prior if desired. Instead of using an informed prior it is more common to start from a uniform prior and tweak it slightly to favor or disfavor having an edge at all (i.e. $p(no\,edge) > \frac{1}{3}$ or $p(no\,edge) < \frac{1}{3}$). [1]

4.1.1 Greedy Search

The most straightforward way to find the optimal DAG structure for our data is to simply perform an exhaustive greedy search over all possible structures[3]. The problem with this method is that when more than 5 variables are present in the data it becomes computationally unrealistic. With n variables we have $\frac{n*(n-1)}{2}$ possible edges and $2^{\frac{n*(n-1)}{2}}$ possible structures (when accounting for direction of the edges). Thus, an exhaustive search could potentially require the inspection of more than 2^{n^2} structures, which is simply unrealistic for anything other than a small n. [1] [8]

4.1.2 Constraint-based Learning

Constraint-based learning is a class of structure learning algorithms that seek to use conditional independence tests to find the DAG that produces the best belief network for our data. This class of algorithms generally works in three steps:

- Reduce the number of possible DAGs (possibly by examining the Markov blanket of each node[4]) [8] [1]
- Find the un-directed edges that will be present in the final DAG (using a conditional independence tests like $\chi - squared$) [8] [1]
- Convert the un-directed edges to directed ones to produce our final DAG [8] [1]

Although this is a general class of algorithms, many of those currently in use owe their roots to the *inductive causation* algorithm. The initial IC algorithm used the final two steps to generate DAGs, but lacked the first pruning step and much of the other optimization that can be presently found in constraint-based learning algorithms. [8]

4.1.3 Score-based Learning

In score-based learning we assign each candidate DAG a score intended to represent its goodness of fit. That score is then maximized in order to find the best possible DAG for our belief network. As was the case with constraint based learning, score-based learning does not describe a single algorithm, but rather a class of algorithms. Each of the algorithms within this class use a score to evaluate the quality of a DAG, but each algorithm performs its search of the DAG space in a different manner. A few of the most common search methods are greedy search, simulated annealing, and genetic algorithms. [8]

Although simulated annealing and genetic algorithms are very interesting ways to go about score-based learning, it is common to default to the easiest solution: greedy search. One greedy search algorithm commonly used to find belief networks is hill climbing. In hill climbing we begin with a sub-optimal DAG form which we try to reach an optimal solution by making iterative changes. When building a DAG each iteration means deleting, reversing, or adding a new edge to the graph and then evaluating the new score. When the score can no longer be improved the algorithm terminates, outputting the final version of the DAG. [8] [1]

While hill climbing in its purest form shines in its simplicity it does a few significant drawbacks. Among those drawbacks the most problematic is that hill climbing is not guaranteed to climb the correct hill. Hill climbing often settles on a local (not global) maximum and is prone to produce sub-par results when starting from a sub-par position. These downsides can be mitigated to a degree with a few clever additions to the hill climbing algorithm, but its important to note some of the issues that can be seen when working with the hill climbing algorithm (and other greedy algorithms). [1]

[3] Some libraries like `pgmpy` even have built in exhaustive search algorithms for finding DAGs [2]
[4] The Markov blanket of a node v consists of the parents, children, and children's parents of v [1]

4.2 Parameter Learning

Once we have learned the structure of our belief network using structure learning we need to optimize the parameters of that structure to align with our data. The two most prominent approaches to optimizing these parameters are maximum likelihood estimation (MLE) and Bayesian estimation. [8]

Maximum likelihood estimation uses the relative frequencies of variable states to build CPDs. MLE maximizes $p(D|BN)$ (where D is our data and BN is our belief network). This approach is generally effective, but when not much data is available it has a tendency to be quite fragile. Returning back to our example with Bat, if we only have a single day's observation using MLE will zero out most of the probabilities in the CPDs, because only a single state will have been observed for hunger, temperature, and ice cream. [8] [1]

Bayesian estimation uses prior CPDs to express our beliefs about the variables. One common prior used in Bayesian estimation is a simple count adjustment to account for small small sizes. Such a prior effectively adds a n occurrences of each variable state to the CPD. In the version of Bat's example where we only have a single day's worth of observations we could use a prior that added a single count to each variable state to avoid zeroing out our probabilities. [8] [1]

5 Naive Bayes

Although it is not frequently introduced as such, the naive Bayes classifier is a specific form of a belief network. A naive Bayes classifier is simply a belief network where we apply the assumption that all variables are conditionally independent. Under this assumption we can then write the joint distribution as: [7]

$$p(y, x) = p(y) \prod_{j=1}^{D} p(x_j|y) \tag{10}$$

The naive Bayes classifier is just one example of a special case of a belief network. Many other well known models like Markov and hidden Markov models are also special cases of belief networks. [7]

6 Data

For this project I used the well known Wisconsin Breast Cancer Diagnostic data set. [4] I chose this data set, because some of the most effective applications of belief networks can be found in the medical field and secondarily because of its relatively small size. Given the computational difficulties associated with structure learning this data set allows for realistic demonstrations of the algorithms presented in section 4 while avoiding painful run times.

7 Implementation

To demonstrate the algorithms described in section 4 I used the `pgmpy` library. The `pgmpy` python library was designed for working with graphical models. The library has implementations of many of the most important algorithms used in graphical models[5]. [2]

Using `pgmpy` I first fit two belief networks to a training set made up of 80% of the original data set (randomly selected). The first belief network was fit using hill climbing (to find the structure) and Bayesian estimation (to find the parameters). The second belief network was fit using constraint-based learning and maximum likelihood estimation.

These two learning pipelines produced two very different looking DAGs. The score-based learning approach with hill climbing output something that looked very similar to a naive Bayes classifier, while the constraint-based approach output a graphical model where the class was only dependent on bland

[5]Many thanks to the authors of `pgmpy` for your extensive documentation

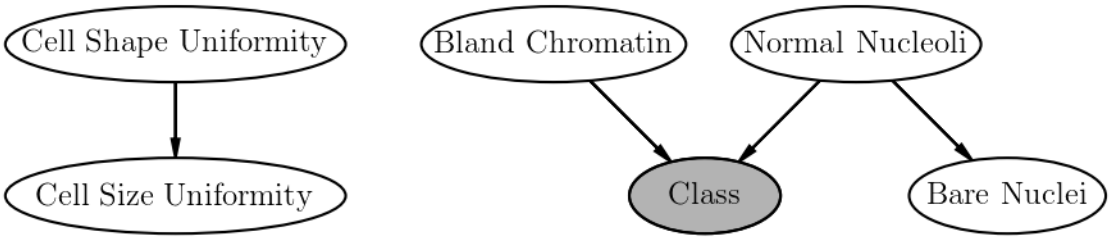

Figure 2: Belief Network Learned Using Constraint-based Learning

chromatin and normal nucleoli[6].

Given the similarity between the model learned using hill climbing and a naive Bayes model, I also implemented a naive Bayes model using `pgmpy's` built in naive Bayes model.

Finally I wanted to compare the predictive power of the of the two belief networks to more well established methods, so I fit logistic regression, random forest, and naive Bayes classifier to the training data using `sklearn`. It should be noted that I scaled all values when working with `sklearn`, so the data processing pipeline was slightly different.

With all 6 models in hand, I tested each of them on the 20% of the data set that was held out as a test set to evaluate the accuracy of the models.

8 Results

As can be seen in table 4, the belief networks performed quite well. The model learned using hill climbing and Bayesian estimation even managed to score the highest accuracy and F1 scores of all models. The Bayes net learned using constraint-based learning preformed the worst in terms of F1 score (indicating it probably is not the best model), but still performed reasonably well overall.

Model	Accuracy	Precision	F1
BN (HC, BE)	**0.97**	0.94	**0.96**
BN (CB, MLE)	0.92	**0.95**	0.88
Naive Bayes (pgmpy)	0.96	0.91	0.95
Naive Bayes (sklearn)	0.95	0.88	0.93
Logistic Regression (CV)	0.96	0.94	0.94
Random Forest	0.95	0.94	0.93

Table 4: Model Results

These results suggest that belief networks can perform at the level of other commonly used models with just a small amount of tuning. They also echo the success belief networks have had in context of medical diagnosis. [7]

9 Conclusion

Belief networks are powerful and fundamental tools in Bayesian machine learning. Bayes nets provide an interpretable probabilistic approach to creating classifiers, and are the basis of many of the more commonly used Bayesian machine learning methods like naive Bayes. [7]

[6]The model output from hill climbing was too large to be shown using a graphic, but can be seen in the `bayes_net.py` file associated with this paper

References

[1] Danish A. Alvi. *Application of Probabilistic Graphical Models in Forecasting Crude Oil Price*. Papers 1804.10869. arXiv.org, Apr. 2018. URL: https://ideas.repec.org/p/arx/papers/1804.10869.html.

[2] Ankur Ankan and Abinash Panda. "pgmpy: Probabilistic graphical models using python." In: *Proceedings of the 14th Python in Science Conference (SCIPY 2015)*. Citeseer. 2015.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[5] Peter D. Hoff. *A First Course in Bayesian Statistical Methods*. 1st. Springer Publishing Company, Incorporated, 2009. ISBN: 0387922997, 9780387922997.

[6] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. Adaptive computation and machine learning. MIT Press, 2009. ISBN: 9780262013192. URL: https://books.google.co.in/books?id=7dzpHCHzNQ4C.

[7] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020, 9780262018029.

[8] Marco Scutari and Jean-Baptiste Denis. *Bayesian Networks with Examples in R*. ISBN 978-1-4822-2558-7, 978-1-4822-2560-0. Boca Raton: Chapman and Hall, 2014.