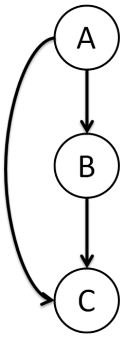
CSC 446 Notes: Lecture 5

Typed by Ian Perera

February 7, 2012

1 Bayesian Networks Continued

A Bayesian network is a convenient graphical model for illustrating conditional probabilities and independence between variables. Take the following network: We see the number of parameters for each node is



P(A)

P(B|A)

P(C|A,B)

A	
0	.6
1	.4

A	B	
0	0	.1
0	1	.5
1	0	.2
1	1	.2

A	B	C	
0	0	0	.1
0	1	0	.2
1	0	0	.1
1	1	0	.1
0	0	1	.1
0	1	1	.2
1	0	1	.1
1	1	1	.1

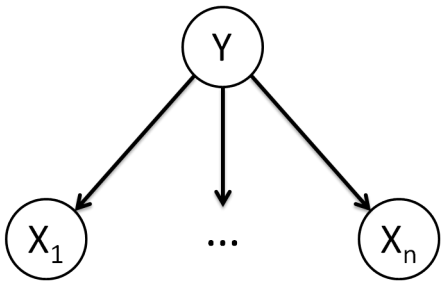
equal to 2^{k+1}, where k is the number of parents for that node. We can compare this to the application of joint probability as the product of conditionals,

$$P(X_1, ...X_N) = \prod_{n=1}^{N} P(X_n|Par(X_n))$$

where the number of parameters is 2^N, with N is the number of variables. We can represent the Naive Bayes assumption, namely,

$$P(Y|X_1, ...X_N) \approx P(Y) \prod_{n=1}^{N} P(X_n|Y)$$

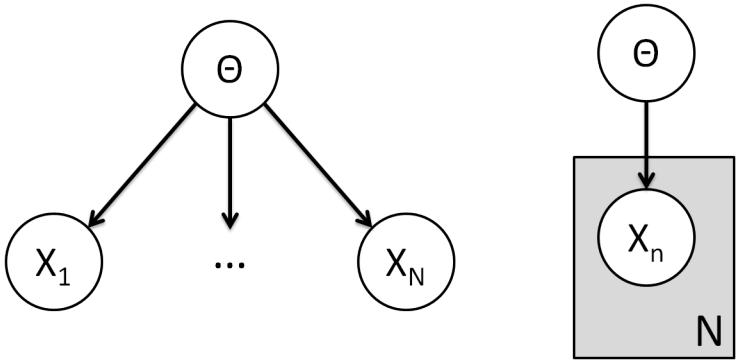
with the following network:



Likewise, we can represent the Dirichlet distribution,

$$P(\theta) = Dir(\alpha) = \frac{1}{Z}\prod_k \theta^{\alpha_k - 1}$$

with the following network (alternate notation on right):



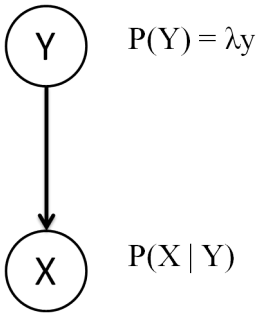
A mixture of Gaussians, with each Gaussian having the distribution

$$\mathcal{N}(x; \mu, \Sigma) = \frac{1}{Z}\exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}$$

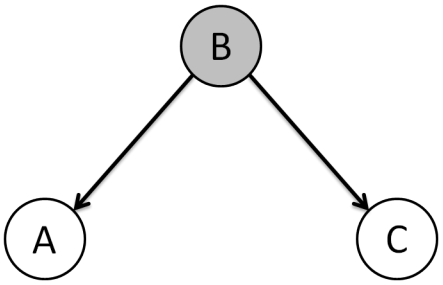
and the mixture distribution

$$P(x) = \sum_{i=1}^{K} \lambda_i \mathcal{N}(x; \mu_i, \Sigma_i)$$

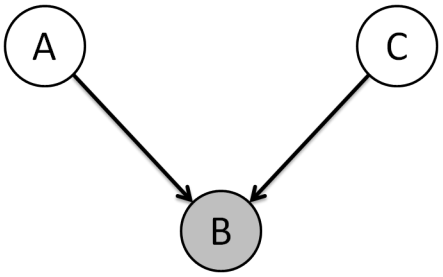
where λ is the weight of the Gaussian, can be represented using the following diagram:



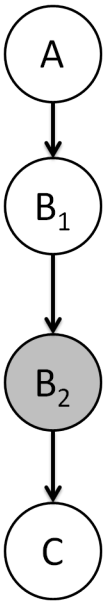
Here, $P(X|Y)$ is the probability of a variable having already chosen the Gaussian y. When determining conditional independence, one can think of knowing a variable in a Bayesian network as "blocking" dependence. In the following diagram, A is not independent of C ($A \not\perp\!\!\!\perp C$), yet, given B, A and C are independent ($A \perp\!\!\!\perp C|B$).



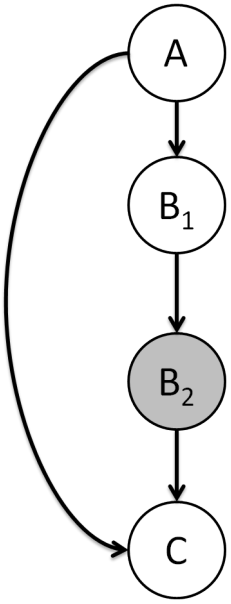
However, if a variable is conditioned on two independent parents, then knowledge of that variable makes those parents no longer independent. This is called "explaining away" evidence. In the following network, $A \perp\!\!\!\perp C$, but $A \not\perp\!\!\!\perp C|B$.



Knowing the value of a variable anywhere along a hierarchy path breaks the dependence between the variable and its ancestor. In the following network, $A \perp\!\!\!\perp C|B_2$:

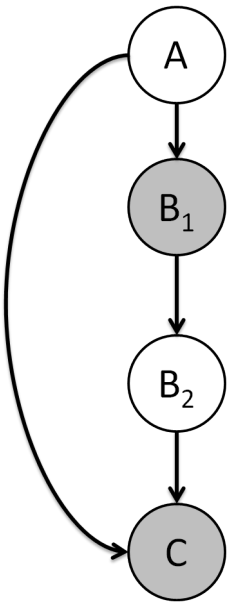


However, if there exists an alternate path, then the two variables are still dependent.For the network

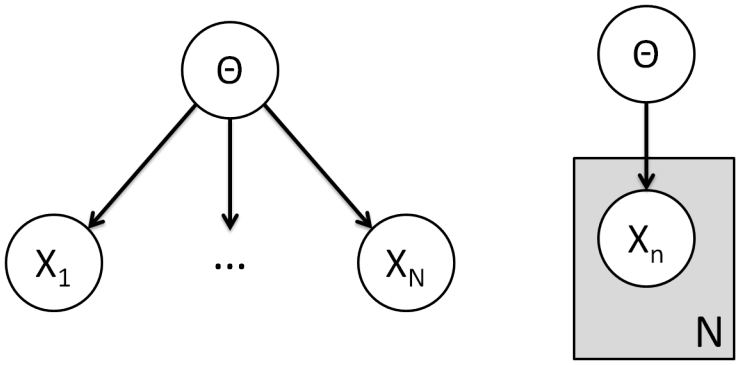


$A \not\perp\!\!\!\perp C|B_2$, since there is still an unobstructed path from A to C.

Knowledge of a variable can also travel up the arrows to cause a dependence on parent variables that were rendered independent by evidence somewhere along the same path. In the following network, $A \perp\!\!\!\perp B_2|B_1$, but $A \not\perp\!\!\!\perp B_2|B_1, C$.



Returning to our Dirichlet distribution, we see that none of the variables are independent of each other, unless θ is known - in which case, all of the variables are independent of each other.



$$P(X_{N+1}, X^N) = P(X^N)P(X_{N+1})$$
$$X_1 \perp\!\!\!\perp X_2|\theta$$
$$P(X_{N+1}) = \theta_{X_{N+1}}$$

2 Perceptrons

A Perceptron is a linear classifier that determines a decision boundary through successive changes to the slope of a line according to a binary feature. The classifier finds values for the weight vector $\mathbf{w}^T$ to solve the equation

$$\mathbf{0} = \mathbf{w}^T\mathbf{x} + \mathbf{b}$$

We define our classification function sign as

$$sign(x) = \begin{cases} -1 & : x < 0 \\ 0 & : x = 0 \\ 1 & : x > 0 \end{cases}$$

We can remove $\mathbf{b}$ from the equation by adding it as an element of $\mathbf{w}$ and adding a 1 to $\mathbf{x}$ in the same spot.

$$\mathbf{w}' = \begin{bmatrix} w_1 \\ \vdots \\ w_N \\ b \end{bmatrix} \quad \mathbf{x}' = \begin{bmatrix} x_1 \\ \vdots \\ x_N \\ 1 \end{bmatrix}$$

$$\mathbf{w'}^T\mathbf{x}' = \mathbf{w}^T\mathbf{x} + \mathbf{b}$$

The next question is, how do we pick $\mathbf{w}$? We have $\mathbf{x}$ as the vector of data points, with $\mathbf{x}^n$ as the nth data point. We have t^n as the classifier output (1 or -1) for the nth data point.

$$t^n = sign(\mathbf{w}^T x)$$

y^n is the true label for the nth data point.

Our goal is to solve the following equation:

$$\underset{\mathbf{w}}{\text{argmax}} \sum_n I(y^n = t^n)$$

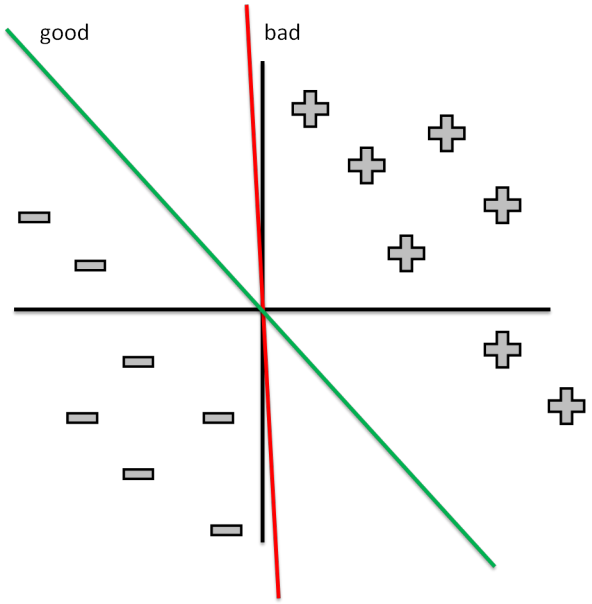
Or, reformulated as minimizing error,

$$\underset{\mathbf{w}}{\text{argmin}} \sum_n I(y^n \neq t^n)$$

To solve this equation, we want to add up all of the positive points to get a vector in that direction. This gives us the Perceptron algorithm:

repeat
  for $n = 1...N$ **do**
    if $t^n \neq y^n$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + y^n \mathbf{x}^n$
    end if
  end for
until $\forall n \; t^n = y^n$ or maxiters

While this algorithm will completely separate linearly separable data, it may not be the best separation (it may not accurately represent the separating axis of the data).



We can solve this problem by replacing our original error function,

$$E = \sum_n I(y^n \neq t^n)$$

with the following error function:

$$E = \sum_n \frac{1}{2}(y^n - t^n)^2 = \sum_n E^n$$

Now, the function E w.r.t. $\mathbf{w}$ is not convex, so we have to use an iterative method. We use Gradient Descent to solve for $\mathbf{w}$, which is the successive application of

$$\mathbf{w} = \mathbf{w} - \frac{\partial E}{\partial \mathbf{w}}$$

$$\frac{\partial E}{\partial \mathbf{w}} = \sum_n \frac{\partial E^n}{\partial \mathbf{w}} = \sum_n \frac{\partial E^n}{\partial t} \frac{\partial t}{\partial \mathbf{w}}$$

However, our sign function is not differentiable at 0, so we replace it with $\tanh(a)$:

$$t^n = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

where

$$a = \mathbf{w}^T \mathbf{x}$$

Plugging in, we get

$$\frac{\partial E}{\partial \mathbf{w}} = \sum_n (y^n - t^n) \frac{\partial t}{\partial a} \frac{\partial a}{\partial \mathbf{w}} \tag{1}$$

$$= \sum_n (y^n - t^n) g'(\mathbf{w}^T \mathbf{x}) \mathbf{x} \tag{2}$$

An alternative solution, which begins making small adjustments immediately, is Stochastic Gradient Descent. Having defined a learning rate η, the algorithm is only slightly different:

repeat
  for $n = 1 \dots N$ **do**
    $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial E^n}{\partial \mathbf{w}}$
  end for
until maxiters