

IBM SPSS Neural Networks 30



Note

Before using this information and the product it supports, read the information in [“Notices” on page 17.](#)

Product Information

This edition applies to version 30, release 0, modification 0 of IBM® SPSS® Statistics and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright International Business Machines Corporation .**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Chapter 1. Neural Networks.....	1
Introduction to Neural Networks.....	1
What Is a Neural Network?.....	1
Neural Network Structure.....	2
Multilayer Perceptron.....	3
Partitions	5
Architecture	5
Training	7
Output	8
Save	9
Export	9
Options	10
Radial Basis Function.....	10
Partitions	12
Architecture	13
Output	13
Save	14
Export	15
Options	15
Notices.....	17
Trademarks.....	18
Index.....	19

Chapter 1. Neural Networks

The following neural network features are included in SPSS Statistics Premium Edition or the Neural Networks option.

Introduction to Neural Networks

Neural networks are the preferred tool for many predictive data mining applications because of their power, flexibility, and ease of use. Predictive neural networks are particularly useful in applications where the underlying process is complex, such as:

- Forecasting consumer demand to streamline production and delivery costs.
- Predicting the probability of response to direct mail marketing to determine which households on a mailing list should be sent an offer.
- Scoring an applicant to determine the risk of extending credit to the applicant.
- Detecting fraudulent transactions in an insurance claims database.

Neural networks used in predictive applications, such as the multilayer perceptron (MLP) and radial basis function (RBF) networks, are supervised in the sense that the model-predicted results can be compared against known values of the target variables. The Neural Networks option allows you to fit MLP and RBF networks and save the resulting models for scoring.

What Is a Neural Network?

The term **neural network** applies to a loosely related family of models, characterized by a large parameter space and flexible structure, descending from studies of brain functioning. As the family grew, most of the new models were designed for nonbiological applications, though much of the associated terminology reflects its origin.

Specific definitions of neural networks are as varied as the fields in which they are used. While no single definition properly covers the entire family of models, for now, consider the following description ¹:

A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

- Knowledge is acquired by the network through a learning process.
- Interneuron connection strengths known as synaptic weights are used to store the knowledge.

For a discussion of why this definition is perhaps too restrictive, see ².

In order to differentiate neural networks from traditional statistical methods using this definition, what is *not* said is just as significant as the actual text of the definition. For example, the traditional linear regression model can acquire knowledge through the least-squares method and store that knowledge in the regression coefficients. In this sense, it is a neural network. In fact, you can argue that linear regression is a special case of certain neural networks. However, linear regression has a rigid model structure and set of assumptions that are imposed before learning from the data.

By contrast, the definition above makes minimal demands on model structure and assumptions. Thus, a neural network can approximate a wide range of statistical models without requiring that you hypothesize in advance certain relationships between the dependent and independent variables. Instead, the form of the relationships is determined during the learning process. If a linear relationship between the dependent and independent variables is appropriate, the results of the neural network should closely

¹ Haykin, S. 1998. *Neural Networks: A Comprehensive Foundation*, 2nd ed. New York: Macmillan College Publishing.

² Ripley, B. D. 1996. *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press.

approximate those of the linear regression model. If a nonlinear relationship is more appropriate, the neural network will automatically approximate the "correct" model structure.

The trade-off for this flexibility is that the synaptic weights of a neural network are not easily interpretable. Thus, if you are trying to explain an underlying process that produces the relationships between the dependent and independent variables, it would be better to use a more traditional statistical model. However, if model interpretability is not important, you can often obtain good model results more quickly using a neural network.

Neural Network Structure

Although neural networks impose minimal demands on model structure and assumptions, it is useful to understand the general **network architecture**. The multilayer perceptron (MLP) or radial basis function (RBF) network is a function of predictors (also called inputs or independent variables) that minimize the prediction error of target variables (also called outputs).

Consider the *bankloan.sav* dataset that ships with the product, in which you want to be able to identify possible defaulters among a pool of loan applicants. An MLP or RBF network applied to this problem is a function of the measurements that minimize the error in predicting default. The following figure is useful for relating the form of this function.

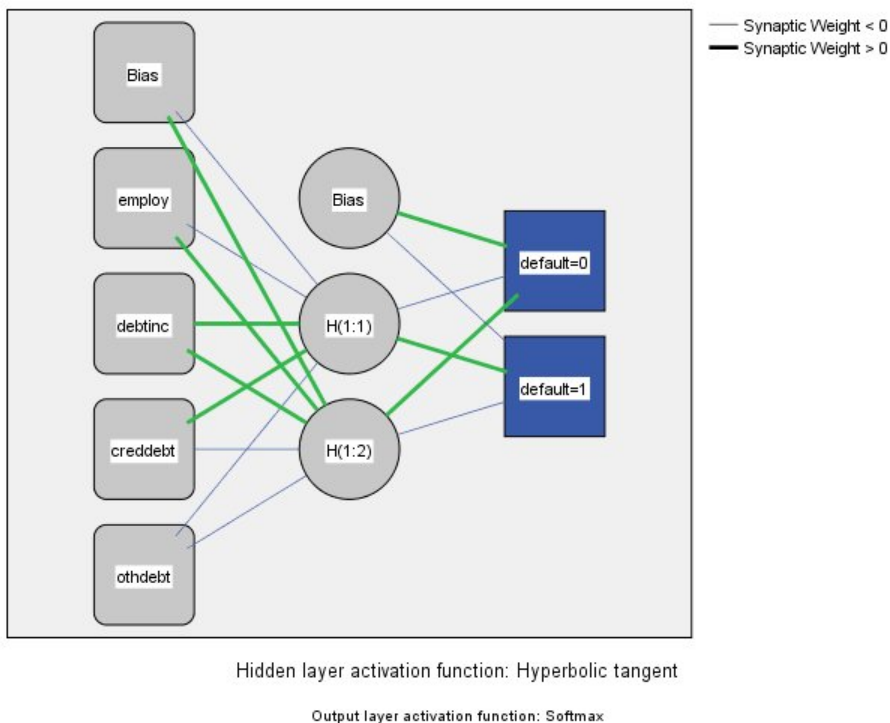


Figure 1. Feedforward architecture with one hidden layer

This structure is known as a **feedforward architecture** because the connections in the network flow forward from the input layer to the output layer without any feedback loops. In this figure:

- The **input layer** contains the predictors.
- The **hidden layer** contains unobservable nodes, or units. The value of each hidden unit is some function of the predictors; the exact form of the function depends in part upon the network type and in part upon user-controllable specifications.
- The **output layer** contains the responses. Since the history of default is a categorical variable with two categories, it is recoded as two indicator variables. Each output unit is some function of the hidden units. Again, the exact form of the function depends in part on the network type and in part on user-controllable specifications.

The MLP network allows a second hidden layer; in that case, each unit of the second hidden layer is a function of the units in the first hidden layer, and each response is a function of the units in the second hidden layer.

Multilayer Perceptron

The Multilayer Perceptron (MLP) procedure produces a predictive model for one or more dependent (target) variables based on the values of the predictor variables.

Examples. Following are two scenarios using the MLP procedure:

A loan officer at a bank needs to be able to identify characteristics that are indicative of people who are likely to default on loans and use those characteristics to identify good and bad credit risks. Using a sample of past customers, she can train a multilayer perceptron, validate the analysis using a holdout sample of past customers, and then use the network to classify prospective customers as good or bad credit risks.

A hospital system is interested in tracking costs and lengths of stay for patients admitted for treatment of myocardial infarction (MI, or "heart attack"). Obtaining accurate estimates of these measures allows the administration to properly manage the available bed space as patients are treated. Using the treatment records of a sample of patients who received treatment for MI, the administrator can train a network to predict both cost and length of stay.












Data Considerations

Dependent variables. The dependent variables can be:

- *Nominal.* A variable can be treated as nominal when its values represent categories with no intrinsic ranking (for example, the department of the company in which an employee works). Examples of nominal variables include region, postal code, and religious affiliation.
- *Ordinal.* A variable can be treated as ordinal when its values represent categories with some intrinsic ranking (for example, levels of service satisfaction from highly dissatisfied to highly satisfied). Examples of ordinal variables include attitude scores representing degree of satisfaction or confidence and preference rating scores.
- *Scale.* A variable can be treated as scale (continuous) when its values represent ordered categories with a meaningful metric, so that distance comparisons between values are appropriate. Examples of scale variables include age in years and income in thousands of dollars.

The procedure assumes that the appropriate measurement level has been assigned to all dependent variables; however, you can temporarily change the measurement level for a variable by right-clicking the variable in the source variable list and selecting a measurement level from the pop-up menu.

An icon next to each variable in the variable list identifies the measurement level and data type:

	Numeric	String	Date	Time
Scale (Continuous)		n/a		
Ordinal				
Nominal				

Predictor variables. Predictors can be specified as factors (categorical) or covariates (scale).

Categorical variable coding. The procedure temporarily recodes categorical predictors and dependent variables using one-of-c coding for the duration of the procedure. If there are c categories of a variable,

then the variable is stored as c vectors, with the first category denoted (1,0,...,0), the next category (0,1,0,...,0), ..., and the final category (0,0,...,0,1).

This coding scheme increases the number of synaptic weights and can result in slower training; however, more "compact" coding methods usually lead to poorly fit neural networks. If your network training is proceeding very slowly, try reducing the number of categories in your categorical predictors by combining similar categories or dropping cases that have extremely rare categories.

All one-of-c coding is based on the training data, even if a testing or holdout sample is defined (see "Partitions" on page 5). Thus, if the testing or holdout samples contain cases with predictor categories that are not present in the training data, then those cases are not used by the procedure or in scoring. If the testing or holdout samples contain cases with dependent variable categories that are not present in the training data, then those cases are not used by the procedure, but they may be scored.

Rescaling. Scale-dependent variables and covariates are rescaled by default to improve network training. All rescaling is performed based on the training data, even if a testing or holdout sample is defined (see "Partitions" on page 5). That is, depending on the type of rescaling, the mean, standard deviation, minimum value, or maximum value of a covariate or dependent variable is computed using only the training data. If you specify a variable to define partitions, it is important that these covariates or dependent variables have similar distributions across the training, testing, and holdout samples.

Frequency weights. Frequency weights are ignored by this procedure.

Replicating results. If you want to replicate your results exactly, use the same initialization value for the random number generator, the same data order, and the same variable order, in addition to using the same procedure settings. More details on this issue follow:

- **Random number generation.** The procedure uses random number generation during random assignment of partitions, random subsampling for initialization of synaptic weights, random subsampling for automatic architecture selection, and the simulated annealing algorithm used in weight initialization and automatic architecture selection. To reproduce the same randomized results in the future, use the same initialization value for the random number generator before each run of the Multilayer Perceptron procedure.
- **Case order.** The Online and Mini-batch training methods (see "Training" on page 7) are explicitly dependent upon case order; however, even Batch training is dependent upon case order because initialization of synaptic weights involves subsampling from the dataset.

To minimize order effects, randomly order the cases. To verify the stability of a given solution, you may want to obtain several different solutions with cases sorted in different random orders. In situations with extremely large file sizes, multiple runs can be performed with a sample of cases sorted in different random orders.

- **Variable order.** Results may be influenced by the order of variables in the factor and covariate lists due to the different pattern of initial values assigned when the variable order is changed. As with case order effects, you might try different variable orders (simply drag and drop within the factor and covariate lists) to assess the stability of a given solution.

Creating a Multilayer Perceptron Network

From the menus choose:

Analyze > Neural Networks > Multilayer Perceptron...

1. Select at least one dependent variable.
2. Select at least one factor or covariate.

Optionally, on the Variables tab you can change the method for rescaling covariates. The choices are:

- **Standardized.** Subtract the mean and divide by the standard deviation, $(x - \text{mean})/s$.
- **Normalized.** Subtract the minimum and divide by the range, $(x - \text{min})/(\text{max} - \text{min})$. Normalized values fall between 0 and 1.
- **Adjusted Normalized.** Adjusted version of subtracting the minimum and dividing by the range, $[2 * (x - \text{min}) / (\text{max} - \text{min})] - 1$. Adjusted normalized values fall between -1 and 1.

- **None.** No rescaling of covariates.

Fields with unknown measurement level

The Measurement Level alert is displayed when the measurement level for one or more variables (fields) in the dataset is unknown. Since measurement level affects the computation of results for this procedure, all variables must have a defined measurement level.

Scan Data. Reads the data in the active dataset and assigns default measurement level to any fields with a currently unknown measurement level. If the dataset is large, that may take some time.

Assign Manually. Opens a dialog that lists all fields with an unknown measurement level. You can use this dialog to assign measurement level to those fields. You can also assign measurement level in Variable View of the Data Editor.

Since measurement level is important for this procedure, you cannot access the dialog to run this procedure until all fields have a defined measurement level.

Partitions

Partition Dataset. This group specifies the method of partitioning the active dataset into training, testing, and holdout samples. The **training sample** comprises the data records used to train the neural network; some percentage of cases in the dataset must be assigned to the training sample in order to obtain a model. The **testing sample** is an independent set of data records used to track errors during training in order to prevent overtraining. It is highly recommended that you create a testing sample, and network training will generally be most efficient if the testing sample is smaller than the training sample. The **holdout sample** is another independent set of data records used to assess the final neural network; the error for the holdout sample gives an "honest" estimate of the predictive ability of the model because the holdout cases were not used to build the model.

- **Randomly assign cases based on relative number of cases.** Specify the relative number (ratio) of cases randomly assigned to each sample (training, testing, and holdout). The **%** column reports the percentage of cases that will be assigned to each sample based on the relative numbers you have specified.

For example, specifying 7, 3, 0 as the relative numbers for training, testing, and holdout samples corresponds to 70%, 30%, and 0%. Specifying 2, 1, 1 as the relative numbers corresponds to 50%, 25%, and 25%; 1, 1, 1 corresponds to dividing the dataset into equal thirds among training, testing, and holdout.

- **Use partitioning variable to assign cases.** Specify a numeric variable that assigns each case in the active dataset to the training, testing, or holdout sample. Cases with a positive value on the variable are assigned to the training sample, cases with a value of 0, to the testing sample, and cases with a negative value, to the holdout sample. Cases with a system-missing value are excluded from the analysis. Any user-missing values for the partition variable are always treated as valid.

Note: Using a partitioning variable will not guarantee identical results in successive runs of the procedure. See "Replicating results" in the main [Multilayer Perceptron](#) topic.

Architecture

The Architecture tab is used to specify the structure of the network. The procedure can select the "best" architecture automatically, or you can specify a custom architecture.

Automatic architecture selection builds a network with one hidden layer. Specify the minimum and maximum number of units allowed in the hidden layer, and the automatic architecture selection computes the "best" number of units in the hidden layer. Automatic architecture selection uses the default activation functions for the hidden and output layers.

Custom architecture selection gives you expert control over the hidden and output layers and can be most useful when you know in advance what architecture you want or when you need to tweak the results of the Automatic architecture selection.

Hidden Layers

The hidden layer contains unobservable network nodes (units). Each hidden unit is a function of the weighted sum of the inputs. The function is the activation function, and the values of the weights are determined by the estimation algorithm. If the network contains a second hidden layer, each hidden unit in the second layer is a function of the weighted sum of the units in the first hidden layer. The same activation function is used in both layers.

Number of Hidden Layers. A multilayer perceptron can have one or two hidden layers.

Activation Function. The activation function "links" the weighted sums of units in a layer to the values of units in the succeeding layer.

- **Hyperbolic tangent.** This function has the form: $\gamma(c) = \tanh(c) = (e^c - e^{-c}) / (e^c + e^{-c})$. It takes real-valued arguments and transforms them to the range $(-1, 1)$. When automatic architecture selection is used, this is the activation function for all units in the hidden layers.
- **Sigmoid.** This function has the form: $\gamma(c) = 1 / (1 + e^{-c})$. It takes real-valued arguments and transforms them to the range $(0, 1)$.

Number of Units. The number of units in each hidden layer can be specified explicitly or determined automatically by the estimation algorithm.

Output Layer

The output layer contains the target (dependent) variables.

Activation Function. The activation function "links" the weighted sums of units in a layer to the values of units in the succeeding layer.

- **Identity.** This function has the form: $\gamma(c) = c$. It takes real-valued arguments and returns them unchanged. When automatic architecture selection is used, this is the activation function for units in the output layer if there are any scale-dependent variables.
- **Softmax.** This function has the form: $\gamma(c_k) = \exp(c_k) / \sum_j \exp(c_j)$. It takes a vector of real-valued arguments and transforms it to a vector whose elements fall in the range $(0, 1)$ and sum to 1. Softmax is available only if all dependent variables are categorical. When automatic architecture selection is used, this is the activation function for units in the output layer if all dependent variables are categorical.
- **Hyperbolic tangent.** This function has the form: $\gamma(c) = \tanh(c) = (e^c - e^{-c}) / (e^c + e^{-c})$. It takes real-valued arguments and transforms them to the range $(-1, 1)$.
- **Sigmoid.** This function has the form: $\gamma(c) = 1 / (1 + e^{-c})$. It takes real-valued arguments and transforms them to the range $(0, 1)$.

Rescaling of Scale Dependent Variables. These controls are available only if at least one scale-dependent variable has been selected.

- **Standardized.** Subtract the mean and divide by the standard deviation, $(x - \text{mean}) / s$.
- **Normalized.** Subtract the minimum and divide by the range, $(x - \text{min}) / (\text{max} - \text{min})$. Normalized values fall between 0 and 1. This is the required rescaling method for scale-dependent variables if the output layer uses the sigmoid activation function. The correction option specifies a small number ϵ that is applied as a correction to the rescaling formula; this correction ensures that all rescaled dependent variable values will be within the range of the activation function. In particular, the values 0 and 1, which occur in the uncorrected formula when x takes its minimum and maximum value, define the limits of the range of the sigmoid function but are not within that range. The corrected formula is $[x - (\text{min} - \epsilon)] / [(\text{max} + \epsilon) - (\text{min} - \epsilon)]$. Specify a number greater than or equal to 0.
- **Adjusted Normalized.** Adjusted version of subtracting the minimum and dividing by the range, $[2 * (x - \text{min}) / (\text{max} - \text{min})] - 1$. Adjusted normalized values fall between -1 and 1 . This is the required rescaling method for scale-dependent variables if the output layer uses the hyperbolic tangent activation function. The correction option specifies a small number ϵ that is applied as a correction to the rescaling formula; this correction ensures that all rescaled dependent variable values will be within the range of the activation function. In particular, the values -1 and 1 , which occur in the uncorrected formula when x takes its minimum and maximum value, define the limits of the range of the hyperbolic tangent function but are not within that range. The corrected formula is $\{2 * [(x - (\text{min} - \epsilon))] / ((\text{max} + \epsilon) - (\text{min} - \epsilon))\} - 1$. Specify a number greater than or equal to 0.

- **None.** No rescaling of scale-dependent variables.

Training

The Training tab is used to specify how the network should be trained. The type of training and the optimization algorithm determine which training options are available.

Type of Training. The training type determines how the network processes the records. Select one of the following training types:

- **Batch.** Updates the synaptic weights only after passing all training data records; that is, batch training uses information from all records in the training dataset. Batch training is often preferred because it directly minimizes the total error; however, batch training may need to update the weights many times until one of the stopping rules is met and hence may need many data passes. It is most useful for "smaller" datasets.
- **Online.** Updates the synaptic weights after every single training data record; that is, online training uses information from one record at a time. Online training continuously gets a record and updates the weights until one of the stopping rules is met. If all the records are used once and none of the stopping rules is met, then the process continues by recycling the data records. Online training is superior to batch for "larger" datasets with associated predictors; that is, if there are many records and many inputs, and their values are not independent of each other, then online training can more quickly obtain a reasonable answer than batch training.
- **Mini-batch.** Divides the training data records into groups of approximately equal size, then updates the synaptic weights after passing one group; that is, mini-batch training uses information from a group of records. Then the process recycles the data group if necessary. Mini-batch training offers a compromise between batch and online training, and it may be best for "medium-size" datasets. The procedure can automatically determine the number of training records per mini-batch, or you can specify an integer greater than 1 and less than or equal to the maximum number of cases to store in memory. You can set the maximum number of cases to store in memory on the [Options](#) tab.

Optimization Algorithm. This is the method used to estimate the synaptic weights.

- **Scaled conjugate gradient.** The assumptions that justify the use of conjugate gradient methods apply only to batch training types, so this method is not available for online or mini-batch training.
- **Gradient descent.** This method must be used with online or mini-batch training; it can also be used with batch training.

Training Options. The training options allow you to fine-tune the optimization algorithm. You generally will not need to change these settings unless the network runs into problems with estimation.

Training options for the scaled conjugate gradient algorithm include:

- **Initial Lambda.** The initial value of the lambda parameter for the scaled conjugate gradient algorithm. Specify a number greater than 0 and less than 0.000001.
- **Initial Sigma.** The initial value of the sigma parameter for the scaled conjugate gradient algorithm. Specify a number greater than 0 and less than 0.0001.
- **Interval Center and Interval Offset.** The interval center (a_0) and interval offset (a) define the interval $[a_0 - a, a_0 + a]$, in which weight vectors are randomly generated when simulated annealing is used. Simulated annealing is used to break out of a local minimum, with the goal of finding the global minimum, during application of the optimization algorithm. This approach is used in weight initialization and automatic architecture selection. Specify a number for the interval center and a number greater than 0 for the interval offset.

Training options for the gradient descent algorithm include:

- **Initial Learning Rate.** The initial value of the learning rate for the gradient descent algorithm. A higher learning rate means that the network will train faster, possibly at the cost of becoming unstable. Specify a number greater than 0.

- **Lower Boundary of Learning Rate.** The lower boundary on the learning rate for the gradient descent algorithm. This setting applies only to online and mini-batch training. Specify a number greater than 0 and less than the initial learning rate.
- **Momentum.** The initial momentum parameter for the gradient descent algorithm. The momentum term helps to prevent instabilities caused by a too-high learning rate. Specify a number greater than 0.
- **Learning rate reduction, in Epochs.** The number of epochs (p), or data passes of the training sample, required to reduce the initial learning rate to the lower boundary of the learning rate when gradient descent is used with online or mini-batch training. This gives you control of the learning rate decay factor $\beta = (1/p K) * \ln(\eta_0/\eta_{low})$, where η_0 is the initial learning rate, η_{low} is the lower bound on the learning rate, and K is the total number of mini-batches (or the number of training records for online training) in the training dataset. Specify an integer greater than 0.

Output

Network Structure. Displays summary information about the neural network.

- **Description.** Displays information about the neural network, including the dependent variables, number of input and output units, number of hidden layers and units, and activation functions.
- **Diagram.** Displays the network diagram as a non-editable chart. Note that as the number of covariates and factor levels increases, the diagram becomes more difficult to interpret.
- **Synaptic weights.** Displays the coefficient estimates that show the relationship between the units in a given layer to the units in the following layer. The synaptic weights are based on the training sample even if the active dataset is partitioned into training, testing, and holdout data. Note that the number of synaptic weights can become rather large and that these weights are generally not used for interpreting network results.

Network Performance. Displays results used to determine whether the model is "good". *Note:* Charts in this group are based on the combined training and testing samples or only on the training sample if there is no testing sample.

- **Model summary.** Displays a summary of the neural network results by partition and overall, including the error, the relative error or percentage of incorrect predictions, the stopping rule used to stop training, and the training time.

The error is the sum-of-squares error when the identity, sigmoid, or hyperbolic tangent activation function is applied to the output layer. It is the cross-entropy error when the softmax activation function is applied to the output layer.

Relative errors or percentages of incorrect predictions are displayed depending on the dependent variable measurement levels. If any dependent variable has scale measurement level, then the average overall relative error (relative to the mean model) is displayed. If all dependent variables are categorical, then the average percentage of incorrect predictions is displayed. Relative errors or percentages of incorrect predictions are also displayed for individual dependent variables.

- **Classification results.** Displays a classification table for each categorical dependent variable by partition and overall. Each table gives the number of cases classified correctly and incorrectly for each dependent variable category. The percentage of the total cases that were correctly classified is also reported.
- **ROC curve.** Displays an ROC (Receiver Operating Characteristic) curve for each categorical dependent variable. It also displays a table giving the area under each curve. For a given dependent variable, the ROC chart displays one curve for each category. If the dependent variable has two categories, then each curve treats the category at issue as the positive state versus the other category. If the dependent variable has more than two categories, then each curve treats the category at issue as the positive state versus the aggregate of all other categories.
- **Cumulative gains chart.** Displays a cumulative gains chart for each categorical dependent variable. The display of one curve for each dependent variable category is the same as for ROC curves.
- **Lift chart.** Displays a lift chart for each categorical dependent variable. The display of one curve for each dependent variable category is the same as for ROC curves.

- **Predicted by observed chart.** Displays a predicted-by-observed-value chart for each dependent variable. For categorical dependent variables, clustered boxplots of predicted pseudo-probabilities are displayed for each response category, with the observed response category as the cluster variable. For scale-dependent variables, a scatterplot is displayed.
- **Residual by predicted chart.** Displays a residual-by-predicted-value chart for each scale-dependent variable. There should be no visible patterns between residuals and predicted values. This chart is produced only for scale-dependent variables.

Case processing summary. Displays the case processing summary table, which summarizes the number of cases included and excluded in the analysis, in total and by training, testing, and holdout samples.

Independent variable importance analysis. Performs a sensitivity analysis, which computes the importance of each predictor in determining the neural network. The analysis is based on the combined training and testing samples or only on the training sample if there is no testing sample. This creates a table and a chart displaying importance and normalized importance for each predictor. Note that sensitivity analysis is computationally expensive and time-consuming if there are large numbers of predictors or cases.

Save

The Save tab is used to save predictions as variables in the dataset.

- **Save predicted value or category for each dependent variable.** This saves the predicted value for scale-dependent variables and the predicted category for categorical dependent variables.
- **Save predicted pseudo-probability or category for each dependent variable.** This saves the predicted pseudo-probabilities for categorical dependent variables. A separate variable is saved for each of the first n categories, where n is specified in the **Categories to Save** column.

Names of Saved Variables. Automatic name generation ensures that you keep all of your work. Custom names allow you to discard/replace results from previous runs without first deleting the saved variables in the Data Editor.

Probabilities and Pseudo-Probabilities

Categorical dependent variables with softmax activation and cross-entropy error will have a predicted value for each category, where each predicted value is the probability that the case belongs to the category.

Categorical dependent variables with sum-of-squares error will have a predicted value for each category, but the predicted values cannot be interpreted as probabilities. The procedure saves these predicted pseudo-probabilities even if any are less than 0 or greater than 1, or the sum for a given dependent variable is not 1.

The ROC, cumulative gains, and lift charts (see “Output ” on page 8) are created based on pseudo-probabilities. In the event that any of the pseudo-probabilities are less than 0 or greater than 1, or the sum for a given variable is not 1, they are first rescaled to be between 0 and 1 and to sum to 1. Pseudo-probabilities are rescaled by dividing by their sum. For example, if a case has predicted pseudo-probabilities of 0.50, 0.60, and 0.40 for a three-category dependent variable, then each pseudo-probability is divided by the sum 1.50 to get 0.33, 0.40, and 0.27.

If any of the pseudo-probabilities are negative, then the absolute value of the lowest is added to all pseudo-probabilities before the above rescaling. For example, if the pseudo-probabilities are -0.30, 0.50, and 1.30, then first add 0.30 to each value to get 0.00, 0.80, and 1.60. Next, divide each new value by the sum 2.40 to get 0.00, 0.33, and 0.67.

Export

The Export tab is used to save the synaptic weight estimates for each dependent variable to an XML (PMML) file. You can use this model file to apply the model information to other data files for scoring purposes. This option is not available if split files have been defined.

Options

User-Missing Values. Factors must have valid values for a case to be included in the analysis. These controls allow you to decide whether user-missing values are treated as valid among factors and categorical dependent variables.

Stopping Rules. These are the rules that determine when to stop training the neural network. Training proceeds through at least one data pass. Training can then be stopped according to the following criteria, which are checked in the listed order. In the stopping rule definitions that follow, a step corresponds to a data pass for the online and mini-batch methods and an iteration for the batch method.

- **Maximum steps without a decrease in error.** The number of steps to allow before checking for a decrease in error. If there is no decrease in error after the specified number of steps, then training stops. Specify an integer greater than 0. You can also specify which data sample is used to compute the error. **Choose automatically** uses the testing sample if it exists and uses the training sample otherwise. Note that batch training guarantees a decrease in the training sample error after each data pass; thus, this option applies only to batch training if a testing sample exists. **Both training and test data** checks the error for each of these samples; this option applies only if a testing sample exists.

Note: After each complete data pass, online and mini-batch training require an extra data pass in order to compute the training error. This extra data pass can slow training considerably, so it is generally recommended that you supply a testing sample and select **Choose automatically** in any case.

- **Maximum training time.** Choose whether to specify a maximum number of minutes for the algorithm to run. Specify a number greater than 0.
- **Maximum Training Epochs.** The maximum number of epochs (data passes) allowed. If the maximum number of epochs is exceeded, then training stops. Specify an integer greater than 0.
- **Minimum relative change in training error.** Training stops if the relative change in the training error compared to the previous step is less than the criterion value. Specify a number greater than 0. For online and mini-batch training, this criterion is ignored if only testing data is used to compute the error.
- **Minimum relative change in training error ratio.** Training stops if the ratio of the training error to the error of the null model is less than the criterion value. The null model predicts the average value for all dependent variables. Specify a number greater than 0. For online and mini-batch training, this criterion is ignored if only testing data is used to compute the error.

Maximum cases to store in memory. This controls the following settings within the multilayer perceptron algorithms. Specify an integer greater than 1.

- In automatic architecture selection, the size of the sample used to determine the network architecture is $\min(1000, memsize)$, where *memsize* is the maximum number of cases to store in memory.
- In mini-batch training with automatic computation of the number of mini-batches, the number of mini-batches is $\min(\max(M/10, 2), memsize)$, where *M* is the number of cases in the training sample.

Radial Basis Function

The Radial Basis Function (RBF) procedure produces a predictive model for one or more dependent (target) variables based on values of predictor variables.

Example. A telecommunications provider has segmented its customer base by service usage patterns, categorizing the customers into four groups. An RBF network using demographic data to predict group membership allows the company to customize offers for individual prospective customers.

Data Considerations

Dependent variables. The dependent variables can be:












- *Nominal.* A variable can be treated as nominal when its values represent categories with no intrinsic ranking (for example, the department of the company in which an employee works). Examples of nominal variables include region, postal code, and religious affiliation.
- *Ordinal.* A variable can be treated as ordinal when its values represent categories with some intrinsic ranking (for example, levels of service satisfaction from highly dissatisfied to highly satisfied). Examples

of ordinal variables include attitude scores representing degree of satisfaction or confidence and preference rating scores.

- **Scale.** A variable can be treated as scale (continuous) when its values represent ordered categories with a meaningful metric, so that distance comparisons between values are appropriate. Examples of scale variables include age in years and income in thousands of dollars.

The procedure assumes that the appropriate measurement level has been assigned to all dependent variables, although you can temporarily change the measurement level for a variable by right-clicking the variable in the source variable list and selecting a measurement level from the pop-up menu.

An icon next to each variable in the variable list identifies the measurement level and data type:

	Numeric	String	Date	Time
Scale (Continuous)		n/a		
Ordinal				
Nominal				

Predictor variables. Predictors can be specified as factors (categorical) or covariates (scale).

Categorical variable coding. The procedure temporarily recodes categorical predictors and dependent variables using one-of-c coding for the duration of the procedure. If there are c categories of a variable, then the variable is stored as c vectors, with the first category denoted $(1,0,\dots,0)$, the next category $(0,1,0,\dots,0)$, ..., and the final category $(0,0,\dots,0,1)$.

This coding scheme increases the number of synaptic weights and can result in slower training, but more "compact" coding methods usually lead to poorly fit neural networks. If your network training is proceeding very slowly, try reducing the number of categories in your categorical predictors by combining similar categories or dropping cases that have extremely rare categories.

All one-of-c coding is based on the training data, even if a testing or holdout sample is defined (see "Partitions" on page 12). Thus, if the testing or holdout samples contain cases with predictor categories that are not present in the training data, then those cases are not used by the procedure or in scoring. If the testing or holdout samples contain cases with dependent variable categories that are not present in the training data, then those cases are not used by the procedure but they may be scored.

Rescaling. Scale dependent variables and covariates are rescaled by default to improve network training. All rescaling is performed based on the training data, even if a testing or holdout sample is defined (see "Partitions" on page 12). That is, depending on the type of rescaling, the mean, standard deviation, minimum value, or maximum value of a covariate or dependent variable are computed using only the training data. If you specify a variable to define partitions, it is important that these covariates or dependent variables have similar distributions across the training, testing, and holdout samples.

Frequency weights. Frequency weights are ignored by this procedure.

Replicating results. If you want to exactly replicate your results, use the same initialization value for the random number generator and the same data order, in addition to using the same procedure settings. More details on this issue follow:

- **Random number generation.** The procedure uses random number generation during random assignment of partitions. To reproduce the same randomized results in the future, use the same initialization value for the random number generator before each run of the Radial Basis Function procedure.
- **Case order.** Results are also dependent on data order because the two-step cluster algorithm is used to determine the radial basis functions.

To minimize order effects, randomly order the cases. To verify the stability of a given solution, you may want to obtain several different solutions with cases sorted in different random orders. In situations with extremely large file sizes, multiple runs can be performed with a sample of cases sorted in different random orders.

Creating a Radial Basis Function Network

From the menus choose:

Analyze > Neural Networks > Radial Basis Function...

1. Select at least one dependent variable.
2. Select at least one factor or covariate.

Optionally, on the Variables tab you can change the method for rescaling covariates. The choices are:

- **Standardized.** Subtract the mean and divide by the standard deviation, $(x - \text{mean})/s$.
- **Normalized.** Subtract the minimum and divide by the range, $(x - \text{min})/(\text{max} - \text{min})$. Normalized values fall between 0 and 1.
- **Adjusted Normalized.** Adjusted version of subtracting the minimum and dividing by the range, $[2 * (x - \text{min})/(\text{max} - \text{min})] - 1$. Adjusted normalized values fall between -1 and 1.
- **None.** No rescaling of covariates.

Fields with unknown measurement level

The Measurement Level alert is displayed when the measurement level for one or more variables (fields) in the dataset is unknown. Since measurement level affects the computation of results for this procedure, all variables must have a defined measurement level.

Scan Data. Reads the data in the active dataset and assigns default measurement level to any fields with a currently unknown measurement level. If the dataset is large, that may take some time.

Assign Manually. Opens a dialog that lists all fields with an unknown measurement level. You can use this dialog to assign measurement level to those fields. You can also assign measurement level in Variable View of the Data Editor.

Since measurement level is important for this procedure, you cannot access the dialog to run this procedure until all fields have a defined measurement level.

Partitions

Partition Dataset. This group specifies the method of partitioning the active dataset into training, testing, and holdout samples. The **training sample** comprises the data records used to train the neural network; some percentage of cases in the dataset must be assigned to the training sample in order to obtain a model. The **testing sample** is an independent set of data records used to track errors during training in order to prevent overtraining. It is highly recommended that you create a testing sample, and network training will generally be most efficient if the testing sample is smaller than the training sample. The **holdout sample** is another independent set of data records used to assess the final neural network; the error for the holdout sample gives an "honest" estimate of the predictive ability of the model because the holdout cases were not used to build the model.

- **Randomly assign cases based on relative number of cases.** Specify the relative number (ratio) of cases randomly assigned to each sample (training, testing, and holdout). The **%** column reports the percentage of cases that will be assigned to each sample based on the relative numbers you have specified.

For example, specifying 7, 3, 0 as the relative numbers for training, testing, and holdout samples corresponds to 70%, 30%, and 0%. Specifying 2, 1, 1 as the relative numbers corresponds to 50%, 25%, and 25%; 1, 1, 1 corresponds to dividing the dataset into equal thirds among training, testing, and holdout.

- **Use partitioning variable to assign cases.** Specify a numeric variable that assigns each case in the active dataset to the training, testing, or holdout sample. Cases with a positive value on the variable are

assigned to the training sample, cases with a value of 0, to the testing sample, and cases with a negative value, to the holdout sample. Cases with a system-missing value are excluded from the analysis. Any user-missing values for the partition variable are always treated as valid.

Architecture

The Architecture tab is used to specify the structure of the network. The procedure creates a neural network with one hidden "radial basis function" layer; in general, it will not be necessary to change these settings.

Number of Units in Hidden Layer. There are three ways of choosing the number of hidden units.

1. **Find the best number of units within an automatically computed range.** The procedure automatically computes the minimum and maximum values of the range and finds the best number of hidden units within the range.

If a testing sample is defined, then the procedure uses the testing data criterion: The best number of hidden units is the one that yields the smallest error in the testing data. If a testing sample is not defined, then the procedure uses the Bayesian information criterion (BIC): The best number of hidden units is the one that yields the smallest BIC based on the training data.

2. **Find the best number of units within a specified range.** You can provide your own range, and the procedure will find the "best" number of hidden units within that range. As before, the best number of hidden units from the range is determined using the testing data criterion or the BIC.
3. **Use a specified number of units.** You can override the use of a range and specify a particular number of units directly.

Activation Function for Hidden Layer. The activation function for the hidden layer is the radial basis function, which "links" the units in a layer to the values of units in the succeeding layer. For the output layer, the activation function is the identity function; thus, the output units are simply weighted sums of the hidden units.

- **Normalized radial basis function.** Uses the softmax activation function so the activations of all hidden units are normalized to sum to 1.
- **Ordinary radial basis function.** Uses the exponential activation function so the activation of the hidden unit is a Gaussian "bump" as a function of the inputs.

Overlap Among Hidden Units. The overlapping factor is a multiplier applied to the width of the radial basis functions. The automatically computed value of the overlapping factor is $1+0.1d$, where d is the number of input units (the sum of the number of categories across all factors and the number of covariates).

Output

Network Structure. Displays summary information about the neural network.

- **Description.** Displays information about the neural network, including the dependent variables, number of input and output units, number of hidden layers and units, and activation functions.
- **Diagram.** Displays the network diagram as a non-editable chart. Note that as the number of covariates and factor levels increases, the diagram becomes more difficult to interpret.
- **Synaptic weights.** Displays the coefficient estimates that show the relationship between the units in a given layer to the units in the following layer. The synaptic weights are based on the training sample even if the active dataset is partitioned into training, testing, and holdout data. Note that the number of synaptic weights can become rather large, and these weights are generally not used for interpreting network results.

Network Performance. Displays results used to determine whether the model is "good." *Note:* Charts in this group are based on the combined training and testing samples or only the training sample if there is no testing sample.

- **Model summary.** Displays a summary of the neural network results by partition and overall, including the error, the relative error or percentage of incorrect predictions, and the training time.

The error is the sum-of-squares error. In addition, relative errors or percentages of incorrect predictions are displayed, depending on the dependent variable measurement levels. If any dependent variable has scale measurement level, then the average overall relative error (relative to the mean model) is displayed. If all dependent variables are categorical, then the average percentage of incorrect predictions is displayed. Relative errors or percentages of incorrect predictions are also displayed for individual dependent variables.

- **Classification results.** Displays a classification table for each categorical dependent variable. Each table gives the number of cases classified correctly and incorrectly for each dependent variable category. The percentage of the total cases that were correctly classified is also reported.
- **ROC curve.** Displays an ROC (Receiver Operating Characteristic) curve for each categorical dependent variable. It also displays a table giving the area under each curve. For a given dependent variable, the ROC chart displays one curve for each category. If the dependent variable has two categories, then each curve treats the category at issue as the positive state versus the other category. If the dependent variable has more than two categories, then each curve treats the category at issue as the positive state versus the aggregate of all other categories.
- **Cumulative gains chart.** Displays a cumulative gains chart for each categorical dependent variable. The display of one curve for each dependent variable category is the same as for ROC curves.
- **Lift chart.** Displays a lift chart for each categorical dependent variable. The display of one curve for each dependent variable category is the same as for ROC curves.
- **Predicted by observed chart.** Displays a predicted-by-observed-value chart for each dependent variable. For categorical dependent variables, clustered boxplots of predicted pseudo-probabilities are displayed for each response category, with the observed response category as the cluster variable. For scale dependent variables, a scatterplot is displayed.
- **Residual by predicted chart.** Displays a residual-by-predicted-value chart for each scale dependent variable. There should be no visible patterns between residuals and predicted values. This chart is produced only for scale dependent variables.

Case processing summary. Displays the case processing summary table, which summarizes the number of cases included and excluded in the analysis, in total and by training, testing, and holdout samples.

Independent variable importance analysis. Performs a sensitivity analysis, which computes the importance of each predictor in determining the neural network. The analysis is based on the combined training and testing samples or only the training sample if there is no testing sample. This creates a table and a chart displaying importance and normalized importance for each predictor. Note that sensitivity analysis is computationally expensive and time-consuming if there is a large number of predictors or cases.

Save

The Save tab is used to save predictions as variables in the dataset.

- **Save predicted value or category for each dependent variable.** This saves the predicted value for scale dependent variables and the predicted category for categorical dependent variables.
- **Save predicted pseudo-probability for each dependent variable.** This saves the predicted pseudo-probabilities for categorical dependent variables. A separate variable is saved for each of the first n categories, where n is specified in the *Categories to Save* column.

Names of Saved Variables. Automatic name generation ensures that you keep all of your work. Custom names allow you to discard or replace results from previous runs without first deleting the saved variables in the Data Editor.

Probabilities and Pseudo-Probabilities

Predicted pseudo-probabilities cannot be interpreted as probabilities because the Radial Basis Function procedure uses the sum-of-squares error and identity activation function for the output layer. The procedure saves these predicted pseudo-probabilities even if any are less than 0 or greater than 1 or the sum for a given dependent variable is not 1.

The ROC, cumulative gains, and lift charts (see “Output ” on page 13) are created based on pseudo-probabilities. In the event that any of the pseudo-probabilities are less than 0 or greater than 1 or the sum for a given variable is not 1, they are first rescaled to be between 0 and 1 and to sum to 1. Pseudo-probabilities are rescaled by dividing by their sum. For example, if a case has predicted pseudo-probabilities of 0.50, 0.60, and 0.40 for a three-category dependent variable, then each pseudo-probability is divided by the sum 1.50 to get 0.33, 0.40, and 0.27.

If any of the pseudo-probabilities are negative, then the absolute value of the lowest is added to all pseudo-probabilities before the above rescaling. For example, if the pseudo-probabilities are -0.30 , $.50$, and 1.30 , then first add 0.30 to each value to get 0.00 , 0.80 , and 1.60 . Next, divide each new value by the sum 2.40 to get 0.00 , 0.33 , and 0.67 .

Export

The Export tab is used to save the synaptic weight estimates for each dependent variable to an XML (PMML) file. You can use this model file to apply the model information to other data files for scoring purposes. This option is not available if split files have been defined.

Options

User-Missing Values. Factors must have valid values for a case to be included in the analysis. These controls allow you to decide whether user-missing values are treated as valid among factors and categorical dependent variables.

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US*

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US*

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© Copyright IBM Corp. 2021. Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. 1989 - 2021. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Index

A

- activation function
 - in Multilayer Perceptron [5](#)
 - in Radial Basis Function [13](#)
- architecture
 - neural networks [2](#)

B

- batch training
 - in Multilayer Perceptron [7](#)

G

- gains chart
 - in Multilayer Perceptron [8](#)
 - in Radial Basis Function [13](#)

H

- hidden layer
 - in Multilayer Perceptron [5](#)
 - in Radial Basis Function [13](#)
- holdout sample
 - in Multilayer Perceptron [5](#)
 - in Radial Basis Function [12](#)

L

- lift chart
 - in Multilayer Perceptron [8](#)
 - in Radial Basis Function [13](#)

M

- mini-batch training
 - in Multilayer Perceptron [7](#)
- missing values
 - in Multilayer Perceptron [10](#)
- Multilayer Perceptron
 - model export [9](#)
 - network architecture [5](#)
 - options [10](#)
 - output [8](#)
 - partitions [5](#)
 - save variables to active dataset [9](#)
 - training [7](#)

N

- network architecture
 - in Multilayer Perceptron [5](#)
 - in Radial Basis Function [13](#)
- network diagram
 - in Multilayer Perceptron [8](#)

- network diagram (*continued*)
 - in Radial Basis Function [13](#)
- network training
 - in Multilayer Perceptron [7](#)
- neural networks
 - architecture [2](#)

O

- online training
 - in Multilayer Perceptron [7](#)
- output layer
 - in Multilayer Perceptron [5](#)
 - in Radial Basis Function [13](#)

R

- Radial Basis Function
 - model export [15](#)
 - network architecture [13](#)
 - options [15](#)
 - output [13](#)
 - partitions [12](#)
 - save variables to active dataset [14](#)
- ROC curve
 - in Multilayer Perceptron [8](#)
 - in Radial Basis Function [13](#)

S

- stopping rules
 - in Multilayer Perceptron [10](#)

T

- testing sample
 - in Multilayer Perceptron [5](#)
 - in Radial Basis Function [12](#)
- training sample
 - in Multilayer Perceptron [5](#)
 - in Radial Basis Function [12](#)

