# Chapter 5
# The BNET Procedure

**Contents**

# Overview: BNET Procedure

The BNET procedure learns a Bayesian network from an input data table in SAS Viya. A Bayesian network is a directed acyclic graphical model in which nodes represent random variables and the links between nodes represent conditional dependency of the random variables. Because the Bayesian network provides conditional independence structure and a conditional probability table at each node, the model has been used successfully as a predictive model in supervised data mining. For more information about Bayesian networks, see Pearl (1988).

The BNET procedure can learn different types of Bayesian network structures, including naive, tree-augmented naive (TAN), Bayesian network-augmented naive (BAN), parent-child Bayesian network, and Markov blanket. PROC BNET performs efficient variable selection through independence tests, and it selects the best model automatically from the specified parameters. It also generates SAS DATA step code or an analytic store to score data. It can load data from multiple nodes and perform computation in parallel.

## PROC BNET Features

The BNET procedure has the following features:

- structure learning through efficient local learning algorithms
- efficient variable selection through independence tests
- automatic selection of the best parameters by using a validation data subset
- learning of different types of Bayesian network structures
- handling of both nominal and interval input variables
- binning of the interval input variables
- handling of missing values
- multithreading during the training and scoring phases

## Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named mysess, and the LIBNAME statement creates the mycas CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section "Introduction to Shared Concepts" on page 11 in Chapter 3, "Shared Concepts."

## Getting Started: BNET Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections "Using CAS Sessions and CAS Engine Librefs" on page 11 and "Loading a SAS Data Set onto a CAS Server" on page 12 in Chapter 3, "Shared Concepts."

Consider a study of the analgesic effects of treatments on elderly patients who have neuralgia. Two test treatments and a placebo are compared. The response variable is whether the patient reported pain or not. Researchers recorded the age and gender of 60 patients and the duration of complaint before the treatment began. The following DATA step creates the data set Neuralgia:

```
Data Neuralgia;
   input Treatment $ Sex $ Age Duration Pain $ @@;
   datalines;
P  F  68   1  No   B  M  74  16  No  P  F  67  30  No
P  M  66  26  Yes  B  F  67  28  No  B  F  77  16  No
A  F  71  12  No   B  F  72  50  No  B  F  76   9  Yes
A  M  71  17  Yes  A  F  63  27  No  A  F  69  18  Yes
B  F  66  12  No   A  M  62  42  No  P  F  64   1  Yes
A  F  64  17  No   P  M  74   4  No  A  F  72  25  No
P  M  70   1  Yes  B  M  66  19  No  B  M  59  29  No
A  F  64  30  No   A  M  70  28  No  A  M  69   1  No
B  F  78   1  No   P  M  83   1  Yes B  F  69  42  No
B  M  75  30  Yes  P  M  77  29  Yes P  F  79  20  Yes
A  M  70  12  No   A  F  69  12  No  B  F  65  14  No
B  M  70   1  No   B  M  67  23  No  A  M  76  25  Yes
```

```
P  M  78  12  Yes  B  M  77   1  Yes B  F  69  24  No
P  M  66   4  Yes  P  F  65  29  No  P  M  60  26  Yes
A  M  78  15  Yes  B  M  75  21  Yes A  F  67  11  No
P  F  72  27  No   P  F  70  13  Yes A  M  75   6  Yes
B  F  65   7  No   P  F  68  27  Yes P  M  68  11  Yes
P  M  67  17  Yes  B  M  70  22  No  A  M  65  15  No
P  F  67   1  Yes  A  M  67  10  No  P  F  72  11  Yes
A  F  74   1  No   B  M  80  21  Yes A  F  69   3  No
;
```

The Neuralgia data set contains five variables: Treatment, Sex, Age, Duration, and Pain. The last variable, Pain, is the target variable. Pain=Yes indicates that the patient felt pain, and Pain=No indicates no pain. The variable Treatment is a nominal variable that has three levels: A and B represent the two test treatments, and P represents the placebo treatment. The gender of the patients is indicated by the nominal variable Sex. The variable Age is the age of the patients, in years, when treatment began. The duration of complaint, in months, before the treatment began is indicated by the variable Duration.

You can load the Neuralgia data set into your CAS session by naming your CAS engine libref in the first statement of the following DATA step:

```
data mycas.neuralgia;
   set neuralgia;
run;
```

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

## Structure Learning

The following statements use the BNET procedure to learn a Bayesian network with Treatment and Sex as nominal variables and Age and Duration as interval variables. The two interval variables are binned into three equal-width levels.

```
proc bnet data=mycas.neuralgia numbin=3 outnetwork=mycas.network;
    target Pain;
    input Treatment Sex/level=nominal;
    input Age Duration/level=interval;
    ods output varselect=varselect;
run;
```

The following statements produce Figure 5.1, which shows the network structure that PROC BNET has learned. There are three variables in the network: Treatment is the parent of Pain, and Pain is the parent of Age. From the structure, you can infer that Pain is dependent on Treatment and is also dependent on Age, but is (conditionally) independent of Sex or Duration.

```
proc print data=mycas.network noobs label;
    var _parentnode_ _childnode_;
    where _type_="STRUCTURE";
run;
```

**Figure 5.1** Network Structure

| Parent Node | Child Node |
|---|---|
| Treatment | Pain |
| Pain | Age |

## Probability Table

The following statements produce Figure 5.2, which shows the conditional probability table for each node in the network. You can use these probability tables for scoring or inferences or both. The conditional probability tables together with the network structure determine the Bayesian network.

```
proc print data=mycas.network noobs label;
    var _parentnode_ _parentcond_ _childnode_ _childcond_ _value_;
    where _type_="PROBABILITY";
run;
```

**Figure 5.2** Probability Table

| Parent Node | Parent Condition | Child Node | Child Condition | Value |
|---|---|---|---|---|
| | | Treatment | A | 0.33333 |
| | | Treatment | B | 0.33333 |
| | | Treatment | P | 0.33333 |
| Treatment | A | Pain | Yes | 0.27273 |
| Treatment | A | Pain | No | 0.72727 |
| Treatment | B | Pain | Yes | 0.27273 |
| Treatment | B | Pain | No | 0.72727 |
| Treatment | P | Pain | Yes | 0.72727 |
| Treatment | P | Pain | No | 0.27273 |
| Pain | Yes | Age | <67 | 0.17857 |
| Pain | Yes | Age | <75 | 0.35714 |
| Pain | Yes | Age | >=75 | 0.46429 |
| Pain | No | Age | <67 | 0.31579 |
| Pain | No | Age | <75 | 0.60526 |
| Pain | No | Age | >=75 | 0.07895 |

## Variable Selection

The network in Figure 5.1 does not include variables Sex or Duration, because PROC BNET automatically selects those variables by using independence tests. PROC BNET produces Figure 5.3, which shows the variable selection results. PROC BNET removes Duration from the network because the $p$-value of the chi-square and G-square statistics of Duration are greater than 0.05 (the default value for the ALPHA= option, which is not specified). Sex is conditionally independent of Pain given Treatment; therefore, PROC BNET also removes it from the network.

**Figure 5.3** Variable Selection

**The BNET Procedure**

**Variable Selection**

| Variable | Selected | Chi-Square | Pr > ChiSq | G-Square | Pr > GSq | Mutual Information | DF | Conditional Variables |
|---|---|---|---|---|---|---|---|---|
| **Sex** | No | 7.20000 | 0.0658 | 7.59454 | 0.0552 | 0.34481 | 3 | Treatment |
| **Treatment** | Yes | 13.71429 | 0.0011 | 14.02297 | 0.0009 | 0.45652 | 2 | |
| **Age** | Yes | 14.60003 | 0.0007 | 15.27118 | 0.0005 | 0.47404 | 2 | |
| **Duration** | No | 2.25795 | 0.3234 | 3.34851 | 0.1874 | 0.23298 | 2 | |

# Syntax: BNET Procedure

The following statements are available in the BNET procedure:

**PROC BNET** < *options* > ;
**AUTOTUNE** < *options* > ;
**CODE FILE**=*filename* ;
**FREQ** *variable* ;
**ID** *variables* ;
**INPUT** *variables* / < **LEVEL=INTERVAL** | **NOMINAL** > ;
**OUTPUT OUT**=*CAS-libref.data-table* < *option* > ;
**PARTITION** *partition-option* ;
**SAVESTATE RSTORE=***CAS-libref.data-table* ;
**TARGET** *variable* ;

The PROC BNET statement, the TARGET statement, and the INPUT statement are required. You can specify only one TARGET statement, but you can specify multiple INPUT statements. The following sections describe the PROC BNET statement and then describe the other statements in alphabetical order.

## PROC BNET Statement

**PROC BNET** < *options* > ;

The PROC BNET statement invokes the procedure. Table 5.1 summarizes important *options* in the PROC BNET statement by function.

**Table 5.1** PROC BNET Statement Options

| Option | Description |
|---|---|
| **Data Options** | |
| DATA= | Specifies the input data set |
| NUMBIN= | Specifies the binning number for interval variables |
| PRESCREENING= | Specifies the initial screening for the input variables |
| VARSELECT= | Specifies the selection for the input variables |

**Table 5.1** *continued*

| Option | Description |
|---|---|
| MISSINGINT= | Specifies how to handle missing values for interval variables |
| MISSINGNOM= | Specifies how to handle missing values for nominal variables |
| **Independence Test Options** | |
| INDEPTEST= | Specifies the methods for independence tests |
| ALPHA= | Specifies the significance level for independence tests by using chi-square or G-square statistics |
| MIALPHA= | Specifies the significant level for independence tests by using mutual information |
| **Structure Learning Options** | |
| STRUCTURE= | Specifies the network structure types |
| PARENTING= | Specifies the structure learning methods |
| MAXPARENTS= | Specifies the maximum number of parents allowed for each node in the network |
| **Model Selection Options** | |
| BESTMODEL | Requests that the best model be selected |

You can specify the following *options*:

**ALPHA=***number*

specifies the significance level for independence tests by using chi-square or G-square statistics. The valid range is 0 to 1, inclusive. If you want to choose the best model among several, you can specify up to five *numbers*, separated by spaces. If you specify multiple *numbers* but you do not specify the BESTMODEL option, PROC BNET uses the first *number* and ignores the remaining *numbers*.

By default, ALPHA = 0.05.

**BESTMODEL**

requests that the best model be selected by using a validation data subset. You can specify the validation data subset by using the PARTITION statement. If you specify this option, you can specify multiple values for the ALPHA=, PRESCREENING=, VARSELECT=, STRUCTURE=, and PARENTING= options. PROC BNET uses the misclassification errors on the validation data to automatically decide the best set of parameter values among these options.

By default, a best model is not selected.

**NOTE:** If you specify BESTMODEL in PROC statement, then AUTOTUNE statement will be ignored.

**DATA=***CAS-libref.data-table*

names the input data table for PROC BNET to use. *CAS-libref.data-table* is a two-level name, where

*CAS-libref* refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME

statement. For more information about *CAS-libref*, see the section "Using CAS Sessions and CAS Engine Librefs" on page 48.

*data-table* specifies the name of the input data table.

**INDEPTEST=ALL | CHIGSQUARE | CHISQUARE | GSQUARE | MI**
specifies the method for independence tests. You can specify the following values:

**ALL** uses the chi-square, the G-square statistics, and the normalized mutual information for independence tests. A variable is independent of the target if both the *p*-values of the chi-square and the G-square statistics are greater than the specified ALPHA= value and the normalized mutual information is less than the value that is specified in MIALPHA= option.

**CHIGSQUARE** uses both the chi-square and the G-square statistics for independence tests. A variable is independent of the target if both the *p*-values of the chi-square and the G-square statistics are greater than the specified ALPHA= value.

**CHISQUARE** uses the chi-square statistics for independence tests. A variable is independent of the target if the *p*-value of the statistics is greater than the specified ALPHA= value.

**GSQUARE** uses the G-square statistics for independence tests. A variable is independent of the target if the *p*-value of the statistics is greater than the specified ALPHA= value.

**MI** uses the normalized mutual information for independence tests. A variable is independent of the target if the normalized mutual information is less than the value that is specified in the MIALPHA= option.

By default, INDEPTEST=CHIGSQUARE.

**MAXPARENTS=***integer*
specifies the maximum number of parents that is allowed for each node in the network structure. The valid range is 1 to 16, inclusive. If you specify the BESTMODEL option, PROC BNET calculates from 1 to *integer* and decides the best number of parents.

By default, MAXPARENTS=5.

**MIALPHA=***number*
specifies the threshold for independence tests by using mutual information. The valid range is 0 to 1, inclusive.

By default, MIALPHA = 0.05.

**MISSINGINT=IGNORE | IMPUTE**
specifies how to handle missing values for all interval input variables. This option applies to training data, validation, testing data and any data used for scoring. You can specify the following values:

**IGNORE** ignores the observations that have missing values in any of the interval variables.

**IMPUTE** replaces the missing values in any interval variable by the mean of the variable.

By default, MISSINGINT=IGNORE.

**MISSINGNOM=IGNORE | IMPUTE | LEVEL**
specifies how to handle the missing values for all nominal input variables. You can specify the following values:

**IGNORE** ignores the observations that have missing values in any of the nominal variables.

**IMPUTE** replaces the missing values in any nominal variable by the mode of the variable.

**LEVEL** treats the missing values in any nominal variable as a separate level of the variable.

By default, MISSINGNOM=IGNORE.

**NTHREADS=***number-of-threads*
specifies the number of threads to use. The default is the minimum CPU count of all the nodes.

**NUMBIN=***integer*
**NBIN=***integer*
specifies the number of binning levels for all interval variables. PROC BNET bins each interval variable into *integer* equal-width levels. The valid range of *integer* is 2 to 1024, inclusive.

By default, NUMBIN=5.

**OUTNETWORK=***CAS-libref.data-table*
**OUTNET=***CAS-libref.data-table*
names the CAS data table to contain the network structure and the probability distributions. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 48.

**PARENTING=BESTONE | BESTSET**
specifies the algorithm for orienting the network structure. You can specify the following values:

**BESTONE** uses a greedy approach to determine the parents of each node; that is, for each node, the best candidate is added as a parent of the node in each iteration.

**BESTSET** determines the best set of variables among possible candidate sets as the parents of each node; that is, instead of adding one variable in an iteration, PROC BNET tests multiple sets of variables together and chooses the best set as the parents of the node.

If you want to choose between the two methods, you can specify both of them and also specify the BESTMODEL option. If you specify both methods but you do not specify the BESTMODEL option, PROC BNET uses the first specified method, and ignores the other.

By default, PARENTING=BESTSET.

**PRESCREENING=0 | 1**
specifies the initial screening for the input variables. You can specify the following values:

**0** uses all the input variables.

**1** uses only the input variables that are dependent on the target.

If you want to choose the best model with or without prescreening, you can specify PRESCREENING=0 1 or PRESCREENING= 1 0 and also specify the BESTMODEL option. If you specify both but you do not specify the BESTMODEL option, PROC BNET uses the first specified value, and ignores the other.

By default, PRESCREENING=1.

**PRINTTARGET**

generates the table, "Predicted Probability Variables," which displays the target variable and the predicted probability variables and the table "Predicted Target Variable" which displays the predicted target variable.

By default, these two tables are not generated.

**STRUCTURE=MB | NAIVE | PC | TAN**

specifies the network structure. Together with the MAXPARENTS= option, this option determines which network structure the procedure learns from the training data. You can specify the following values:

**MB** learns the Markov blanket of the target variable. The Markov blanket includes the parents, the children, and the other parents of the children. After learning the Markov blanket, PROC BNET further determines the parents of the target, the links from the parents to the children, and the links among the children. When you specify STRUCTURE=MB, the procedure learns the Markov blanket regardless of the values of PRESCREENING= and VARSELECT= options.

**NAIVE** assumes a naive Bayesian network structure (that is, the target has a direct link to each input variable). If MAXPARENTS=1, the structure is a naive Bayesian network (NB). If MAXPARENTS is greater than 1, the structure is a Bayesian network-augmented naive Bayesian network (BAN).

**PC** learns the parent-child Bayesian network structure (PC). STRUCTURE=PC differs from STRUCTURE=NAIVE in that some input variables could be learned as the parents of the target variable. In addition, links from the parents to the children and among the children are also possible in PC.

**TAN** learns the tree-augmented naive Bayesian network structure. The TAN structure includes a direct link from the target to each input variable plus a tree structure among the input variables.

If you want to choose the best structure among several structures, you can specify multiple values in any combination, separated by spaces, and also specify the BESTMODEL option. If you specify multiple structures but you do not specify the BESTMODEL option, PROC BNET uses the first value that you specify, and ignores the rest.

By default, STRUCTURE=PC.

**VARSELECT=0 | 1 | 2 | 3**

specifies how input variables are selected beyond the prescreening. You can specify the following values:

**0** uses all input variables that remain after the initial screening is performed as specified in the PRESCREENING= option.

**1** tests each input variable for conditional independence of the target variable given any other input variable. This type of selection uses only the variables that are conditionally dependent on the target given any other input variable.

**2** tests each input variable further for conditional independence of the target variable given any subset of other input variables. This type of selection uses only the variables that are conditionally dependent on the target given any subset of other input variables.

**3** determines the Markov blanket of the target variable and uses only the variables in the Markov blanket.

If you specify VARSELECT=1, 2, or 3, PROC BNET automatically tests each input variable for unconditional independence of the target regardless of the value of the PRESCREENING= option. If no variables are left at a particular variable selection level, PROC BNET rolls back to the previous level. For example, if you specify VARSELECT=3 and there are no variables in the Markov blanket of the target, PROC BNET uses the variables from the previous level, VARSELECT=2.

If you want to choose the best model among different levels of variable selections, you can specify any combination of values for the VARSELECT= option and also specify the BESTMODEL option. If you specify multiple values for the VARSELECT= but you do not specify the BESTMODEL option, PROC BNET uses the first specified value, and ignores the remaining values.

By default, VARSELECT=1.

## AUTOTUNE Statement

**AUTOTUNE** < *options* > **;**

The AUTOTUNE statement searches for the best combination of values for the ALPHA, INDEPTEST, MAXPARENTS, MIALPHA, MISSINGINT, MISSINGNOM, NUMBIN, PARENTING, PRESCREENING, STRUCTURE, and VARSELECT options in the PROC BNET statement.

Table 5.2 summarizes the *options* that you can specify in the AUTOTUNE statement. For more information about all options except the TUNINGPARAMETERS= option, see the option's description in the section "AUTOTUNE Statement" on page 14 in Chapter 3, "Shared Concepts." The TUNINGPARAMETERS= option is described following Table 5.2.

**NOTE:** Processing the AUTOTUNE statement is computationally expensive and requires a significant amount of time.

**NOTE:** If you specify both the AUTOTUNE statement and the BESTMODEL option in the PROC BNET statement, the AUTOTUNE statement is ignored.

**Table 5.2** AUTOTUNE Statement Options

| Option | Description |
|---|---|
| EVALHISTORY= | Specifies how to report the evaluation history of the tuner |
| FRACTION= | Specifies the fraction of observations to use for validation |
| KFOLD= | Specifies the number of folds for *k*-fold cross validation |
| MAXBAYES= | Specifies the maximum number of points in the kriging model |
| MAXEVALS= | Specifies the maximum number of evaluations |
| MAXITER= | Specifies the maximum number of iterations when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN |
| MAXTIME= | Specifies the maximum time for all iterations |
| MAXTRAINTIME= | Specifies the maximum time for a model train |
| NPARALLEL= | Specifies the number of parallel sessions |
| NSUBSESSIONWORKERS= | Specifies the number of workers in parallel sessions |
| OBJECTIVE= | Specifies the objective function |
| POPSIZE= | Specifies the population size when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN |
| SAMPLESIZE= | Specifies the sample size when SEARCHMETHOD=LHS or SEARCHMETHOD=RANDOM |
| SEARCHMETHOD= | Specifies the search method that the optimizer uses |
| TARGETEVENT= | Specifies the target event for ROC-based calculations |
| TRAINFRACTION= | Specifies the fraction of observations to use for training |
| TUNINGPARAMETERS= | Specifies the custom tuning parameters |
| USEPARAMETERS= | Specifies how to handle the TUNINGPARAMETERS= option |

**TUNINGPARAMETERS=(***suboption* | . . . | < *suboption* >**)**

**TUNEPARMS=(***suboption* | . . . | < *suboption* >**)**

specifies which parameters to tune and which ranges to tune over. If you specify USEPARAMETERS=STANDARD, this option is ignored.

You can specify one or more of the following *suboptions*:

**ALPHA (LB=***number* **UB=***number* **VALUES=***value-list* **INIT=***number* **EXCLUDE)**

specifies information about the significance level for independence tests by using chi-square or G-square statistics, where *number* or any value in the *value-list* must be a real number in the range 0 to 1, inclusive. For more information, see the ALPHA= option in the PROC BNET statement.

You can specify the following additional suboptions:

**LB=***number*

specifies the minimum significance level to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=0.01.

**UB=***number*
specifies the maximum significance level to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=0.99.

**VALUES=***value-list*
specifies a list of significance levels to consider during tuning, where *value-list* is a space-separated list of numbers in the range 0 to 1. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

**INIT=***number*
specifies the initial significance level for the tuner to use.

By default, INIT=0.05.

**EXCLUDE**
excludes the significance level from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**INDEPTEST (VALUES=***value-list* **INIT=***value* **EXCLUDE)**
specifies information about the method to use for independence tests. For more information, see the INDEPTEST= option in the PROC BNET statement.

You can specify the following additional suboptions:

**VALUES=***value-list*
specifies a list of methods to use for independence tests, where *value-list* is a space-separated list of CHIGSQUARE, CHISQUARE, MI, and GSQUARE.

**INIT=CHIGSQUARE | CHISQUARE | MI | GSQUARE**
specifies the initial method to use for independence tests.

By default, INIT=CHIGSQUARE.

**EXCLUDE**
excludes the INDEPTEST suboption from the tuning process. If you specify EXCLUDE, any specified VALUES= and INIT= suboptions are ignored.

**MAXPARENTS (LB=***number* **UB=***number* **VALUES=***value-list* **INIT=***number* **EXCLUDE)**
specifies information about the maximum number of parents that are allowed for each node in the network structure. The valid range is 1 to 16, inclusive. For more information, see the MAXPARENTS= option in the PROC BNET statement.

You can specify the following additional suboptions:

**LB=***number*
specifies a lower bound on the maximum number of parents to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=1.

**UB=***number*
specifies an upper bound on the maximum number of parents to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=16.

**VALUES=***value-list*
specifies a list of values to consider for the maximum number of parents in the network structure, where *value-list* is a space-separated list of numbers. If you specify this suboption, you cannot specify the LB= and UB= suboptions.

**INIT=***number*
specifies the initial maximum number of parents in the network structure.

By default, INIT=5.

**EXCLUDE**
excludes the maximum number of parents from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**MIALPHA (LB=***number* **UB=***number* **VALUES=***value-list* **INIT=***number* **EXCLUDE)**
specifies information about the threshold for independence tests by using mutual information. For more information, see the MIALPHA= option in the PROC BNET statement.

You can specify the following additional suboptions:

**LB=***number*
specifies the minimum threshold value to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=0.

**UB=***number*
specifies the maximum threshold value to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=1.

**VALUES=***value-list*
specifies a list of threshold values to consider during tuning, where *value-list* is a space-separated list of numbers in the range 0 to 1. If you specify this suboption, you cannot specify the LB= and UB= suboptions.

**INIT=***number*
specifies the initial threshold value for the tuner to use.

By default, INIT=0.05.

**EXCLUDE**
excludes the threshold value from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**MISSINGINT (VALUES=***value-list* **INIT=***value* **EXCLUDE)**
specifies information about how to handle missing values for interval variables during the tuning process. For more information, see the MISSINGINT= option in the PROC BNET statement.

You can specify the following additional suboptions:

**VALUES=***value-list*
specifies a list of values for the tuner to try for the MISSINGINT suboption, where you can specify IMPUTE or IGNORE (or both) in a space-separated *value-list*.

**INIT=IMPUTE | IGNORE**
specifies whether to start tuning by imputing or ignoring missing values of interval variables.

By default, INIT=IGNORE.

**EXCLUDE**
excludes the MISSINGINT suboption from the tuning process. If you specify EXCLUDE, any specified VALUES= and INIT= suboptions are ignored.

**MISSINGNOM (VALUES=***value-list* **INIT=***value* **EXCLUDE)**
specifies information about how to handle missing values for nominal variables during the tuning process. For more information, see the MISSINGNOM= option in the PROC BNET statement.

You can specify the following additional suboptions:

**VALUES=***value-list*
specifies a list of values for the tuner to handle missing values for nominal variables, where *value-list* is a combination of one or more of the following values in a space-separated list: IMPUTE, IGNORE, and LEVEL.

**INIT=IMPUTE | IGNORE | LEVEL**
specifies the initial value to use in tuning the MISSINGNOM suboption.

By default, INIT=IGNORE.

**EXCLUDE**
excludes the MISSINGNOM suboption from the tuning process. If you specify EXCLUDE, any specified VALUES= and INIT= suboptions are ignored.

**NUMBIN (LB=***number* **UB=***number* **VALUES=***value-list* **INIT=***number* **EXCLUDE)**
specifies information about the number of binning levels for all interval variables. The valid range is 2 to 20. For more information, see the NUMBIN= option in the PROC BNET statement.

You can specify the following additional suboptions:

**LB=***number*
specifies a lower bound of binning levels to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=2.

**UB=***number*
specifies an upper bound of binning levels to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=20.

**VALUES=***value-list*
specifies a list of values of binning levels to consider, where *value-list* is a space-separated list of numbers. If you specify this suboption, you cannot specify the LB= and UB= suboptions.

**INIT=***number*
specifies the initial number of binning levels.

By default, INIT=5.

**EXCLUDE**
excludes the number of binning levels from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**PARENTING (VALUES=***value-list* **INIT=***value* **EXCLUDE)**
specifies information about the algorithm for orienting the network structure. For more information, see the PARENTING= option in the PROC BNET statement.

You can specify the following additional suboptions:

**VALUES=***value-list*
specifies a list of algorithms for orienting the network structure during tuning, where *value-list* is a space-separated list of BESTONE and BESTSET.

**INIT=BESTONE | BESTSET**
specifies the initial algorithm for orienting the network structure for the tuner to use.

By default, INIT=BESTSET.

**EXCLUDE**
excludes the algorithm selection for orienting the network structure from the tuning process. If you specify this suboption, any specified VALUES= and INIT= suboptions are ignored.

**PRESCREENING (VALUES=***value-list* **INIT=***value* **EXCLUDE)**
specifies information about the initial screening for the input variables. For more information, see the PRESCREENING= option in the PROC BNET statement.

You can specify the following additional suboptions:

**VALUES=***value-list*
specifies a list of initial screening values for the input variables, where *value-list* is a space-separated list of 0 and 1.

**INIT=***number*
specifies the initial screening for the input variables.

By default, INIT=1.

**EXCLUDE**
excludes the initial screening for the input variables from the tuning process. If you specify this suboption, any specified VALUES= and INIT= suboptions are ignored.

**STRUCTURE (VALUES=***value-list* **INIT=***value* **EXCLUDE)**
specifies information about the network structure. For more information, see the STRUCTURE= option in the PROC BNET statement.

You can specify the following additional suboptions:

**VALUES=***value-list*
specifies a list of structures, where *value-list* is a space-separated list of MB, NAIVE, PC, and TAN.

**INIT=MB | NAIVE | PC | TAN**
specifies the initial network structure.

By default, INIT=PC.

**EXCLUDE**
excludes the network structure from the tuning process. If you specify this suboption, any specified VALUES= and INIT= suboptions are ignored.

**VARSELECT (VALUES=***value-list* **INIT=***value* **EXCLUDE)**
specifies information about how to select input variables during the tuning process after the prescreening. For more information, see the VARSELECT= option in the PROC BNET statement.

You can specify the following additional suboptions:

**VALUES=***value-list*
specifies a list of input variables to consider for the tuning process after the prescreening, where *value-list* is a space-separated list of 0, 1, 2, and 3.

**INIT=***number*
specifies the initial value to consider for the tuning process after the prescreening.

By default, INIT=1.

**EXCLUDE**
excludes the VARSELECT suboption from the tuning process. If you specify EXCLUDE, any specified VALUES= and INIT= suboptions are ignored.

## CODE Statement

**CODE FILE=***filename* **;**

The CODE statement is optional in PROC BNET. If you use a CODE statement, SAS DATA step code is generated and stored in a file that can be used for scoring purposes.

## FREQ Statement

**FREQ** *variable* **;**

The *variable* in the FREQ statement identifies a numeric variable in the data set that contains the frequency of occurrence for each observation. The BNET procedure treats each observation as if it appeared *n* times, where *n* is the value of the *variable* for the observation. If *n* is not an integer, it is truncated to an integer. If *n* is less than 1 or is missing, the observation is ignored. When the FREQ statement is not specified, each observation is assigned a frequency of 1.

## ID Statement

**ID** *variables* **;**

The optional ID statement lists one or more variables from the input data set to be copied to the prediction output data set. The ID statement accepts both numeric and character variables. The variables in an ID statement can also appear in any other statements.

## INPUT Statement

**INPUT** *variables* </**LEVEL=INTERVAL** | **NOMINAL** > **;**

The INPUT statement specifies one or more *variables* as input variables. You can specify multiple INPUT statements. PROC BNET does not support duplicate input variables. If the INPUT statement contains a duplicate variable, PROC BNET returns an error message and then exits. You can specify the following option in each INPUT statement:

**LEVEL=INTERVAL | NOMINAL**
specifies the type of all the variables in the current INPUT statement. You can specify the following values:

**NOMINAL** treats all the variables in the current INPUT statement as nominal variables.

**INTERVAL** treats all the variables in the current INPUT statement as interval variables.

By default, LEVEL=INTERVAL for numeric variables and LEVEL=NOMINAL for categorical variables.

## OUTPUT Statement

**OUTPUT OUT=***CAS-libref.data-table* < *options* > **;**

The OUTPUT statement creates a data table to contain the predicted target values of the input table.

You must specify the following option:

**OUT=***CAS-libref.data-table*
names the output data table for PROC BNET to use. You must specify this option before any other options. *CAS-libref.data-table* is a two-level name, where

*CAS-libref* refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section "Using CAS Sessions and CAS Engine Librefs" on page 48.

*data-table* specifies the name of the output data table.

This table includes variables that are specified either in the COPYVARS= option or in the ID statement. If you specify PARTITION statement, the output includes one more column, _ROLE_. _ROLE_ is a reserved name. If it exist in the input data table and you specify it in the COPYVARS= option or in the ID statement, you need to use the ROLE= option to change the generated column's name.

You can also specify the following *options*:

**COPYVAR=***variable*
**COPYVARS=(***variables***)**
lists one or more variables from the input data table to be copied to the output data table.

**ROLE=***rolename*
renames the generated column _ROLE_ in the output data table to the specified *role*.

## PARTITION Statement

**PARTITION** *partition-option* **;**

The PARTITION statement specifies how observations in the input data set are logically partitioned into disjoint subsets for model training, validation, and testing. For more information, see the section "Using Validation and Test Data" on page 21 in Chapter 3, "Shared Concepts." Either you can designate a variable in the input data table and a set of formatted values of that variable to determine the role of each observation, or you can specify proportions to use for randomly assigning observations to each role.

You must specify exactly one of the following *partition-options*:

**FRACTION(< TEST=***fraction* **> < VALIDATE=***fraction* **> < SEED=***number* **>)**
randomly assigns specified proportions of the observations in the input data table to the roles. You specify the proportions for testing and validation by using the TEST= and VALIDATE= suboptions. If you specify both the TEST= and VALIDATE= suboptions, then the sum of the specified fractions must be less than 1 and the remaining fraction of the observations are assigned to the training role. The SEED= option specifies an integer that is used to start the pseudorandom number generator for random partitioning of data for training, testing, and validation. If you do not specify SEED=*number* or if *number* is less than or equal to 0, the seed is generated by reading the time of day from the computer's clock.

**ROLE=***variable* **(< TEST='***value***' > < TRAIN='***value***' > < VALIDATE='***value***' >)**

**ROLEVAR=***variable* **(< TEST='***value***' > < TRAIN='***value***' > < VALIDATE='***value***' >)**

names the *variable* in the input data table whose values are used to assign roles to each observation. This *variable* cannot also appear as an analysis variable in other statements or options. The TEST=, TRAIN=, and VALIDATE= suboptions specify the formatted values of this variable that are used to assign observation roles. If you do not specify the TRAIN= suboption, then all observations whose role is not determined by the TEST= or VALIDATE= suboption are assigned to the training role.

## SAVESTATE Statement

**SAVESTATE RSTORE=***CAS-libref.data-table* **;**

The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to score new data. For more information, see Chapter 4, "The ASTORE Procedure."

You must specify the following option:

**RSTORE=***CAS-libref.data-table*

specifies a data table in which to save the analytic store for the model. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 48.

## TARGET Statement

**TARGET** *variable* **;**

The TARGET statement names the *variable* that PROC BNET predicts. PROC BNET treats the TARGET *variable* as nominal.

The target values are levelized in descending order.

# Details: BNET Procedure

## Independence Tests

Both variable selection and structure learning require either independence tests between two variables or conditional independence tests given some other variables. PROC BNET supports independence tests by using the chi-square statistic, G-square statistic, normalized mutual information, or some combination of them.

Given two nominal variables X and Y (or interval variables after binning) that have levels $r$ and $c$, respectively, the chi-square statistic is computed as

$$\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(O_{ij}-E_{ij})^2}{E_{ij}}$$

where $O_{ij}$ is an observed frequency in a cell of the contingency table of the two variables and $E_{ij}$ is the expected frequency of the cell. The degrees of freedom for the test is $(r-1)\times(c-1)$. If the $p$-value of the test statistic is greater than the specified significance level, the two variables are considered to be independent.

Similarly, the G-square statistic is calculated as

$$G^2 = 2\sum_{i=1}^{r}\sum_{j=1}^{c} O_{ij}\ln(\frac{O_{ij}}{E_{ij}})$$

where $O_{ij}$ is an observed frequency, $E\_ij$ is the expected frequency, and the degrees of freedom for the test is $(r-1)\times(c-1)$. If the $p$-value of the test statistic is greater than the specified significance level, the two variables are considered to be independent.

The mutual information between X and Y is defined as

$$I(X,Y) = \sum_{x=1}^{r}\sum_{y=1}^{c} p(x,y)\ln(\frac{p(x,y)}{p(x)p(y)})$$

where $p(x,y) = \frac{O_{xy}}{N}$ is the joint distribution function of X and Y; $p(x) = \frac{\sum_{y=1}^{c} O_{xy}}{N}$ and $p(y) = \frac{\sum_{x=1}^{r} O_{xy}}{N}$ are the marginal probability distributions of X and Y, respectively; and $N$ is the total number of observations in the training data.

The mutual information is then normalized to be between 0 and 1 as

$$NI(X,Y) = \sqrt{1-e^{-2I(X,Y)}}$$

where $I(X,Y)$ is the mutual information between $X$ and $Y$.

If the value of the normalized mutual information is less than the specified significance level, the two variables are considered to be independent.

to test the conditional independence between two variables X and Y given a subset S of other variables ($X \notin S$ and $Y \notin S$), the corresponding statistics are summed for each value combination of $S$, and the corresponding degrees of freedom for the chi-square and G-square statistics is $(r-1)\times(c-1)\times q$, where $q$ is the number of value combinations for $S$.

The following PROC BNET options are related to independence tests:

- The INDEPTEST= option specifies which test statistic or combination of them to use.
- The ALPHA= option specifies the significance level for the chi-square and the G-square statistics.
- The MIALPHA= option specifies the significance level for the normalized mutual information.

## Variable Selection

A Bayesian network is a graphical model that consists of two parts, $< G, P >$, where $G$ is a directed acyclic graph (DAG) whose nodes correspond to the random variables in $U$ ($U$ is the set of input variables plus the target variable in PROC BNET and $P$ is a set of local probability distributions, one for each node conditional on each value combination of the parents). The joint probability distribution of $U$ can be factorized to the product of the local probability distributions; that is $p(U) = \Pi_{X \in U} p(X|\pi(X))$, where $\pi(X)$ are the parents of $X$. It is assumed that the network structure $G$ and the probability distribution $P$ are faithful to each other; that is, every conditional independence in the structure $G$ is also present in $P$, and vice versa.

Given a target variable $T$, a Markov blanket of $T$ is defined as a subset of input variables $MB \subseteq U - \{T\}$ such that $T$ is conditionally independent of each of the remaining input variables $X \in U - MB - \{T\}$ given $MB$. Under the faithful assumption, the Markov blanket of $T$ is unique. According to the definition of the Markov blanket, the probability distribution of $T$ is completely determined by its Markov blanket; therefore, the Markov blanket can be used for variable selection.

PROC BNET supports two types of variable selection: one by independence tests between each input variable and the target (when PRESCREENING=1) and the other by conditional independence tests between each input variable and the target given any subset of other input variables (when VARSELECT=1, 2, or 3).

PROC BNET uses specialized data structures to efficiently compute the contingency tables for any variable combination, and it uses dynamic candidate generation to reduce the false candidates (variable combinations).

## Structure Learning

In general, there are two approaches to learning the network structure: one is score-based, and the other is constraint-based. The score-based approach uses a score function to measure how well a structure fits the training data and tries to find the structure that has the best score. The constraint-based approach uses independence tests to determine the edges and the directions.

PROC BNET uses both score-based and constraint-based approaches to learn the network structure. It uses the BIC (Bayesian information criterion) score, which is defined as

$$\mathrm{BIC}(G, D) = N \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} p(\pi_{ij}) p(X_i = v_{ik}|\pi_{ij}) \ln p(X_i = v_{ik}|\pi_{ij}) - \frac{M}{2} \ln N$$

where $G$ is a network, $D$ is the training data set, $N$ is the number of observations in $D$, $n$ is the number of variables, $X_i$ is a random variable, $r_i$ is the number of levels for $X_i$, $v_{ik}$ is the $k$th value of $X_i$, $q_i$ is the number of value combinations of $X_i$'s parents, $\pi_{ij}$ is the $j$th value combination of $X_i$'s parents, and $M = \sum_{i=1}^{n} (r_i - 1) \times q_i$ is the number of parameters for the probability distributions.

PROC BNET uses independence tests to determine the edges and the directions as follows. Assume that you have three variables, $X$, $Y$ and $Z$, and that it has been determined (using independent tests) that there are edges between $X$ and $Z$ and $Y$ and $Z$, but no edge between $X$ and $Y$. If $X$ is conditionally dependent of $Y$ given any subset of variables $S = \{Z\} \bigcup S', S' \subseteq U - \{X, Y, Z\}$, then the direction between $X$ and $Z$ is $X \rightarrow Z$ and the direction between $Y$ and $Z$ is $Y \rightarrow Z$. Notice that using independence tests alone might not be able

to orient all edges because some structures are equivalent with respect to conditional independence tests. For example, $X \leftarrow Y \leftarrow Z$, $X \rightarrow Y \rightarrow Z$, and $X \leftarrow Y \rightarrow Z$ belong to the same equivalence class. In these cases, PROC BNET uses the BIC score to determine the directions of the edges.

PROC BNET learns different types of network structures: naive Bayesian (NB), tree-augmented naive (TAN), Bayesian network-augmented naive (BAN), parent-child Bayesian network (PC), and Markov blanket (MB). Based on the network structure that is specified, it uses different algorithms. For example, if you specify STRUCTURE=TAN, the procedure uses the maximum spanning tree to learn the tree structure, where the weight for an edge is the mutual information between the two nodes. PROC BNET uses either the BESTONE or BESTSET value of the PARENTING= option to learn the other network structures (BAN, PC, MB).

PROC BNET orders the input variables based on the BIC score with the target. The BIC score of an input variable $X$ with the target is defined as

$$\mathrm{BIC}(X, T) = \max(\mathrm{BIC}(X \rightarrow T), \mathrm{BIC}(T \rightarrow X))$$

where $\mathrm{BIC}(X \rightarrow T)$ is the BIC score when $X$ is the parent of $T$ (ignoring all the remaining variables) and $\mathrm{BIC}(T \rightarrow X)$ is the BIC score when $X$ is the child of $T$ (ignoring all the remaining variables).

PROC BNET learns the parents of the target first for structures PC and MB. Then it learns the parents of the input variable that has the highest BIC score with the target. It continues learning the parents of the input variable that has the next highest BIC score, and so on. When learning the parents of a node, it first determines the edges by using independence tests. Then it orients the edges by using both independence tests and the BIC score. PROC BNET uses the BIC score not only for orienting the edges but also for controlling the network complexity, because a complex network that has more parents is penalized in the BIC score.

Both the BESTONE and BESTSET value of the PARENTING= option try to find the local optimum structure for each node. BESTONE adds the best candidate variable to the parents at each iteration, whereas BESTSET tries to pick the best set of variables among the candidate sets.

If you have many input variables, structure learning can be time consuming, because the number of variable combinations is exponential. Therefore, variable selection is strongly recommended.

## Parameter Learning

Parameter learning determines the probability distribution for each node in a network structure. In PROC BNET, the probability distribution is discrete because the interval variables are binned.

You can use the resulting probability distribution table to score an observation $(x_1, x_2, \ldots, x_{n-1})$ as

$$\begin{aligned} \mathrm{argmax}_c\, p(T = c | x_1, x_2, \ldots, x_{n-1}) &= p(x_1, x_2, \ldots, x_{n-1} | T = c) \times K \\ &= \Pi_i\, p(x_i | \pi(X_i)) \times K \end{aligned}$$

where $c$ is a level of the target variable, $\pi(X_i)$ are the parents of $X_i$, $K$ is a constant, and $X_n = T$ (target) for convenience.

To estimate the parameters $p(x_i|\pi(X_i))$, PROC BNET uses an additive smoothing technique:

$$\hat{p}(x_i|\pi(X_i)) = \frac{\text{counts\_of}(xi, pi(X_i)) + \alpha}{\text{counts\_of}(pi(X_i)) + \alpha \times n_i}$$

where $\alpha$ is the smoothing parameter ($\alpha = 0$ corresponds to no smoothing, and PROC BNET uses $\alpha = 1$) and $n_i$ is the number of possible values for $X_i$.

The general reason for smoothing is to avoid overfitting the data. The case where the count of some class is 0 is just a particular case of overfitting. PROC BNET still smooths the probabilities when every class is observed.

## Displayed Output

The following sections describe the output that PROC BNET produces by default. The output is organized into various tables, which are discussed in the order of their appearance.

### Model Information

The "Model Information" table contains the initial training settings, such as significance level, structure, and number of bins.

### Fit Statistics

The "Fit Statistics" table contains the fit statistics of the Bayesian network.

### Number of Observations

The "Number of Observations" table contains the number of observations and the number of observations used.

### Variable Level

The "Variable Level" table contains the details of each level of the variables. The columns include the observed target, predicted event, predicted nonevent, and total numbers of events or nonevents for the training data.

### Variable Order

The "Variable Order" table contains the order of the input variables based on the BIC score with the target.

### Variable Information

The "Variable Information" table contains the variable information such as number of levels, number of missing values, and so on.

### Iteration Report

The “Iteration Report” table contains the variable selection results.

### Validation Information

The “Validation Information” table contains the validation results. If the PARTITION statement is specified, then the misclassification errors mean misclassification errors in the validation data; if not, then the misclassification errors mean misclassification errors in the training data.

## ODS Table Names

Each table that the BNET procedure creates has a name associated with it. You must use this name to refer to the table when you use the ODS statements. These names are listed in Table 5.3.

**Table 5.3** ODS Tables Produced by PROC BNET

| Table Name | Description | Statement | Option |
|---|---|---|---|
| FitStatistics | Contains the fit statistics of the network | PROC BNET | Default |
| ModelInfo | Contains the initial training settings, such as significance level, structure, and number of bins | PROC BNET | Default |
| NObs | Contains the number of observations for training, validation and testing, and so on | PROC BNET | Default |
| PredIntoName | Predicted target variable | PROC BNET | PRINTTARGET |
| PredProbName | Predicted probability variables | PROC BNET | PRINTTARGET |
| ValidInfo | Contains the validation results | PROC BNET | BESTMODEL |
| VarInfo | Contains the variable information such as number of levels, number of missing values, and so on | PROC BNET | Default |
| VarLevel | Contains the details of each level of the variables | PROC BNET | Default |
| VarOrder | Contains the order of the input variables | PROC BNET | Default |
| VarSelect | Contains the variable selection results | PROC BNET | Default |

# Examples: BNET Procedure

## Example 5.1: Naive Bayesian Network

This example shows how you can use PROC BNET to learn a naive Bayesian network for the Iris data that is available in the Sashelp library. The following DATA step loads the Iris data into your CAS session. This DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.iris;
   set sashelp.iris;
run;
```

The following statements specify MAXPARENTS=1, PRESCREENING=0, and VARSELECT=0 to request that PROC BNET use only one parent for each node and use all the input variables:

```
proc bnet data=mycas.iris numbin=3 structure=Naive maxparents=1
        prescreening=0 varselect=0
        outnetwork=mycas.network;
    target Species;
    input PetalWidth PetalLength SepalLength SepalWidth/level=interval;
run;
```

The following statements produce Output 5.1.1, which shows the network structure. In the structure, Species is the parent of PetalWidth, PetalLength, SepalLength, and SepalWidth.

```
proc print data=mycas.network noobs label;
    var _parentnode_ _childnode_;
    where _type_="STRUCTURE";
run;
```

**Output 5.1.1** Naive Bayesian Network Structure

| Parent Node | Child Node |
|---|---|
| Species | PetalLength |
| Species | PetalWidth |
| Species | SepalLength |
| Species | SepalWidth |

## Example 5.2: Tree-Augmented Naive Bayesian Network

This example also uses the Iris data set that is available in the Sashelp library. In the following statements, STRUCTURE=TAN results in a tree-augmented Bayesian network:

```
data mycas.iris;
   set sashelp.iris;
run;
```

```
proc bnet data=mycas.iris numbin=3 structure=TAN
            prescreening=0 varselect=0
            outnetwork=mycas.network;
    target Species;
    input PetalWidth PetalLength SepalLength SepalWidth/level=interval;
run;
```

The following statements produce Output 5.2.1, which shows the network structure. In the structure, Species is a parent of PetalWidth, PetalLength, SepalLength, and SepalWidth. In addition, PetalWidth is a parent PetalLength, SepalWidth is a parent of SepalLength, and PetalWidth is a parent of SepalWidth.

```
proc print data=mycas.network noobs label;
    var _parentnode_ _childnode_;
    where _type_="STRUCTURE";
run;
```

**Output 5.2.1** TAN Network Structure

| Parent Node | Child Node |
|---|---|
| Species | PetalLength |
| PetalWidth | PetalLength |
| Species | PetalWidth |
| Species | SepalLength |
| SepalWidth | SepalLength |
| Species | SepalWidth |
| PetalWidth | SepalWidth |

## Example 5.3: Parent-Child Bayesian Network

This example also uses the Iris data set that is available in the Sashelp library. In the following statements, STRUCTURE=PC results in a parent-child Bayesian network:

```
data mycas.iris;
   set sashelp.iris;
run;

proc bnet data=mycas.iris numbin=3 structure=PC
            prescreening=0 varselect=0
            outnetwork=mycas.network;
    target Species;
    input PetalWidth PetalLength SepalLength SepalWidth/level=interval;
run;
```

The following statements produce Output 5.3.1, which shows the network structure. In the structure, PetalLength is the parent of Species, and Species is the parent of PetalWidth, SepalLength, and SepalWidth.

```
proc print data=mycas.network noobs label;
    var _parentnode_ _childnode_;
    where _type_="STRUCTURE";
run;
```

**Output 5.3.1** Parent-Child Network Structure

| Parent Node | Child Node |
|---|---|
| PetalLength | Species |
| Species | PetalWidth |
| Species | SepalLength |
| Species | SepalWidth |

## Example 5.4: Markov Blanket

This example uses the HMEQ sample data set that is available in the Sampsio library to learn a Markov blanket Bayesian network, which is specified by STRUCTURE=MB:

```
data mycas.hmeq;
   set sampsio.hmeq;
run;

 proc bnet data=mycas.hmeq indeptest=MI mialpha=0.2 structure=MB nbin=5
         missingint=IMPUTE missingnom=LEVEL
         outnetwork=mycas.network;
     target Bad;
     input Reason Job Delinq Derog Ninq/level=nominal;
     input Loan Mortdue Value Yoj Clage Clno Debtinc/level=interval;
 run;
```

The following statements produce Output 5.4.1, which shows the network structure. In the structure, Bad is a parent of Delinq and Derog, Delinq and Ninq are the other parents of Derog, and Ninq is the other parent of Delinq.

```
proc print data=mycas.network noobs label;
    var _parentnode_ _childnode_;
    where _type_="STRUCTURE";
run;
```

**Output 5.4.1** Markov Blanket Network Structure

| Parent Node | Child Node |
|---|---|
| BAD | DELINQ |
| NINQ | DELINQ |
| BAD | DEROG |
| DELINQ | DEROG |
| NINQ | DEROG |

## Example 5.5: Bayesian Network-Augmented Naive Bayesian Network

This example also uses the HMEQ sample data set that is available in the Sampsio library to learn a BAN structure, which is specified by STRUCTURE=NAIVE:

```
data mycas.hmeq;
   set sampsio.hmeq;
run;

 proc bnet data=mycas.hmeq numbin=10 alpha=0.1 structure=Naive
             missingint=IMPUTE missingnom=LEVEL
             outnetwork=mycas.network;
     target Bad;
     input Reason Job Delinq Derog Ninq/level=nominal;
     input Loan Mortdue Value Yoj Clage Clno Debtinc/level=interval;
 run;
```

The following statements produce Output 5.5.1, which shows the network structure. In the structure, Bad is a parent of Delinq, Derog, Job, Ninq, Clage, Clno, Loan, and Mortdue. In addition, Delinq is the other parent of Derog, and Job is a parent of both Mortdue and Clno.

```
 proc print data=mycas.network noobs label;
     var _parentnode_ _childnode_;
     where _type_="STRUCTURE";
 run;
```

**Output 5.5.1** BAN Network Structure

| Parent Node | Child Node |
|---|---|
| BAD | DELINQ |
| BAD | DEROG |
| DELINQ | DEROG |
| BAD | JOB |
| BAD | NINQ |
| BAD | CLAGE |
| BAD | CLNO |
| JOB | CLNO |
| BAD | LOAN |
| BAD | MORTDUE |
| JOB | MORTDUE |

## Example 5.6: Model Selection

This example uses the German Credit sample data that is available in the Sampsio library to learn the best Bayesian network model among all network structures: naive, TAN, PC, and MB, with or without variable selection. PROC BNET also tries to choose the best value for the MAXPARENTS= option. About 30% of the input data is used for validation.

```
data dmagecr_part;
  set sampsio.dmagecr;
  seed=12345;
  if ranuni(seed) < 0.7 then partind=1;
  else partind=0;
  id=_N_;
```

```
run;

data mycas.dmagecr;
   set dmagecr_part;
run;

 proc bnet data=mycas.dmagecr numbin=10 alpha=0.05
          structure=Naive TAN PC MB varselect=0 1 bestmodel
          outnetwork=mycas.network;
     target Good_Bad;
     input  Checking History Purpose Savings Employed Installp Marital Coapp
            Resident Property Other Housing Existcr Job Depends Telephon
            Foreign/level=nominal;
     input Age Amount Duration/level=interval;
     partition rolevar= PartInd (TRAIN='1' VALIDATE='0' TEST='2');
     ods output nobs=nobs fitstatistics=fit validinfo=validinfo;
 run;
```

Output 5.6.1 shows the information of number of observations: 695 observations are used for training and 305 are used for validation.

**Output 5.6.1** Model Selection: Number of Observations

**The BNET Procedure**

| Number of Observations | |
|---|---|
| Number of Observations Read | 1000 |
| Number of Observations Used | 1000 |
| Number of Observations Used for Training | 695 |
| Number of Observations Used for Validation | 305 |
| Number of Observations Used for Testing | 0 |

Output 5.6.2 shows the fit statistics. In the resulting network, there are 13 nodes and 13 links between the nodes, and the number of parameters is 129.

**Output 5.6.2** Model Selection: Fit Statistics

| Fit Statistics | |
|---|---|
| Number of Nodes | 13 |
| Number of Links | 13 |
| Average Degree | 2 |
| Maximum Number of Parents in Network | 2 |
| Number of Parameters | 129 |
| Score | -11092.55 |
| Validation Misclassification Rate | 0.23934426 |
| Test Misclassification Rate | . |

Output 5.6.3 shows the validation results for each parameter combination. The PC Bayesian network structure has misclassified 73 observations out of 305 validation observations when VARSELECT=0 and MAXPARENTS is greater than or equal to 2. The TAN structure has 77 misclassification errors when VARSELECT=0 and MAXPARENTS=2. The naive Bayesian network has 81 misclassification errors when VARSELECT=0 and MAXPARENTS is greater than or equal to 2. The MB Bayesian network structure has 113 misclassification errors.

**Output 5.6.3** Model Selection: Validation Information

| Validation Information | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Best Model | Misclassification Rate | Misclassification Errors | N Observations for Assessment | Significance Threshold | Prescreening | Variable Selection | Structure | Parenting Method |
| Yes | 0.23934 | 73 | 305 | 0.05 | 1 | 0 | PC | BestSet |
| | 0.23934 | 73 | 305 | 0.05 | 1 | 0 | PC | BestSet |
| | 0.23934 | 73 | 305 | 0.05 | 1 | 0 | PC | BestSet |
| | 0.23934 | 73 | 305 | 0.05 | 1 | 0 | PC | BestSet |
| | 0.25246 | 77 | 305 | 0.05 | 1 | 0 | TAN | BestSet |
| | 0.26557 | 81 | 305 | 0.05 | 1 | 0 | Naive | BestSet |
| | 0.26557 | 81 | 305 | 0.05 | 1 | 0 | Naive | BestSet |
| | 0.26557 | 81 | 305 | 0.05 | 1 | 0 | Naive | BestSet |
| | 0.26557 | 81 | 305 | 0.05 | 1 | 0 | Naive | BestSet |
| | 0.26885 | 82 | 305 | 0.05 | 1 | 1 | PC | BestSet |
| | 0.26885 | 82 | 305 | 0.05 | 1 | 1 | PC | BestSet |
| | 0.26885 | 82 | 305 | 0.05 | 1 | 1 | PC | BestSet |
| | 0.26885 | 82 | 305 | 0.05 | 1 | 1 | PC | BestSet |
| | 0.28197 | 86 | 305 | 0.05 | 1 | 0 | PC | BestSet |
| | 0.28197 | 86 | 305 | 0.05 | 1 | 0 | Naive | BestSet |
| | 0.34754 | 106 | 305 | 0.05 | 1 | 1 | PC | BestSet |
| | 0.34754 | 106 | 305 | 0.05 | 1 | 1 | Naive | BestSet |
| | 0.34754 | 106 | 305 | 0.05 | 1 | 1 | Naive | BestSet |
| | 0.34754 | 106 | 305 | 0.05 | 1 | 1 | Naive | BestSet |
| | 0.34754 | 106 | 305 | 0.05 | 1 | 1 | Naive | BestSet |
| | 0.34754 | 106 | 305 | 0.05 | 1 | 1 | Naive | BestSet |
| | 0.35738 | 109 | 305 | 0.05 | 1 | 1 | TAN | BestSet |
| | 0.37049 | 113 | 305 | 0.05 | 1 | 3 | MB | BestSet |
| | 0.37049 | 113 | 305 | 0.05 | 1 | 3 | MB | BestSet |
| | 0.37049 | 113 | 305 | 0.05 | 1 | 3 | MB | BestSet |
| | 0.37049 | 113 | 305 | 0.05 | 1 | 3 | MB | BestSet |
| | 0.37049 | 113 | 305 | 0.05 | 1 | 3 | MB | BestSet |

**Output 5.6.3** *continued*

| Validation Information | Max N Parents |
|---|---|
| | 2 |
| | 3 |
| | 4 |
| | 5 |
| | 2 |
| | 2 |
| | 3 |
| | 4 |
| | 5 |
| | 2 |
| | 3 |
| | 4 |
| | 5 |
| | 1 |
| | 1 |
| | 1 |
| | 1 |
| | 2 |
| | 3 |
| | 4 |
| | 5 |
| | 2 |
| | 1 |
| | 2 |
| | 3 |
| | 4 |
| | 5 |

The following statements produce Output 5.6.4, which shows that PROC BNET has learned a PC Bayesian network structure. In the structure, Checking is the parent of Good_Bad, Housing is the parent of Property, and Good_Bad is the parent of all the selected input variables except its parent Checking.

```
proc print data=mycas.network noobs label;
    var _parentnode_ _childnode_;
    where _type_="STRUCTURE";
run;
```

**Output 5.6.4** Model Selection: Best Structure

| Parent Node | Child Node |
|---|---|
| checking | good_bad |
| good_bad | employed |
| good_bad | foreign |
| good_bad | history |
| good_bad | housing |
| good_bad | other |
| good_bad | property |
| housing | property |
| good_bad | purpose |
| good_bad | savings |
| good_bad | age |
| good_bad | amount |
| good_bad | duration |

# References

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco: Morgan Kaufmann.