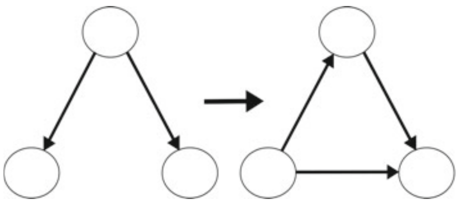
Chapter 8
Bayesian Networks: Learning



8.1 Introduction

Learning a Bayesian network includes two aspects: learning the structure and learning the parameters. When the structure is known, parameter learning consists in estimating the conditional probability tables (CPTs) from data. For structure learning there are two main types of methods: (i) global methods based on search and score, and (ii) local methods that use conditional independence tests. Next we describe both aspects, starting with parameter learning.

8.2 Parameter Learning

For this section we assume that the structure of the Bayesian network is known, and will focus on how to learn its parameters; later we will see how to learn the structure. We will consider that all the variables in the model are discrete; in the case of continuous variables, some discretization techniques will be described.

L. E. Sucar, *Probabilistic Graphical Models*, Advances in Computer Vision and Pattern Recognition, https://doi.org/10.1007/978-3-030-61943-5_8

If we have *sufficient* and complete data for all the variables, and we assume the topology of the BN is known, parameter learning is straight forward. The CPT for each variable can be estimated from the data based on the frequency of each value (or combination of values) obtaining a *maximum likelihood* (ML) estimator of the parameters. For example, to estimate the CPT of B given it has two parents, A, C:

$$P(B_i \mid A_j, C_k) \sim N_{i,j,k}/N_{j,k} \tag{8.1}$$

where $N_{i,j,k}$ is the number of cases in the database in which $B = B_i$, $A = A_j$ and $C = C_k$, and $N_{j,k}$ is the total number of cases in which $A = A_j$ and $C = C_k$.

8.2.1 Smoothing

When we estimate probabilities from data, it can sometimes happen that a particular event never occurs in the data set. This leads to the corresponding probability value being zero, implying an *impossible* case; if in the inference process this probability is considered, it will also make the result zero. This situation occurs, in many cases, because there is insufficient data to have a robust estimate of the parameters, and not because it is an impossible event.

The previous situation can be avoided by using some type of *smoothing* for the probabilities, eliminating zero probability values. There are several smoothing techniques, one of the most common and simplest is *Laplacian* smoothing.

Laplacian smoothing consists in initializing the probabilities to a uniform distribution, and then updating these values based on the data. Consider a discrete variable, X, with k possible values. Initially, each probability will be set to $P(x_i) = 1/k$. Then, consider a data set with N samples, in which the value x_i occurs m times; the estimate of its probability will be the following:

$$P(x_i) = (1 + m)/(k + N) \tag{8.2}$$

8.2.2 Parameter Uncertainty

If there is not sufficient data, a situation common in practice, we have uncertainty in the parameters. This uncertainty can be modeled using a second order probability distribution, and could be propagated in the inference process so we have an estimate of the uncertainty in the resulting probabilities. For binary variables, the uncertainty in the parameters can be modeled using a Beta distribution:

$$\beta(a, b) = \frac{(a + b + 1)!}{a!b!} x^a (1 - x)^b \tag{8.3}$$

For multivalued variables, uncertainty in the parameters can be represented by an extension of the Beta known as the Dirichlet distribution.

For the binary case, the expected value of the Beta distribution is given by: $P(b_i) = a + 1/a + b + 2$, where a and b are the parameters of the Beta distribution. This representation could also be used when we have experts' estimates of the probabilities. The parameters of the Beta distribution can represent a measure of *confidence* in the expert's estimates, expressed by varying the term $a + b$ with the same probability value. For instance:

- Complete ignorance: $a = b = 0$.
- Low confidence: $a + b$ *small* (10).
- Medium confidence: $a + b$ *intermediate* (100).
- High confidence: $a + b$ large (1000).

This representation could be used to combine experts' estimations with data. For example, to approximate the probability value of a binary variable, b_i we can use:

$$P(b_i) = k + a + 1/n + a + b + 2 \tag{8.4}$$

Where $a/a + b$ represents the expert's estimate, and k/n is the probability obtained from the data (k is the number of times b_i occurs in n samples).

For example, assume an expert gives an estimate of 0.7 for a certain parameter, and that the experimental data provides 40 positive cases among 100 samples. The parameter estimation for different confidences assigned to the expert will be the following:

Low confidence ($a + b = 10$): $P(b_i) = \frac{40+7+1}{100+10+2} = 0.43$
Medium confidence ($a + b = 100$): $P(b_i) = \frac{40+70+1}{100+100+2} = 0.55$
High confidence ($a + b = 1000$): $P(b_i) = \frac{40+700+1}{100+1000+2} = 0.67$

We observe that in the first case the estimate is dominated by the data, while in the third case the probability is closer to the expert's estimate; the second case provides a compromise.

8.2.3 Missing Data

Another common situation is to have incomplete data. There are two basic cases:

Missing values: In some registers there are missing values for one or more variables.
Hidden nodes: A variable or set of variables in the model for which there is no data at all.

For dealing with missing values, there are several alternatives:

1. Eliminate the registers with missing values.

2. Consider a special “unknown” value.
3. Substitute the missing value by the most common value (mode) of the variable.
4. Estimate the missing value based on the values of the other variables in the corresponding register.

The first and second alternatives are acceptable if there is sufficient data, otherwise we could be discarding useful information. The third alternative does not consider the other variables and as a result, it could bias the model. In general the best alternative is the fourth option. In this case, we first learn the parameters of the BN based on the complete registers, and then complete the data and re-estimate the parameters, applying the following process. For each register with missing values:

1. Instantiate all the known variables in the register.
2. Through probabilistic inference obtain the posterior probabilities of the missing variables.
3. Assign to each variable the value with highest posterior probability.
4. Add this completed register to the database and re-estimate the parameters.

An alternative to the previous process is that instead of assigning the value with the highest probability, we assign a *partial* case for each value of the variable proportional to the posterior probability.

For hidden nodes, the approach to estimate their parameters is based on the *Expectation–Maximization* (EM) technique.

8.2.3.1 Hidden Nodes: EM

The EM algorithm is a statistical technique used for parameter estimation when there are non-observable variables. It consists of two phases which are repeated iteratively:

E step: the missing data values are estimated based on the current parameters.
M step: the parameters are updated based on the estimated data.

The algorithm starts by initializing the missing parameters with random values.

Given a database with one or more hidden nodes, $H_1, H_2, \ldots, H_k$, the EM algorithm to estimate their CPTs is the following:

1. Obtain the CPTs for all the *complete* variables (the values of the variable and all its parents are in the database) based on a ML estimator.
2. Initialize the unknown parameters with random values.
3. Considering the actual parameters, estimate the values of the hidden nodes based on the known variables via probabilistic inference.
4. Use the estimated values for the hidden nodes to complete/update the database.
5. Re-estimate the parameters for the hidden nodes with the updated data.
6. Repeat 3–5 until converge (no significant changes in the parameters).

The EM algorithm optimizes the unknown parameters and gives a *local maximum* (the final estimates depend on the initialization).

Table 8.1 Data for the golf example with missing values for *Temperature* and a hidden variable, *Wind*

Outlook	Temperature 1	Humidity	Wind	Play
Sunny	xxx	High	–	N
Sunny	High	High	–	N
Overcast	High	High	–	P
Rainy	Medium	High	–	P
Rainy	Low	Normal	–	P
Rainy	Low	Normal	–	N
Overcast	Low	Normal	–	P
Sunny	Medium	High	–	N
Sunny	xxx	Normal	–	P
Rainy	Medium	Normal	–	P
Sunny	Medium	Normal	–	P
Overcast	Medium	High	–	P
Overcast	High	Normal	–	P
Rainy	Medium	High	–	N

8.2.3.2 Example

We now illustrate how to handle missing values and hidden variables using data from the *Golf* example (see Table 8.1). In this data set, there are some missing values for the variable *Temperature* (registers 1 and 9), and there is no information about *Wind*, which is a hidden node. We first illustrate how to fill-in the missing values for temperature and then how to manage the hidden node.

Assume that we learn a naive Bayes classifier (a NBC is a particular type of BN) based on the available data (12 complete registers without the wind variable), considering *Play* as the class variable and the other variables as attributes. Then, based on this model, we can estimate the probability of temperature for the registers in which it is missing, via probabilistic inference using the values of the other variables in the corresponding registers as evidence. Note that although the model is a NBC for *Play*, we can also apply probabilistic inference to estimate the value of an attribute given the class and the other attributes.That is:

Register 1: $P(Temperature \mid sunny, high, N)$
Register 9: $P(Temperature \mid sunny, normal, P)$

Then we can select as the value of temperature the one with highest posterior probability, and fill-in the missing values, as shown in Table 8.2.

For the case of the hidden node, *Wind*, we cannot obtain the corresponding CPT from the NBC, $P(Wind \mid Play)$, as there are no values for wind. However, we can apply the EM procedure, where we first pose initial random parameters for the CPT, which could be, for example, a uniform distribution:

Table 8.2 Data for the golf example after completing the missing values for *Temperature* and one iteration of the EM procedure to estimate the values of *Wind*

Outlook	Temperature 1	Humidity	Wind	Play
Sunny	Medium	High	No	N
Sunny	High	High	No	N
Overcast	High	High	No	P
Rainy	Medium	High	No	P
Rainy	Low	Normal	Yes	P
Rainy	Low	Normal	Yes	N
Overcast	Low	Normal	Yes	P
Sunny	Medium	High	No	N
Sunny	Medium	Normal	No	P
Rainy	Medium	Normal	No	P
Sunny	Medium	Normal	Yes	P
Overcast	Medium	High	Yes	P
Overcast	High	Normal	Yes	P
Rainy	Medium	High	Yes	N

$$P(Wind \mid Play) = \begin{matrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{matrix}$$

Given this CPT we have a complete, initial model for the NBC, and can estimate the probability of wind for each register based on the values of the other variables in the register. By selecting the highest probability value for each register, we can fill-in the table, as depicted in Table 8.2. Based on this new data table, we re-estimate the parameters, and obtain a new CPT:

$$P(Wind \mid Play) = \begin{matrix} 0.60 & 0.44 \\ 0.40 & 0.56 \end{matrix}$$

This completes one cycle of the EM algorithm; the process is then repeated until all parameters in the CPT have *almost* no change from the previous iteration. At this point, the EM procedure has converged, and we have an estimate of the missing parameters of the BN.

8.2.4 Discretization

Usually Bayesian networks consider discrete or nominal variables. Although there are some developments for continuous variables, these are restricted to certain distributions, in particular Gaussian variables and linear relations. An alternative to include continuous variables in BNs is to discretize them; that is, transform them to nominal variables. Discretization methods can be (i) unsupervised and (ii) supervised.

8.2.4.1 Unsupervised Discretization

Unsupervised techniques do not consider the task for which the model is going to be used (e.g., classification), such that the intervals for each variable are determined independently. The two main types of unsupervised discretization approaches are: equal width and equal data.

Equal width consists in dividing the range of a variable, $[Xmin; Xmax]$, in k equal bins; such that each bin has a size of $[Xmax - Xmin]/k$. The number of intervals, k, is usually set by the user.

Equal data divides the range of the variable in k intervals, such that each interval includes the same number of data points from the training data. In other words, if there are n data points, each interval will contain n/k data points; this means that the intervals will not necessarily have the same width.

A way to determine how many intervals, specially useful for Bayesian classifiers, is Proportional k-Interval Discretization (PKID) [14]. This strategy seeks a trade-off between the bias and variance of the parameter estimates by adjusting the number and size of intervals to the number of training instances. Given a continuous variable with N training instances, it is discretize to $\sqrt{N}$ intervals, with $\sqrt{N}$ instances in each interval. This method was compared experimentally with other discretization methods for implementing naive Bayesian classifiers; PKID achieves the lowest mean error [14].

8.2.4.2 Supervised Discretization

Supervised discretization considers the task to be performed with the model, such that the variables are discretized to optimize this task, for instance classification accuracy. If we consider a BN for classification, i.e. a Bayesian classifier, then the supervised approach can be directly applied. Assuming continuous attribute variables, these are discretized according to the class values. This can be posed as an optimization problem.

Consider the attribute variable X with range $[Xmin; Xmax]$ and a class variable C with m values $c_1, c_2, \ldots, c_m$. Given n training samples, so that each one has a value for C and X, the problem is to determine the *optimal* partition of X such that the classifier precision is maximized. This a combinatorial problem that is computationally complex, and can be solved using a search process as follows:

1. Generate all potential divisions in X which correspond to a value in $[Xmin; Xmax]$ where there is a change in the class value.
2. Based on the potential division points generate an initial set of n intervals.

3. Test the classification accuracy of the Bayesian classifier (usually on a different set of data known as a validation set) according to the current discretization.
4. Modify the discretization by partitioning an interval or joining two intervals.
5. Repeat (3) and (4) until the accuracy of the classifier cannot be improved or some other termination criteria occurs.

Different search approaches can be used, including basic ones such as *hill-climbing* or more sophisticated methods like *genetic algorithms*.

The previous algorithm does not apply for the general case of a Bayesian network which can be used to predict different variables based on different evidence variables. In this case, there is a supervised method [8] that discretizes continuous attributes while it learns the structure of the BN. The method is based on the *Minimum Description Length* (MDL) principle –described in Sect. 8.3.3. For each continuous variable, the number of intervals is determined according to its neighbors in the network. The objective is to minimize the MDL (a compromise between the precision and complexity of the model), using a search and test approach analogous to the process for Bayesian classifiers. This is repeated iteratively for all continuous variables in the network.

8.3 Structure Learning

Structure learning consists in obtaining the topology of the BN from the data. This is a complex problem because: (i) the number of possible structures is *huge* even with a few variables (it is super-exponential on the number of variables; for example, for 10 variables the number of possible DAGs is in the order of 4×10^{18}), and (ii) a very large database is required to obtain good estimates of the statistical measures on which all methods depend.

For the particular case of a tree structure, there is a method that guarantees the *best* tree. For the general case several methods have been proposed, which can be divided into two main classes:

1. Global methods: these [3, 5] perform a heuristic search over the space of network structures, starting from some initial structure, and generating a variation of the structure at each step. The *best* structure is selected based on a score that measures how well the model represents the data. Common scores are BIC [3] and MDL [5].
2. Local methods: these are based on evaluating the independence relations between subsets of variables given the data, to sequentially obtain the structure of the network. The most well known variant of this approach is the PC algorithm [12].

Both classes of methods obtain similar results with *enough* data. Local methods tend to be more sensitive when there are few data samples, and global methods tend to be more computationally complex.

Next we review the tree learning algorithm developed by Chow and Liu [2] and its extension to polytrees. Then we present the techniques for learning a general structure.

8.3.1 Tree Learning

Chow and Liu [2] developed a method for approximating any multi-variable probability distribution as a product of second order distributions, which is the basis for learning tree-structured BNs. The joint probability of n random variables can be approximated as:

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^{n} P(X_i \mid X_{j(i)}) \tag{8.5}$$

where $X_{j(i)}$ is the parent of X_i in the tree.

The problem consists in obtaining the *best* tree, that is, the tree structure that best approximates the real distribution. A measure of how close the approximation is based on the information difference between the real distribution (P) and the tree approximation (P^*) is as follows:

$$DI(P, P^*) = \sum_{X} P(\mathbf{X}) log(P(\mathbf{X})/P^*(\mathbf{X})) \tag{8.6}$$

Thus, now the problem consists in finding the tree that minimizes DI. However, evaluating this for all possible trees is very expensive. Chow and Liu proposed an alternative based on the mutual information between pairs of variables.

The mutual information between any pair of variables is defined as:

$$I(X_i, X_j) = \sum_{X_i, X_j} P(X_i, X_j) log(P(X_i, X_j)/P(X_i)P(X_j)) \tag{8.7}$$

Given a tree-structured BN with variables $X_1, X_2, \dots, X_n$, we define its *weight*, W, as the sum of the mutual information of the arcs (pairs of variable) that constitute the tree:

$$W(X_1, X_2, \dots, X_n) = \sum_{i=1}^{n-1} I(X_i, X_j) \tag{8.8}$$

where X_j is the parent of X_i in the tree (a tree with n nodes has $n - 1$ arcs).

It can be shown [2] that minimizing DI is equivalent to maximizing W. Therefore, obtaining the optimal tree is equivalent to finding the *maximum weight spanning tree*, using the following algorithm:

Table 8.3 Mutual information in descending order for the golf example

No.	Var 1	Var 2	Mutual Info.
1	Temperature	Outlook	0.2856
2	Play	Outlook	0.0743
3	Play	Humidity	0.0456
4	Play	Wind	0.0074
5	Humidity	Outlook	0.0060
6	Wind	Temperature	0.0052
7	Wind	Outlook	0.0017
8	Play	Temperature	0.0003
9	Humidity	Temperature	0
10	Wind	Humidity	0

1. Obtain the mutual information (I) between all pairs of variables (for n variables, there are $n(n-1)/2$ pairs).
2. Order the mutual information values in descending order.
3. Select the pair with maximum I and connect the two variables with an arc, this constitutes the initial tree
4. Add the pair with the next highest I to the tree, while they do not make a cycle; otherwise skip it and continue with the following pair.
5. Repeat 4 until all the variables are in the tree ($n-1$ arcs).

This algorithm obtains the *skeleton* of the tree; that is, it does not provide the direction of the arcs in the BN. The directions of the links have to be obtained using external semantics or using higher order dependency tests (see Sect. 8.3.2).

To illustrate the tree learning method consider the classic *golf* example with 5 variables: *play, outlook, humidity, temperature, wind*. Given some data, we obtain the mutual information shown in Table 8.3.

In this case, we select the first 4 pairs (arcs) and obtain the tree in Fig. 8.1, where the directions were assigned arbitrarily.

8.3.2 *Learning a Polytree*

Rebane and Pearl [11] developed a method that can be used to direct the arcs in the skeleton, and in general, learn a *polytree* BN. The algorithm is based on independence tests for variable triplets, and in this way it can distinguish *convergent* substructures; once one or more substructures of this type are detected in the skeleton, it can direct additional arcs by applying the independence tests to neighboring nodes. However, there is no guarantee of obtaining the direction for all the arcs in the tree. This same idea is used in the PC algorithm for learning general structures.

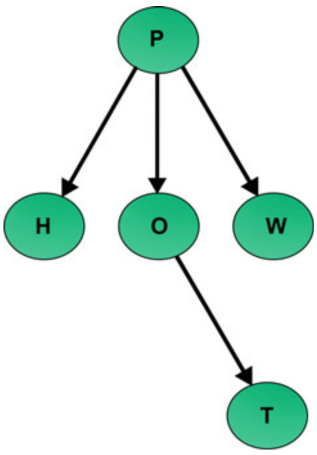


Fig. 8.1 Tree structure obtained for the golf example (P is play, O is outlook, H is humidity, T is temperature, and W is wind). Arc directions are set arbitrarily

The algorithm begins with the skeleton (undirected structure) obtained with the Chow and Liu algorithm. Subsequently, the direction of the arcs is learned using independence tests for variable triplets. Given three variables, there are three possibilities:

1. Sequential arcs: $X \rightarrow Y \rightarrow Z$.
2. Divergent arcs: $X \leftarrow Y \rightarrow Z$.
3. Convergent arcs: $X \rightarrow Y \leftarrow Z$.

The first two cases are indistinguishable under statistical independence testing; that is, they are equivalent. In both cases, X and Z are independent given Y. However the third case is different, since X and Z are NOT independent given Y. Consequently, this case can be used to determine the directions of the two arcs that connect these three variables; additionally, we can apply this knowledge to learn the directions of other arcs using independence tests. With this in mind, the following algorithm can be used for learning polytrees:

1. Obtain the skeleton using the Chow and Liu algorithm.
2. Iterate over the network until a convergent variable triplet is found. We will call the variable to which the arcs converge a *multi-parent node*.
3. Starting with a multi-parent node, determine the directions of other arcs using independence tests for variable triplets. Continue this procedure until it is no longer possible (causal base).
4. Repeat 2–3 until no other directions can be determined.
5. If any arcs are left undirected, use the external semantics to infer their directions.

To illustrate this algorithm, let us consider the golf example again, with the obtained skeleton (undirected structure). Suppose that the variable triplet H, P, W falls in the convergent case. Then, the arcs will be directed such that H points to P and W points to P. Subsequently, the dependence between H and W is measured with respect to O given P. If H and W are independent from O given P then there

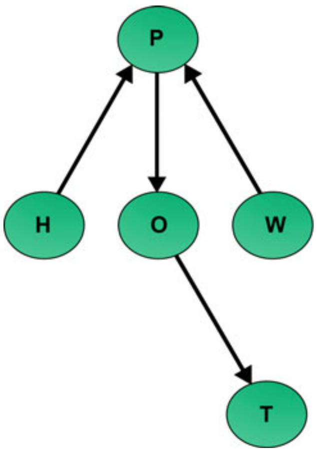


Fig. 8.2 A polytree obtained for the golf example using the Rebane and Pearl algorithm

will be an arc that points from P to O. Finally, the dependence relation between P and T given O is tested, and if they are again found to be independent, then the arc points from O to T. Figure 8.2 shows the resulting structure.

8.3.3 Search and Score Techniques

The previous methods are restricted to tree and polytree structures; in this and the following section we will cover general structure learning techniques, starting with global approaches.

Global methods search for the *best* structure based on a global metric. That is, different structures are generated and these are evaluated with respect to the data using some scoring method. There are different variants of these methods, all of which basically depend on two aspects: (i) a fitness measure between the structure and the data, and (ii) a method for searching for the best structure.

8.3.3.1 Scoring Functions

There are several possible fitness measures or scoring functions. Two desirable properties for scoring functions are [1]:

Decomposability: a scoring function is decomposable if the value assigned to each structure can be expressed as a sum (in the logarithmic space) of local values that depend only on each node and its parents. This is important for efficiency reasons during the search process; given this property when a local change is made to the structure, only a part of the score has to be re-evaluated.

Score Equivalence: a scoring function S is score equivalent if it assigns the same value to all DAGs that are represented by the same essential graph. In this way, the result of evaluating an equivalence class will be the same regardless of the DAG that is selected from this class. The structures of two BNs correspond to the same essential graph if they are equivalent in terms of the independence relations they represent. For instance, the following structures: $X \rightarrow Y \rightarrow Z$ and $X \leftarrow Y \rightarrow Z$ correspond to the same essential graph ($I(X, Y, Z)$).

Next we describe some common scoring functions, including: the maximum likelihood (ML), the Bayesian information criterion (BIC), the Bayesian score (BD), and the minimum description length (MDL) criterion.

The maximum likelihood score selects the structure that maximizes the probability of the data, D, given the structure, G:

$$G^* = ArgMax_G[P(D \mid \Theta_G, G_i)] \tag{8.9}$$

where G_i is the candidate structure and Θ_G the corresponding vector of parameters (probability of each variable given its parents according to the structure).

The direct application of the ML score might result in a highly complex network, which usually implies overfitting the data (poor generalization) and also makes inference more complex. Therefore, a way to penalize complex models is required.

A commonly used scoring function that includes a penalty term is the Bayesian Information Criterion or BIC defined as:

$$BIC = logP(D \mid \Theta_G, G_i) - \frac{d}{2}logN \tag{8.10}$$

where d is the number of parameters in the BN and N the number of cases in the data. An advantage of this metric is that it does not require a prior probability specification and it is related to the MDL measure, compromising between the precision and complexity of the model. However, given the high penalty on the complexity of the model, it tends to choose structures that are too simple.

Bayesian Scores

An alternative metric is obtained by following a Bayesian approach, obtaining the posterior probability of the structure given the data with the Bayes rule:

$$P(G_i \mid D) = P(G_i)P(D \mid G_i)/P(D) \tag{8.11}$$

Given that P(D) is a constant that does not depend on the structure, it can be discarded from the metric to obtain de Bayesian or BD score:

$$BD = P(G_i)P(D \mid G_i) \tag{8.12}$$

$P(G_i)$ is the prior probability of the model. This can be specified by an expert or defined such that simpler structures are preferred; or just set to a uniform distribution.

The BDe score is a variation of the BD score which makes the following assumptions: (i) the parameters are independent and have a prior Dirichlet distribution, (ii) equivalent structures have the same score, (iii) the data samples are independent and identically distributed (iid). Under these assumptions the *virtual counts* required to compute the score can be estimated as:

$$N_{ijk} = P(X_i = k, Pa(X_i) = j \mid G_i, \Theta_G) \times N' \tag{8.13}$$

This is the estimated count of a certain *configuration*: $X_i = k$ given $Pa(X_i) = j$; N' is the equivalent sample size.

By assuming that the hyper parameters of the priors are one, we can further simplify the calculation of the Bayesian score, and obtain what is known as the K2 metric.[1] This score is decomposable and it is calculated for each variable X_i given its parents $Pa(X_i)$:

$$S_i = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} \alpha_{ijk}! \tag{8.14}$$

Where r_i is the number of values of X_i, q_i is the number of possible configurations for the parents of X_i, α_{ijk} is the number of cases in the database where $X_i = k$ and $Pa(X_i) = j$, and N_{ij} is the number of cases in the database where $Pa(X_i) = j$.

This metric provides a practical alternative for evaluating a BN. Another common alternative that is based on the MDL principle is described next.

MDL

The MDL measure makes a compromise between accuracy and model complexity. Accuracy is estimated by measuring the mutual information between each variable and its parents (an extension of the tree learning algorithm). Model complexity is evaluated by counting the number of parameters. A constant, α within [0, 1], is used to balance the weight of each aspect, that is, accuracy against complexity. The fitness measure is given by the following equation:

$$MC = \alpha(W/Wmax) + (1 - \alpha)(1 - L/Lmax) \tag{8.15}$$

where W represents the accuracy of the model, and L the complexity. $Wmax$ and $Lmax$ represent the maximum accuracy and complexity, respectively. To determine the maximums, usually an upper bound is set on the number of parents each node is allowed to have. A value of $\alpha = 0.5$ gives equal importance to the model complexity and accuracy, while a value near 0 gives more importance to the complexity, and a value near 1 more importance to accuracy.

Complexity is given by the number of parameters required for representing the model, which can be measured with the following equation:

$$L = S[k log_2 n + d(S - 1)F] \tag{8.16}$$

[1] K2 is an algorithm for learning BNs described below.

where n is the number of variables in the BN, k is the average number of parents per variable, S is the average number of values per variable, F is the average number of values per parent variable, and d the number of bits per parameter. For example, consider a BN with 16 variables, all variables are binary and have in average 3 parents, and each parameter is represented by 16 bits. Then:

$$L = 2 \times [3 \times log_2(16) + 16 \times (2 - 1) \times 2] = 2 \times [12 + 32] = 88$$

The accuracy can be estimated based on the 'weight' of each node; this is analogous to the weights in the methodology for learning trees. In this case, the weight of each node, X_i, is estimated based on its mutual information with its parents, $Pa(X_i)$:

$$w(X_i, Pa(X_i)) = \sum_{xi} P(X_i, Pa(X_i)) log[P(X_i, Pa(X_i))/P(X_i)P(Pa(X_i))] \tag{8.17}$$

and the weight (accuracy) total is given by the sum of the weights for each node:

$$W = \sum_i w(X_i, Pa(X_i)) \tag{8.18}$$

8.3.3.2 Search Algorithms

Once a fitness measure for the structure has been established, we need to establish a method for choosing the 'best' structure among the possible options. Since the number of possible structures is exponential on the number of variables, it is impossible to evaluate every structure. To limit the number of structures that are evaluated, a heuristic search is carried out. Several different search methods can be applied. One common strategy is to use a *hill climbing* approach, where we begin with a simple tree structure that is improved until we obtain the 'best' structure. A basic greedy algorithm to search for the best structure is the following:

1. Generate an initial structure—tree
2. Calculate the fitness measure of the initial structure.
3. Add/invert an arc from the current structure.
4. Calculate the fitness measure of the new structure.
5. If the fitness improves, keep the change; if not, return to the previous structure.
6. Repeat 3–5 until no further improvements exist.

The previous algorithm is not guaranteed to find the optimum structure, since it is possible to reach only a local maximum. Figure 8.3 illustrates the search procedure for the golf example, starting with a tree structure that is improved until the final structure is obtained. Other search methods, such as genetic algorithms, simulated annealing, bidirectional searches, etc., can also be applied to obtain the best structure.

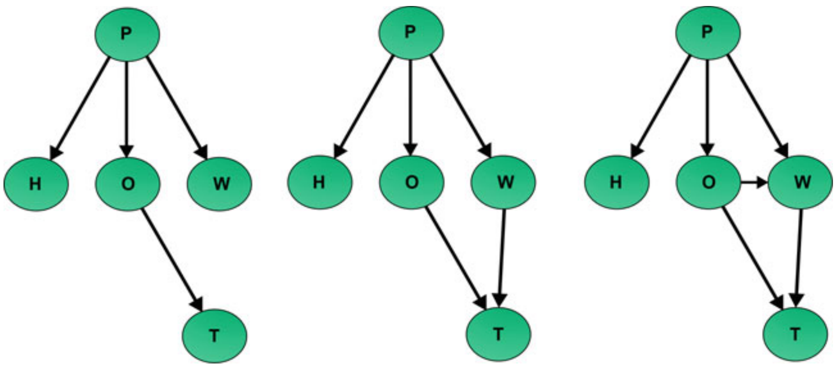


Fig. 8.3 A few steps in the procedure for learning the structure of the "golf" example, beginning with a tree structure (left) until the final structure (right) is obtained

An alternative to reduce the number of potential structures to be evaluated, is to set an ordering on the variables, known as a *causal ordering*. Given this ordering, the arcs in the network are restricted to follow this order; that is, there could be NO arc from V_j to V_i if $j > i$ according to the ordering. The K2 algorithm [3] takes advantage of this, providing and efficient and popular method for learning BNs.

8.3.3.3 The K2 Algorithm

Given a causal ordering for all the variables, learning the best structure is equivalent to selecting the best set of parents for each node independently. Initially, each variable has no parents. Then, the K2 algorithm incrementally adds parents to each node, as long as it increases the global score. When adding parents to any node does not increase the score, the search stops. Also, given a causal ordering it guarantees that there are not cycles in the graph.

Algorithm 8.1 provides a summary of the K2 procedure. The inputs to the algorithm are the set of n variables with a causal ordering, $X_1, X_2, \ldots, X_n$, a data base D containing m cases, and, usually, a restriction on the maximum number of parents for each variable, u. The output is the set of parents, $Pa(X_i)$, for each variable, which defines the structure of the network. Starting from the first variable according to the ordering, the algorithm tests all possible parents of a variable that have not been added, and includes the one that makes the maximum increment in the score of the network. This is repeated until there is no additional parent that increases the score; for every node on the network.

Algorithm 8.1 The K2 Algorithm

```
Require: Set of variables $X$ with a causal ordering, scoring function $S$, and maximum parents $u$
Ensure: Set of parents for each variable, $Pa(X_i)$
  for $i = 1$ to $n$ do
    $oldScore = S(i, Pa(X_i))$
    $incrementScore = true$
    $Pa(X_i) = \emptyset$
    while $incrementScore$ and $|Pa(X_i)| < u$ do
      let $Z$ be the node in $Predecessors(X_i) - Pa(X_i)$ that maximizes $S$
      $newScore = S(i, Pa(X_i) \cup Z)$
      if $newScore > oldScore$ then
        $oldScore = newScore$
        $Pa(X_i) = Pa(X_i) \cup Z$
      else
        $incrementScore = false$
      end if
    end while
  end for
  return $Pa(X_1), Pa(X_2)...Pa(X_n)$
```

8.3.4 Independence Tests Techniques

The other class of structure learning techniques use a *local* approach instead of the *global* one used by the score and search techniques. The basic idea is to apply independence tests to sets of variables to recover the structure of the BN. An example of this type of techniques is the Chow and Liu algorithm for trees. Next we present a method for learning general structures, the PC algorithm.

8.3.4.1 The PC Algorithm

The PC algorithm [12] first recovers the skeleton (underlying undirected graph) of the BN, and then it determines the orientation of the edges.

To determine the skeleton, it starts from a fully connected undirected graph, and determines the conditional independence of each pair of variables given some subset of the other variables. For this it assumes that there is a procedure that can determine if two variables, X, Y, are independent given a subset of variables, $\mathbf{S}$, that is, $I(X, Y \mid \mathbf{S})$. An alternative for this procedure is the conditional cross entropy measure. If this measure is below a threshold value set according to a certain confidence level, the edge between the pair of variables is eliminated. These tests are iterated for all pairs of variables in the graph.

In the second phase the direction of the edges are set based on conditional independence tests between variable triplets. It proceeds by looking for substructures in the graph of the form $X - Z - Y$ such that there is no edge $X - Y$. If X, Y are not independent given Z, it orients the edges creating a V-structure $X \rightarrow Z \leftarrow Y$. Once all the V-structures are found, it attempts to orient the other edges based on independence tests and avoiding cycles. Algorithm 8.2 summarizes the basic procedure.[2]

Algorithm 8.2 The PC algorithm

```
Require: Set of variables $\mathbf{X}$, Independence test $I$
Ensure: Directed Acyclic Graph $G$
 1: Initialize a complete undirected graph $G'$
 2: i = 0
 3: repeat
 4:   for $X \in \mathbf{X}$ do
 5:     for $Y \in ADJ(X)$ do
 6:       for $S \subseteq ADJ(X) - \{Y\}, \mid S \mid = i$ do
 7:         if $I(X, Y \mid S)$ then
 8:           Remove the edge $X - Y$ from $G'$
 9:         end if
10:       end for
11:     end for
12:   end for
13:   i = i + 1
14: until $\mid ADJ(X) \mid \leq i, \forall X$
15: Orient edges in $G'$
16: Return $G$
```

If the set of independencies are faithful to a graph[3] and the independence tests are perfect, the algorithm produces a graph equivalent to the original one; that is, the BN structure that generated the data.

The independence test techniques rely on having *enough* data for obtaining good estimates from the independence tests. Search and score algorithms are more robust with respect to the size of the data set, however their performance is also affected by the size and quality of the available data. An alternative for when there is not *sufficient* data, is to combine expert knowledge and data.

8.4 Combining Expert Knowledge and Data

When domain expertise is available, this can be combined with learning algorithms to improve the model. In the case of parameter learning, we can combine data and expert estimates based on the Beta or Dirichlet distributions as described in Sect. 8.2.

For structure learning, there are two basic approaches to combine expert knowledge and data:

[2] $ADJ(X)$ is the set of nodes adjacent to X in the graph.

[3] The *Faithfulness Condition* can be thought of as the assumption that conditional independence relations are due to the causal structure rather than to accidents in parameter values [12].

- Use expert knowledge as *restrictions* to reduce the search space for the learning algorithm.
- Start from a structure proposed by an expert and use data to validate and improve this structure.

There are several ways to use expert knowledge to aid the structure learning algorithm, such as:

1. Define an ordering for the variables (causal order), such that there could be an arc from X_i to X_j only if X_j is after X_i according to the specified ordering.
2. Define restrictions in terms of directed arcs that must exist between two variables, i.e. $X_i \rightarrow X_j$.
3. Define restrictions in terms of an arc between two variables that could be directed either way.
4. Define restrictions in terms of pairs of variables that are not directly related, that is, there must be no arc between X_i and X_j.
5. Combinations of the previous restrictions.

Several variants of both types of techniques, search and score and independence tests, incorporate the previous restrictions.

In the case of the second approach, an example was presented in Chap. 4, with the structural improvement algorithm. This technique starts from a naive Bayes structure which is improved by eliminating, joining or inserting variables. This idea can be extended to general BN structures, in particular for tree-structured BNs.

8.5 Transfer Learning

Another alternative when there is not enough data for certain application is to *transfer* knowledge and/or data for related domains, what is known as *transfer learning*. Next we describe an approach based on transfer learning for Bayesian networks.

Luis et al. [7] proposed a transfer learning method that learns a BN, structure and parameters, for a target task, from data from this task and from other related auxiliary tasks. The structure learning method is based on the PC algorithm, combining the dependency measures obtained from data in the target task, with those obtained from data in the auxiliary tasks. The combination function takes into account the consistency between these measures.

For structure learning, the method modifies the independence tests of the PC algorithm by combining the independence measures in the target task with that of the *closest* auxiliary task. The similarity measure considers both, global and local similarity. The global similarity considers all pair-wise conditional independencies in the models, while the local similarity includes only those for specific variables. So the combined independence measure $I(X, \mathbf{S}, Y)$ is a linear weighted combination of the independence measures in the target and auxiliary tasks, considering the confidence and similarity measures. An alternative PC algorithm is defined, based on the combined independence measures.

The parameter learning technique uses an aggregation process, combining the parameters estimated from the target task, with those estimated from the auxiliary tasks. Based on linear combination techniques, they proposed two variants: (i) Distance-based linear pool, which takes into account the distance of the auxiliary parameters to the target parameters, and (ii) Local linear pool, which only includes auxiliary parameters that are close to the target one, weighted by the amount of data in each auxiliary task.

To combine the CPT for a variable X from an auxiliary task, it is necessary that the variable has the same parents in the target task. If they do not have the same parents, the substructure in the auxiliary task is transformed to match the target one.

The Distance-based linear pool (DBPL) makes a weighted linear combination of the parameters estimated from data of the target task, with an average of the parameters of the auxiliary tasks:

$$P^{'}_{target} = C_i \times P_{target} + (1 - C_i) \times P_{auxiliary} \tag{8.19}$$

where C_i is a factor defined according to the similarity between the parameters estimated for that target task and those of the auxiliary tasks.

The Local linear pool (LoLP) takes into account only the most similar parameters from the auxiliary tasks, and weights them according to their confidence based on the amount of data. It is based also on Eq. 8.19, but only considering the parameters from the closest auxiliary tasks, those for which the difference with the parameters of the target task is below certain threshold.

For more details on the method see [7].

8.6 Applications

There are many domains in which learning Bayesian networks has been applied to get a better understanding of the domain or make predictions based on partial observations; for example medicine, finance, industry and the environment, among others. Next we present two application examples: modeling the air pollution in Mexico City and agricultural planning for coffee production.

8.6.1 Air Pollution Model for Mexico City

Air quality in Mexico City is a major problem. There, air pollution is one of the highest in the world, with a high average of daily emissions of several primary pollutants, such as hydrocarbons, nitrogen oxides, carbon monoxide and others. The pollution is due primarily to transportation and industrial emissions. When the primary pollutants are exposed to sunshine, they undergo chemical reactions and yield a variety of secondary pollutants, ozone being the most important. Besides the health problems it may cause, ozone is considered an indicator of the air quality in urban areas.

The air quality is monitored in 25 stations within Mexico City, with five of these being the most complete. Nine variables are measured in each of the 5 main stations, including: wind direction and velocity, temperature, relative humidity, sulfur dioxide, carbon monoxide, nitrogen dioxide and ozone. These are measured every minute 24 hours a day, and are averaged every hour.

It is important to be able to forecast the pollution level several hours, or even a day in advance for several reasons, including:

1. To be able to take emergency measures if the pollution level is going to be above a certain threshold.
2. Helping industry make contingency plans in advance in order to minimize the cost of the emergency measures.
3. To estimate the pollution in an area where there are no measurements.
4. To take preventive actions in some places, such as schools, in order to reduce the health hazards produced by high pollution levels.

In Mexico City, the ozone level is used as a global indicator for the air quality within the different parts of the city. The concentrations of ozone are given in IMECA (Mexican air quality index). It is important to predict the daily ozone, or at least, predict it several hours in advance using the other variables measured at different stations.

It is useful to know the dependencies between the different variables that are measured, and specially their influence in the ozone concentration. This will provide a better understanding of the problem with several potential benefits:

- Determine which factors are more important for the ozone concentration in Mexico City.
- Simplify the estimation problem, by taking into account only the relevant information.
- Discover the most critical primary causes of pollution in Mexico City; these could help in future plans to reduce pollution.

We started by applying a learning algorithm to obtain an initial structure of the phenomena [13]. For this we considered 47 variables: 9 measurements for each of the 5 stations, plus the hour and month in which they were recorded. We used nearly 400 random samples, and applied the Chow and Liu algorithm to obtain the tree structure that best approximates the data distribution. This tree-structured Bayesian network is shown in Fig. 8.4.

We then considered the ozone in one station (Pedregal) as unknown, and we estimate it, one hour in advance, using the other measurements. Thus we make *ozone-Pedregal* the hypothesis variable and consider it as the root in the probabilistic tree, as shown in Fig. 8.4. From this initial structure we can get an idea of the relevance or

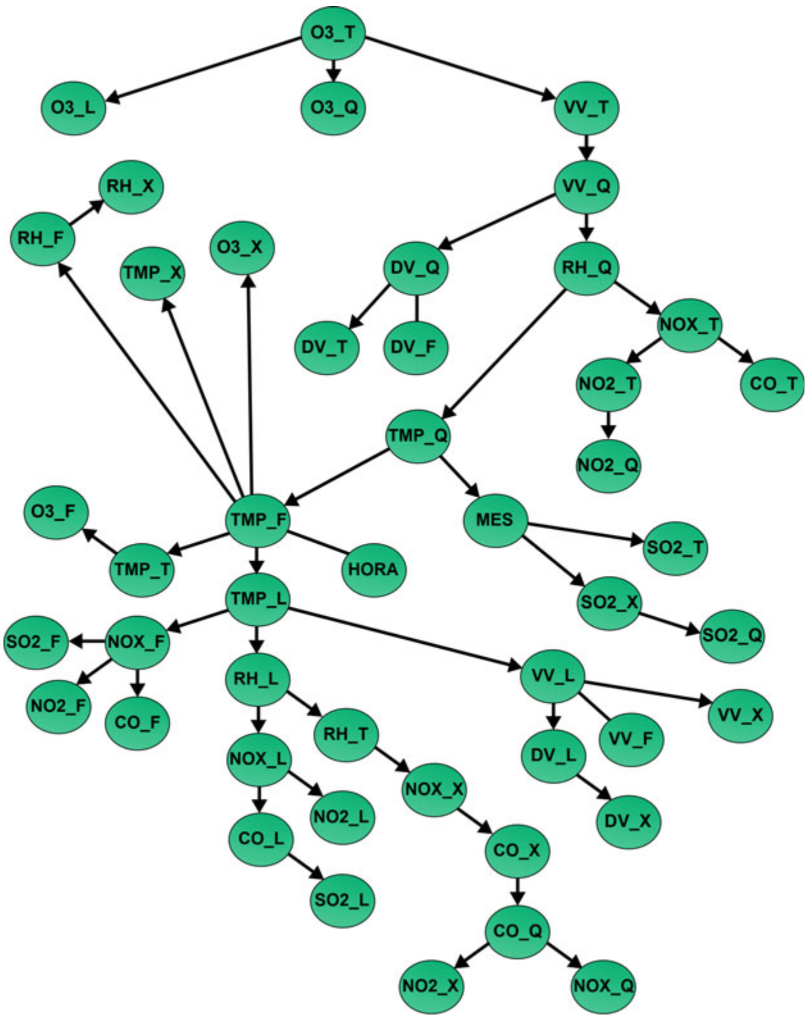


Fig. 8.4 A Bayesian tree that represents the ozone phenomena in 5 stations in Mexico City. The nodes represent the 47 variables according to the following nomenclature. For the measured variables, each name is formed by two parts, "measurement-station", using the following abbreviations: the measurements, O3-ozone, SO2-sulfur dioxide, CO-carbon monoxide, NO2-nitrogen dioxide, NOX-nitrogen oxides, VV-wind velocity, DV-wind direction, TMP-temperature, RH-relative humidity; the monitoring stations, T-Pedregal, F-Tlalnepantla, Q-Merced, L-Xalostoc, X-Cerro de la Estrella. The other two variables correspond to the time when the measurements were taken, and they are: HORA-hour, MES-month

influence of the other variables for estimating *ozone-Pedregal*. The nodes "closest" to the root are the most important ones, and the "far-away" nodes are less important.

In this case we observe that there are 3 variables (ozone-Merced, ozone-Xalostoc, and wind velocity in Pedregal) that have the greatest influence in *ozone-Pedregal*. Furthermore, if the tree structure is a good approximation to the "real" structure, these 3 nodes make *ozone-Pedregal* independent from the rest of the variables (see Fig. 8.4). Thus, as a first test of this structure, we estimated *ozone-Pedregal* using only these 3 variables. We carried out two experiments: (1) estimate *ozone-Pedregal* using 100 random samples taken from the training data, and (2) estimate *ozone-Pedregal* with another 100 samples taken from separate data, not used for training. We observe that even with only three parameters, the estimations are quite good. For training data the average error (absolute difference between the real and the estimated ozone concentration) is 11 IMECA or 12%, and for non-training data it is 26 IMECA or 22%.

An interesting observation from the obtained structure is that *ozone-Pedregal* (located in the south of the city) is related basically to three variables, *ozone-Merced* and *ozone-Xalostoc* (located in the center and north of the city), and the wind velocity in Pedregal. Otherwise stated, pollution in the south depends basically on the pollution in the center and north of the city (where there is more transit and industry) and the wind velocity –which carries the pollution from the north to the south. This phenomenon was already known, but it was discovered automatically by learning a BN. Other, not so well known relations, can also be discovered and could be useful for making decisions to control pollution and take emergency measures.

8.6.2 Agricultural Planning Using Bayesian Networks

Climate data plays a key role in planning in the agricultural sector. However, data at the required spatial or temporal resolution is often lacking, or certain values are missing. An alternative is to learn a Bayesian network from the available data, and then use it to generate data for missing variables. This work is in the domain of coffee production in Central America and southern Mexico [6]. The variables considered are: precipitation, maximum and minimum air temperature, wind speed, and solar radiation; which are used to infer missing values for monthly relative humidity and relative humidity for the driest month.

The data was obtained from the surface reanalysis dataset, Climate Forecast System Reanalysis (CFSR), which includes daily values for the variables precipitation (mm), air temperature (°C, minimum and maximum at 2 m), wind speed (m/s, at 10 m), surface solar radiation (MJ/m^2) and relative humidity (%, at 2 m). The spatial resolution is 38 km $\times$ 38 km per pixel and data are available from 1979 to 2014

The BN model was built using the software package Netica (Version 6.04, Norsys Software Corp., Vancouver, BC, Canada). For each selected variable, nodes were created and discretized. Continuous variables were discretized in intervals of equal

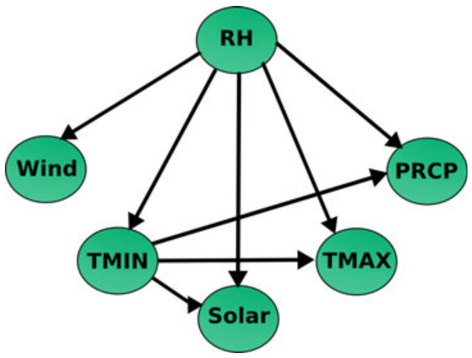


Fig. 8.5 Bayesian network model to infer monthly relative humidity. Notation: RH: relative humidity, TMAX: maximum temperature, TMIN: minimum temperature, PRCP: total precipitation, Wind: wind speed, Solar: surface solar radiation

length; the number of intervals was selected for each variable according to the data distribution.

Once the variables were discretized, the graphical model was learned from 80% of the cases of the dataset (n = 180,530). The relative humidity node was set as the target variable, and the BN topology was restricted to a Tree Augmented Naive Bayes (TAN) classifier, considering *relative humidity* as the class. The resulting BN structure is depicted in Fig. 8.5. The parameters of this model were learned from the same dataset using Laplacian smoothing.

The model was validated in two ways. Firstly, they explored the capability of the model to infer relative humidity for all the months in a year, and the relative humidity of the driest month, based on all other variables. Secondly, they estimated relative humidity with the variables Solar and Wind missing, because these are hardly registered in the study region's weather stations. The rest of the data not used for training (45,190 cases) was used to evaluate the model under the two previous scenarios. In general, when comparing inferred values to the actual values, the metrics bias (less than the unit) and RMSE $\leq 4.1\%$ indicate a very close agreement. As expected, the best model performance was obtained when information on all other variables was available; however, even with missing variables (Solar and Wind), the results were very good (RMSE $\leq 5\%$).

8.7 Additional Reading

A general book on learning Bayesian networks is [9]; Heckerman [4] has comprehensive tutorial on learning BNs. The tree and polytree learning algorithms are described in [10]. A general introduction to learning BNs from a statistical perspective is given in [12]. An analysis of different scoring functions is presented in [1].

8.8 Exercises

1. The table below gives the original data for the golf example using numeric values for some of the variables. Discretize these variables using 3 intervals for each variable: (a) use equal width discretization, (b) use equal data.

Outlook	Temperature	Humidity	Wind	Play
sunny	19	high	5	N
sunny	25	high	3	N
overcast	26	high	3	P
rainy	17	high	6	P
rainy	11	normal	15	P
rainy	7	normal	17	N
overcast	8	normal	11	P
sunny	20	high	7	N
sunny	19	normal	1	P
rainy	22	normal	5	P
sunny	21	normal	20	P
overcast	22	high	18	P
overcast	28	normal	16	P
rainy	18	high	3	N

2. Using the discretized data from the previous problem, obtain the CPTs for the Bayesian network structure given in Fig. 8.2.
3. Discretize the continuous variables in the table of Problem 1 to maximize the accuracy of the prediction of *Play*, considering a naive Bayes classifier. Build an initial classifier with the discrete attributes, then apply the supervised discretization to *Temperature*, and then to *Wind*.
4. Obtain the CPTs for a naive Bayes classifier based on the data (only the complete registers) in Table 8.1, Sect. 8.2.3.2.
5. Estimate the missing values of *Temperature* for the example in Sect. 8.2.3.2. Re-estimate the CPTs including these values.
6. Continue the EM algorithm for the example in Sect. 8.2.3.2 until convergence. Show the final CPT, $P(Wind \mid Play)$, and the final data table.
7. Based on the data for the golf example in Table 8.2, learn the skeleton of a tree BN using Chow and Liu's algorithm.
8. Obtain the directions for the arcs of the skeleton from the previous problem by applying the polytree learning technique.
9. Based on the same dataset from the previous problem, Table 8.2, learn a BN using the PC algorithm. Use the conditional mutual information measure (seen in Chap. 2) for testing conditional independence.
10. Given the dataset of the golf example, Table 8.2, learn a BN using the K2 algorithm. Define a causal ordering for the variables, and use as an initial structure the tree obtained with Chow and Liu's algorithm. Compare the structure obtained with that of the previous problem.

11. Given the dataset in the following table, (a) learn a naive Bayesian classifier considering C as the class, (b) learn a tree-structured BN and fix the directions of the arcs considering C as the root, (c) compare the structures of both models.

A1	A2	A3	C
0	0	0	0
0	1	1	1
0	1	0	1
0	0	1	1
0	0	0	0
0	1	1	0
1	1	0	1
0	0	0	1
0	1	0	0
0	1	1	0

12. For the dataset in the previous problem, use the Laplacian smoothing technique to obtain the conditional probability tables for both, the NBC and the tree BN. Compare the tables to the ones obtained without smoothing.
13. *** Develop a program that implements the polytree learning algorithm. Apply it to the golf data and compare with the results from Exercise 2.
14. *** Implement a program for learning a BN from data using a score and search technique based on the MDL scoring function, and another based on independence tests (PC algorithm). Apply both to different datasets and compare the results.
15. *** Develop a program for implementing the transfer learning approach for BNs. To evaluate it, select a BN as target task, and build several auxiliary tasks by modifying the original structure, adding, eliminating or inverting arcs. Generate *few* data samples for the target task (using logic sampling) and more data for the auxiliary tasks.

References

1. De Campos, L.M.: A scoring function for learning Bayesian networks based on mutual information and conditional independence tests. J. Mach. Learn. Res. **7**, 2149–2187 (2006)
2. Chow, C.K., Liu, C.N.: Approximating discrete probability distributions with dependence trees. IEEE Trans. Inf. Theory **14**, 462–467 (1968)
3. Cooper, G.F., Herskovitz, E.: A Bayesian method for the induction of probabilistic networks from data. Mach. Learn. **9**(4), 309–348 (1992)
4. Heckerman, D.: A tutorial on learning with Bayesian networks. Innovations in Bayesian Networks, pp. 33–82. Springer, Netherlands (2008)
5. Lam, W., Bacchus, F.: Learning Bayesian belief networks: an approach based on the MDL principle. Comput. Intell. **10**, 269–293 (1994)
6. Lara-Estrada, L., Rasche, L., Sucar, L.E., Schneider, U.A.: Inferring missing climate data for agricultural planning using Bayesian networks. Land (2018)
7. Luis, R., Sucar, L.E., Morales, E.F.: Inductive transfer for learning Bayesian networks. Mach. Learn. **79**, 227–255 (2010)

8. Martínez, M., Sucar, L.E.: Learning an optimal Naive Bayes classifier. In: 18th International Conference on Pattern Recognition (ICPR), vol. 3, pp. 1236–1239 (2006)
9. Neapolitan, R.E.: Learning Bayesian Networks. Prentice Hall, New Jersey (2004)
10. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Francisco (1988)
11. Rebane, G., Pearl, J.: The recovery of causal poly-trees from statistical data. In: Kanal, L.N., Levitt, T.S., Lemmer, J.F. (eds.) Uncertainty in Artificial Intelligence, pp. 175–182 (1987)
12. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search. Springer, Berlin (1993)
13. Sucar, L.E., Ruiz-Suarez, J.C.: Forecasting air pollution with causal probabilistic networks. In: Barnett, V., Turkman, K.F. (eds.) Statistics for the Environment 3: Statistical Aspects of Pollution, pp. 185–197. Wiley, Chichester (2007)
14. Yang, Y., Webb, G.I.: Proportional k-interval discretization for Naive-Bayes classifiers. In: 12th European Conference on Machine Learning (ECML), pp. 564–575 (2001)