Bayesian Networks

Gwyn, Durvasula, Hurst, and Arora

UC Berkeley

2020

Where We Are

- Previously within this course we have covered some of the ideas of graph theory with basic neural networks. State Transition Diagrams in 16A also gave us a slight flavour of this.
- Continuing with the ideas of using graphs to encode information and quickly and efficiently compute classifications and decisions, we now introduce Bayesian Networks.
- Similar to how neural networks propagate values down their graph, so do Bayes Nets. In a Bayes Net, every node represents concrete real features, and the values we propagate down are the conditional probabilities of that node based on the nodes above (the parent nodes).
- Bayes Nets also utilize hyper-parameter turning to solve for the most accurate graph model, and data regularization as we will see later on.

Probability Distributions

- The *probability distribution* of some variable or experiment lists the probability of each possible outcome of that experiment (or value of that variable).
- An event is anything with randomness, such as flipping a coin or rolling a die (strictly speaking, events need not be random, like a die with six on all sides).
- Here's the probability distribution for the outcome of rolling a die:

Value	Probability
1	1/6
2	1/6
3	1/6
4	1/6
5	1/6
6	1/6

Probability Distributions

- We can calculate the probability of an *event* (a collection of different outcomes) by counting the number of outcomes making up the event and dividing by the total number of outcomes.
  - As an example, there are three outcomes of a dice roll that are *even* (an even outcome would be an example of an event). The probability that a dice roll comes up even is

    $$\frac{3}{6} = \frac{1}{2}$$

    as we'd expect.
- We can also define probability distributions for *continuous variables*, like how far a football goes when you throw it.
  - An event, in this case, could be something like the football travelling more than 20 yards.

Conditional Probabilities

- Sometimes one random variable might have some effect on or relationship with another one. We can compute the *conditional probability* of one random variable based on the value of another. We denote this by $\mathbb{P}[A \mid B = b]$, where A and B are events.
- As an example, consider the sum of two fair dice rolls. The probability that this sum equals 12 is equal to the probability that each dice roll was a 6, which is

  $$\mathbb{P}[S = 12] = \mathbb{P}[D_1 = 6]\mathbb{P}[D_2 = 6] = \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36}$$

  However, the probability that the sum equals 12 *conditioned* on the first dice roll having been a 3 is 0, as no matter what, the sum cannot equal 12 in this case.
- Because the probability distribution of S changes based on D_1, we say that S and D_1 are *dependent* (the opposite case being called *independent*).

Bayes' Law

- To solve for conditional probabilities, we make use of the fact that

$$\mathbb{P}[E_1 \cap E_2] = \mathbb{P}[E_1]\mathbb{P}[E_2 \mid E_1] = \mathbb{P}[E_2]\mathbb{P}[E_1 \mid E_2]$$

- Some quick algebra gives *Bayes' law*:

$$\mathbb{P}[E_2 \mid E_1] = \frac{\mathbb{P}[E_2]\mathbb{P}[E_1 \mid E_2]}{\mathbb{P}[E_1]}$$

- We often expand Bayes' law further when actually using it to obtain the form

$$\begin{aligned}\mathbb{P}[E_2 = b \mid E_1 = a] &= \frac{\mathbb{P}[E_2 = b]\mathbb{P}[E_1 = a \mid E_2 = b]}{\sum_{e_2} \mathbb{P}[E_1 = a \cap E_2 = e_2]} \\ &= \frac{\mathbb{P}[E_2 = b]\mathbb{P}[E_1 = a \mid E_2 = b]}{\sum_{e_2} \mathbb{P}[E_2 = e_2]\mathbb{P}[E_1 = a \mid E_2 = e_2]}\end{aligned}$$

Bayes' Nets

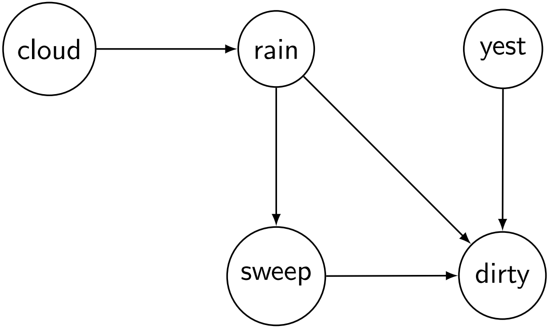

Figure: A Bayesian network modelling clouds, rain, the presence of a street sweeping machine, my car's prior cleanliness, and my car's current cleanliness.

Nodes and Edges

- Each *node* of our network represents a random variable.
  - We might have a node called `raining` which is `true` if it rains today and `false` otherwise.
  - or a node called `dinner` which could be one of `steak`, `salad`, or `sushi`.
- *Edges* or *arrows* in our network symbolize *real-world dependence* of one variable on another.

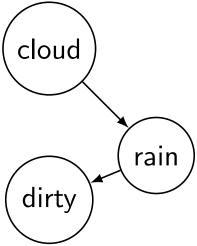

Figure: Clouds give the possibility of rain. Rain can make my car dirty.

Directed Acyclic Graphs (DAGs)

- Bayes' nets have the structure of a *graph*, which is nothing more than a collection of nodes and edges like we just described.
- Some graphs have edges that aren't arrows, but Bayes' nets don't.
- Bayes' nets have the important property of not containing any cycles. As an example, the network on the previous slide is just, but this one is not:

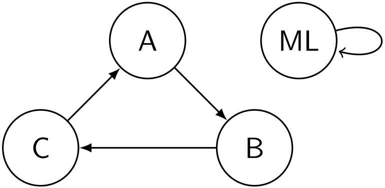

Figure: A directed graph with two cycles (ML) and (A, B, C) (course dependencies).

Conditional Probability Tables

- Each node in our network has a set of *parents*, which are the nodes that point to it and hence affect its value.
- To be completely specific, we need to store the *conditional distributions* of each node based on the values of its parent nodes.
- We do this with a *conditional probability table (CPT)*. For example:

dirty		
Condition	$\mathbb{P}$[yes]	$\mathbb{P}$[no]
R, Y, S	0.9	0.01
$R, Y, \neg S$	0.6	0.1
$R, \neg Y, S$	0.6	0.1
$R, \neg Y, \neg S$	0.3	0.7
$\neg R, Y, S$	0.7	0.3
$\neg R, Y, \neg S$	0.5	0.5
$\neg R, \neg Y, S$	0.4	0.6
$\neg R, \neg Y, \neg S$	0.05	0.95

Inference

- Once we have a complete Bayes' net (directed graph *and* conditional probability table for each node), we want to use it.
- The fundamental calculation we need to perform is to calculate the *joint distribution* of all nodes in the network (this is the distribution of all their values together).
- The conditional dependencies represented by our graph along with the distributions contained in our CPTs are sufficient to allow us to write

$$\mathbb{P}[v_1, \ldots, v_n] = \prod_{i=1}^{n} \mathbb{P}[V_i = v_i \mid P_1 = p_1, \ldots, P_{k_i} = p_{k_i}]$$

where the nodes P_j are the parents of the node V_i.

Inference

- We can use this approach to enable us to learn about unknown variables given known ones.
- Say we know the values of $K_1, K_2, \ldots, K_n$ to be $k_1, \ldots, k_n$ respectively, and we are interested in the variable I. Then we can calculate

$$\mathbb{P}[I = i] = \frac{\mathbb{P}[I = i \cap K_1 = k_1, \ldots, K_n = k_n]}{\mathbb{P}[K_1 = k_1, \ldots, K_n = k_n]}$$

  using the law of conditional probability.
- If we wanted to predict the value of I, then, we could simply choose the value i which maximizes the value of the above fraction.

Variable Elimination

- Calculating values of the joint distribution over and over again can become very slow, and even a single calculation can come to be slow if we have many nodes and many arrows in our Bayes' net.
- We can capitalize on the independence relationships given by the Bayes' net structure via the *variable elimination* algorithm.
- We can go through each variable in our network, identify all of the conditional probabilities that include that variable, and compute these probabilities for *all possible values* of the associated variables (which, in the general case, is not all of the variables).
- In the nicest cases (e.g. a Bayes' net which includes separate notes for variables at different times), we can reduce the exponential brute-force algorithm down to polynomial time.

Data for Parameter Learning

- The CPTs are almost never available within practical use cases of Bayes' nets.
- Rather, the entire point is to *learn* what the conditional probability distributions are.
- We start with a data set:

Cloud	Rain	Sweep	Dirty	yest
T	T	F	T	F
T	F	F	T	T
F	T	T	F	T
T	F	F	T	F
F	F	T	T	F
F	F	F	F	T
F	T	T	F	T
T	T	T	T	T

Figure: Small data set for our prior Bayes' Net, with complete entries.

What are our parameters

- If the structure of our Bayesian network is already known to us, then we also know the structure of the CPT for each node. We are missing, however, the numerical entries in the CPT.
- That is, we start with

dirty		
Condition	$\mathbb{P}$[yes]	$\mathbb{P}$[no]
R, Y, S		
$R, Y, \neg S$		
$R, \neg Y, S$		
$R, \neg Y, \neg S$		
$\neg R, Y, S$		
$\neg R, Y, \neg S$		
$\neg R, \neg Y, S$		
$\neg R, \neg Y, \neg S$		

Empirical Distribution

- Note that the parameters making u the CPT that we are training are equal to conditional probabilities. We can estimate these probabilities as

  $$\theta_i^n = \mathbb{P}[N = n | S = s_i] = \frac{\#D(N = n, S = S_i)}{\#D(S = S_i)}$$

  where $D(\texttt{condition})$ signifies the set of all of our data points which satisfy `condition`.
- In other words, we are using the *empirical distribution* of an event or variable to estimate the *actual distribution*, and then we apply the law of conditional probability to estimate the conditional probability of one event on another.

Algorithm for Parameter Learning

- for every node/feature in the network
    - create a CPT
    - for every value N which the node can take on
        - for every combination of values the parents can take on
        - `numerator=denominator=0`
        - for all samples in dataset
        - if sample satisfies conditions on N and S increment numerator and denominator
        - else if sample satisfies conditions on S, increment denominator.

Small or generic datasets

- Within our naïve approach to parameter learning, if a particular even is not within our training data, then its conditional probability of occurring will be set to zero. The majority of the time, this does not actually represent the true probability of an even occurring (we just have rare events and a small sample size).
- It is not in our best interest the majority of the time to have a distribution of zero, and for this reason we make use of Laplace smoothing.

Laplace smoothing

- Laplace smoothing introduces a hyperparameter k to our Bayes' net.
- What Laplace smoothing does is assume every possible event occurs at least k times, essentially augmenting our data set to assure that each event has some nonzero probability.
- We can implement this rather simply with the following change to the empirical distribution:

$$(\text{Laplace})\theta_i^n = \frac{\#D(N = n, S = S_i) + K}{\#D(S = S_i) + (K \cdot B)}$$

where B is the size of the domain for our argument.

The story so far ...

- We often know the structure of a Bayes' net through domain knowledge or some other prior information.
- We might know the CPT entries in advance; otherwise, we estimate them.
- But what occurs if we don't have any priors or any knowledge of the structure of our Bayes' net? In that case, we don't even know the shape of the CPTs, let alone the values that fill them.
- The ability to "estimate" or learn the structure of Bayesian network is hence a crucial aspect of using such a model. But how?

Yes! Data is your friend

- Like we did in parameter learning, we can leverage our data to find a Bayes Net Structure.
- Remember, a Bayes Net is a DAG. So, using the same idea as in parameter learning, we want to find a DAG (let's call it G), that maximizes the likelihood of the data (let's call it D) we have.
- We want to consider all DAG's G, and choose the one that maximizes the log likelihood (LL) of our data D. Mathematically, we can define the LL as:

$$\mathcal{L}\mathcal{L}(G; D) = \prod_{d \in D} \mathbb{P}(d \mid G)$$

Intuition

- Intuitively, our data comes from an underlying true structure G_{true} , and thus G_{true} is the G that maximizes $LL(G; D)$ (along with a regularization term to encourage sparse and simple structures to avoid overfitting).
- Our goal has now become to find a G that ideally $= G_{true}$, or is as close to it as possible.

Algorithm 1: Score and Search

- Consider all possible DAG's G given the variables you have, and assign each of them a score $= LL(G; D)$. Choose the G with the highest score.
- This should be pretty intuitive, but as you may have noticed, its runtime is exponential with regards to the number of variables (Super Exponential).

Algorithm 2: Approximate Learning

- Score and Search is great and makes sense, however its runtime is horrible.
- Approximate Learning Algorithms reduce the search space of Score and Search via methods like:
  - Imposing constraints on the structure of the output. (Eg: Has to be a tree, cannot have more than x edges, etc.)
  - Heuristics and A* search.
  - Dynamic Programming
  - Hill Climbing Methods, Simulated Annealing etc.
- What does this mean for runtime?
- Accuracy/Performance?

Algorithm 2a: Chow Liu Trees

- Output must be a tree.
- Procedure:
  - Add every edge possible with a weight equal to the mutual information (I) between the two variables in question. A higher weight means a higher mutual dependency between the two variables.
  - Using all the edges and their weights, construct a maximum weight spanning tree - ie choose the edges that form a sparse fully connected structure while trying to use edges to encode high mutual dependencies between variables.
  - Choose any root node and assign edge directions such that you have a Directed Tree where each node only has one parent except the root node.
  - Note: Edge directions don't matter for likelihood as Mutual Information is symmetric.

Chow Liu Trees continued

- One of the quickest approximate algorithms.
- Runtime $= O(v^2)$ where v is the number of variables in the BN.
- Equation to find weight I of the edge (u, v):

$$I(u, v) = \Sigma_{u \in U} \Sigma_{v \in V} \mathbb{P}[u, v] \log_2 \frac{\mathbb{P}[u, v]}{\mathbb{P}[u]\mathbb{P}[v]}$$

- Using these weights construct a Maximum Weight Spanning Tree: A tree such that the sum of the weights of the edges it has is maximised.

Algorithm 3: Constraint Learning

- Yet another class of algorithms for Bayes Net Structure Learning (BNSL).
- Involve calculating correlation or co-occurrence to identify an undirected backbone of edges that could exist in true G. Prune these edges systematically until a DAG is reached.
- Not as intuitive as the previous algorithms as there is no probabilistic model or function over here. These class of algorithms have been described as you should be aware they exist but are not in scope.

Conclusion

- Often times, we may not know the structure of a Bayes Net but may have data from which we can learn the structure.
- Learning consists of finding a DAG that maximizes the log likelihood of the data being generated from that DAG structure. Thus we have a way of scoring each DAG.
- Number of DAG's is exponential with number of variables, so we resort to approximate methods such as the Chow Liu Tree Algorithms.
- There is an inherent tradeoff between runtime of algorithms and how 'good' a structure they can recover.
- You now know a lot about Bayes Nets!

References

- https://www.biostat.wisc.edu/ craven/cs760/lectures/BNs-2.pdf
- https://www.jmlr.org/papers/volume7/niculescu06a/niculescu06a.pdf
- https://arxiv.org/ftp/arxiv/papers/1304/1304.2736.pdf