TreeNet™

An exclusive implementation of Jerome Friedman’s MART methodology

Robust Multi-Tree Technology for Data Mining, Predictive Modeling and Data Processing

Copyright Salford Systems 2001 – 2005
TreeNet™ is based on MART™, licensed exclusively to Salford Systems

TreeNet™ is a trademark of Salford Systems
MART™ is a trademark of Jerll, Inc

4740 Murphy Canyon Road
San Diego, California 92123, USA
619.543.8880 TEL
619.543.8888 FAX
www.salford-systems.com

Developers of CART®, MARS®, PRIM™, RandomForests™, HotSpotDetector™ and other award-winning data mining software tools
Copyright

Copyright 2001-2005, Salford Systems; all rights reserved worldwide. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise without the express written permission of Salford Systems.

Limited Warranty

Salford Systems warrants for a period of ninety (90) days from the date of delivery that, under normal use, and without unauthorized modification, the program substantially conforms to the accompanying specifications and any Salford Systems authorized advertising material; that, under normal use, the magnetic media upon which this program is recorded will not be defective; and that the user documentation is substantially complete and contains the information Salford Systems deems necessary to use the program.

If, during the ninety (90) day period, a demonstrable defect in the program’s magnetic media or documentation should appear, you may return the software to Salford Systems for repair or replacement, at Salford Systems option. If Salford Systems cannot repair the defect or replace the software with functionally equivalent software within sixty (60) days of Salford Systems receipt of the defective software, then you shall be entitled to a full refund of the license fee. Salford Systems cannot and does not warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free. Salford Systems disclaims any and all liability for special, incidental, or consequential damages, including loss of profit, arising out of or with respect to the use, operation, or support of this program, even if Salford Systems has been apprised of the possibility of such damages.

Trademarks

TreeNet™ is a trademark of Salford Systems. MART™ is a trademark of JerIl, Inc and is exclusively licensed to Salford Systems. CART® is a registered trademark of California Statistical Software, Inc., and is exclusively licensed to Salford Systems. DBMS/COPY™ is a trademark of Conceptual Software. All other trademarks and registered trademarks mentioned herein are the property of their respective owners.
# Treenet Features and Options

- Case Weights ........................................................................................................ 111
- Setting Reporting, Random Number and Directory Options .................................. 111
- Controlling Report Contents ..................................................................................... 112
- Controlling Random-Number Seed Values ................................................................. 112
- Treenet Options .......................................................................................................... 113
- Directory Preferences ................................................................................................ 114
- Saving, Opening, and Viewing Grove Files ................................................................. 115
- Saving and Printing Classic Text Output ................................................................. 116
- Report Writer .............................................................................................................. 117
- Default Options ........................................................................................................... 117
- Pre-configured ............................................................................................................. 118
- Target Class ................................................................................................................ 118
- Printing and Saving Reports ......................................................................................... 119
- Data Information .......................................................................................................... 119
- Data Viewer ................................................................................................................ 120
- Online Help .................................................................................................................. 121

# Command Reference

- Category .................................................................................................................... 123
- Cdf .............................................................................................................................. 124
- Charset ....................................................................................................................... 125
- Class ........................................................................................................................... 126
- Cw ............................................................................................................................... 128
- Datainfo ...................................................................................................................... 129
- Descriptive .................................................................................................................. 130
- Discrete ....................................................................................................................... 131
- Error ............................................................................................................................. 132
- Exclude ......................................................................................................................... 133
- Format .......................................................................................................................... 134
- Grove ........................................................................................................................... 135
- Help .............................................................................................................................. 136
- Histogram ..................................................................................................................... 137
- Idvar .............................................................................................................................. 138
- Keep ............................................................................................................................. 139
- Limit ............................................................................................................................. 140
- Loptions ....................................................................................................................... 141
- Mart ............................................................................................................................... 142
- Misclass ......................................................................................................................... 146
- Model ............................................................................................................................ 147
- Names ........................................................................................................................... 148
- New ............................................................................................................................... 149
- Note ............................................................................................................................... 150
- Options ......................................................................................................................... 151
- Output ........................................................................................................................... 152
- Quit ............................................................................................................................... 153
- Rem ............................................................................................................................... 154
- Save ............................................................................................................................... 155
- Seed ............................................................................................................................... 156
- Select ............................................................................................................................ 157
- Submit ........................................................................................................................... 157
- Translate ....................................................................................................................... 159
- Use ............................................................................................................................... 160
- Weight .......................................................................................................................... 161
- Xplot .............................................................................................................................. 162

# Basic Programming Language

- Basic Programming Language ...................................................................................... 164
- Getting Started with Basic Programming Language ................................................... 164
- Basic: Overview of Basic Components ........................................................................ 165
Introducing TreeNet

This section provides a brief introduction to TreeNet and answers some frequently asked questions.
Preface

TreeNet is a revolutionary advance in data mining technology developed by Jerome Friedman, one of the world’s outstanding data mining researchers. TreeNet offers exceptional accuracy, blazing speed, and a high degree of fault tolerance for dirty and incomplete data. It can handle both classification and regression problems and has been proven to be remarkably effective in traditional numeric data mining and text mining. (See our section on text mining for further details.) This preface introduces TreeNet and answers some basic questions about the methodology.

How does TreeNet work and what does a TreeNet model look like?

A TreeNet model normally consists of from several hundred to several thousand small trees, each typically containing about six terminal nodes. Each tree is devoted to contributing a small portion of the overall model and the final model prediction is constructed by adding up all the individual tree contributions. You can think of the TreeNet model as a black box that has been proven to deliver exceptionally accurate models. TreeNet offers detailed self-testing to document its reliability on your own data. In many ways the TreeNet model is not that mysterious, although it is undeniably complex. You do not need to concern yourself with that complexity because all results, performance measures, and explanatory graphs can be grasped by anyone familiar with basic data mining principles. However, for those wishing to better understand the details of TreeNet model construction, we provide an outline here.

The model is similar in spirit to a long series expansion (such as a Fourier or Taylor’s series) - a sum of factors that becomes progressively more accurate as the expansion continues. The expansion can be written as:

\[ F(X) = F_0 + \beta_1 T_1(X) + \beta_2 T_2(X) + \ldots + \beta_M T_M(X) \]

where each \( T_i \) is a small tree. You should read this as a weighted sum of terms, each of which is obtained from the appropriate terminal node of a small tree.

As an example we display the first few terms from a regression model based on a well-known data set; the Boston Housing data extracted from the 1970 US Census. This data set is usually used to build models predicting the median value of houses in a neighborhood based on quality of life variables (such as crime rate, school quality, and socioeconomic status of the neighborhood) and a few core housing descriptors such as typical house size and age.

```
Response = 22.533

Tree 1
+13.541, if RM>6.835
-2.812, if otherwise

Tree 2
+2.607, if LSTAT<14.76
-5.291, if otherwise
```
The model begins with an estimate of mean home value (in 1970) of $22,533 and uses this as a baseline from which adjustments will be made to reflect characteristics of the housing and the neighborhood. In the first term the model states that the mean value would be adjusted upwards by $13,541 for larger homes, and adjusted upwards again by $2,607 for neighborhoods with good socioeconomic status indicators. In practice, the adjustments are usually much smaller than shown in this regression example and hundreds of adjustments may be needed. The final model is thus a collection of weighted and summed trees.

Although this example is a regression, exactly the same scheme is used for classification problems. For binary classification problems, a yes or no response is determined by whether the sign of the predicted outcome is positive or negative. For multi-class problems, a score is developed separately for each class via class-specific expansions, and the scores are converted into a set of probabilities of class membership.

The Boston housing regression example above uses the smallest possible two-node tree in each stage. More complicated models tracking complex interactions are possible with three or more nodes at each stage. The TreeNet default uses a six-node tree, but the optimal tree size for any specific modeling task can only be determined via trial and error. We have worked on problems that are handled perfectly well with two-node trees, and one of our award-winning data mining models use nine-node trees. Fortunately, TreeNet models run very quickly, so it is easy to experiment with a few different-sized trees to determine which will work best for your problem.
What are the advantages of TreeNet?

In our experience TreeNet is the closest tool we have ever encountered to a fully automated statistician. TreeNet can deal with substantial data quality issues, decide how a model should be constructed, select variables, detect interactions, address missing values, ignore suspicious data values, prevent the model from being dominated by just a few ultra-powerful variables, and resist overfitting. Typically, the end result is a model far more accurate than any that could be constructed using other data mining tools and at least as accurate as models painstakingly built by hand by experts working for weeks or months. Of course there is no substitute for a good understanding of the subject matter and problems being addressed and a reasonable familiarity with the data is always required for reliable results. However, given that, TreeNet can give you a substantial advantage in reaching high performance models quickly. Here is a summary of TreeNet's key features:

- Automatic selection from thousands of candidate predictors
  - No prior variable selection or data reduction is required
- Ability to handle data without preprocessing
  - Data do not need to be rescaled, transformed, or modified in any way
  - Resistance to outliers in predictors or the target variable
  - Automatic handling of missing values
  - General robustness to dirty and partially inaccurate data
- High Speed:
  - Trees are grown quickly; small trees are grown extraordinarily quickly
  - TreeNet is able to focus on the data that are not easily predictable as the model evolves
    - Thus, as additional trees are grown, less and less data needs to be processed
    - TreeNet is frequently able to focus much of its training on just 20% of the available data (all accomplished automatically without user intervention)
- Resistance to OverTraining
- When working with large data bases even models with 2,000 trees show little evidence of overtraining

TreeNet’s robustness extends to the most serious of all data errors: when the target variable itself is sometimes incorrect, for example, when a “yes” is incorrectly recorded as a “no,” or vice versa. In the machine learning world such data are said to be contaminated with erroneous target labels. For example, in medicine there is some risk that patients labeled as healthy are in fact ill and vice versa. This type of data error can be very challenging for conventional data mining methods and can be catastrophic for conventional forms of “boosting.” In contrast, TreeNet is generally immune to such errors as it dynamically rejects training data points too much at variance with the existing model.

In addition, TreeNet adds the advantage of a degree of accuracy usually not attainable by a single model or by ensembles such as bagging or conventional boosting. Independent real world tests in text mining, fraud detection, and credit worthiness have shown TreeNet to be dramatically more accurate on test data than other competing methods.
Of course no one method can be best for all problems in all contexts. Typically, if TreeNet is not well suited for a problem it will yield accuracies on par with that achievable with a single CART tree.

What are the advantages of TreeNet over a neural net?
Several of our most avid TreeNet users are former neural network advocates who have discovered how much faster and easier it is to get their modeling done with TreeNet while sacrificing nothing in the area of accuracy. In contrast to neural networks, TreeNet is not overly sensitive to data errors and needs no time-consuming data preparation, preprocessing or imputation of missing values. TreeNet is especially adept in dealing with errors in the target variable, a type of data error that could be catastrophic for a neural net. TreeNet is resistant to overtraining and can be 10 to 100 times faster than a neural net. Finally, TreeNet is not troubled by hundreds or thousands of predictors.

Can a neural net do anything a TreeNet cannot?
Yes. Version 2.0 of TreeNet cannot accept more than one target variable at a time. To model a collection of targets, a separate TreeNet model must be developed for each target independently. Also, neural nets can simultaneously estimate a function and its derivatives whereas TreeNet is not designed to estimate function derivatives.

What is the technology underlying TreeNet and how does it differ from boosting?
TreeNet is a proprietary methodology invented by Jerome Friedman known as “stochastic gradient boosting” and the trade secrets of the technology are embedded exclusively in TreeNet. Others may eventually try to develop technology based on the public descriptions offered by Professor Friedman, but hundreds of technical details remain exclusive to Salford Systems.

What is the TreeNet track record?
TreeNet was developed in 1997 by Stanford’s Jerome Friedman, a co-author of CART®, the author of MARS™ and PRIM, and the inventor of Projection Pursuit regression. TreeNet technology has been tested in a broad range of industrial and research settings and has demonstrated considerable benefits. In tests in which TreeNet was pitted against expert modeling teams using a variety of standard data mining tools, TreeNet was able to deliver results within a few hours comparable to or better than results that required months of hands-on development by expert data mining teams.

Recently TreeNet was designated “Most Accurate” in the KDDCup 2004 data mining competition (sponsored by the Association for Computing Machinery’s data mining SIG). TreeNet also won first place in all four competitive categories in the 2002 Duke University/NCR Teradata Churn modeling competition.

How does TreeNet fit into the Salford Systems data mining solution?
The Salford Systems data mining solution rests on two groups of technologies: CART, MARS and PRIM for accurate easy-to-understand models and TreeNet and
RandomForests for ultra-high performing, but potentially very complex, models interpreted via supporting graphical displays. Even in circumstances where interpretability and transparency are mandatory and a model must be expressible in the form of rules, TreeNet can serve a useful function by benchmarking the maximum achievable accuracy against which interpretable models can be compared.

**Typographical Conventions**

The following typographical conventions help you relate written material to information you see on your screen:

- Reference to menu names (File menu) or menu items (Close command) appear in the **bold** font.

- When you are asked to choose an item from the main menu, this is written as “**File–Close**”, meaning go to the **File** menu and choose the **Close** menu item. There are also scenarios where a menu item has multiple sub-menus. In these cases we write “**File–Open>Data File…**”, meaning go to the **File** menu, select **Open**, and choose the **Data File…** sub-menu.

- Likewise, items on popup menus and in dialog windows are shown in the **bold** font. In addition, you will be given more detailed instructions on where to find the corresponding item.

- References to variable names are shown in a monospaced font and are always displayed in upper case (for example; **VAR_A, VAR_B, VAR_C**).

- There are several locations in this documentation where you may read “see [1] for more technical discussion,” or something similar. This indication refers to bibliography [1] listed at the end of this manual.


This technical report contains a detailed technical discussion of TreeNet and can be found on our web site at the following address:


- Warning notes or “things to know” are denoted by a ⚡ sign.

- Additional notes or comments are denoted by a ➔ sign.
Getting Started

This chapter provides a brief introduction on how to install TreeNet and set up your first TreeNet run.
Installation Instructions
This chapter provides instructions for installing and starting TreeNet for Windows 2000, Windows XP, and later versions of Windows. We also provide instruction for our UNIX and Linux command line versions.

Although TreeNet can be run on other earlier versions of Windows we do not recommend or support such installations. You will see much better performance and support for our graphics with the newer operating systems.

Minimum System Requirements
To install and run TreeNet for Windows, the minimum hardware you need includes:

- Pentium processor or higher.
- 128 MB of random-access memory (RAM). We highly recommend no less than 512 MB to reduce the use of virtual memory.
- Hard disk with 40 MB of free space for program files, data file access utility, and sample data files.
- Additional hard disk space for scratch files (with the required space contingent on the size of the input data set).
- CD-ROM or DVD drive.
- Windows 2000, Windows XP, or later versions of Windows

Recommended System Configuration
Because TreeNet is extremely CPU intensive, the faster your CPU, the faster TreeNet will run. For optimal performance, we strongly recommend that TreeNet run on a machine with a system configuration equal to, or greater than, the following.

- Pentium IV processor running 1.0+ GHz or similar AMD chip
- 512 MB of DDR SDRAM
- 2 GIG of additional hard disk space available for virtual memory and temporary files
- Windows 2000 or Windows XP
- CD-ROM or DVD drive
- Hard disk with 40 MB of free space for program files, data file access utility, and sample data files
Installation Procedure for Windows

To install TreeNet:

1. Insert the CD labeled TreeNet 2.0 into your CD-ROM drive. If Autorun is enabled on your system, the installation starts automatically and you can skip steps 2 and 3.
2. From the start menu, select Run.
3. In the Run dialog box, type D:\SETUP (substitute the appropriate drive letter of your CD-ROM if other than D).
4. Follow the instructions on the screen.

The installation program prompts you to select a type of setup:

- **Typical**: The Typical installation provides you with all application software, tools, documentation, and sample data files that are normally available. All components will be installed within the directory structure defined during the installation procedure.

- **Compact**: Choose the Compact installation if you wish to install the TreeNet application only. WARNING: this setup type will not install the data file access program that allows TreeNet accessibility to different file formats. Without this ability, the only format that will be accessible is CSV (comma-separated value) and the SYSTAT file format (.SYD, .SYS).

- **Custom**: Choose the Custom installation if you would like to choose specific components available for installation. To include a particular option, click the mouse once on the desired option. Be sure that a checkmark appears in the appropriate box to ensure the item will be included as part of the installation.

By default, TreeNet is installed in `C:\Program Files\TreeNet 2.0`. Each component of the TreeNet installation is installed in a subfolder under `TreeNet 2.0`.

Ensuring Proper Permissions using Windows 2000 and Windows XP.

If you are installing TreeNet on a machine that uses security permissions, please read the following note.

- You must belong to the power user group on Win-NT and Win2000 to be able to run TreeNet. This is due to the way licensing works on these platforms (the license information must be written to a system folder to which you need write access).
Starting and Running TreeNet
Start TreeNet by clicking Start and selecting the TreeNet program group icon.

TreeNet takes advantage of Windows’ preemptive multi-tasking ability, so you can start a TreeNet run and then switch to other Windows tasks. Be aware that performance in TreeNet and your other active applications will decrease as you open additional applications. If TreeNet is running slowly you may want to close down other applications.

Note: Some stages of the TreeNet model building process may share the CPU only intermittently with other applications.

Preparing Your Data for TreeNet
TreeNet has two different modes of operation when reading in data.

**Mode 1:** In this mode TreeNet uses built-in algorithms to read legacy SYSTAT files or comma-separated ASCII files.

**Mode 2:** In this mode TreeNet reads in data using various third party Database drivers. This mode allows you direct access to different data formats, including SAS, SPSS, Excel, etc.

PLEASE REFER TO THE README FILE INCLUDED IN THE INSTALLER YOU MAY HAVE DOWNLOADED OR WHICH IS ON YOUR TREENET CD-ROM FOR FURTHER INFORMATION ON THE DATABASE DRIVERS.

These two modes are discussed in Chapter 2. Please refer to this chapter for details on this topic.

Setting up Working Directories
TreeNet always uses user-specified directories for different input and output files. First choose Edit—Options, then select the Directories tab to access/change the default locations:
### Input Files
- **Data** – input data sets for modeling and data exploration
- **Grove** – TreeNet model files for scoring and prediction
- **Command** – command files (modeling scripts)

### Output Files
- **Grove** – TreeNet model files to be created
- **Prediction results** – output data sets from scoring
- **Run report** – classic plain text output

### Temporary Files
- **Temporary** – where TreeNet will create additional temporary files as needed

**Note:** Make sure that the drive where the temporary folder is located has plenty of free space (at least the size of the largest data set you are planning to use, and preferably several times that size).

**Note:** Depending on your preferences, you may choose one of two working styles:
- using identical locations for both input and output files
- using separate locations for input and output files

**Note:** You should visit the TreeNet temporary file location periodically to remove files that may have been inadvertently left behind after a TreeNet session was ended.

**Note:** Useful audit trails of your TreeNet sessions are automatically saved to the temporary directory. You can safely delete these text files if you have no need for them but they can be lifesavers if you need to retrieve a log of your work.
Sample Data Set
A sample data set TREENET.CSV is supplied as part of the TreeNet installation. It is located in the Sample Data folder.

The following table lists the major groups of variables in the data set.

<table>
<thead>
<tr>
<th>Variable Names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1, … , X10</td>
<td>Continuous predictors</td>
</tr>
<tr>
<td>Z1$, Z2$</td>
<td>Categorical predictors (character)</td>
</tr>
<tr>
<td>Y1</td>
<td>Continuous target</td>
</tr>
<tr>
<td>Y2</td>
<td>Binary target coded +1 and -1</td>
</tr>
<tr>
<td>Y3</td>
<td>Categorical target with 4 levels: 1, 2, 3, and 4</td>
</tr>
<tr>
<td>W</td>
<td>Weight variable</td>
</tr>
<tr>
<td>T</td>
<td>Learn/Test dummy indicator (0 – learn, 1 – test)</td>
</tr>
</tbody>
</table>

In our first run we will try to predict the continuous target Y1 using the continuous predictors X1–X10. We chose a regression problem for this first example because regression problems are somewhat simpler and produce fewer reports than classification problems.

The scatter matrix for selected variables follows.

The plots suggest only a strong nonlinear relationship between X1 and Y1. It is not clear whether there is any relationship between the remaining predictors and the target.
An attempt to run a stepwise multiple linear regression of Y1 on X1–X10 selects five predictors, X1, X2, X6, X8, X9. Unfortunately, this model only has R-squared equal to .0179, making it practically useless. Later we will discover that this model fails to pick important predictors and at the same time mistakenly selects some irrelevant predictors. (You don’t need to do this to follow along. Right now just trust us.)

Exploiting possible periodic non-linearity in X1 as suggested by the corresponding plot may slightly improve the results, but the remaining relationships remain obscure.

Scatter Plot Matrix for Selected Variables
Reading Data In

To open the input file \texttt{TREENET.CSV}

1. Select \textbf{Open>Data File…} from the \textbf{File} menu.

   - You may simply click on the button in the toolbar.

2.) Use the \textbf{Open Data File} dialog window to navigate into the Sample Data folder in the TreeNet 1.0 installation directory.

   - To speed up further access to the given location, you may want to set up the corresponding working directory first. (See the section titled \textit{Setting Up Working Directories} earlier in this manual.)

3.) Choose \textbf{Delimited Text (*.csv,*.dat,.txt)} in the \textbf{File of type:} selection box.

4.) Highlight \texttt{TREENET.CSV}.

5.) Finally, click [Open] to load the data set into TreeNet.

   - Make sure that the data set is not currently being accessed by another application. Otherwise, the data loading will fail. This is a common source of problems when dealing with Excel and ASCII files (see later).

Setting Up the Model

When the data set has been successfully opened, the \textbf{Model Setup} dialog window will appear. The Model Setup tabs are the primary control center for various aspects of the TreeNet model-building process. In this example, we will enter information into the \textbf{Model} tab and \textbf{Testing} tab only and then create a model using TreeNet default settings.

   - In most forms of the TreeNet program, only the TreeNet button is available for selection. However, if you are running a combined TreeNet/CART/RF program, make sure that the \textbf{[TreeNet]} button is pressed at the bottom of the dialog window.
For this simple regression analysis, we complete the following four simple steps.

1.) In the Model tab we mark Y1 as the target by clicking the check box in the column labeled Target.

2.) Further, we mark X1-X10 as the predictors by clicking the check boxes in the column labeled Predictor.

3.) Next we choose the Regression radio button in the section labeled Tree Type.

4.) Select the Testing tab and choose the Variable separates learn and test samples radio button. Choose variable T in the selection list box.
Running TreeNet

To begin the TreeNet analysis, click the **[Start]** button. The following progress indicator will appear on the screen while the model is being built, letting you know how much time the analysis should take and approximately how much time remains.

Once the analysis is complete, text output will appear in the **TreeNet Output** window, and a new window, the **TreeNet Results**, opens.

The top portion reports various statistics about the current run. In our case all 10 predictors were used, individual trees had 6 terminal nodes, and 200 trees were grown with the optimal model keeping 132 trees.

« Note that the R-squared of the optimal on TEST data is 0.334, quite a bit better than an ordinary regression.

The bottom portion displays train and test mean absolute error curves as the TreeNet run progresses. The exact numeric minimum on the test data was attained at the 132-tree model highlighted by the green beam.

« To view the train and test mean squared error curves simply press the **[LS]** button.
When the results suggest that the test curve may continue going down, the TreeNet run may be resumed for an additional specified number of trees by simply clicking the [More Trees...] button.

**Viewing the Results**

The overall performance of the TreeNet regression model is summarized in model summary reports. To access the reports, click the [Summary...] button at the bottom of the TreeNet Results window.

As illustrated below, the TreeNet Summary contains gains charts and variable importance rankings. The summary report initially displayed is the Gains Chart. The Gains Chart tab displays gains information for the optimal model. This report is discussed in greater detail later in this manual.

If you use a test sample, [Learn], and [Test] will appear in the lower portion of the Gains Chart results tab. To view gains charts for the test sample, click [Test], and to view gains charts for the test sample, click [Learn].

The next TreeNet Reports displays the variable importance rankings, as illustrated below. The scores reflect the contribution each variable makes in predicting the target variable. The Variable Importance tab displays summary information about the predictive power of all selected variables on a 100% scale (for further details, see [1] for more technical discussion.)

In our example, we can see that X1, X2, X3, and X4 clearly stand out as important.
Note that the TreeNet model correctly stacks the only four variables that matter at the top of this report. The third ranked variable has a score of 35.35 and the fifth ranked has a somewhat lower score of 33.34. With such a pattern we might try rerunning the model using only the clearly most important variables.

- The various summary reports, including Variable Importance rankings, R-squared, and gains charts apply to a SINGLE model. In the example above this is the model that consists of exactly 132 trees.

- Although the best model is the one normally recommended for further study there is nothing preventing you from looking at other models. In fact you can access results for any model in the TreeNet sequence of models, which means that you can review results for models with 1,2,3,4,….199,or 200 included trees.

- To economize on resources we display the overall performance for every size of model beginning with the 1-tree model, but we only display detailed summary reports and graphs for select models. You can control how many such models are accompanied with details.

Now let us look at the various selections of plots. To view the selections available, click the [Display Plots...] button at the bottom of the TreeNet Results window. The TreeNet Plot Selection window will appear. The predictors are reported according to their importance. Clicking on the “+” signs expands any of the four selection items.
Double-click on any of the selections to view individual plots. Alternatively, highlight a selection and click the [Select...] button. The **One-Variable Dependence** plots for X1 and X3 are shown below.

- The [Scatter] button is pressed to view individual data points.
- By default, TreeNet automatically generates plots for the top three most important variables. You may change this number in the TreeNet tab of the Options window available through the Edit -> Options menu.

One-variable dependence plots are constructed by randomly selecting a subset of 200 data points and then averaging out all other variables except for the variable in focus.

- If the underlying relationship is additive in nature (no interactions), the one-variable plots simply represent the main effects in our regression model.

Notice how the periodic structure in X1 has been uncovered.
It is interesting to note that even though the underlying TreeNet model is by the nature of its construction piece-wise constant, having hundreds of overlapping trees creates nearly continuous dependency patterns.

When we have reason to suspect the presence of two-way interactions, it is beneficial to check the two-variable dependence plots. Two-variable plots for X1 and X2 are shown below.

The 3-D plots may be rotated using the buttons to obtain a better view. In addition, you may use the [Slice] or [All Slices] buttons to view either single or multiple 2-D surface slices.

Additional 2-D and 3-D plots can be generated immediately after the current model has been built. The new plots can be requested for any model (not necessarily the optimal one) and can include any available predictor. Press the [Create Plots...] button to gain access to this powerful feature. The procedure is described in more detail in a separate chapter later in this manual.

The various summary reports, including Variable Importance rankings, R-squared, and gains charts and all the 2D and 3D graphs displayed above apply to a SINGLE model. In the example above this is the model that consists of exactly 132 trees.

Occasionally you may have some reason for wanting to work with a model quite a bit smaller than the optimal model. We know of one large scale commercial user of TreeNet that opted to deploy a 30-tree model when an 800-tree model was optimal.

If you decide to work with a cut-back model, you can view graphs and displays for the smaller model, but you may have to rerun the model and either stop its evolution early or explicitly request that details be saved for many sub-models.

Viewing Results for Non-optimal Trees

The optimal model(s) are always accompanied by a comprehensive set of diagnostics, reports and graphs. In addition, TreeNet includes complete reports for selected other models and partial reports for all models. In the case of regression, complete reports
are always available for two potentially different models: the model with the number of trees yielding the optimal Least Squares, and the model containing the number of trees yielding the optimal LAD. A partial model summary is computed for all models but contains only core performance measures and the variable importance ranking. You may select any model by pressing [Ctrl] + arrow keys

⚠️ It is possible to increase the number of models with full summaries in the TreeNet tab of the Model Setup dialog.

⚠️ Requesting too many complete summaries might result in a significant program slowdown and may cause memory problems. Remember that each model might require 50 to several hundred graphs and if you save them all for 2000 models you can easily bog down your machine.
Reading Data

This chapter covers typical situations you may encounter while accessing your data in TreeNet
General Comments.

The following requirements must be met to read your data successfully in TreeNet:

- Data must be organized into a “flat file” with rows for observations (cases) and columns for variables (features).
- The maximum number of cells (rows x columns) allowed in the analysis will be limited by your license.
- The maximum number of variables allowed in the analysis is practically unlimited. The default maximum is 32768. To access more, consult the help system.
- TreeNet is case insensitive for variable names; all reports show variables in upper case.
- TreeNet supports both character and numeric variable values.
- Variable names must not exceed 32 characters.
- Variable names must have only letters, numbers, or underscores (spaces, %, *, &, -, $, etc. are NOT ALLOWED). If characters other than letters, numbers, or underscores are encountered, TreeNet will attempt to remedy the problem by substituting the illegal characters with underscores. The only exception is that character variables in ASCII files must end with a $ sign (see the next section).
- Variable names must start with a letter.

Be especially careful to follow the variable name requirements, because failure to do so may cause TreeNet to operate improperly. When you experience difficulties reading your data, first make sure that the variable names are legal.

Methods for Reading Data

TreeNet has two different modes of operation when reading in data.

Mode 1: Reading ASCII files:

In this mode TreeNet uses built-in algorithms to read legacy SYSTAT files or comma-separated ASCII files.

Mode 2: Using Database driver facilities

In this mode TreeNet reads in data using various database drivers. This mode allows you direct access to different data formats, including SAS, SPSS, Excel, etc.

If you have licensed a previous version of TreeNet or other Salford Systems products you may have obtained the DBMSCOPY drivers. If so, you have the option of continuing to use those drivers.

We are now supplying the Circle Systems Stat/Transfer drivers in all Salford Systems products and recommend that you eventually migrate to these drivers.

Reading ASCII files (Mode 1)

TreeNet has the built-in capability to read various forms of delimited raw ASCII text files. This built-in capability is most appropriate for datasets composed of numeric and quoted
character data, using a comma for the delimiter. Optionally, spaces, tabs or semicolons instead of commas can separate the data, although a single delimiter must be used throughout the text data file.

ASCII files are best accessed in mode 1 (using the internal ASCII reader). An attempt to read ASCII files in mode 2 (using database drivers) may fail unless you have a specially formatted dictionary file, in addition to an ASCII file.

ASCII files must have one observation per line with the first line containing variable names (see the necessary requirements for variable names in the previous section). As previously noted, variable names and values are usually separated using the comma (",") character. For example:

```
DPV, PRED1, CHAR2$, PRED3, CHAR4$, PRED5, PRED6, PRED7, PRED8, PRED9, PRED10, IDVAR
0, -2.32, "MALE", -3.05, "B", -0.0039, -0.32, 0.17, 0.051, -0.70, -0.0039, 1
0, -2.32, "FEMALE", -2.97, "O", 0.94, 1.59, -0.80, -1.86, -0.68, 0.940687, 2
1, -2.31, "MALE", -2.96, "H", 0.05398, 0.875059, -1.0656, 0.102, 0.35215, 0.0539858, 3
1, -2.28, "FEMALE", -2.9567, "O", -1.27, 0.83, 0.200, 0.0645709, 1.62013, -1.2781, 4
```

Character variables are indicated by either placing a '$' at the end of the variable name (e.g., POLPARTY$), or surrounding the character data with quotes (e.g., "REPUBLICAN"), or both.

TreeNet uses the following assumptions to distinguish numeric variables from character variables in ASCII files:

- When a variable name ends with "$", or if it is surrounded by quotes (either ' or ") on the first record, or both, it is processed as a character variable. In this case, a $ will be added to the variable name if needed.
- Otherwise, the variable is treated as numeric. If a character value is found for a numeric variable, it is converted to the numeric missing value.
- It is safest to use "$" to indicate character fields. Quoting character fields is necessary if "$" is not used at the end of the variable name or if the character data string contains commas (which would otherwise be construed as field separators).
- Character variables are automatically treated as discrete (categorical). Logically, this is because only numeric values can be continuous in nature.
- When a variable name does not end with a $ sign, the variable is treated as numeric. In this case, if a character value is encountered it is automatically replaced by a missing value.

A sample ASCII file TREENET.CSV comes as part of the TreeNet distribution and resides in the Sample Data folder.

To open TREENET.CSV you should:

1. Click File–Open> Data File...
2. In the Open Data File dialog window choose Delimited Text (*.csv, *.dat, *.txt).
3. When you double click on treenet.csv, the Model Setup dialog window should appear.

- The Open Data File dialog lists only those files that match the selected extension in the File of type: selection box. In mode 1 (when File–Use Database Drivers… is unchecked), choosing a file type that does not match the specified File of type: will fail. You must select an explicit data format in either mode 1 or 2 to activate the corresponding data access driver (because different data formats may have conflicting extensions).

- We recommend using Delimited Text (*.csv, *.dat, *.txt) selection to read ASCII files (Mode 1). Selecting ASCII (*.dat) in the File Types selection box will be using DBMS/COPY driver (Mode 2) to access your ASCII files and will fail unless you have the dictionary file.

Variable Naming

Acceptable variable names have a maximum of 32 characters, must be composed of letters, numbers and underscores, and must begin with a letter.

- Spaces are not permitted when reading raw ASCII text files. When using database drivers, spaces are permitted only when the selected data file format allows them. However, in most cases the space will be converted and displayed as an underscore.

Examples of acceptable and unacceptable variable names.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Acceptability</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGE_1</td>
<td>OK</td>
</tr>
<tr>
<td>GENDER</td>
<td>OK</td>
</tr>
<tr>
<td>POLPARTY</td>
<td>OK</td>
</tr>
<tr>
<td>1WORLD</td>
<td>Unacceptable; leading character other than letter</td>
</tr>
<tr>
<td>%WEIGHT</td>
<td>Unacceptable; leading character other than letter</td>
</tr>
<tr>
<td>SOCIAL_SECURITY_NUMBER_AND_ACCOUNT</td>
<td>Unacceptable, too long. Variable name will be truncated to 32 characters.</td>
</tr>
<tr>
<td>SALT&amp;PEPPER</td>
<td>Unacceptable, “&amp;” not letter, number or underscore. This character will be replaced with an underscore.</td>
</tr>
</tbody>
</table>

Character variable names are required to end in an additional ‘$’, so if a character variable name does not end with ‘$’, it will be added by TreeNet as the data are read:

```
NAME$
SSNUMBER$
```

Numeric variables may optionally have subscripts from 0 to 99 but TreeNet does not use them in any special way:
<table>
<thead>
<tr>
<th>CREDIT(1)</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCORE(99)</td>
<td>OK</td>
</tr>
<tr>
<td>ARRAY(0)</td>
<td>OK</td>
</tr>
<tr>
<td>ARRAY(100)</td>
<td>Unacceptable; parenthesis will be replaced with underscore. Subscript may not exceed 99.</td>
</tr>
<tr>
<td>(1)</td>
<td>Unacceptable; parenthesis will be replaced with underscore.</td>
</tr>
<tr>
<td>x()</td>
<td>Unacceptable; parenthesis will be replaced with underscore.</td>
</tr>
<tr>
<td>x(1)(2)</td>
<td>Unacceptable; parenthesis will be replaced with underscore.</td>
</tr>
</tbody>
</table>

atório When using raw ASCII text input data, TreeNet does not check for, or alter, duplicate variable names in your dataset.

Using Database Drivers (Mode 2)

Unless your data are in the legacy SYSTAT format or in the comma-separated ASCII format, mode 2 must be used to access your data. Mode 2 is activated whenever **File—Use DATABASE DRIVERS**… is checked.

Accessing your data regardless of the original file format

All Salford Systems’ data analysis software can access data in many file formats. By default, this capability is enabled during the installation procedure. To confirm this feature is enabled, select the **File** menu and make sure that the **Use DATABASE DRIVERS** item is checked. If the menu item is not checked, selecting the item turns the check mark on, thereby enabling this feature.

TreeNet, as well as other Salford Systems’ applications, employs built-in DATABASE DRIVERS functionality to enable you to open a large variety of file formats, including Excel, SAS, S-Plus, Access, etc. Raw text files can also be opened directly without the use of DATABASE DRIVERS. Users who are using raw ASCII text file should read the section above titled “Reading ASCII Files.”

In mode 2, the **File of type:** selection box in the **Open Data File** window contains a wide selection of supported data formats. Choose the corresponding data format first to see your files.
Setting up Dynamic Link to DBMS/COPY™ Database Drivers

In earlier versions of our products we made use of the DBMS/COPY™ Database Drivers and we continue to offer this as one option, although we also offer database drivers from other vendors. Regardless of which set of drivers you elect to use you may need to assist TreeNet in locating them. In the example below, we illustrate how the database driver location would be specified for DBMS/COPY™.

TreeNet may fail to connect with the DATABASE DRIVERS. When this happens, an attempt to click File–Open>Data File... will result in the display of the following information window.

Click [Yes], and the following window will appear, allowing you to try to locate DBMS/COPY.

Within the Browse for Folder dialog, navigate to the location of the DBMS/COPY drivers. (By default, the DBMS/COPY folder is in the TreeNet installation folder, but you
may also select a standalone DBMS/COPY location if it is also installed on your system. If you already have TreeNet version 1.0 or other Salford products you may already have a copy of the DBMS/COPY™ Database Drivers installed.

Once you have located the DBMC/COPY directory, click the [OK] button. The DBMS/COPY drivers should now be linked to TreeNet and the Open Data File dialog window should appear. Unless something unexpected happens with your system, the link will be automatically established the next time you start up TreeNet. You should only have to go through this procedure once.

To permanently unlink DBMS/COPY drivers from TreeNet, remove the DBMSdll= entry in the cart.ini file located in the WINDOWS folder on your system drive (making sure that TreeNet is closed when you do this).

Making sure that you are using the most up-to-date file access drivers
If you used the Custom installation procedure, and did not install DATABASE DRIVERS, you may have an older version of DATABASE DRIVERS that does not support recently-introduced file formats. We recommend that you run the Custom installation procedure and install the DATABASE DRIVERS component.

Reading Excel Files
Excel files are easily accessible in mode 2 using DATABASE DRIVERS. However, care must be exercised when doing this. Make sure that the following requirements are met:

- The Excel file must contain only a single data sheet; no charts, macros or other items are allowed.
- The Excel data format limits the number of variables to 256 and the number of records to 65535. It is advisable to limit your Excel data files to no more than about 250 columns and no more than about 65000 rows. Going all the way up to the limit of the format may create problems.
- The Excel file must not be currently open in Excel, otherwise the operating system will block any access to it by an external application such as TreeNet. On some operating systems, if the Excel file was recently open in Excel, the Excel application must be closed to entirely release the file to be opened by TreeNet.
- The First row must contain legal variable names (see the beginning of this chapter for details).
- Missing values must be represented by blank cells (no spaces or any other invisible characters are allowed).
- Any cell with a character value will cause the entire column to be treated as a character variable (will show up ending in a $ sign within the Model Setup). This situation may be difficult to notice right away, especially in large files.
- Any cell explicitly declared as a character format in Excel will automatically render the entire column as character even though the value itself might look like a number—such cases are extremely difficult to track down.
- It is best to use the cut-and-paste-values technique to replace all formulas in your spreadsheet with actual values. Formulas have sometimes been reported to cause problems in reading data correctly.
- Alternatively, you may save a copy of your Excel file as a comma delimited file (.CSV) and use the **File of type**: Delimited Text (*.csv, *.dat, *.txt) (**caution**: make sure no commas are part of the data values).

**SQL and relational databases**

To access data stored in a SQL-based system, you will need to extract an appropriate flat file using your favorite database management tools. Direct access to your relational database and tools for extracting your data will be provided in a later version of TreeNet.
3 ........................

Regression Models

This chapter covers in depth how to use TreeNet to build regression models.
General Remarks on Regression Runs

- If you are primarily interested in predicting 0/1, YES/NO or other binary targets please go directly to the next chapter.

Regression is used to model a continuous outcome. The predictors can be a mix of continuous and categorical variables (including text) and do not require any special preparation. By the nature of the algorithm, TreeNet models are invariant to monotone transformations of continuous predictors so you never need concern yourself with transforms of the predictors. However, it can be worthwhile to consider transforming your target variable, by taking a logarithm or a square root, for example.

TreeNet naturally supports three distinct regression loss functions: least squares loss (LS), least absolute deviation loss (LAD), and Huber-M loss (a hybrid of LS and LAD). In particular, it is possible to build three generally different model sequences depending on the loss function used.

For any given model sequence, TreeNet reports two separate error profiles: mean squared error and mean absolute error. In the presence of a test sample (cross validation), the optimal model is defined as the smallest test (cross-validated) mean squared error for the LS loss and the smallest test (cross-validated) mean absolute error for the LAD and Huber-M losses. When no independent testing is specified (exploratory runs), the optimal model is always the one with the largest number of trees.

- It is always possible to manually select a model of a different size for subsequent scoring, translation, and reporting.

Model Setup – Model

Regression modeling in TreeNet is controlled from the Model Setup dialog. This window automatically appears after a data file has been successfully accessed. You may also request the Model Setup dialog using the Model–Construct Model… menu or by simply clicking the toolbar icon.

- A data set must be currently open in TreeNet for this operation to work.
- The Model menu item is accessible when the Classic Output window is active.

In the Model Setup–Model tab we specify our core model variables. Variables specified in this dialog include a target (dependent), predictors (independent), categorical designation, and optionally a single case-weight.

The selection of variables and tree type for our regression is as follows:

1. Choose the Regression radio button in the section titled Tree Type.
2. Select the target variable by placing a check mark in the corresponding variable check box under the Target column.

- There can be only one target at a time in TreeNet.
You may change the Sort: order to Alphabetically when it is convenient.

3. Select potential predictors by placing check marks in the corresponding variable’s check boxes under the Predictor column.

You may highlight multiple variables first by using a left mouse click and/or the <Shift> or <Ctrl> keys while selecting in the Variable Name column. Once selected you may either check or uncheck the corresponding check marks all at once for the selected variables by using the check box at the bottom of the appropriate column.

4. Optionally, check a single weight variable using a check box under the Weight column.

There can be only one weight variable at a time.

5. Indicate which numeric predictors are categorical by placing check marks under the Categorical column for the corresponding variables.

Correctly declaring variables as categorical is less important for TreeNet than for CART or other modeling methods. TreeNet is capable of uncovering the true relationship between the target and a categorical predictor whether the declaration is made or not. However, some graphs will be much easier to read if the declarations are made. For high-level categorical predictors, however, it may be advantageous to treat such predictors as numeric. This is discussed further in a later chapter.

The Model tab will now appear as follows:

All character variables (variables ending in $) will be automatically checked as categorical and cannot be unchecked.
Using high-level categorical variables may significantly increase the running time and impose severe memory requirements. Make sure that a continuous variable has not been accidentally checked as categorical.

Consider letting TreeNet treat a high-level categorical variable as a continuous predictor if it has more than about 30 levels. This is unlikely to hurt the model and may actually improve results quite a bit. This is possible only if the HLC is coded as numeric rather than text.

Model Setup – Testing
The Testing tab specifies how the resulting TreeNet model will be evaluated. The six following test modes are available in TreeNet.

- **No independent testing – exploratory tree.** In this mode no external evaluation will be performed. The entire data set will be used for training; therefore, the final results will include only training data and the best model is almost invariably the largest.

- **Fraction of cases is selected at random for testing.** The specified fraction of data will be set aside for testing. Final results will be reported for both train and test data.

- **Test sample is contained in a separate file.** The data for testing will be contained in the specified file. Use this mode when you want to use one data file for training and a separate data file for testing. When this mode is activated, a file selection dialog window will appear.

- **V-fold Cross Validation.** Usually used with small datasets when one cannot afford to reserve some data for testing. After the entire dataset has been used for learning purposes, it is partitioned into ten bins. At each fold in 10-fold CV, nine bins are used as a training set and the remaining bin is used as a test set. After all 10 folds are completed, the test results from each fold are averaged to get a fair test estimate of the all-data model performance.

- **Variable determines cross-validation samples.** Same as the above option but the user has full control over bin creation by introducing a special variable with integer values 1 through 10 to mark the bins. You can opt for a different number of folds.

- **Variable separates learn and test samples.** This is the most flexible mode. All you need is a binary variable coded as 0 or 1. When its value is 0, the corresponding observation will go into the learn data; otherwise, the observation will go into the test data.

Use extreme care when creating your own CV bins. Each bin should be of the same size and balanced on the target variable.

10-fold cross-validation runs are on average 10 times slower than using a test sample. For small data sets you may not even notice the difference but for large data sets using cross validation will substantially increase your run times.

Use “Cross Validation” when the supply of data is limited.
Use the indicator variable mode when you want complete control over which part of the data is used for testing. You may also create multiple indicator variables in the same data set to experiment with different random seeds or learn/test ratios.

In our example we are using **A variable separates learn and test samples**. The sample data set already has a test indicator variable, called T. The **Testing** tab now appears as follows:

![Model Setup – TreeNet](image)

**Model Setup – TreeNet**

The **Model Setup–TreeNet** tab contains important parameter settings used in the model-building process. This setup dialog allows the user to control multiple aspects of the TreeNet modeling process. Each of the parameters displayed in the following illustration are discussed in detail below.
Learn rate:
Controls the rate at which the model is updated after each training stage. This is also referred to as “shrinkage” in Friedman’s original articles.

The default is AUTO, and the allowable range is 0.0001 to 1.0 inclusive. AUTO is calculated as follows:

\[
AUTO\ value = \max(0.01, 0.1\times\min(1, nl/10000))
\]
where \( nl \) = number of LEARN records.

This default uses very slow learn rates for small data sets and uses 0.10 for all data sets with more than 10,000 records. We strongly recommend that you experiment with different learn rates, especially rates slower than the default for larger data sets. See [1] for more technical discussion.

High learn rates and especially values close to 1 typically result in overfit models with poor performance.

Values much smaller than .01 significantly slow down the learning process and might be reserved for overnight runs.

Subsample fraction:
This setting controls the proportion of data used for learning at each training cycle. A new random draw is conducted at each cycle, which not only speeds up the modeling time but also guards against overfitting, and explains occasional minor local increases in the learn error curve as a TreeNet run progresses.

The default is 0.5, and the allowable range is 0.01 to 1.0 inclusive.

It may be necessary to use values greater than the .5 default with small training files. Unlike the learn rate, using a sampling rate of 1.0 is not necessarily
catastrophic, but values less than 1.0 are still strongly recommended. You may need to experiment to determine the best rate.

- See [1] for more technical discussion.
- Sampling rates that are too small can hurt accuracy substantially while yielding no benefits other than speed.

Influence trimming factor:
This setting is ignored in regression runs.

M-Regression breakdown parameter:
This setting is used only when the regression loss criterion is set to Huber-M. It sets the relative threshold beyond which the least squares deviation criterion is replaced by the least absolute deviation criterion.

- Lowering this value from the default 0.9 can be helpful especially if you are concerned about outliers.
- See [1] for more technical discussion.

The default is 0.9 and the range is 0.01 to 1.0 inclusive.

Optimal Logistic Model Selection Criterion:
This setting is ignored in regression runs.

Number of trees to use:
A TreeNet model is essentially a collection of trees of a fixed size. New trees are created as the modeling process progresses. This option specifies how many trees should be grown before the modeling process ends.

- Theory suggests that having a minimum of several hundred trees is needed to achieve the best predictive results as well as to exhaust the extraction of useful information from the internal data structure. Experience suggests that several thousand trees may be needed to fully exploit the predictability inherent in the data.
- If the test set error is going down even when the largest number of trees specified is reached, you should consider requesting more trees.
- We recommend expanding the number of trees in the model whenever the “optimal” model is anywhere close to the maximum grown.

Maximum number of trees, including restart continuations:
This parameter controls the largest possible number of trees that are allowed in any single TreeNet model, including requests for more trees. You will rarely need to change this setting.

Maximum nodes per tree:
This setting controls the fixed size of each individual tree used in the TreeNet model. At each step, a fixed-size tree will be grown to correct the current prediction error. Sizes usually used by TreeNet experts range from two to 12 nodes.

- The lowest possible setting is two nodes. In this case each individual tree will contain a single split, resulting in two terminal nodes. Such trees are known as “stumps” in the machine-learning literature.
- A TreeNet model constructed from 2-node trees is incapable of detecting interactions. However, it does reveal the best you can do with a “main effects” or additive model and is a very useful benchmark model.
- The higher the maximum nodes setting, the greater the likelihood that higher order interactions will be detected.

Growing large trees will slow down your runs and possibly bog down your machine so we don’t recommend pushing much beyond 12 nodes.

**Minimum number of training observations in terminal nodes:**
This setting controls how small individual terminal nodes are allowed to be. In our example, the value setting of 10 indicates that no terminal nodes with fewer than 10 observations are allowed in the model. In smaller data sets you may need to lower this to a value like 5 or 3.

**Minimum number of most-optimal models to save summary results for:**
This setting controls how many models will have complete summary results reported and saved. When this is set to one, the full summary results will only be saved for the single best model per each available optimality criterion (mean squared error and mean absolute error). Settings above one will include additional next best optimal models uniformly distributed among the available optimality criteria.

Setting this parameter to a very large value may severely increase the running time of the modeling step and have memory implications. We recommend small values such as 1, 5 or 10. You may want to use larger numbers only if you are planning on manually selecting a model from the TreeNet sequence of models.

**Regression loss criterion:**
Three different loss functions are available for regression modeling.

- **Huber-M** – this loss function sums either squared deviation or absolute deviation for each observation depending on the relative magnitude of the deviation. Because of this, it combines the best properties of the other two loss functions and was chosen as the default setting.
  - See [1] for more technical discussion.

- **Least absolute deviation** – is loss function is the sum of absolute deviations. It is best used when many outliers are present (optimal for the double exponential distribution of errors); otherwise, it may produce inferior results.
- **Least squares** – is loss function is the sum of squared deviations. It has a number of theoretical advantages (optimal for the Gaussian distribution of errors) but may perform poorly when outliers are present.

  - Regression loss criterion settings are ignored when TreeNet is used for logistic regression and classification modeling (following chapters).

**Model Setup – TreeNet – Defaults**

The group of buttons in the lower right corner of the TreeNet tab allows the user to specify new defaults for any subsequent TreeNet run. The defaults are saved in the TreeNet initialization file and persist from session to session.

- [Std. Defaults] Press this button to restore the original “factory” settings.
- [Save Defaults] Press this button to save the current settings as the new defaults.
- [Recall Defaults] Press this button to recall previously-saved defaults.

For example, you might decide that your typical TreeNet runs should use a 0.01 learn rate, 1000 trees, two-nodes (to force additive models), and the least squares loss criterion. Just set these parameters and press [Save Defaults]. In future runs, TreeNet will automatically assume the new settings unless you manually override them.

**Model Setup – Save Grove**

Any time during model setup, the [Save Grove...] button can be used to save the TreeNet model to a “grove” file. The grove file, a binary file that contains all the information about the TreeNet model, is needed if you want to apply the model to a new data set or translate the model into one of the supported languages. The grove file is the recommended repository for any model, TreeNet to display all your graphical results as well as supporting scoring of data sets at some future time.

After the [Save Grove...] button is pressed, the **Specify Grove File** dialog window appears:

![Specify Grove File](image)

Type in the name of the grove file to be created and click the [Open] button.

- The grove file will be created on your hard drive automatically, but only after the model-building process is completely finished.
You can also save the grove file interactively from the Results window (see below) after the run is complete.

**TreeNet Results**

A TreeNet modeling run starts after the [Start] button is pressed. A progress report window appears. After all requested trees have been grown, the TreeNet Results window automatically appears:

Information available in the TreeNet results window includes the following information:

- General information about run settings is displayed in the upper section of this window: training data file, target variable, number of predictors, requested tree size and the total number of trees to be grown, type of loss function.

- TreeNet also reports the number of trees in the model at which the test set error attains its minimum. For the current run, the optimal model is attained when 193 trees are grown.

- The lower part of the window displays the run history: the tree number on the horizontal axis and train/test error profiles on the vertical axis. You can switch between the least squares and the least absolute deviation error profiles using the [LS] and [LAD] buttons. Note how the test error curve goes slightly above the learn error curve and how they both agree.

  The green line marks the optimal model – the model having the smallest test mean squared deviation (when LS loss is used) or the smallest test mean absolute deviation (when LAD or Huber-M loss is used). The actual number of trees and the corresponding value of the error function are reported above the line. In our example, the optimal model has 193 trees with a mean absolute error of 0.294 on the test data.

- The R-squared for the optimal tree is reported.
R-squared is reported on the test data and has a traditional least squares deviation definition. It is rare but not impossible to get negative values of R-squared for poorly predictable data, especially when the LAD or Huber-M losses are used.

Summary Reports – General

The TreeNet Summary window is activated when the [Summary...] button is pressed in the TreeNet Results window. TreeNet provides variable importance information for all models and gains information for a selected subset of models, which always includes the optimal least squares and the optimal least absolute deviation models.

You can specify the number of models for which the full results are available in the TreeNet tab of the Model Setup window. The default value of “1” retains detailed graphs and reports only for the optimal model.

You can select alternative (and on-optimal) models by using the arrow keys or making a left mouse click. Models without the full summary (variable importance only) will appear as dashed lines. You can jump directly among the models with the full summary available by using the combination of <Ctrl> key and the corresponding arrow keys or mouse clicks.

Consider a non-optimal model if you must work with a cut-down small-sized model to get the highest speed in deployment situations (such as on-line scoring). Remember that you do sacrifice accuracy when you select a sub-optimal model.

Summary Reports – Gains

The Gains Chart tab is located in the TreeNet Reports window. The gains chart and the corresponding table are often used to judge the overall performance of a regression model on a given data set. It appears as follows:

Gains charts were originally created to help assess the performance of classification models. However, if the dependent variable is non-negative then the gains chart does reveal something about model performance. Looking at the top bin (the top row) of the table, we see that in this bin the top 10% of the data as ranked by model predictions
have a mean value that is a certain multiple of the sample average (the LIFT). The
greater the lift, the more successfully the model correctly ranks the data from the
highest to the lowest value of the target.

The following generic algorithm explains the assumptions and calculations involved in
the process.

### Computing Gains

**Assumptions:**
- **TARGET** denotes the actual response that must be known for each case.
- **RESPONSE** denotes the response values predicted by the given model.
- **K** denotes the predefined number of bins.

1. Sort the data by descending **RESPONSE**.
2. Divide the data into **K** bins such that each bin has an approximately equal number of
weighted cases.

- **Target Bin Average** – the weighted mean of the **TARGET** in the given bin.
- **% Target in Bin** – the weighted sum of the **TARGET** in the given bin divided by
  the overall weighted sum of the **TARGET**.
- **Cum % Target** – the simple cumulate of the **% Target in Bin**.
- **Cases in Bin** – the weighted number of cases in the given bin.
- **% Pop** – the weighted number of cases in the given bin divided by the total
  weighted number of cases.
- **Cum % Pop** – the simple cumulate of the **% Pop**.
- **Cum Lift** – **Cum % Target** divided by **Cum % Pop**.
- **Lift Pop** – **% Target in Bin** divided by **% Pop**.

The graph to the left displays:
- When the **[Gains]** button is pressed – **Cum % Target** versus **Cum % Pop**.
- When the **[Lift]** button is pressed – **Lift Pop** versus **Cum % Pop**.
- When the **[Cum. Lift]** button is pressed – **Cum Lift** versus **Cum % Pop**.

The number of bins can be changed using the control.

When the test data are available, both learn and test gains charts can be displayed
using either the **[Learn]** or the **[Test]** buttons.

One may use the following simple interpretation of gains. Assume that a regression
model is built to predict the amount of purchases by each individual in a population in
response to a certain mail offer. If every member of the population is given the offer,
the total sales would be **G** (Total Sales=**G**). Now assume that we would actually like to
send mail offers to only 10% of the population. When no model is available and offers
are mailed purely randomly, we should expect to collect only about 1/10th of **G**. In the
presence of a model, instead of random mailing, one could send offers to the 10% of
the population in the top bin. In this case the expected volume of sales would be \textbf{Lift Pop} times 1/10\textsuperscript{th} of \textbf{G}. The interpretation is similar for the other bins and cumulative lift.

- Gains based on the test data are usually more realistic than gains based on the training data.
- Test gains are not available when independent testing is disabled (Test tab in the Model Setup).
- When the TARGET has negative values, it is adjusted upwards by its smallest value prior to computing gains to produce sensible displays. However, in this case the interpretation of gains might become cumbersome.

**Summary Reports – Variable Importance**

The \textbf{TreeNet Summary} window also contains a \textbf{Variable Importance} tab.

The raw variable importance score is based on the improvements of all splits associated with a given variable across all trees in the current model. Note that smaller models will tend to have fewer variables with non-zero importance scores and larger models will generally tend to utilize many more of the available predictors.

The raw importance scores are rescaled so that the most important always gets a score of 100. The relative importance \textbf{Score} provides a relative measure of each variable’s contribution to the model’s predictive power. In our example, the continuous predictors \textit{X1}, \textit{X2}, \textit{X3}, \textit{X4} and the categorical predictors \textit{Z1$}, \textit{Z2$} were clearly identified as important. The remaining predictors are reported to have smaller contributions. This conclusion is further supported by the analysis of contribution plots described in a separate chapter.

When the \texttt{[Importance]} button is pressed, the following display is shown:

- The rows in the table can be re-sorted by clicking on the corresponding column header.
In addition to the variable importance, this tab gives access to the summary single stats display. When the [Single Stats] button is pressed, the following display is shown:

This report simply collects all generated Single Stats Plots on one display. A more detailed explanation is provided in the subsequent chapters.

**Other Controls**

The [More Trees...] button can be used to request more trees. This operation is only available immediately after the current set of trees has been grown. This feature is not available when cross validation is used.

The [Save Grove...] button is used to save the model results to a grove file. The saved grove can later be opened for viewing (File->Open>>Open Grove... menu) or used for scoring and translation (see later chapters).

The [Create Plots...] and [Display Plots...] buttons are used in plot operations discussed later on.
Binary Logistic Models

This chapter covers in depth how to use TreeNet to build binary logistic models.
General Remarks on Binary Logistic Runs

TreeNet “Binary Logistic” is the recommended way to model any binary outcome and is probably the most popular use of TreeNet. Although most analysts think of this type of model as CLASSIFICATION, in TreeNet we consider it as a special type of regression. This is because we want to produce a PROBABILITY of response for and not just a simple prediction of “YES” or “NO.” TreeNet’s ability to produce extremely accurate rankings of data from “most likely” to “least likely” underpins its strong performance in many data mining settings.

The first point to make is that this chapter is concerned exclusively with the binary or two-valued target variable. This variable is often coded as “0” and “1” or “YES” and “NO” or “RESPONSE” and “NONRESPONSE,” etc. Such models are by far the most common in data mining applications and are used to predict:

- Default in credit risk scoring
- Response in targeted marketing campaigns
- Fraud in insurance, credit card transactions, and loan applications
- Disease (or abnormal versus normal) in biomedical settings

If you want to analyze such data then the logistic regression style of TreeNet is likely to give you the best results.

Best Models

There is no such thing as a “best model” absent a context and an objective. To help you decide which model is best we provide you with four measures of model performance:

- Classification Accuracy: This is based on a straightforward tally of how often the model tags a record correctly or incorrectly. If you intend to use the model to simply tag data as either “YES” or “NO” then this criterion will help you select the best model.

- ROC: area under the ROC curve. This is the most commonly-used model criterion in machine learning and is a measure of overall model performance tied closely to the ability of the model to correctly RANK records from most likely to least likely to be a “1” or a “0.”

- Cross-entropy or Log-Likelihood: Similar to ROC and offered for technical users who prefer this measure for technical reasons.

- Lift in the top X-percentile: Focuses on the ability of the model to concentrate the responders or the “1”s in the highest ranking records. This criterion is useful if the model is intended to be used exclusively to identify, say, the top 1%, 5% or 10% of a data set. In this case, we can safely ignore model performance in the lower percentiles.
When you run TreeNet the software builds its model in stages, adding a tree at each stage in an effort to refine performance. Typically, the first tree yields a modest performance, the second tree improves on it, the third tree improves further, and so on. At some point in the process adding further trees does no good and may even do some harm. The entire set of trees grown is called the model sequence, and after the specified number of trees has been generated, we want to determine how many of the trees to keep. To assist in this process we track the four performance criteria at each stage of the modeling sequence and identify the best number of trees to keep for each of the performance criteria.

- It is not possible to tell in advance how many trees will be optimal.
- The best number of trees can vary quite a bit depending on which performance criterion you select.
- Typically, the best model for classification accuracy has many fewer trees than the best model for best probability estimates (CxEE) or overall performance (ROC). Best lift is also usually accomplished with relatively few trees.
- The best model is normally determined by performance on test data or via cross-validation. When no independent testing is specified (exploratory runs), the optimal model is always set to the largest number of trees.
- It is possible to manually select a non-optimal model of any size for graphical display, obtaining summary reports, scoring and translation.

**Model Setup – Model**

Logistic regression modeling in TreeNet is controlled from the Model Setup dialog. This window automatically appears after a data file has been successfully accessed. You can also request the Model Setup dialog using the Model–Construct Model… menu or by clicking the toolbar icon.

- A data set must be currently open in TreeNet for this operation to work.
- The Model menu item is accessible when the Classic Output window is active.

In the Model Setup–Model tab we specify our core model variables. Variables specified in this dialog include a target (dependent), predictors (independent), categorical designation, and optionally a single case-weight.

The selection of variables and tree type for our model is as follows:

1. Choose the Logistic Binary radio button in the section titled Tree Type.
2. Select the target variable by placing a check mark in the corresponding variable check box under the Target column.

- There can be only one target at a time in TreeNet.
- You may change the Sort: order to Alphabetically when it is convenient.
3. Select potential predictors by placing check marks in the corresponding variable's check boxes under the **Predictor** column.

- You may highlight multiple variables first by using a left mouse click and/or the <Shift> or <Ctrl> keys while selecting in the Variable Name column. Once selected you may either check or uncheck the corresponding check marks all at once for the selected variables by using the check box at the bottom of the appropriate column.

4. Optionally check a weight variable using a check box under the **Weight** column.

- There can be only one weight variable active at a time.

5. Indicate which predictors are categorical by placing check marks under the **Categorical** column for the corresponding variables.

The **Model** tab will now appear as follows:

- Correctly declaring variables as categorical is less important for TreeNet than for CART or other modeling methods. TreeNet is capable of uncovering the true relationship between the target and a categorical predictor whether the declaration is made or not. However, some graphs will be much easier to read if the declarations are made. For high level categorical predictors, however, it may advantageous to treat such predictors as numeric. This is discussed further in a later chapter.

- All character variables (variables ending in $) will be automatically checked as categorical and cannot be unchecked.

- Using high-level categorical variables may significantly increase the running time and impose severe memory requirements. Make sure that a continuous variable has not been accidentally checked as categorical.
Consider letting TreeNet treat a high-level categorical variable as a continuous predictor if it has more than about 30 levels. This is unlikely to hurt the model and may actually improve results quite a bit. This is possible only if the HLC is coded as numeric rather than text.

Model Setup – Testing

The Testing tab specifies how the resulting TreeNet model will be evaluated. The six following testing modes are available in TreeNet.

- **No independent testing – exploratory tree.** In this mode no external evaluation will be performed. The entire data set will be used for training; therefore, the final results will include only training data and the best model is almost invariably the largest.

- **Fraction of cases is selected at random for testing.** The specified fraction of data will be set aside for testing. Final results will be reported for both train and test data.

- **Test sample is contained in a separate file.** The data for testing will be contained in the specified file. Use this mode when you want to use one data file for training and a separate data file for testing. When this mode is activated, a file selection dialog window will appear.

- **V-fold Cross Validation.** Usually used with small datasets when one cannot afford to reserve some data for testing. The entire dataset is used for learning purposes, then is partitioned into ten bins. At each fold in 10-fold CV, nine bins are used as a training set and the remaining bin is used as a test set. After all 10 folds are completed, the test results from each fold are averaged to get a fair test estimate of the all-data model performance.

- **Variable determines cross-validation samples.** Same as the above option but the user has full control over bin creation by introducing a special variable with integer values 1 through 10 to mark the bins. You can opt for a different number of folds.

- **Use extreme care when creating your own CV bins. Each bin should be of the same size and balanced on the target variable.**

- **10-fold cross validation runs are on average 10 times slower than using a test sample. For small data sets you may not even notice the difference but for large data sets using cross validation will substantially increase your run times**

- **Variable separates learn and test samples.** This is the most flexible mode. All you need is a binary variable coded as 0 or 1. When its value is 0, the corresponding observation will go into the learn data; otherwise, the observation will go into the test data.

- **Use “Cross Validation” when the supply of data is limited.**

- **Use the indicator variable mode when you want complete control over which part of the data is used for testing. You may also create multiple indicator variables in the same data set to experiment with different random seeds or learn/test ratios.**
In our example we are using **A variable separates learn and test samples**. The sample data set already has a test indicator variable, called T. The **Testing** tab now appears as follows:

![Model Setup – TreeNet](image)

**Model Setup – TreeNet**

The **Model Setup–TreeNet** tab contains important parameter settings used in the model-building process. This setup dialog allows the user to control multiple aspects of the TreeNet modeling process. Each of the parameters displayed in the following illustration is discussed in detail below.
Learn rate:
Controls the rate at which the model is updated after each training stage. This is also referred to as “shrinkage” in Friedman’s original articles.

The default is AUTO, and the allowable range is 0.0001 to 1.0 inclusive. AUTO is calculated as follows:

\[
\text{AUTO value} = \max(0.01, 0.1 \times \min(1, \text{nl}/10000))
\]

where \( \text{nl} = \text{number of LEARN records} \).

This default uses very slow learn rates for small data sets and uses 0.10 for all data sets with more than 10,000 records. We strongly recommend that you experiment with different learn rates, especially rates slower than the default for larger data sets. See [1] for more technical discussion.

High learn rates and especially values close to 1 typically result in overfit models with poor performance.

Values much smaller than .01 significantly slow down the learning process and might be reserved for overnight runs.

Subsample fraction:
This setting controls the proportion of data used for learning at each training cycle. A new random draw is conducted at each cycle, which not only speeds up the modeling time but also guards against overfitting, and explains occasional minor local increases in the learn error curve as a TreeNet run progresses.

The default is 0.5, and the allowable range is 0.01 to 1.0 inclusive.

It may be necessary to use values greater than the .5 default with small training files. Unlike the learn rate, using a sampling rate of 1.0 is not necessarily catastrophic, but values less than 1.0 are still strongly recommended. You may need to experiment to determine the best rate.

See [1] for more technical discussion.

Sampling rates that are too small can hurt accuracy substantially while yielding no benefits other than speed.

Influence trimming factor:
Influence trimming is a form of data selection designed to focus the TreeNet learning process on the most important data records. It is the mechanism by which suspicious data are ignored.

If you have no concerns about the quality of your data you might consider turning this feature off by setting the factor to 0.

It is always worth experimenting with a few values of this factor to see what works best. We recommend settings of 0, 0.9, and even 0.5.
The Influence trimming factor can vary between 0 and 1, with larger values generally having larger effects.

- Setting influence trimming to zero will disable this feature.
- See [1] for more technical discussion.

### M-Regression breakdown parameter:
This setting is ignored in binary logistic runs.

### Optimal Logistic Model Selection Criterion:
By selecting “Logistic Regression” you select a specific way to generate the individual TreeNet trees but you still have four ways to evaluate model performance. See the discussion on Best Models above for details.

Note that the “lift in the given percentile” and “misclassification rate” measures also require additional user’s input, namely the percentile and the threshold (both expressed as fractions between 0 and 1).

- The sequence of trees generated by TreeNet during the model-building phase is determined by control parameters such as the number of nodes and the learn rate. The model selection criterion determines the size of the model extracted from this sequence.
- Changing the model selection criterion does not change the actual model sequence. The only thing that can change is the optimal model size. Optimal lift models will generally be much smaller than the optimal CXE models in the same sequence.

### Number of trees to use:
A TreeNet model is essentially a collection of trees of a fixed size. New trees are created as the modeling process progresses. This option specifies how many trees should be grown before the modeling process ends.

- You should expect to grow hundreds or even thousands of trees to get a maximum performance model.
- The default setting for 200 trees is designed to facilitate interactive model selection and provide very rapid feedback. Once you have a good feel for your data and how you want to tackle your modeling effort, be prepared to allow for 500, 1000, 2000 or more trees.
- Be sure to use a slow learn rate when growing many trees.
- If the “optimal” model contains close to the maximum number of trees allowed (e.g., more than ¾ of the maximum) consider growing more trees.

### Maximum number of trees including restart continuations:
This parameter controls the largest possible number of trees that are allowed to be grown in any single interactive TreeNet model. You will rarely need to change this setting.
**Maximum nodes per tree:**
This setting controls the fixed size of each individual tree used in the TreeNet model. At each step, a fixed-size tree will be grown to correct the current prediction error.

- You can grow TreeNet models made up of two-node trees. These tiny trees (also known as “stumps”) can be surprisingly powerful and are well worth experimenting with.

- Two-node trees contain only a single variable and a single split. Because only one variable is ever involved in a model update, the models cannot detect interactions and thus can be described as main-effects additive models.

- More than two nodes are required to detect interactions and the default six-node tree appears to do an excellent job.

- Nothing prevents you from requesting trees with quite a few nodes (e.g., more than 12) but doing so is likely to undermine TreeNet’s slow-learning strategy. Large trees can also place unnecessary burdens on your computer’s memory resources.

**Minimum number of training observations in terminal nodes:**
This setting controls how small individual terminal nodes are allowed to be. In our example, the value setting of 10 indicates that no terminal nodes with fewer than 10 observations are allowed in the model.

When working with small training samples it may be vital to lower this setting to values such as five or even three.

**Minimum number of most-optimal models to save summary results for:**
If you generate a model with 1000 trees you actually have access to 1000 different models beginning with a model containing one tree, then a model containing two trees, etc. Each of these models potentially comes with a complete set of summary reports and graphs. To conserve resources we store details for only the best models but give you the option of producing extensive reports on as many different models as you would like to review.

This setting controls how many models will have complete summary results reported and saved. When this setting is set to one, the full summary results will only be saved for the single best model for each of the four optimality criteria. Settings above one will include additional next best optimal models uniformly distributed among the available optimality criteria.

- Setting this parameter to a large value will increase the running time for your models and possibly cause you to run out of memory.

- We recommend keeping this setting well below 10 for routine modeling.

**Regression loss criterion:**
This setting is ignored in the binary logistic regression runs.
**Model Setup – TreeNet – Defaults**

The group of buttons in the lower right corner of the TreeNet tab allows the user to specify new defaults for any subsequent TreeNet run. The defaults are saved in the TreeNet initialization file and persist from session to session.

- **[Std. Defaults]** Press this button to restore the original “factory” settings.
- **[Save Defaults]** Press this button to save the current settings as the new defaults.
- **[Recall Defaults]** Press this button to recall previously-saved defaults.

For example, you might decide that your typical TreeNet runs should use a 0.01 learn rate, 1000 trees, two-nodes (to force additive models), and the least squares loss criterion. Just set these parameters and press [Save Defaults]. In future sessions, TreeNet will automatically assume the new settings unless you manually override them.

**Model Setup – Save Grove**

Any time during model setup, the **[Save Grove...]** button can be used to save the currently active TreeNet model to a grove file. The grove file, a binary file that contains all the information about the TreeNet model, must be saved if you want to apply the model to a new data set or translate the model into one of the supported languages at a later time.

After the **[Save Grove...]** button is pressed, the **Specify Grove File** dialog window appears:

![Specify Grove File](image)

Type in the name of the grove file to be created and click the **[Open]** button.

- The grove file will be created on your hard drive only after the model building process is completely finished.
- You can also save the grove file from the Results window (see below) after the run is finished.

**Model Setup – Class Weights**

Class weights allow you to specify weights for each member of a class. For example, an observation from class 1 could count twice as much as an observation from class 0.
Class weights are distinct from individual case or record weights and both types of weights can be used at the same time.

The following three class weights options for the logistic binary model are available:

- **BALANCED** - As the name implies, this setting is intended to rebalance unequal class sizes. Automatic reweighting ensures that the sum of all weights in each class are equal, eliminating any need for manual balancing of classes via record selection or definition of weights.

- **UNIT** - This setting takes the relative sizes of the two classes as given and does not attempt to compensate by adjusting weights for the smaller class. This is the default setting and is recommended for the binary logistic regression.

- **SPECIFY** - Lets the user specify class weights. This is the most flexible option. You can set any positive values for each class.

While class weights can influence how a TreeNet model evolves and how it will perform in prediction, these weights will not influence any reports involving record or class counts. The class weights are “virtual” and are used to influence the model building process.

In most TreeNet runs, we recommend using UNIT class weights. This simplifies reading the output and usually produces superior results.

TreeNet Results
A TreeNet modeling run starts after the [Start] button is pressed. A progress report window appears. After the trees have been grown, the TreeNet Results window automatically appears:
Information available in the TreeNet results window includes the following information:

1. General information about run settings is displayed in the upper section of this window: training data file, target variable, number of predictors, requested tree size and the total number of trees to be grown, type of loss function.

2. TreeNet also reports the number of trees in the model at which the test set error criterion is optimized. For the current run, the optimal entropy model is attained when 146 trees are grown.

3. The lower part of the window displays the run history: the tree number on the horizontal axis and train/test error profiles on the vertical axis. You can switch among the four error profiles using the [Entropy], [Misclass], [ROC], and [Lift] buttons. Note that the test error curve lies slightly above the learn error curve and that they are in close agreement.

   A vertical green beam marks the optimal model with respect to the user-specified optimality criterion. The actual number of trees and the corresponding value of the criterion are reported just above the graph. In our example, the optimal model has a 0.297 average negative log-likelihood.

4. An additional profile can be requested by pressing the [Sample] button. The resulting curve shows the actual percentage of learn data used at each iteration.

   - The profile always starts at the "subsample fraction" (0.5 by default).
   - When “influence trimming” is allowed (by setting a non-zero value for influence trimming), the profile should decline to reflect the impact of record selection.

**Summary Reports – General**

The TreeNet Summary window is activated when the [Summary...] button is pressed in the TreeNet Results window. TreeNet provides variable importance information for models of all sizes and gains, misclassification, prediction success information and
dependency graphs for a selected subset of model sizes, always including the optimal models corresponding to each of the four evaluation criteria.

- You can specify the number of models for which the full results are available in the TreeNet tab of the Model Setup window.

You can select alternative models by using the arrow keys or making a left mouse click. Models without the full summary (variable importance only) will appear as dashed lines. You can jump directly among the models with the full summary available by using the combination of <Ctrl> key and the corresponding arrow keys or mouse clicks.

### Summary Reports – Gains

The **Gains Chart** tab is located in the TreeNet Reports window. The gains chart and the corresponding table are often used to judge the overall performance of a model on a given data set. It appears as follows:

The following table defines the columns of this report.

### Gains Tables

**TARGET** denotes the actual response (0 or 1, YES or NO, etc) and it must be known for each case.

**RESPONSE** denotes the scores predicted by the given model.

**K** denotes the predetermined number of bins.

1. Sort the data by descending **RESPONSE**.
2. Divide the data into **K** bins such that each bin has an approximately equal number of weighted cases.

**% of Bin Class 1** – the percentage of 1s in the given bin.

**% Class 1** – the weighted number of class 1 cases in the bin divided by the total weighted count of class 1 cases.
**Cum % Class 1** – the simple cumulate of the % Class 1 column. This is also known as **sensitivity**, assuming that the classification threshold is set at the corresponding bin boundary.

**Cases in Bin** – the weighted number of cases in the given bin.

% Pop – the weighted number of cases in the given bin divided by the total weighted number of cases.

**Cum % Pop** – the simple cumulate of the % Pop column.

**Cum Lift** – Cum % Class 1 divided by Cum % Pop.

**Lift Pop** – % of Bin Class 1 divided by % Pop.

The graph to the left displays:

- When the [Gains] button is pressed – **Cum % Class 1** versus **Cum % Pop**.
- When the [Lift] button is pressed – **Lift Pop** versus **Cum % Pop**.
- When the [Cum. Lift] button is pressed – **Cum Lift** versus **Cum % Pop**.
- When the [ROC] button is pressed – **Cum % Class 1** versus (1 – **specificity**) (the specificity is defined similar to the sensitivity but with respect to the class 0).

The number of bins can be changed using the control.

The class in focus (the class that is referred to as 1s) can be selected using the **Class** selection box.

- The ROC Integral value shown when the [ROC] button is pressed is an approximation based on the binned scores. The exact value of the ROC integral is reported on the ROC profile.
- The ROC curve, while similar to the gains curve, has a number of theoretical advantages, especially when considering the model performance on the dominant class (in which case the gains curve may degenerate into the 45-degree line).

When the test data are available, both learn and test gains and ROC charts can be displayed using either the [Learn] or the [Test] buttons.

One may use the following simple interpretation of gains. Assume that a scoring model is built to predict potential responders to a certain mail order. If every member of the population is given the offer, all responders would be targeted. Now assume that we would actually like to send mail offers to only 10% of the population. When no model is available and offers are mailed purely randomly, we should expect to collect only about \(\frac{1}{10}\) of the responders. In the presence of a model, instead of random mailing, one could send offers to the 10% of the population in the top bin (the highest scores). In this case the expected percent of covered responders would be **Lift Pop** times \(\frac{1}{10}\). The interpretation is similar for the other bins and cumulative lift.

- This is a special case (0/1 response) of the regression gains described earlier on.
- Gains based on the test data are usually more realistic than gains based on the training data.
Test gains are not available when independent testing is disabled (Test tab in the Model Setup).

Summary Reports – Variable Importance

The TreeNet Summary window also contains the Variable Importance tab.

The raw variable importance score is computed as the cumulative sum of improvements of all splits associated with the given variable across all trees up to a specific model size. Note that, smaller models will typically involve fewer variables, thus limiting the number of variables with non-zero importance scores, whereas larger models will often utilize a large number of predictors.

The relative importance Score, expressed on a scale of 0 to 100, simply rescales the importances so that the most important variable always gets a score of 100. All other variables are rescaled to reflect their importance relative to the most important. In our example, the continuous predictors X1, X2, X3, X4 and the categorical predictors Z1$, Z2$ were clearly identified as important. The remaining predictors are reported to have rather small contributions and most probably reflect random noise patterns.

This conclusion is further supported by the analysis of contribution plots described in a separate chapter.

When the [Importance] button is pressed, the following display is shown:

The rows in the table can be re-sorted differently by clicking on the corresponding column header. Thus, you can sort the variables by name and change the direction of the sort (ascending or descending).

In addition to the variable importance, this tab gives access to the summary single stats display ([Single Stats] button). This report simply collects all generated Single Stats Plots on one display. A more detailed explanation is provided in subsequent chapters.
Summary Reports – Misclassification

The Misclassification tab contains summary misclassification rates by class for the train and test data (if available).

The prediction success table displayed below shows the following information:

<table>
<thead>
<tr>
<th>ACTUAL CLASS</th>
<th>Total number of cases in the class</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL CASES</td>
<td></td>
</tr>
<tr>
<td>PERCENT CORRECT</td>
<td>Percent of cases for the class that were classified correctly</td>
</tr>
<tr>
<td>-1 N=n</td>
<td>Number of Class -1 cases classified in each class, where n is the total number of cases predicted (correctly or incorrectly) as Class -1</td>
</tr>
<tr>
<td>1 N= n</td>
<td>Number of Class 1 cases classified in each class, where n is the total number of cases predicted as Class 1.</td>
</tr>
</tbody>
</table>

In our example, the model is about 84% accurate in class 1, and 94% accurate in class -1.
To switch to the learn sample prediction success table, click on [Learn]. Similarly, to view row or column percentages rather than counts, click the [Row %] or [Column %] buttons.

**Other Controls**

The [More Trees...] button can be used to further evolve the model by growing more trees. This is useful when the optimal model contains a large number of trees relative to the number grown. If you allowed for 200 trees and the optimal model contains more than 150, you should consider growing more trees. This process can be repeated many times. More trees are available immediately after the current run is complete, but may not be available after you perform some other actions such as opening a saved grove file.

This feature is not available when cross validation is used.

The [Save Grove...] button is used to save the model results to a grove file on the hard drive. The saved grove can later be opened for viewing (File-Open>Open Grove... menu) or used for scoring and translation (see later chapters).

The [Create Plots...] and [Display Plots...] buttons are used in plot operations discussed later.
Classification Models

This chapter covers in depth how to use TreeNet to build classification models.
General Remarks on Classification Runs

Most classification models are developed for binary response problems for which we try to predict a 0/1 or YES/NO response. If you need to generate such models please refer to the previous chapter where the preferred method for such modeling is discussed.

If you are trying to predict a class (more than two class) outcome with TreeNet, then you must use the methods discussed in this chapter. You can also use this chapter’s methods for the two-class problem if you insist.

The classification mode is used to model multi-level discrete outcomes \( (K \text{ levels}) \). For example, we might want to classify potential borrowers into three groups: 1) Never Late, 2) Slow payers, and 3) Defaulters. The levels do not have to follow any inherent order and usually they will not. There can be any number of target levels but there are practical reasons to avoid using TreeNet for much more than 50 levels. If you must predict for more than 50 levels, see the discussion of this topic at the end of this chapter.

The TreeNet classification model is inherently structured to predict a probability for each of the possible outcomes, and naturally supports multi-level modeling. For three target classes A,B,C, TreeNet will develop three separate binary models:

- A vs (B or C)
- B vs (A or C)
- C vs (A or B)

each making use of its own preferred variables and with its own accuracy. The three separate models are ultimately combined into a single overall model but the separate nature of the components allows for different variables to be important for each level.

See [1] for more technical discussion.

For any given model sequence, TreeNet reports two separate error profiles: cross-entropy (synonym for the multinomial likelihood) and the overall misclassification error using the majority class assignment rule. The optimal model is always defined in terms of the likelihood. The classification profile is only given for reference and may degenerate to a flat line on problems with a single dominant class (because the majority rule will keep assigning all records to the single class).

When no independent testing is specified (exploratory runs), the optimal model is always set to the largest number of trees.

It is possible to manually select a model of different size for subsequent scoring and translation.

Classification mode is designed to handle true multinomial classification problems. While binary targets can be naturally used in classification runs, logistic regression runs (described in the previous chapter) are specifically tuned to exploit special characteristics of the binary case and provide wider functionality and reporting.
On binary targets, the model sequences, including the likelihood and classification profiles, of the logistic regression runs should be similar to the classification runs.

**Model Setup – Model**

Classification modeling in TreeNet is controlled from the **Model Setup** dialog. This window automatically appears after a data file has been successfully accessed. You may also request the **Model Setup** dialog using the **Model–Construct Model…** menu or by clicking the toolbar icon.

- A data set must be currently open for this operation to work.
- The Model menu item is accessible when the Classic Output window is active.

In the **Model Setup–Model** tab we specify our core model variables. Variables specified in this dialog include a target (dependent), predictors (independent), categorical designation, and optionally a single case weight.

The selection of variables and tree type for our model is as follows:

1. Choose the **Classification** radio button in the section titled **Tree Type**.
2. Select the target variable by placing a check mark in the corresponding variable check box under the **Target** column.
   - There can be only one target at a time in TreeNet.
   - You may change the **Sort**: order to **Alphabetically** when it is convenient.
3. Select potential predictors by placing check marks in the corresponding variable's check boxes under the **Predictor** column.
   - You may highlight multiple variables by using a left mouse click and/or the <Shift> or <Ctrl> keys while selecting in the Variable Name column. Once selected, you may either check or uncheck the corresponding check marks all at once for the selected variables by using the check box at the bottom of the appropriate column.
4. Optionally, select a weight variable using a check box under the **Weight** column.
   - There can be only one weight variable at a time.
5. Indicate which predictors are categorical by placing check marks in the **Categorical** column for the corresponding variables.

The **Model** tab will now appear as follows:
Correctly declaring variables as categorical is less important for TreeNet than for CART or other modeling methods. TreeNet is capable of uncovering the true relationship between the target and a categorical predictor whether the declaration is made or not. However, some graphs will be much easier to read if the declarations are made. For high level categorical predictors, however, it may be advantageous to treat such predictors as numeric. This is discussed further in a later chapter.

Consider letting TreeNet treat a high-level categorical variable as a continuous predictor if it has more than about 30 levels. This is unlikely to hurt the model and may actually improve results quite a bit. This is possible only if the HLC is coded as numeric rather than text.

All character variables (variables ending in $) will be automatically checked as categorical and cannot be unchecked.

Using high-level categorical variables may significantly increase the running time and impose severe memory requirements. Make sure that a continuous variable has not been accidentally checked as categorical.

TreeNet needs to build a separate model for every level of the multi-class target. This means that if you have a 50-level target you will be building 50 separate models. While we have successfully run models for 250-level targets they can be quite time consuming.

**Model Setup – Testing**

The Testing tab indicates how TreeNet models are to be evaluated. The following six testing modes are available in TreeNet.
- **No independent testing – exploratory tree.** In this mode no external evaluation will be performed. The entire data set will be used for training; therefore, the final results will include only training data and the best model is almost the largest.

- **Fraction of cases is selected at random for testing.** The specified fraction of data will be set aside for testing. Final results will be reported for both train and test data.

- **Test sample is contained in a separate file.** The data for testing will be contained in the specified file. Use this mode when you want to use one data file for training and a separate data file for testing. When this mode is activated, a file selection dialog window will appear.

- **V-fold Cross Validation.** Usually used with small datasets when one cannot afford to reserve some data for testing. The entire dataset is used for learning purposes and then is partitioned into ten bins. At each fold in 10-fold CV, nine bins are used as a training set and the remaining bin is used as a test set. After all 10 folds are completed, the test results from each fold are averaged to get a fair test estimate of the all-data model performance.

- **Variable determines cross-validation samples.** Same as the above option but the user has full control over bin creation by introducing a special variable with integer values 1 through 10 to mark the bins. You can opt for a different number of folds.

  ♦ Use extreme care when creating your own CV bins. Each bin should be of the same size and balanced on the target variable.

  ♦ 10-fold cross-validation runs are on average 10 times slower than using a test sample. For small data sets you may not even notice the difference but for large data sets using cross validation will substantially increase your run times.

- **Variable separates learn and test samples.** This is the most flexible mode. All you need is a binary variable coded as 0 or 1. When its value is 0, the corresponding observation will go into the learn data; otherwise, the observation will go into the test data.

  ✤ Use “Cross Validation” when the supply of data is limited.

  ✤ Use the indicator variable mode when you want complete control over which part of the data is used for testing. You may also create multiple indicator variables in the same data set to experiment with different random seeds or learn/test ratios.

In our example we are using **A variable separates learn and test samples.** The sample data set already has a test indicator variable, called T. The **Testing** tab now appears as follows:
Model Setup – TreeNet

The Model Setup–TreeNet tab contains important parameter settings used in the model-building process. This setup dialog allows the user to control multiple aspects of the TreeNet modeling process. Each of the parameters displayed in the following illustration is discussed in detail below.

**Learn rate:**
Controls the rate at which the model is updated after each training stage. This is also referred to as “shrinkage” in Friedman’s original articles.

The default is AUTO, and the allowable range is 0.0001 to 1.0 inclusive. AUTO is calculated as follows:
\[ AUTO \text{ value} = \max(0.01, 0.1 \times \min(1, nl/10000)) \]
\[ \text{where nl = number of LEARN records.} \]

- This default uses very slow learn rates for small data sets and uses 0.10 for all data sets with more than 10,000 records. We strongly recommend that you experiment with different learn rates, especially rates slower than the default for larger data sets. See [1] for a more technical discussion.

- High learn rates and especially values close to 1 typically result in overfit models with poor performance.

- Values much smaller than .01 significantly slow down the learning process and might be reserved for overnight runs.

**Subsample fraction:**

This setting controls the proportion of data used for learning at each training cycle. A new random draw is conducted at each cycle, which not only speeds up the modeling time but also guards against overfitting, and explains occasional minor local increases in the learn error curve as a TreeNet run progresses.

The default is 0.5, and the allowable range is 0.01 to 1.0 inclusive.

- It may be necessary to use values greater than the .5 default with small training files. Unlike the learn rate, using a sampling rate of 1.0 is not necessarily catastrophic, but values less than 1.0 are still strongly recommended. You may need to experiment to determine the best rate.

- See [1] for a more technical discussion.

- Sampling rates that are too small can hurt accuracy substantially while yielding no benefits other than speed.

**Influence trimming factor:**

Influence trimming is a form of data selection designed to focus the TreeNet learning process on the most important data records. It is the mechanism by which suspicious data are ignored.

- If you have no concerns about the quality of your data you might consider turning this feature off by setting the factor to 0.

- It is always worth experimenting with a few values of this factor to see what works best. We recommend settings of 0, 0.9, and even 0.5.

The Influence trimming factor can vary between 0 and 1, with larger values generally having larger effects.

- Setting influence trimming to zero will disable this feature.

- See [1] for a more technical discussion.
M-Regression breakdown parameter:
This setting is ignored in classification runs.

Optimal Logistic Model Selection Criterion:
This setting is currently ignored in classification runs. The optimal model is always defined in terms of the test cross-entropy (likelihood).

Number of trees to use:
A TreeNet model is essentially a collection of trees of a fixed size. New trees are created as the modeling process progresses. This option specifies how many trees should be grown before the modeling process ends.

- You should expect to grow hundreds or even thousands of trees to get a maximum performance model.
- The default setting for 200 trees is designed to facilitate interactive model selection and provide very rapid feedback. Once you have a good feel for your data and how you want to tackle your modeling effort, be prepared to allow for 500, 1000, 2000 or more trees.
- Be sure to use a slow learn rate when growing many trees.
- If the “optimal” model contains close to the maximum number of trees allowed (e.g., more than ¾ of the maximum) consider growing more trees.

Maximum number of trees including restart continuations:
This parameter controls the largest number of trees allowed to be grown in any single interactive TreeNet model. You will rarely need to change this setting.

Maximum nodes per tree:
This setting controls the fixed size of each individual tree used in the TreeNet model. At each step, a fixed-size tree will be grown to correct the current prediction error.

- You can grow TreeNet models made up of two-node trees. These tiny trees (also known as “stumps”) can be surprisingly powerful and are well worth experimenting with.
- Two-node trees contain only a single variable and a single split. Because only one variable is ever involved in a model update, the models cannot detect interactions and thus can be described as main-effects additive models.
- More than two nodes are required to detect interactions and the default six-node tree appears to do an excellent job.
- Nothing prevents you from requesting trees with quite a few nodes (e.g., more than 12) but doing so is likely to undermine TreeNet’s slow-learning strategy. Large trees can also place unnecessary burdens on your computer’s memory resources.

Minimum number of training observations in terminal nodes:
This setting controls how small individual terminal nodes are allowed to be. In our example, the value setting of 10 indicates that no terminal nodes with fewer than 10 observations are allowed in the model.

When working with small training samples it may be vital to lower this setting to values such as five or even three.

**Minimum number of most-optimal models to save summary results for:**

If you generate a model with 1000 trees you actually have access to 1000 different models beginning with a model containing one tree, then a model containing two trees, etc. Each of these models potentially comes with a complete set of summary reports and graphs. To conserve resources we store details for only the best models but give you the option of producing extensive reports on as many different models as you would like to review.

This setting controls how many models will have complete summary results reported and saved. When this setting is set to one, the full summary results will be saved only for the single best model per each optimality criterion (likelihood and misclassification error). Settings above one will include additional next best optimal models uniformly distributed among the available optimality criteria.

⚠️ Setting this parameter to a large value will increase the running time for your models and possibly cause you to run out of memory.

⚠️ We recommend keeping this setting well below 10 for routine modeling.

**Regression loss criterion:**

This setting is ignored in the classification runs.

---

**Model Setup – TreeNet – Defaults**

The group of buttons in the lower right corner of the TreeNet tab allows the user to specify new defaults for any subsequent TreeNet run. The defaults are saved in the TreeNet initialization file and persist from session to session.

- **[Std. Defaults]** Press this button to restore the original “factory” settings.
- **[Save Defaults]** Press this button to save the current settings as the new defaults.
- **[Recall Defaults]** Press this button to recall previously-saved defaults.

For example, you might decide that your typical TreeNet runs should use a 0.01 learning rate, 1000 trees, two-nodes (to force additive models), and the least squares loss criterion. Just set these parameters and press [Save Defaults]. In future sessions, TreeNet will automatically assume the new settings unless you manually override them.

---

**Model Setup – Save Grove**

Any time during model setup, the **[Save Grove...]** button can be used to save the currently active TreeNet model to a grove file. The grove file, a binary file that contains all the information about the TreeNet model, must be saved if you want to apply the
model to a new data set or translate the model into one of the supported languages at a later time.

After the [Save Grove...] button is pressed, the Specify Grove File dialog window appears:

![Specify Grove File dialog window]

Type in the name of the grove file to be created and click the [Open] button.

- The grove file will be created on your hard drive only after the model building process is completely finished. This is because it is only after the [Start] button is pressed that the GUI issues the required sequence of commands to be executed by the TreeNet engine.

- You can also save the grove file from the Results window (see below) after the run is finished.

**Model Setup – Class Weights**

Class weights allow you to specify weights for each member of a class. For example, an observation from class A could count twice as much as an observation from class B and half as much as an observation from class C. Class weights are distinct from individual case or record weights and both types of weights can be used at the same time.

The following three class weights options are available for the logistic binary model:

- **BALANCED** - As the name implies this setting is intended to rebalance unequal class sizes. Automatic reweighting ensures that the sum of all weights in each class are equal. This eliminates any need for manual balancing of classes via record selection or definition of weights.

- **UNIT** - This setting takes the relative sizes of the classes as given and does not attempt to compensate by adjusting weights for the smaller classes. This is the default setting.

- **SPECIFY** - Lets the user specify class weights. This is the most flexible option. You can set any positive values for each class.

- While class weights can influence how a TreeNet model evolves and how it will perform in prediction, these weights will not influence any reports involving record
or class counts. The class weights are “virtual” and are used to influence the model building process.

In classification runs it may be vital to use BALANCED class weights.

Model Setup – Costs

By default, TreeNet uses the majority or plurality rule to make class assignments. In other words, each case is assigned to the class having the largest predicted probability for that record. However, for many cases, regardless of their predictor values, the probability of belonging to one of the smaller classes may remain low. For example, suppose we are predicting the choice a mobile phone consumer makes from a menu of 20 hand sets. If one of those hand sets is rarely chosen, it may never be predicted as the choice with a large enough probability.

The costs are designed to introduce a certain asymmetry into the class assignment process. Classes with higher costs will no longer be required to have the largest predicted probability to get the class assignment. The higher the costs, the smaller the resulting within-class misclassification rate. Of course, this is usually accompanied by accuracy reduction in the other classes.

Unlike class weights, costs do not change the underlying model sequence. Costs can thus be viewed as a passive post-processing reporting feature.
The impact of costs will be illustrated later in this chapter.

**TreeNet Results**

A TreeNet modeling run starts after the [Start] button is pressed. A progress report window appears. After the trees have been grown, the TreeNet Results window automatically appears:

Information available in the TreeNet results window includes the following information:

1. General information about run settings is displayed in the upper section of this window: training data file, target variable, number of predictors, requested tree size and the total number of trees to be grown, type of loss function.
2. TreeNet also reports the number of trees in the model at which the test set error criterion is optimized. For the current run the optimal entropy model is attained when 200 trees are grown.

3. The lower part of the window displays the run history: the tree number on the horizontal axis and train/test error profiles on the vertical axis. You can switch between the four error profiles using the [Entropy] and [Misclass] buttons. Note how the test likelihood goes slightly above the learn likelihood and how they both agree.

   - The misclassification profile is only shown when the true test data are available; it does not contain the train data performance.

   The green line marks the optimal model with respect to the test likelihood. The actual number of trees and the corresponding value of the likelihood are reported above the line. In our example, the optimal 192-tree model has 0.654 average negative log-likelihood.

4. An additional profile can be requested by pressing the [Sample] button. The resulting curve shows the actual percentage of learn data used at each iteration.

   - The profile always starts at the “subsample fraction” (0.5 by default).

   - When the “influence trimming” is allowed (non-zero value), the profile should go down to reflect the impact of influence trimming in terms of sample size reduction.

Summary Reports – General

The TreeNet Summary window is activated when the [Summary...] button is pressed in the TreeNet Results window. TreeNet provides variable importance information for all models and gains, misclassification, and prediction success information for a selected subset of models, which always includes the optimal models corresponding to the likelihood and classification criteria.

   - You can specify the number of models for which the full results are available in the TreeNet tab of the Model Setup window.

You can select alternative models by using the arrow keys or making a left mouse click. Models without the full summary (variable importance only) will appear as dashed lines. You can jump directly among the models with the full summary available by using the combination of <Ctrl> key and the corresponding arrow keys or mouse clicks.

Summary Reports – Gains

The Gains Chart tab is located in the TreeNet Reports window. The gains chart and the corresponding table are often used to judge the overall performance of a model on a given data set. It appears as follows:
The calculation of gains is described in great detail in the previous chapter. Here we should just mention that in multinomial modeling one should first put one of the target classes in focus and treat the remaining classes as the aggregate opposite class. This is because the whole concept of gains and ROC is inherently binary.

The class in focus (the class that is referred to as 1s) can be selected using the Class selection box.

- The ROC Integral value shown when the [ROC] button is pressed is an approximation based on the binned scores. The exact value of the ROC integral is not available in classification runs.

- The ROC curve, while similar to the gains curve, has a number of theoretical advantages, especially when considering the model performance on the dominant class in focus (in which case the gains curve may degenerate into the 45-degree line).

When the test data are available, both learn and test gains and ROC charts can be displayed using either the [Learn] or the [Test] buttons.

- Gains based on the test data are usually more realistic than gains based on the training data.

- Test gains are not available when independent testing is disabled (Test tab in the Model Setup).

**Summary Reports – Variable Importance**

The TreeNet Summary window also contains the Variable Importance tab.

The raw variable importance score is computed as the cumulative sum of improvements of all splits associated with the given variable across all trees up to the current model size. In particular, smaller models will have fewer variables with non-zero importance scores whereas larger models will generally tend to utilize almost all the available predictors.
The relative importance **Score**, expressed on the percentage scale, is simply the raw score scaled to the largest raw score. It provides a relative measure of each variable’s contribution into the model’s predictive power.

An important thing to remember is that in K-level classification runs TreeNet internally builds K separate model sequences – one sequence for each class in focus. As a result, K fully-functional variable importance tables are available, one for each target class. For example, some variables can be important in separating class 1 from the rest and not important in separating class 2 from the rest.

When the [Importance] button is pressed, the following display is shown:

The rows in the table can be sorted differently by simply clicking on the corresponding column header.

You can either view the combined variable importance scores averaged across all K runs (“Average all classes” is checked) or request individual variable importance tables for target class to have a more in-depth look.

Again, a TreeNet model mostly evolves around the continuous predictors X1, X2, X3, X4, and the categorical predictors Z1$ and Z2$.

Alternatively, you can highlight a variable in the importance list and then switch to the “By class for:” radio button to see the relative contribution of the selected variable into individual class separation. For example, requesting such a report for Z1$ gives the following table:
It follows from here that $Z1\$ is mostly contributing to separating levels 3 and 4 and not levels 1 and 2.

In addition to the variable importance, the tab gives access to the summary single stats display ([Single Stats] button). This report simply collects all generated Single Stats Plots on one display. A more detailed explanation is provided in subsequent chapters.

**Summary Reports – Misclassification**

The **Misclassification** tab contains summary prediction success tables for the train and test data (if available).

**Summary Reports – Prediction Success**

The **Prediction Success** tab contains the summary prediction success tables for the train and test data (if available) in a slightly different visual form.

The prediction success table displayed below shows the following information:
In our example, the model is about 90% accurate in class 1, 73% accurate in class 2, etc.

To switch to the learn sample prediction success table, click on [Learn]. Similarly, to view row or column percentages rather than counts, click the [Row %] or [Column %] buttons.

**Costs – Illustration**

We have noticed that class 2 has the lowest accuracy in the current run. Let us change the costs to improve the accuracy in this class:
Now we have about 81% accuracy in class 2. The accuracy in the other classes dropped somewhat, as expected.

The new run has the identical model sequence, as can be seen from the likelihood profile and the optimal model definition. However, the classification error profile has changed reflecting new cost-adjusted class assignments based on the same original probability scores.

Other Controls
The [More Trees...] button can be used to request more trees. This is useful when the optimal model has the largest number of trees and it is clear that the error profile keeps improving. This operation is only available immediately after the current run until a major action item takes place (for example, request to load another grove file, issuing the NEW command, starting another run, etc.)
This feature is not available when cross validation is used.

The [Save Grove...] button is used to save the model results to a grove file on the hard drive. The saved grove can later be opened for viewing (File-Open>Open Grove… menu) or used for scoring and translation (see later chapters).

The [Create Plots...] and [Display Plots...] buttons are used in plot operations discussed later.
This chapter explains how to apply a model to new data.
Using TreeNet Models for Predicting

This section describes how to score or predict using a TreeNet model. When scoring, we are predicting values for a target variable using either entirely new previously-unseen data or old (e.g., learn/training) data.

The process of using a TreeNet to predict a target variable is known as APPLYing data (see the MART APPLY command), or scoring data. Each observation is processed case-by-case, using every tree. The splitting criteria for each tree are applied and the case is given a predicted response (regression models), or a predicted probability and final class assignment (binary logistic and classification models).

The process of using TreeNet to obtain a piece of code that can be used externally to score data in a different programming environment is called Translate Model.

Scoring usually results in an output data file being created.

In the regression runs, TreeNet can include the following special variables in the output dataset:

- **RESPONSE** — predicted continuous response
- **<TARGET>** — the original target variable, if available
- **RESIDUAL** — difference between the target variable and RESPONSE unless the target variable is not present in the data set to be scored
- **IMPUTED** — indicator whether at least some of the modeling variables contained missing values and had to be imputed (Possible character values are “yes” and “no”.)

In the binary logistic and classification runs TreeNet can include the following special variables in the output dataset:

- **RESPONSE** — predicted categorical response (class assignment)
- **<TARGET>** — the original target variable, if available
- **CORRECT** — dummy indicator whether the RESPONSE agrees with the original target unless the target is not present in the dataset to be scored
- **IMPUTED** — indicator whether at least some of the modeling variables contained missing values and had to be imputed (Possible character values are “yes” and “no”.)
- **PROB<N>** — predicted probability for each class <N> takes consecutive integer values starting with 1. The correspondence between the classes and values of <N> is given in the part of the TreeNet classic output with the title “Ordering of the predicted probabilities in the SAVE dataset.”
In addition, the user may request up to 50 ID variables (see below) and all modeling variables (predictors) to the output dataset ("Include model information" checkbox below).

**Model–Score Data**

To apply a TreeNet model to new data, the model must first be saved into a grove file prior to proceeding with the scoring process (see **Model Setup–Save Grove**). Once a grove file is generated and saved, the scoring can proceed.

- It is also possible to score data on the fly using the currently-opened grove. However, at the end of the current session this information will be lost unless the grove is saved.

The **Score Data** dialog box can be activated by any of the following:

- Pressing the **Score…** button in the “Model Setup” window.
- Using **Model – Score Data…** menu
- Pressing the icon on the toolbar

Use the following sequence of steps to score data:

1. Specify the data file you want to score in the **Data file:** entry by pressing the **[Select]** button and following the standard file selection dialog.
   - You don’t need to do this if you want to score the currently-opened modeling file.
2. Similarly, specify the grove file to be used for scoring in the **Grove** entry.
   - When you request scoring while having the current grove active, the entry is automatically filled in. This allows you to do a quick “on the fly” scoring of the data.
3. Select the output data set in the **Save results to a file** entry by pressing the **[Select]** button. This activates the standard Save File dialog.
   - When DATABASE DRIVERS mode is on, you may request saving to any of the multiple supported data formats.
4. Decide whether you want to activate one of the following options:
   - **Include Model Info** – when this option is checked, all variables used in scoring will be automatically included in the output file.
   - **Include predicted probabilities** – when this option is checked, the predicted probabilities will also be written to the output file for the binary logistic and classification models.
   - This option is ignored in regression runs.
   - **Single precision** – for SYSTAT datasets only, if checked – all numeric values will be saved in single precision (4 bytes) instead of double precision (8 bytes default). This may reduce data set size by as much as 50%.
5. In the subsequent section select the desired model size (number of trees). By default, the optimal number of trees will be used. Press the **[Select…]** button to
specify a different model size. You can always return to the default size by pressing the [Optimal Tree] button.

6. Select the ID variables, if any. Simply highlight the corresponding variables in the variable selection list and click the corresponding [Select...] button. This option is also useful when you only want to include a small subset of the original variables to the output dataset. Make sure that in this case the “Include Model Info” checkbox is not checked.

7. Select the weight variable, if any.
   - The weight variable will have no effect on the output scores. It will only change the counts reported in the GUI summary reports and the classic output.

8. Use a special case selection dialog available by pressing the [Select Cases...] button if you want to score only part of the data.

9. Click [OK] to start the scoring process. The following scoring progress window appears:
Scoring Results

After scoring is finished, the scoring results dialog window is displayed and an output data file is created. Once this process is completed, a TreeNet Apply results dialog opens and a text report appears in the Classic Output window. The content of both the GUI and text output for an Apply run will vary depending on whether the target variable is continuous or categorical and whether you are using new or training data. The TreeNet Apply dialog for an Apply run using a categorical target variable with a learn and test sample is displayed below.

The window may contain Gains information (when the target is not missing) and a Prediction Success table (not included for regression models).

- Both reports will be affected when a weight variable is supplied.

Translate—Exporting TreeNet Model Information

TreeNet includes the ability to export the model information contained within the binary grove file (discussed earlier in this chapter), including splitting rules, to SAS®-compatible and C programming languages. The files containing the exported code can be used outside of TreeNet for scoring data.

- Translation mode is disabled in the evaluation versions of TreeNet. Contact Salford Systems if you want to see working examples.

To translate a TreeNet model, the model must first be saved into a grove file prior to proceeding with the scoring process (see Model Setup—Save Grove). Once a grove file is generated and saved, the translation can proceed.

- It is also possible to translate data on the fly using the currently-opened grove. However, at the end of the current session this information will be lost unless the grove is saved.

The Model Translation dialog box can be activated by any of the following:
Using **Model – Translate Model**... menu

- Pressing the icon on the toolbar

Use the following sequence of steps to translate a TreeNet model:

1. Specify the grove file you want to score in the **Grove** entry by pressing the [Select] button and following the standard file selection dialog.
   - When you request translation while the current grove is active, the entry is automatically filled in. This allows you to do a quick “on the fly” translation of the model.

2. Choose the translation language under **Translation Options**.
   - SAS-compatible language mode has additional SAS-specific options.

3. Select the output file in the **Save Output To file** entry by pressing the [Select] button. This activates the standard Save File dialog.

4. Click **[OK]** to start the translation process.

The resulting code appears both in the “Classic Output” and the output file you have specified.
Working with Plots

This chapter explains how to generate and interpret TreeNet plots.
Plots–Introduction

A TreeNet model usually contains hundreds to thousands of trees, making any attempt to understand the model by inspecting individual trees impossible. To assist model interpretation, TreeNet offers graphs displaying how each variable affects the predicted outcomes. The 2D plots graph the predicted response against a single predictor and the 3D plots graph the response surface for more than a pair of predictors.

By default, TreeNet automatically generates 2D and 3D plots for the top three most important variables. This default setting can be changed in the TreeNet tab of the Edit-Options… dialog: you can either allow or disallow automatic plot generation and increase the number of top important predictors used in plots.

- Command-line equivalents will be MART PLOTS and MART MV.
- About half the time needed to complete a TreeNet analysis is devoted to preparing these graphs. You can save quite a bit of time by disabling them until you actually plan to study them.
- Graphs are not necessary if all you are doing is searching for an accurate model and experimenting with various TreeNet control settings. Disabling automatic plot generation can cut run times dramatically.

Automatically-generated plots become part of a grove and can be viewed at any time by pressing the [Display Plots...] button.

Immediately after the current model is built, TreeNet remains in a special waiting state called the “TreeNet Loop.” While in this state, the following operations can be performed an unlimited number of times:

- Creating new plots using the [Create Plots...] button. Optionally, you may delete the existing queue of plots.
- Requesting more trees using the [More Trees...] button (not allowed for cross-validation runs).
- Saving the current content into a grove file on the disk using the [Save Grove...] button. Thus, you can save multiple grove files containing different collections of plots.

Newly-created plots also become part of a grove and can be viewed at any time by pressing the [Display Plots...] button.

The following actions force TreeNet to exit the TreeNet loop so that the above operations are no longer available:

- Starting a new run.
- Issuing the NEW command.
- Loading another grove through File-Open>Open Grove…
- Data scoring.
• Model translation.

Plots–Display Plots
When the [Display Plots...] button is pressed, the TreeNet Plot Selection window will appear:

There are four main categories:

• One-variable dependence plots. Here you will find a collection of 2D plots based on one predictor at a time.

• Two-variable dependence plots. Here you will find a collection of 3D plots based on a pair of predictors at a time.

• One-variable stats.

• Two-variable stats.

The last two categories contain information about the degree of additive versus multiplicative contribution of the corresponding plot.

⚠️ Stats are only available for automatically generated plots. Because they have a rather narrow scope, their use has been deprecated and is no longer supported for plots requested during the TreeNet loop.

⚠️ See [1] for an in-depth description of stats.

Each category can be expanded first to the model sizes and then to the individual plots. Simply double click on the plot in order to see it. For example, clicking on X1 will result in the following 2D display:
You can switch between the [Line] and the [Scatter] representations of the plot.

By default, 2D plots are based on a randomly-selected 200 observations.

Use the MART PP command to change this default.

For categorical variables, the plot will be displayed as a bar chart.

Clicking on Z2$,X1 will result in the following 3D display:
You can manipulate the plot rotation buttons to get a better perspective. In addition, 2D slices can be requested using the [Slice] and [All Slices] buttons. Finally, alternative displays can be obtained by combining the [Mesh], [Shaded], [Contours], and [Zones] buttons.

**Plots–Create Plots**

While in the TreeNet loop, you may request creation of additional 2D and 3D plots for any specified variable or group of variables. You can also change the model size (number of trees) for which the plots are to be generated and clear the existing collection of plots. At any moment, the existing collection of plots can be saved as part of a grove file to be viewed later on.

When the [Create Plots...] button is pressed, the Create Plots window will appear:

![Create Plots Window](image)

**Data file:** section simply reports the currently-open data file. Likewise, the **Grove** section reports the current TreeNet grove.

In the section named **"Select Number of Trees Used for Scoring,"** choose the model size (number of trees) for which to generate plots. Click the [Optimal Tree] button to request the optimal model.
Select “One Variable Dependence” and/or “Two Variable Dependence” to request 2D and/or 3D plots.

You will find the list of current model variables sorted by their importance on the left side of the “Select Variables to Plot” section. Use a left mouse click to highlight a variable or a group of variables which you are interested. Then press the [Select] button to include the selected variables in the right panel.

Alternatively, put a checkmark in the box called “Use top <N> variables” and specify the number of variables you wish to include.

Similarly, you may highlight the currently-selected variables in the right panel and then click the [Remove] button to delete them from the selection list.

Optionally, press the [Clear Existing] button to erase all existing plots. Otherwise, the new plots will be added to the existing collection of plots.

Finally, press the [Create Plots] button to generate the plots you requested. After some delay, TreeNet Plot Selection will appear. You can now view the plots or save them in a grove. This operation can be repeated many times.

[Screenshots – a few plots examples]
Working with Command Language

This chapter provides insight into the essentials of TreeNet configuration and gives an important practical introduction to using command files.
Introduction
This chapter describes the situations in which a Windows user may want to take advantage of the two alternative modes of control in TreeNet, command-line and batch, and provides a guide to using these two control modes. For users running TreeNet on a UNIX platform, this chapter contains a detailed guide to command syntax and options and describes how the Windows version may assist you in learning the command-line language.

The following picture illustrates common channels of interaction between a user and TreeNet.

First, note that TreeNet itself is a sophisticated analytical engine that is controlled via command sequences sent to its input and that may generate various pieces of output when requested.

A less experienced user may communicate with the engine via GUI front and back ends. The GUI front end provides a set of setup screens and “knows” how to issue the right command sequences according to the user’s input. It is also possible to request the GUI front end to save command sequences into an external command file.

The GUI back end captures the results produced by the engine and displays various plots, tables, and reports. Most of these can be directly saved to the hard drive for future reference.

The whole cycle (marked by the large arrows) is completely automated so that the user does not need to worry about what is taking place underneath.

A more demanding user may write separate command files with or without the help of the GUI front end. This feature is especially attractive for creating an audit trail or for combining various sequences of commands to form an automated process. Given that
the current release of TreeNet for UNIX is entirely command-line driven, the user running TreeNet for UNIX will fall into this category.

The TreeNet engine reads data off the hard drive for modeling or scoring, takes grove files for scoring, or executes command files when requested. In addition, the engine may generate new data with scoring information added, create grove files for models, and save classic text output.

The following sections provide in-depth discussion.

**Alternative Control Modes in TreeNet for Windows**

In addition to controlling TreeNet with the graphical user interface (GUI), you can control the program via commands issued at the command prompt or via submission of a command (.cmd) file. This built-in flexibility enables you to avoid repetition, create an audit trail, and take advantage of the BASIC programming language.

**Avoiding Repetition:** You may need to interact with several dialogs to define your model and set model estimation options. This is particularly true when a model has a large number of variables or many categorical variables, or when more than just a few options must be set to build the desired model. Suppose that a series of runs are to be accomplished, with little variation between each. A batch command file, containing the commands that define the basic model and options, provides an easy way to perform many TreeNet command functions in one user step. For each run in the series, the “core” batch command file can be submitted to TreeNet, followed by the few graphical user interface selections necessary for the particular run in question.

**Creating an Audit Trail:** The Command Log window can help you create an audit trail when one is needed. Imagine not being able to reproduce a particular analysis track, perhaps because the specific set of options used to create a model (e.g., the name of the data set itself) was never recorded. The updated command log provides you with the entire command set necessary to exactly reproduce your analysis, provided the input data do not change.

**Taking Advantage of TreeNet Built-In Programming Language:** TreeNet offers an integrated BASIC programming language that allows the user to define new variables, modify existing variables, access mathematical, statistical and probability distribution functions, and define flexible criteria to control case deletion and the partitioning of data into learn and test samples. BASIC commands are implemented through the command interface, either interactively or via batch command files.

Small BASIC programs are defined near the beginning of your analysis session, after you have opened your dataset but before you estimate (or apply) the model and usually before defining the list of predictor variables. BASIC is powerful enough that in many cases users do not need to resort to a stand-alone data manipulation program. See Appendix II for more on BASIC Programming Language.

**Command-Line Mode**

Choosing **Command Prompt** from the **File** menu allows you to enter commands directly from the keyboard. Switching to the command-line mode also enables you to
access the integrated BASIC programming language. See Appendix II for a detailed description of the BASIC programming language.

**Creating and Submitting Batch Files**

The TreeNet Notepad can be used to create and edit command files. Use the File – New Notepad… menu to open a new Notepad window. From the Notepad, you can submit part or all of an open file. To submit a section of the command file, move the cursor to the first line of the selected section and select Submit Current Line to End from the File menu. To submit the entire command file, select Submit Window from the File menu (or click on the Submit icon in the toolbar). After you submit the file, the analysis proceeds as if you had clicked on the [Start] button in the GUI—the progress report window appears and, after the analysis is complete, the Results dialog.

To submit an existing batch file, choose Submit Command File from the File menu. In the Submit Command File dialog that appears, specify the ASCII text file from which command input is to be read and then click on [Open]. To facilitate multiple TreeNet runs, the TreeNet results are directed only to the TreeNet output window in text form (i.e., the GUI Results dialog does not appear).

Each of these topics is discussed in more detail below.

**Command Log**

Most GUI dialog and menu selections have command analogs that are automatically sent to the Command Log and can be viewed, edited, resubmitted and saved via the Command Log window. When the command log is first opened (by selecting Open Command Log… from the View menu), all the commands for the current TreeNet session are displayed. Subsequently, by selecting Update Command Log from the View menu, the most recent commands are added to the Command Log window.

After computing a TreeNet model, the entire set of commands can be archived by updating the command log, highlighting and copying the commands to the Notepad (or saving directly to a text file), then pasting into your text application. Alternatively, you can edit the text commands, deleting or adding new commands, and then resubmit the analysis by selecting either Submit Window or Submit Current Line to End from the File menu.

**View – Open Command Log**

Within a single work session TreeNet keeps a complete log of all the commands given to the engine. You may access this command list at any time through the View—Open Command Log menu.
This feature is very helpful for learning command syntax and writing your own command files. All you need to do is set up run options using the GUI front end and then read the corresponding command sequence from the Command Log.

You may save the Command Log into a command file on your hard drive using the File—Save menu. If you do this before exiting a TreeNet session, the resulting command file will contain the audit trail of the entire session.

The Command Log Window supports the cut-and-paste technique.

**File—New Notepad**

TreeNet GUI offers a simple text editor to write your own command files. You may open multiple instances of the Notepad window using the File—New Notepad... menu. You may also open the existing command file using the File—Open>Command File... menu.

You may use the cut-and-paste technique to grab command sequences from the Command Log Window and then edit them in the notepad window.

**File—Submit Window**

This menu item allows you to submit a command sequence from a TreeNet Notepad window to the TreeNet engine. Using this channel does not suppress the results window generated by the GUI back end.
This option is also available for the Command Log Window. In this case the entire session will be reproduced.

Submitting multiple runs may produce too many open windows, seriously affecting your system's performance. In this case, saving the contents of the notepad window into a command file and then using the File—Submit Command File… menu item (see the following section) is preferable.

File–Submit Command File
This menu item allows you to submit a command file (*.cmd) directly to the TreeNet engine. Using this channel completely suppresses all output sent to the GUI back end.

Use this mode when you want to execute multiple runs without cluttering the GUI with multiple results windows (which may slow things down and drag the system to a halt).

Consider using the OUTPUT command to save the classic text result to an ASCII text file.

Consider using the GROVE command to save the GUI results (which are usually generated using interactive mode.)

Command Syntax Conventions
TreeNet command syntax follows the following conventions:

- Commands are case insensitive.
- Each command takes one line starting with a reserved keyword.
- A command may be split over multiple lines using comma “,” as the line continuation character.
- No line may exceed 256 characters.

Example: Regression Modeling Run
Below is a sample command file RUN_REG.CMD to conduct a regression run.

```
REM Example
NEW
OUTPUT "run_reg.dat"
NOTE "TN REGRESSION run"
USE "treenet.csv"
GROVE "grobe_reg.grv"
LOPTONS MEANS = NO, NOPRINT = NO, PREDICTIONS = NO, TIMING = YES, PLOTS = NO
FORMAT = 7
MODEL Y1
KEEP X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 Z1$ Z2$
WEIGHT U
ERROR SEPVAR=T
MART TREES=200, LEARNRATE=0.1, LOSS=LS, NODES=0, MINCHILD=10, SUBSAMPLE=0.5
MART PLOTS = YES, YES, YES, YES, NO = 10, 10, 10, 10
MART ST = 1000000, SC = 100000
MART GO
NEW
OUTPUT *
```
Detailed descriptions of each command are given in Appendix I, “Command Reference.” The highlights are:

**Line 1:** REM – a simple comment, the entire line is ignored by the parser.

**Line 2:** NEW – forces TreeNet to clear all previous settings and prepare for a fresh run.

**Line 3:** OUTPUT – redirects Classic Output to the text file called “run.dat.”

Equivalent to the File–Save>Save Output...

**Line 4:** NOTE – prints the supplied text line in the classic output file.

**Line 5:** USE – specifies the data file (equivalent of the File–Open menu item). Note that in this case the data file is in the ASCII format.

**Line 6:** GROVE – requests saving the model grove file.

Equivalent to the [Save Grove...] button in the Model Setup dialog.

**Line 7:** LOPTIONS – controls various pieces of information that can be included in the classic output (descriptive statistics, timing, etc.).

**Line 8:** FORMAT – sets the number of digits after the decimal point to be reported in the classic output.

**Line 9:** MODEL – sets the target variable.

**Line 10:** KEEP – sets the list of predictors.

**Line 11:** WEIGHT – sets the weight variable, if any.

Lines 9, 10, and 11 correspond to the Model tab in the Model Setup window.

**Line 12:** ERROR – sets the testing method.

Alternative useful settings:

ERROR EXPLORE
ERROR PROPORTION=.5
ERROR FILE="<path and file name>"

This corresponds to the Testing tab in the Model Setup window.

**Line 13:** MART – sets TreeNet parameters:

- TREES: number of trees to grow
- NODES: number of terminal nodes per tree
- MINCHILD: smallest node count allowed
- LEARNRATE: learning rate
- SUBSAMPLE: sampling rate
- LOSS: loss function
These correspond to the TreeNet tab in the Model Setup window.

Alternative values for LOSS would be HUBER or LAD.

**Line 14:** MART PLOTS – controls how many single plots, pair plots, single stats and pair stats will be automatically generated by the TreeNet engine for this modeling run.

**Line 15:** MART ST – controls how much memory will be allocated for the current run. You should increase these values if TreeNet complains about not having enough tree or categorical space to finish the run.

**Line 16:** MART GO – signals the TreeNet engine to start the modeling process.

This is equivalent to pressing the [Start] button in the Model Setup window.

**Line 17:** NEW – resets all internal settings. This command is needed to ensure that the TreeNet loop is finished and all output files are closed.

Delete this command from the command file if you want to create new plots or request more trees using the GUI after the command file has been submitted.

**Line 18:** OUTPUT * – closes the classic output file. No more lines of text will be added to the file.

Using File –Submit Window executes this regression run.

**Example: Binary Logistic Regression Modeling Run**

Below is a sample command file RUN_LOGIT.CMD to conduct a binary logistic regression run.

```plaintext
1 REM Example
2 NEW
3 OUTPUT "run_logit.dat"
4 NOTE "TH BINARY LOGISTIC run"
5 USE "treenet.csv"
6 GROUP "group_logit_grv"
7 OPTIONS MEANS = NO, NOPRINT = NO, PREDICTIONS = NO, TIMING = YES, PLOTS = NO
8 FORMAT = 7
9 MODEL Y2
10 KEEP X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X22
12 WEIGHT U
13 ERROR SEPARAT
14 GW UNIT
15 MART TREES=200, LEARNRATE=0.1, OPTIMAL=ROC, MODES=6, MINCHILD=10, SUBSAMPLE=0.5
16 MART BINARY=LOGISTIC, CTHRESH=0.5, LTHRESH=0.1
17 MART PLOTS = YES, YES, YES, YES, MU = 10, 10, 10, 10
18 MART ST = 1000000, SC = 10000
19 MART GO
20 NEW
21 OUTPUT *
```

From the command file perspective, binary logistic regression differs from a regression run in the following key items:

- If the target has two numeric levels, it must appear on the CATEGORY list.
LOSM option is ignored. Instead you should use the OPTIMAL option to specify the desired model evaluation criterion.

You should use the BINARY=LOGISTIC option to ensure this mode of operation.

You may set up CTHRESH and LTHRESH options to supply classification and lift thresholds.

Below is a sample binary logistic regression run followed by line-by-line highlights.

Detailed descriptions of each command are given in Appendix I, “Command Reference.” The highlights are:

**Line 1:** REM – a simple comment, the entire line is ignored by the parser.

**Line 2:** NEW – forces TreeNet to clear all previous settings and prepare for a fresh run.

**Line 3:** OUTPUT – redirects Classic Output to the text file called “run.dat”.

Equivalent to the File–Save>Save Output…

**Line 4:** NOTE – prints the supplied text line in the classic output file.

**Line 5:** USE – specifies the data file (equivalent of the File–Open menu item). Note that in this case the data file is in the ASCII format.

**Line 6:** GROVE – requests saving the model grove file.

Equivalent to the [Save Grove…] button in the Model Setup dialog.

**Line 7:** LOPTIONS – controls various pieces of information that can be included into the classic output (descriptive statistics, timing, etc.).

**Line 8:** FORMAT – sets the number of digits after the decimal point to be reported in the classic output.

**Line 9:** MODEL – sets the target variable.

**Line 10:** KEEP – sets the list of predictors.

**Line 11:** CATEGORY – specifies that the target is categorical. Any additional categorical numeric variables must appear here as well.

**Line 12:** WEIGHT – sets the weight variable, if any.

Lines 9, 10, 11, and 12 correspond to the Model tab in the Model Setup window.

**Line 13:** ERROR – sets the testing method.

Alternative useful settings:

ERROR EXPLORE
ERROR PROPORTION=.5
ERROR FILE="<path and file name>"
This corresponds to the Testing tab in the Model Setup window.

Model tab in the Model Setup window.

**Line 14:** CW – sets the class weights.

**Line 15:** MART – sets TreeNet parameters:

- **TREES:** number of trees to grow
- **NODES:** number of terminal nodes per tree
- **MINCHILD:** smallest node count allowed
- **LEARNRATE:** learning rate
- **SUBSAMPLE:** sampling rate
- **OPTIMAL:** optimality criterion

These correspond to the TreeNet tab in the Model Setup window.

**Line 16:** MART BINARY – forces binary logistic run and then sets classification and lift thresholds.

**Line 17:** MART PLOTS – controls how many single plots, pair plots, single stats and pair stats will be automatically generated by the TreeNet engine for this modeling run.

**Line 18:** MART ST – controls how much memory will be allocated for the current run. You should increase these values if TreeNet complains about not having enough tree or categorical space to finish the run.

**Line 19:** MART GO – signals the TreeNet engine to start the modeling process.

This is equivalent to pressing the [Start] button in the Model Setup window.

**Line 20:** NEW – resets all internal settings. This command is needed to ensure that the TreeNet loop is finished and all output files are closed.

Delete this command from the command file if you want to create new plots or request more trees using the GUI after the command file has been submitted.

**Line 21:** OUTPUT * – closes the classic output file. No more lines of text will be added to the file.

Using File –Submit Window executes this binary logistic regression run.

**Example: Classification Modeling Run**

Below is a sample command file `RUN_CLASS.CMD` to conduct a classification run.
From the command file perspective, the classification run differs from the binary logistic regression run in the following key items:

- You should use the BINARY=CLASS option to ensure this mode of operation.
- OPTIMAL, CTHRESH, and LTHRESH options are ignored.
- You may use the MISCLASS command to set misclassification costs.

Below is a sample classification run followed by line-by-line highlights.

Detailed descriptions of each command are given in Appendix I, “Command Reference.” The highlights are:

**Line 1**: REM – a simple comment, the entire line is ignored by the parser.

**Line 2**: NEW – forces TreeNet to clear all previous settings and prepare for a fresh run.

**Line 3**: OUTPUT – redirects Classic Output to the text file called “run.dat”.

Embed: Equivalent to the File–Save>Save Output...

**Line 4**: NOTE – prints the supplied text line in the classic output file.

**Line 5**: USE – specifies the data file (equivalent of the File– Open menu item). Note that in this case the data file is in the ASCII format.

**Line 6**: GROVE – requests saving the model grove file.

Embed: Equivalent to the [Save Grove...] button in the Model Setup dialog.

**Line 7**: LOPTIONS – controls various pieces of information that can be included into the classic output (descriptive statistics, timing, etc.).
Line 8: FORMAT – sets the number of digits after the decimal point to be reported in the classic output.

Line 9: MODEL – sets the target variable.

Line 10: KEEP – sets the list of predictors.

Line 11: CATEGORY – specifies that the target is categorical. Any additional categorical numeric variables must appear here as well.

Line 12: WEIGHT – sets the weight variable, if any.

- Lines 9, 10, 11, and 12 correspond to the Model tab in the Model Setup window.

Line 13: ERROR – sets the testing method.

Alternative useful settings:

ERROR EXPLORE
ERROR PROPORTION=.5
ERROR FILE="<path and file name>"

- This corresponds to the Testing tab in the Model Setup window.
- Model tab in the Model Setup window.

Line 14: CW – sets the class weights.

Line 15: MART – sets TreeNet parameters:

- TREES: number of trees to grow
- NODES: number of terminal nodes per tree
- MINCHILD: smallest node count allowed
- LEARNRATE: learning rate
- SUBSAMPLE: sampling rate

- These correspond to the TreeNet tab in the Model Setup window.

Line 16: MART BINARY – forces classification run.

Line 17: MART PLOTS – controls how many single plots, pair plots, single stats and pair stats will be automatically generated by the TreeNet engine for this modeling run.

Line 18: MART ST – controls how much memory will be allocated for the current run. You should increase these values if TreeNet complains about not having enough tree or categorical space to finish the run.

Lines 19, 20, 21: MISCLASS – set misclassification costs.

- Multiple MISCLASSIFY commands are allowed.
- The default value for any cell in the cost matrix is always one.
This is equivalent to the Costs tab in the Model Setup window.

Setting an individual cell value to zero or to a negative number results in an error. Use very small positive numbers when zero costs are needed.

**Line 22:** MART GO – signals the TreeNet engine to start the modeling process.

This is equivalent to pressing the [Start] button in the **Model Setup** window.

**Line 23:** NEW – resets all internal settings. This command is needed to ensure that the TreeNet loop is finished and all output files are closed.

Delete this command from the command file if you want to create new plots or request more trees using the GUI after the command file has been submitted.

**Line 24:** OUTPUT * – closes the classic output file. No more lines of text will be added to the file.

Using **File – Submit Window** executes this classification run.

**Example: Scoring Run**

Command files for scoring TreeNet models are practically identical for different types of models. Below is a sample command file **SCORE_LOGIT.CMD** to score a binary logistic regression model. Other examples of the scoring command files (SCORE_REG.CMD and SCORE_CLASS.CMD) are also included in the installation folder.

```
1 REM Example
2 NEW
3 OUTPUT "score_logit.dat"
4 NOTE "TN BINARY LOGISTIC SCORING run"
5 USE "treenet.csv"
6 GROVE "grove_logit.grv"
7 SAVE "score_logit.csv"
8 FORMAT = 7
9 WEIGHT V
10 IDVAR T
11 MART APPLY, PROB=YES
12 OUTPUT *
```

Detailed descriptions of each command are given in Appendix I, “Command Reference.” The highlights are:

**Line 1:** REM – a simple comment, the entire line is ignored by the parser.

**Line 2:** NEW – forces TreeNet to clear all previous settings and prepare for a fresh run.

**Line 3:** OUTPUT – redirects Classic Output to the text file called “run.dat.”

**Line 4:** NOTE – prints the supplied text line in the classic output file.
Line 5: USE – specifies the data file (equivalent of the File–Open menu item). Note that in this case the data file is in the ASCII format.

Line 6: GROVE – requests the model grove file.

Line 7: SAVE – specifies the output dataset.

Example: Use the SAVE “score.csv” / MODEL command to include the original model variables in the output dataset.

Line 8: FORMAT – sets the number of digits after the decimal point to be reported in the classic output.

Line 9: WEIGHT – sets the weight variable, if any.

Line 10: IDVAR – requests the addition of ID variables to the output dataset.

Line 11: MART APPLY – triggers the scoring process.

Line 12: OUTPUT * – closes the classic output file. No more lines of text will be added to the file.

Example: Translate Run

Command files for translating TreeNet models are identical for different types of models. Below is a sample command file TRANSLATE_SAS.CMD to translate a regression model into SAS-compatible language. You will find a sample C translation command file (TRANSLATE_C.CMD) in the main installation folder.

```plaintext
1 REM Example
2 NEW
3 OUTPUT "translate.sas"
4 GROVE "grove_reg.grv"
5 TRANSLATE LANGUAGE=SAS
6 OUTPUT *
```

Line 1: REM – a simple comment, the entire line is ignored by the parser.

Line 2: NEW – forces TreeNet to clear all previous settings and prepare for a fresh run.

Line 3: OUTPUT – redirects Classic Output to the text file called “translate.sas.” This file will contain the SAS-compatible folder.

Line 4: GROVE – requests the model grove file.

Line 5: TRANSLATE – triggers the translation process.

Line 6: OUTPUT * – closes the classic output file. No more lines of text will be added to the file.
TreeNet Features and Options

This chapter provides an orientation to features not covered in previous chapters.
Case Weights

In addition to selecting target and predictor variables, the **Model Setup—Model** tab now allows you to specify a case-weighting variable.

Case weights are stored as a variable in the dataset and typically vary from observation to observation. An observation’s case weight can, in some sense, be thought of as a repetition factor. A missing, negative or zero case weight causes the observation to be deleted, just as if the target variable were missing. Case weights may take on fractional values (e.g., 1.5, 27.75, 0.529, 13.001) and whole numbers (e.g., 1, 2, 10, 100).

To select a variable as the case weight, place a check in the box from the Weight column that corresponds to the weight variable (from all those in your dataset). In the example below, the variable named W has been selected as the case-weighting variable.

There can be only one weight variable at a time.

If you are using a test sample contained in a separate dataset, the case weight variable must exist and have the same name in that dataset as in your main (learn sample) dataset.

For command line users, the variable containing observation case weights is specified with the `WEIGHT=<variable>` command, which is issued after the USE command and before the MART GO command.

Setting Reporting, Random Number and Directory Options

This section is a guide to the reporting and other fine-tuning controls you may want to set before you grow your trees. These parameters are contained in the **Options** dialog, accessed by selecting Options from the Edit menu (or clicking the Options toolbar icon ).
If you are in the Model Setup dialog box, you must first click on the [Continue] button to access Options... from the Edit menu.

Controlling Report Contents

The parameters controlling the contents of the TreeNet Output can be set in the first tab on the Options dialog, Reporting tab. The default Reporting settings are shown below:

The current distribution of TreeNet contains portions of the Options dialog that are intentionally disabled (grayed out). Do not be troubled by this. The disabled portions of the dialog are components of the CART analysis and are not available in the version of TreeNet you are running.

The following items in the Reporting dialog, the Report Preferences group box, allows you to turn on and off the following:

- Summary stats for all model variables - mean, standard deviation, min, max, etc.
- Exponential notation for near-zero values - scientific notation used for values close to zero
- Decimal places - precision to which the numerical output is printed

Controlling Random-Number Seed Values

As illustrated below, the Options–Random Number tab allows you to set the random-number seed and to specify whether the seed is to remain in effect after a TreeNet model is generated, or after data are dropped down a tree. The random number seed is used whenever TreeNet needs to select records randomly, such as in the model-building process or in the random separation of data into learn and test samples. Normally the seed is reset to 13579, 12345, 131 on start up and after each tree is constructed or after data are dropped down a tree. Resetting the random number seed after each run ensures that if you repeat a run with the same control parameters you will get the same results.
If you prevent the random seed from being reset, then its value at the beginning of the next run will be whatever it ended up being at the end of the last run. The seed will retain its latest value after a model is built if you click on the **Retain to default values after each run** radio button.

---

**TreeNet Options**

The parameters controlling the number of plots and stats that are generated automatically following a MART GO or MORETREES are set in the **Options–TreeNet** tab. The four check boxes displayed in the section titled Plot Creation turn on/off plot creation corresponding to singleplots, pairplots, singlestats and pairstats, respectively.

By default, pair- and singleplots (Variable and Bivariate dependence), and pair- and singlestats (Variable and Bivariate interaction statistics), are automatically generated for all variables that affect the model (i.e., those with nonzero variable importance). However, to speed your analysis you may wish to have pairplots and pairstats automatically generated for the most important 15 predictors, but allow singleplots and stats for all predictors. This would be requested by setting the **Options–TreeNet** tab as shown below.
The default setting or **Max. number of important variable to plot** is zero. In this case, the value 0=all.

The other options available in the **Options–TreeNet** tab are **Memory Allocation** values for tree storage and categorical split storage. In certain situations, TreeNet may generate an error message indicating that one of these memory allocation blocks has been exhausted. These two option settings allow the user to increase the available memory allocation for these storage blocks.

- **Trees** Specifies amount of memory to allocate to tree storage.
  The default is 1000000.

- **Categorical Splits** Specifies amount of memory to allocate to categorical split storage.
  The default is 100000.

We recommend doubling or quadrupling both settings if you receive a message indicating that the defaults were insufficiently large.

**Directory Preferences**

The **Options–Directories** tab allows you to set default directory preferences for input (data, grove and command), output (grove, prediction results, and text report), and temporary files. As shown below, all input and output directories are initially set to the directory housing TreeNet; the temporary directory is your machine’s temporary Windows directory.
To change any of the default directories, click on the [...] button next to the appropriate directory path and specify a new directory in the Select Default Directory dialog box. TreeNet will retain default directory settings in subsequent analysis sessions.

- If you will be processing large data sets be sure to select a temporary directory on a drive with plenty of free space. If you are on a network use a local drive for temporary files if you have enough free space; this will give you better performance.

**Saving, Opening, and Viewing Grove Files**

TreeNet allows you to save a binary file that contains all TreeNet model information. This file is referred to as a “Grove” file. The Grove file can be saved and then later reloaded. To save a Grove file, the user must click the [Save Tree Information...] button from the Model Setup dialog and specify a file name prior to starting the modeling process. In the Save Grove File dialog box, click on the File name: text box to change the default file name. The file extension is .GRV and should not be changed. Select the directory in which the Navigator file should be saved and click on [Save].
To open a Grove file you have previously saved, select the File–Open>Open Grove... menu. In the Open TreeNet Grove File dialog box, specify the name and directory location of the .GRV file and click on [Open].

![Open TreeNet Grove File dialog box](image)

Opening a Grove file in subsequent sessions allows you to continue your exploration of detailed and summary reports for each of the optimal TreeNet models only. Reopening the file does not reload the model setup specifications in the GUI dialogs. To save your model setup specifications, save the settings in a command file prior to exiting TreeNet. The commands, by default stored in TreeNet’s command log, can be accessed by selecting Open Command Log... from the View menu (or by clicking the Open Command Log toolbar icon). To save the command log, select Save from the File menu. To then reload your setting in the Model Dialog, simply submit the command log. The last set of model setup commands in the command file appears in the tabbed Model Setup dialogs. For more on using the Command Log, see the Command Log section above.

**Saving and Printing Classic Text Output**

By default, TreeNet text output is sent to the TreeNet Output window. To simultaneously save the text output to a file:

1. Select Log Results to>File... from the File menu.
2. Click on the File name: text box in the Text Results to File dialog box to set the file name, as illustrated below.
3. Select the directory in which the file should be saved.
4. Click on [Save].

![Text Results to File dialog box](image)
All subsequent output will be recorded in the selected file. To stop sending the output to a file, select Log Results to>Window… from the File menu. To save the current contents of the Output window to a file after you have built a tree, select Save>Save Output… from the File menu and follow steps 2 through 4 above. To save a particular section of the output, highlight that section and select Copy from the Edit menu (or from the toolbar). Paste the copied text to the Notepad by selecting New Notepad… from the File menu and then save the notepad contents by selecting Save As… from the File menu. Alternatively, after you copy the text, paste it to another application such as Microsoft Word or Excel.

Report Writer

TreeNet includes Report Writer. Report Writer is a report generator, word processor and text editor that allows you to construct custom reports from result diagrams, tables and graphs as well as from the "classic" TreeNet output appearing in the Output window.

Using the Report Writer is easy! One way is to copy certain reports and diagrams to the Report Window as you view the TreeNet Results dialog or Output windows. Once a processing is complete, a TreeNet Results window appears, allowing you to explore the performance with a variety of graphic reports, statics and diagrams. Virtually any graph, table, grid display, or diagram can be copied to the Report Writer. Simply right-click the item you wish to add to the Report Writer and select Add to Report. The selection will appear at the bottom of the Report window.

TreeNet also produces “classic” output for those users more comfortable with a text-based summary of the model and its performance. To add any (or all) of TreeNet’s classic output to the Report Writer window, highlight text in the classic output window, copy it to the Windows clipboard (Ctrl-C), switch to the Report Writer window and paste (Cntl-V) at the point you want text inserted. This way you can combine those TreeNet result elements you find most useful—either graphic in nature and originating in the TreeNet Results dialog, or textual in nature from the classic output—into a single custom Report.

Default Options

In the Report Options dialog, the currently-selected reporting items and the Automatic Report checkbox can be saved as a default group of settings for future TreeNet sessions by clicking the [Set Default] button. These default options will then persist from session to session because they are saved in the TreeNet preference file (cart.ini). You may recall these settings any time with the Use Default button.
Pre-configured

Additionally, TreeNet can produce a “stock report” with the click of a button. You decide which components of TreeNet output would be the most useful to you on the **Report—Set Report Options**... menu—just select those that are most useful. The stock report will be the same for all TreeNet Results in the session until you visit the **Report Contents** dialog again. (In addition, the currently-open TreeNet Results dialogs are listed and individual ones can be excluded or added to the list that will appear in the report when the [**Report Now**] button is clicked.)

A stock report for the currently-active (i.e., foreground) TreeNet Results can be generated by choosing **Report—Report Current**. If the active window is not a results window, the **Report Current** menu item will be disabled. Furthermore, if you have several TreeNet Results windows open, you can generate a report for all the trees (in the order in which they were built) by choosing the **Report—Report All** menu item.

**Target Class**

Reports summarizing class performance (e.g., gains charts) require a target class. For binary models (i.e., 0/1 or 1/2), the second level is assumed to be the target class. For multinomial models (e.g., 1, 2, 3, 4), the lowest class is assumed to be the target class.
Printing and Saving Reports
Once you have generated a report it may be printed or previewed by using the Print…, Print Setup… and Print Preview… options on the File menu.

To save a report to a file, use the File–Save As… option. The contents of the Report Window can be saved in two formats: Rich text format (.rtf), text, or text with line breaks (.txt). The rich text (.rtf) can be read by most other word processors and maintains the integrity of any graphics imbedded in the report. The two text formats do not retain graph and diagram images or table formatting.

Data Information
TreeNet provides a GUI facility for viewing information on the currently-open data file. Groups of descriptive statistics are provided for each variable (numeric and character). Information groups available for viewing include the following:

DESCRIPTIVE: N, N missing, N = 0, N <> 0, N Distinct Values, Mean, Std Deviation, Skewness, Coeff Variation, Cond. Mean, Sum of Weights, Sum, Variance, Kurtosis, Std Error Mean

LOCATION: Mean, Median, Range

VARIABILITY: Std Deviation, Variance, Intrqrt Range

QUANTILES: 100% Max, 99%, 95%, 90%, 75% Q3, 50% Median, 25% Q1, 10%, 5%, 1%, 0% Min
FREQUENCY TABLES: Most (Top 5 in Pop.), Least (Bottom 5 in Pop.), All

The DataInfo window is opened by selecting the View–Data Info… menu item. Once the window is open and active, use the [+ ] and [–] toggles to expand and contract each information group.

For additional control of the DataInfo window you need to issue the DATAINFO command from the command line. Details for the DATAINFO command can be found in Appendix I: Command Reference.

For additional insight into the use of TreeNet command language, and working in command line mode, see Chapter 6 (Working with Command Language).

Data Viewer

If you have opened your dataset using DATABASE DRIVERS, TreeNet’s Data Viewer allows you to view (but not edit or print) the data as a spreadsheet—handy for investigating data anomalies or seeing the pattern of missing values. Because the Data Viewer is a facility of DATABASE DRIVERS it is available only for data files that are opened using the DATABASE DRIVERS translators. To ensure this, remember to have File–Use DATABASE DRIVERS checked.

The Data Viewer window is opened by selecting the View–View Data… menu item or clicking on the View Data toolbar icon (it looks like a little spreadsheet).

Only one data file can be displayed at a time.
Because opening an ASCII text file does not use the DATABASE DRIVERS facility to open this form of data, the Data Viewer is not available for ASCII text files.

Online Help
The Help menu provides comprehensive on-line version of this documentation.

The About TreeNet for Windows selection displays information about the version number and work space.
Appendix I.

Command Reference

This appendix provides a command language reference including syntax and examples.
CATEGORY

Purpose

The CATEGORY command indicates whether the target variable is categorical and identifies which predictors are categorical.

The command syntax is:

\[ \text{CATEGORY } \var{var1}, \var{var2} \]

Examples:

\begin{verbatim}
MODEL LOW
CATEGORY LOW  (categorical dependent variable indicates classification TreeNet)

MODEL SEGMENT
CATEGORY SEGMENT
\end{verbatim}

CATEGORY is also used to identify categorical predictor variables. TreeNet will determine the number of distinct values for you. Example:

\begin{verbatim}
MODEL LOW
CATEGORY LOW, AGE, RACE, EDUC
\end{verbatim}
CDF

Purpose

The CDF command evaluates one or more distribution, density, or inverse distribution functions at specified values.

This command is for your convenience only and does not affect TreeNet runs.

For cumulative distribution functions the syntax is:

CDF [ NORMAL = z | T = t, dof | F = f, dof1, dof2 |
   CHI-SQUARE = chisq, dof | EXPONENTIAL = x | GAMMA = gamma, p |
   BETA = beta, p, q | LOGISTIC = x | STUDENTIZED = x, p, q |
   WEIBULL = x, p, q | BINOMIAL = x, p, q | POISSON = x, p ]

To generate density values, use the syntax above with the DENSITY option:

CDF DENSITY [ distribution_name = user-specified-value(s) ]

To generate inverse cdf values, specify an 'alpha' value between 0 and 1:

CDF INVERSE [ NORMAL=alpha | T=alpha, dof | POISSON=alpha, p |
   F=alpha, dof1, dof2 | CHI-SQUARE=alpha, dof | EXPONENTIAL=alpha |
   GAMMA=alpha, p | BETA=alpha, p, q | LOGISTIC=alpha |
   STUDENTIZED=alpha, p, q | WEIBULL = alpha, p, q |
   BINOMIAL=alpha, p, q ]

CDF NORMAL=-2.16, DENSITY NORMAL=-2.5, INVERSE CHISQ=.8,3
CHARSET

Purpose

The CHARSET command allows you to select which type of characters to use for character graphics (as opposed to high-resolution SYGRAPH graphics). You may choose either IBM screen and printer GRAPHICS characters or GENERIC characters that will print on any printer.

⚠️ This command only affects the classic text output.

⚠️ GRAPHICS characters do not print correctly on some printers; if you have problems, switch to GENERIC.

The command syntax is:

   CHARSET  GRAPHICS | GENERIC

Examples:

   CHARSET GRAPHICS
   CHAR GENERIC
The **CLASS** command assigns labels to specific levels of categorical variables (target or predictor). Labels are not limited in their length, although in some reports they will be truncated due to space limitations. For instance, if variable DRINK takes on the values 0, 1, 2, and 3 in the data, you might wish to assign labels to those levels:

```
CATEGORY DRINK
CLASS DRINK 0=tea 1='Columbian coffee' 2="soda pop", 
              3='Cold German Beer!'
```

Class labels will appear in the node detail, misclassification reports, terminal node reports, and in most instances where the numeric levels would normally show up, in lieu of the numeric levels themselves.

It is not necessary to specify labels for all levels of a categorical variable -- any levels without a label will show up as numbers.

**The command syntax is:**

```
CLASS <variable> <level>=<string>, <level>=<string>, ...
```

You may issue separate CLASS commands for each variable, such as:

```plaintext
CLASS PARTY 1=Repub 2=Democratic 3="Peace and Freedom"
CLASS GENDER 0=female 1=male
CLASS EVAL$ "G"="Good", "F"="Fair", "P"="Poor"
```

or you may combine them in a single command, separating variables with a slash:

```plaintext
CLASS PARTY 1=Repub 2=Democratic, 
            3="Peace and Freedom" / GENDER 0=female 1=male /,
            EVAL$ "G"="Good", "F"="Fair", "P"="Poor"
```

Note that the label "Peace and Freedom" requires quotes, since it contains spaces. Labels consisting only of numbers and letters can be listed without quotes, but if so any letters will be converted to uppercase.

Note also that all class labels for a given variable must be defined at once, since the `<variable>` token that leads the list of classes clears out any existing class labels for the variable.

Variable groups that are composed of one type of variable only (i.e., numeric or character) may be used in the CLASS command similarly to variable names, e.g.:

```
GROUP CREDITEVAL = EVAL3MO, EVAL6MO, EVAL1YR, EVAL3YR 
CATEGORY CREDITEVAL
CLASS CREDITEVAL 0="n/a", 1="Poor", 2="Fair", 3="Good"
```

Class labels are reset with the USE command. They are preserved in a TreeNet grove file. They will not carry over from a BUILD run to a CASE run unless in a continuation of
the BUILD session. To reset all class labels, issue the CLASS command with no options:

```
CLASS
```

To see a summary of class labels issue the command:

```
CLASS  _TABLE_
```
**CW**

*Purpose*

The **CW** command specifies prior class weights for binary logistic regression and classification runs.

*The command syntax is:*

```
CW [ BALANCED | UNIT | SPECIFY <class1>=<x1>, <class2>=<x2>, ... ]
```

in which `<x1>`, `<x2>`, ... is a vector of real numbers. The options set class weights as follows:

**BALANCE** attempts to balance the influence of the target classes relative to one another.

**UNIT** imposes class weights of 1.0 for all target classes, in effect using no class weights. UNIT is the default.

**SPECIFY** `<class1>=<x1>, <class2>=<x2>, ...`

class weights set to any strictly positive numbers. A value must be assigned to each class. For character classes, the class value must be in quotes. The SPECIFY option requires that the dependent variable already be identified on the MODEL command.

*Examples:*

```
CW SPECIFY "COKE"=1, "Pepsi"=2, "H2O"=4, "7UP"=1   (explicit list, let TreeNet rescale)
CW UNIT                        (the default)
```
**DATAINFO**

*Purpose*

The **DATAINFO** command generates descriptive statistics for numeric and character variables. Its simplest form is:

```
DATAINFO
```

The full command syntax is:

```
DATAINFO <varlist> / [ CHARACTER | NUMERIC, 
   EXTREMES = <n>, TABLES ]
```

This command is for your convenience only and does not affect TreeNet runs.

*Examples:*

To indicate particular variables:

```
DATAINFO GENDER$, WAGES, LOGWAGES
```

To generate statistics only for numeric variables, and for each such variable to list the extreme 15 values:

```
DATAINFO / NUMERIC, EXTREMES = 15
```

To produce full frequency tabulations, use the TABLES option:

```
DATAINFO POLPARTY$ / TABLES
```

Variable groups may be used in the CATEGORY command similarly to variable names, e.g.:

```
GROUP GRADES = ROSHREC$, SOPHREC$, JUNIOR$, SENIOR$, PSAT, SAT, MCAT
DATAINFO GRADES
```

Caution: if you have ordered variables (with many distinct values) included in the **DATAINFO**, the TABLES option can generate huge output.

The default is:

```
DATAINFO / EXTREMES = 5
```
DESCRIPTIVE

Purpose

The DESCRIPITIVE command specifies what statistics are computed and printed during the initial pass through the input data. The statistics will not appear in the output unless the command LOPTIONS MEANS=YES command is issued. By default, the mean, N, SD and sum of each variable will appear when LOPTIONS MEANS=YES is used. To indicate that only the N, MIN and MAX should appear in descriptive statistics tables, use the commands:

DESCRIPTIVE N, MIN, MAX
LOPTIONS MEANS=YES

This command only affects the classic text output.

The command syntax is:

DESCRIPTIVE MEAN=<YES|NO>, N=<YES|NO>, SD=<YES|NO>, UM=<YES|NO>, MIN=<YES|NO>, MAX=<YES|NO>, MISSING=<YES|NO>, ALL

ALL will turn on all statistics and MISSING will produce the fraction of observations with missing data.
DISCRETE

Purpose

The DISCRETE command sets options specific to discrete or categorical variables.

This command only affects the classic text output.

The command syntax is:

DISCRETE [TABLES = NONE | SIMPLE | DETAILED ,
CASE = MIXED | UPPER | LOWER ,
MISSING = MISSING | LEGAL ,
REFERENCE = FIRST | LAST ,
MAXTERNARY = \(<n>\) ,
ALLLEVELS = YES | NO

TABLES controls whether frequency tables should be printed following data preprocessing. SIMPLE generates a listing of the levels encountered for each discrete variable and total counts (across learn and test samples). DETAILED breaks down counts by learn and test sample, and also by the dependent variable for classification trees. The default is SIMPLE.

CASE controls whether character strings are case-converted. The default is MIXED.

MISSING controls whether missing values for discrete variables are treated as truly MISSING or are considered a legal and distinct level.

REFERENCE specifies which level is considered the reference, or "left out" level. In MARS, a reference level is only needed when computing an OLS model for comparative purposes prior to the MARS model. By default, the FIRST level according to the ORDER and SORT criteria is considered the reference level. You may wish to change this to the LAST level to reach agreement with some other OLS programs.

MAXTERNARY specifies the maximum number of allowable nodes in the ternary trees used to accumulate distinct levels in discrete variables. The default is 4000. You should only consider increasing this parameter if the program is unable to obtain a complete tabulation of one or more of your discrete variables.

ALLLEVELS By default, node statistics will not list discrete variable levels for a node that is not represented (N=0) in that node. Specifying ALLLEVELS=YES results in a complete tabulation of levels including those with N=0 in the node.

The default is

DISCRETE TABLES=SIMPLE, CASE=MIXED, MISSING=MISSING, REFERENCE=FIRST, ALLLEVELS=NO
**ERROR**

*Purpose*

The `ERROR` command specifies the method used to measure true regression error and misclassification rates.

*The command syntax is:*

```
ERROR [ EXPLORATORY | PROPORTION = <x> | CROSS = <n> |
        CV = <var> | SEPVAR = <var> | FILE = <filename> ]
```

`<x>` is between 0 and 1, `<n>` is an integer, `<var>` is a variable and `<filename>` is any valid file (file formats other than ASCII text and legacy SYSTAT require appropriate pseudo extension).

**EXPLORATORY** no independent testing -- resubstitution estimate.

**PROPORTION** fraction of cases selected at random for testing.

**SEPVAR** named variable separates learn and test samples. The test value is 1 for numeric SEPVAR variables and "TEST" or "test" for character SEPVAR variables.

**FILE** test sample is contained in a separate SYSTAT data file.

**CROSS** `<n>`-fold cross-validation.

**CV** named variable specifies cross-validation bins. The values must be consecutive integers starting with 1.

*Examples:*

```
ERROR PROPORTION=.25       (select 25% of cases at random for test)
ERROR FILE=SHARP            (test cases are found in file SHARP.SYS)
```
EXCLUDE

Purpose

The EXCLUDE command specifies a list of independent variables to exclude from the analysis.

The command syntax is:

\[
\text{EXCLUDE} \ <\text{varlist}\>
\]

in which \text{<varlist>} is a list of variables NOT to be used in the model building process. All other variables will be used.

See the MODEL and KEEP commands for other ways to restrict the list of candidate predictor variables.

Examples:

MODEL CHOICE
   EXCLUDE ID, SSN, ATTITUDE (all numeric variables except ID, SSN and ATTITUDE can be used in the TreeNet process)
FORMAT

Purpose

The FORMAT command controls the number of digits that are displayed to the right of the decimal point in analysis output. You may select from 1 to 9 digits, or 0 digits, or -1 for no digits and no decimal point. The default is 3.

This command only affects the classic text output.

The UNDERFLOW option prints tiny numbers (those that would appear to be zero in the chosen precision) in scientific (exponential) notation.

The command syntax is:

FORMAT <#> [/UNDERFLOW]

Examples:

FORMAT=5
FORMAT=0
FORMAT=9/UNDERFLOW  (print tiny numbers with exponents)
GROVE

Purpose

The GROVE command names a grove file in which to store the next model or to use in the next TRANSLATE or MART APPLY operation. Its syntax is:

The command syntax is:

GROVE <file>

Examples:

GROVE MYDATA  (assumes MYDATA.GRV)
GROVE '\MONTHLY\SURVEY.GRV'
HELP

Purpose

The HELP command provides information about TreeNet commands. You can abbreviate the name of the command.

⚠️ This command only affects the classic text output.

The command syntax is:

```
HELP [command]
```

Examples:

```
HELP  (lists commands available for the current procedure)
```
HISTOGRAM

Purpose

The HISTOGRAM command produces low resolution density plots.

This command only affects the classic text output.

The command syntax is:

```plaintext
HISTOGRAM <var1> [, <var2> , <var3> , . . . ,
/ FULL, TICKS | GRID, WEIGHTED, NORMALIZED, BIG ]
```

The plot is normally a half screen high: the FULL and BIG options will increase it to a full screen (24 lines) or a full page (60 lines).

TICKS and GRID add two kinds of horizontal and vertical grids.

WEIGHTED requests plots weighted by the WEIGHT command variable.

NORMALIZED scales the vertical axis to 0 to 1 (or -1 to 1).

Examples:

```plaintext
HISTOGRAM IQ / FULL, GRID
HISTOGRAM LEVEL(4-7) / NORMALIZED
```

Only numerical variables may be specified.

Variable groups may be used in the HISTOGRAM command similarly to variable names.
**IDVAR**

*Purpose*

The **IDVAR** command lists extra variables to save in the next dataset to be SAVED. These can be any variables from the USE dataset that are not in the model. (Model variables are saved with the SAVE / MODEL option.)

*The command syntax is:*

If every case in your file has a unique identifier, say SSN, you could specify:

```
IDVAR SSN
SAVE "WATER.CSV"
```

The file WATER.CSV will include the variable SSN in addition to its normal contents.

If you want to include all the non-model and model variables in the saved dataset, you would issue:

```
IDVAR / ALL
SAVE <"filename"> / MODEL
```

Variable groups may be used in the IDVAR command similarly to variable names.
**KEEP**

*Purpose*

The **KEEP** command specifies a list of independent variables.

*The command syntax is:*

```
KEEP <indep_list>
```

in which `<indep_list>` is a list of potential predictor variables. If no `<indep_list>` is specified, all numeric variables are considered for node splitting (unless an EXCLUDE command or `<indep_list>` is included on the MODEL statement).

Independent variables may be separated by spaces, commas, or + signs.

See the MODEL and EXCLUDE commands for other ways to restrict the list of candidate predictor variables.

*Examples:*

```
MODEL CLASS
  KEEP AGE-IQ, EDUC, FACTOR(3-8), RACE (selected variables)

MODEL CHOICE
  KEEP FOOD+AGE+HEIGHT-WAIST
```
LIMIT

Purpose

The **LIMIT** command allows tree growth limits to be set.

*The command syntax is:*

```
LIMIT LEARN=\( n | AUTO \), TEST=\( n | AUTO \),
DATASET=\( n \), ERRORSET=\( n \)
```

in which \( n \) is a whole number.

**LEARN**

maximum number of cases to allow into the learning set. By default no limit is in effect. AUTO removes current limit.

**TEST**

maximum number of cases to allow into the test set. By default no limit is in effect. AUTO removes current limit.

*Examples:*

```
LIMIT LEARN=20000, TEST=5000
```
LOPTIONS

Purpose

The LOPTIONS command toggles several “logical” options on and off.

The command syntax is:

LOPTIONS MEANS=YES|NO, TIMING=YES|NO, NOPRINT
    PREDICTION_SUCCESS=YES|NO,
    GAINS=YES|NO, ROC=YES|NO, PS=YES|NO,
    UNS=YES|NO, UNR=YES|NO,
    PLOTS = YES|NO / "<plot_character>"

MEANS controls printing of summary stats for all model variables.

TIMING reports CPU time on selected platforms.

NOPRINT omits node specific output and prints only summary tables.

PREDICTION_SUCCESS requests the prediction success table.

GAINS and ROC toggle the printing of gains and ROC charts for classification models. Binary models always show these charts.

PS toggles printing of the pruning sequence when a tree is built.

UNS selects unsupervised learning for models without a target variable.

UNR specifies that, if unsupervised learning is conducted, the "copy" data are created by sampling with replacement.

PLOTS toggles summary plots and allows a user specified plotting symbol.

To turn an option ON the '=YES' portion is not needed.

LOPTIONS MEANS, TIMING (turn MEANS printing and CPU timing on)
LOPTIONS MEANS (turn MEANS printing on)
LOPTIONS MEANS=NO (turn MEANS printing off)
**MART**

*Purpose*

The **MART** command sets modeling options for TreeNet models. It also launches an initial modeling run or a continuous of the most recent modeling run.

*The command syntax is:*

```
MART [ GO | MORETREES | APPLY | SHRINK=<ntrees>,
TREES=<ntrees>, MAXTREES=<n>, NODES=<n>, MINCHILD=<n>, TEST=YES|NO,
LOSS=LAD|LS|HUBER, BINARY=CLASS|LOGISTIC, FULLREPORT=YES|NO,
ST=<n>, SC=<n>, LEARNRATE=<x|AUTO>, SUBSAMPLE=<x>, INFLUENCE=<x>,
BREAKDOWN=<x>, GB=<n>, PP=<n,n,n,n>, MV=<n,n,n,n>, MP=<n,n,n,n>,
PLOTS=<yes|no>,<yes|no>,<yes|no>,<yes|no>, SEED=<n>, LIST=<n>,
PF="<filename>", CTHRESHOLD=<DATA|x>, LTHRESHOLD=<x>,
STT=<yes|no>, SRL=<yes|no>, SIT=<yes|no>, FR=<n> ]
```

*The default settings are:*

```
MART TREES=200, MAXTREES=10000, NODES=6, MINCHILD=10, TEST=HUBER,
ST=1000000, SC=100000, TEST=YES, GB=10, FULLREPORT=NO,
LEARNRATE=AUTO, SUBSAMPLE=0.5, INFLUENCE=0.1, BREAKDOWN=0.9,
PLOTS=YES,YES,YES, YES, MV=0,0,0,0, MP=0,0,0,0, PP=200,1000,100,200,
PROB=YES
```

The **GO** option indicates that a TreeNet model should be launched after processing all the options on the command.

The **MORETREES** option indicates that the latest TreeNet model should be continued adding more trees to the model.

The **SHRINK** option is deprecated in TreeNet 2.0. This command follows a GO or MORETREES, directs TreeNet to generate reports and plots for a smaller model than the optimal one. The number of trees you choose must be between 2 and the optimal number determine by TreeNet during the most recent GO or MORETREES.

The **APPLY** option, applies the TreeNet model to new data (scoring). Both a USE file and GROVE (.grv) file must be.

```
USE [<INPUT DATA FILENAME>]
GROVE [<MODEL GROVE FILENAME>]
SAVE [<PREDICTION OUTPUT FILENAME>]
MART APPLY
```

**TREES**

Maximum number of trees (iterations) in this run. The default is 200, the minimum is 1.

**MAXTREES**

Maximum number of trees in all runs associated with this model (including MORETREES continuations). The default is 10000, the minimum is 2.

**NODES**

Maximum number of terminal nodes in a tree. The default is 6, the minimum is 2.

**MINCHILD**

Minimum number of training observations in each terminal node. The default is 10, the minimum is 1.
LOSS  Optimization loss criterion, which only affects regression models. The allowable options are:

- **LAD**: least absolute deviation
- **LS**: least squares
- **HUBER**: Huber-M

The default is HUBER.

OPTIMAL  Optimality criterion for logistic models. The options are:

- **CXE**: cross-entropy
- **LIFT**: lift at the LTHRESHOLD point
- **ROC**: maximize area under ROC curve
- **CLASS**: classification error using CTHRESHOLD

The default is CXE.

BINARY  Classification method for binary models. The options are:

- **CLASS**: classification
- **LOGISTIC**: logistic likelihood

The default is LOGISTIC.

ST  Amount of memory to allocate to tree storage. The default is 1000000.

SC  Amount of memory to allocate to categorical split storage. The default is 100000.

LEARNRATE  Regularization shrinkage factor. The default is AUTO, and the allowable range is 0.0001 to 1.0 inclusive.

SUBSAMPLE  Fraction of training sample observations randomly sampled at each iteration. The default is 0.5, and the allowable range is 0.0 to 1.0 inclusive.

INFLUENCE  Influence trimming speed-up (classification or LOGISTIC only). At each iteration, ignore the observations with the smallest influence whose sum is less than INFLUENCE times the total influence. The default is 0.1 and the minimum is 0.0.

BREAKDOWN  1 - breakdown parameter for M-regression. The default is 0.9 and the range is 0.01 to 1.0 inclusive.

TEST  With TEST=YES, if an explicit test sample is not created via ERROR PROPORTION=, ERROR SEPVAR= or ERROR FILE=, TreeNet will extract 20% of the learn sample to serve as a test sample. In this case, using TEST=NO will force TreeNet to proceed without a test sample.

FULLREPORT  Only applies to classification models. Full reports include the complete NxN prediction success matrix and variable importance table for each level of the target. Both of these have as many columns as levels of the target and can be large. Abbreviated reports instead show misclassification counts by target class and average variable importance. FULLREPORT also generates "class importance" tables
identifying, for each variable, the target classes for which it offers the
greatest separation.

**GB**
Defines the number of bins in gains charts in the text output.

**PROB=**YES|NO Specifies whether predicted probabilities are to be saved in the output
dataset created during GO or APPLY.

**CTHRESHOLD** determines the probability threshold for separating classes
when computing classification error rates. The default is 0.5.
DATA bases the threshold on the learn sample target distribution.
The range is 0.0 to 1.0 exclusive (0.0 and 1.0 are not permitted).
Logistic models only.

**LTHRESHOLD** determines the threshold for computing lift. The default is 0.1.
The range is 0.0 (excluded) to 1.0. Logistic models only.

**SEED** controls the seed for the next MART operation. The default is 987654321.

**STT, SRL**
For logistic models, when the integrated ROC, lift and prediction
success threshold table are computed for the learn sample, the
computations can be based on the entire learn sample or the
sub-sampled learn sample. STT controls this for threshold tables, SRL
controls it for integrated ROC and lift. The default is NO for both, i.e.,
use the full learn sample for these computations.

There is no comparable control for test sample versions of these
measures since no sampling is done on test data.

**SIT** controls whether individual tree responses are saved during
MART APPLY. The default is NO.

**LIST**
gains, ROC, prediction success and threshold tables are computed
for the final LIST-most trees in the MART model. The default is
1000, meaning that gains, ROC, prediction success and threshold
tables will be created for the last 1000 trees in the model or, if there
are fewer than 1000 trees in the model, for all trees.

If you encounter insufficient memory situations while building a TreeNet
model with many trees, you might try reducing this parameter.

**FR** controls how many "good models" are identified for purposes of
generating complete results including PS matrices, gains, roc, and
threshold tables. The default is 1, which means the best single model
for each available optimality criterion is tracked (i.e., four for logistic
models, two for classification and regression models). If FR is set to,
say, 100 and the model is a logistic one (with four optimality criteria)
then TreeNet will track the best 100 models equally across the four
optimality criteria (about 25 models per criterion), accounting for
models which are considered good by several criteria. A high value for
FR will result in a potentially lengthy "reconciliation" after the tree
building is done and will increase memory requirements of TreeNet as
well as the size of the grove file, but it makes full results available for a
greater number of models that are of potential interest to the analyst.

The PP, PLOTS, MV and MP options, each of which have four options following them,
work together to control whether (PLOTS), how (PP) and how many (MV, MP)
singleplots, pairplots, singlestats and pairstats (respectively) are generated
automatically following a GO, MORETREES or SHRINK.

PP

Defines the number of randomly selected points used to construct
single- and pair- plots and stats. Four numbers must be given be
specified corresponding to singleplots, pairplots, singlestats and
pairstats, respectively. The maximum is 5000 (anything larger is
reduced to 5000).

PLOTS

Controls if plots and stats are generated automatically following a GO,
MORETREES or SHRINK. Four YES or NO options must be given
corresponding to singleplots, pairplots, singlestats and pairstats,
respectively.

MV

By default, pair- and singleplots and -stats, if generated via the PLOTS
option, are generated for all variables that affect the model (i.e., those
nonzero variable importance). However, to speed your analysis you
may wish to, say, have pairplots and pairstats automatically generated
for the most important 15 predictors but allow singleplots and stats for
all predictors. This would be requested with MV=0,15,0,15. Note: MV
controls how many predictor stats and plots are computed. PLOTS
controls whether they are generated automatically or not.

MP

Places an upper bound on the number of plots that will be generated
for stats and plots.

PF

specifies an XML-like text file in which TreeNet stores the data upon
which the error rate profile, pair plots and single plots are based. E.g.,

MART ... PF="c:\mywork\plotdata\job3.txt" ...
MISCLASS

Purpose

The MISCLASS command specifies misclassification costs. To specify other than unit costs, use one of the following command forms.

The command syntax is:

MISCLASS COST = <x> CLASSIFY <n1,n2,...> AS <m> [/ COST=. .. CLASSIFY..]
MISCLASS COST = <x> CLASSIFY <n> AS <m1,m2,...> [/ COST=. .. CLASSIFY..]

Examples:

The cost of misclassifying a class 2 case as a class 4 case is 4.5:

MISCLASS COST=4.5 CLASSIFY 2 AS 4

The cost of misclassifying a case from classes 1, 2, 3, 5 or 8 as a class 6 case is 2.75:

MISCLASS COST=2.75 CLASSIFY 1-3,5,8 AS 6

MISCLASS commands are cumulative -- each command will specify a part of the misclassification matrix. To reset the matrix use

MISCLASS UNIT
MODEL

Purpose

The MODEL command specifies the dependent variable.

The command syntax is:

MODEL <depvar> [ = <indep_list> ]

in which <depvar> is the dependent variable and <indep_list> is an optional list of potential predictor variables. If no <indep_list> is specified, all variables are used for TreeNet processing (unless KEEP or EXCLUDE commands are used).

Examples:

MODEL DIGIT (all non-character variables used in tree generation)
MODEL WAGE = AGE - IQ, EDUC, FACTOR(3-8), RACE (selected variables)
MODEL CLASS = PRED(8) + VARA-VARZ + PRED(1-3)

See the KEEP and EXCLUDE commands for another way to restrict the list of candidate predictor variables.
NAMES

Purpose

The NAMES command lists the variables on the data set.

This command only affects the classic text output.

The command syntax is:

NAMES
NEW

Purpose

The NEW command resets all TreeNet specific options while leaving TreeNet global options (USE file, PRINT settings, etc.) in effect.

The command syntax is:

```
NEW
```
NOTE

Purpose

The NOTE command lets you write comments on your output. A note can span any number of lines, but no line may be more than 150 characters long. You can embed an apostrophe in a note if you enclose the line in double quotation marks. You can embed double quotation marks if you enclose the line in apostrophes (single quotation marks). A number without quotation marks sends the corresponding ASCII character to the current output device.

This command only affects the classic text output.

The command syntax is:

```
NOTE <#> '<$>', '<...>', <#>
```

Examples:

```
NOTE 'THIS IS A COMMENT.' 'This is second line of comment.',
    "It's the third line here!'"
NOTE 'This the top of a new page' (subsequent NOTE creates line break).
```
OPTIONS

Purpose

The OPTIONS command displays the SYSTAT options currently in effect including the currently used file, any weighting, grouping or selection in effect, short, medium or long output, current graphics character set, number of decimal places to which output prints, and the output destination.

The command syntax is:

OPTIONS
OUTPUT

Purpose

The **OUTPUT** command routes output to the screen (the video display), a file, or a printer. If you send output to a file and specify a simple filename, TreeNet automatically gives the file a "DAT" extension. If you supply a complete path name for the file you must enclose the name in quotes. If you send output to a file or a printer, analysis results also appear on the display.

If the screen pauses waiting for you to hit [Enter] or [Return], output to a file or a printer will also pause.

*The command syntax is:*

```
OUTPUT * | <file>
```

*Examples:*

```
OUTPUT *         (sends subsequent output to screen only)
OUTPUT FILE1     (sends output to FILE1.DAT)
OUTPUT 'C:\REPORTS\NEWOUT.LST'
```
QUIT

Purpose

The QUIT command returns you to the operating system.

The command syntax is:

QUIT
REM

Purpose

The REM command is for comments. All subsequent text on that line is ignored. The REM command is especially useful when writing programs in BASIC and in the writing of command files.

The command syntax is:

   REM <text>

Examples:

   REM  This is a comment line and is not executed
SAVE

Purpose

The SAVE command specifies the output dataset in scoring runs. You may specify the root of the filename if the file will be created in the current directory, or the directory specified with Utilities/Defaults/Path or the FPATH command. If you specify a path, you must provide the complete filename with the appropriate extension, and surround the whole pathname/filename with single or double quotation marks.

The command syntax is:

```
SAVE <file>
```

Examples:

```
SAVE MYDATA  (saves MYDATA.SYS)
SAVE '\MONTHLY\SURVEY.SYS'
```
SEED

Purpose

The SEED command allows you to set the random number seed to a certain value as well as specify that the seed remain in effect after the model is built. Normally the seed is reset to 13579, 12345, 131 upon starting up TreeNet.

*The command syntax is:*

```
SEED I, J, K, RETAIN | NORETAIN
```

All three values I, J, K must be given. Legal values include all whole numbers between 1 and 30000. If RETAIN is not specified the seed will be reset to 13579, 12345, 131 after the current tree is completed.

If RETAIN is specified, the seed will keep its latest value after the tree is built.

*Examples:*

```
SEED 1, 99, 7773
SEED RETAIN
SEED 35, 784, 29954, NORETAIN
```
SELECT

Purpose

The SELECT command selects cases from a file for analysis. You may specify up to ten simple conditions; SYSTAT then selects those cases in the data file that meet all the conditions (that is, the conditions are linked by logical AND).

Specify each condition as variable name, logical relation, and a constant value. The variable name must come first. The six possible logical relations are =, <>, <, >, <=, and >=. You must enclose character values in quotes. Character comparisons are case sensitive.

The command syntax is:

```
SELECT <var$> <relation> '<string$>'
```

or

```
SELECT <var> <relation> <#>
```

Examples:

```
SELECT GROUP=2
SELECT GROUP<>.  
SELECT AGE>=21, AGE<65
SELECT SEX$='Female', AGE>=25
```
SUBMIT

Purpose

The SUBMIT command lets you send a text (not binary) command file to TreeNet for processing in batch mode. The commands are executed as if you had typed them from the keyboard. If the file of commands is in the current directory (or the directory specified with Utilities/Defaults/Path) and has a .CMD extension, you need only specify the basic filename (without the extension). Otherwise, specify a pathname and the complete filename enclosed in single or double quotation marks.

The command syntax is:

SUBMIT <file> [/ECHO ]

The ECHO option displays the commands on the screen as TreeNet reads them from the SUBMIT file.

Note that screen output is automatically scrolled when you SUBMIT commands. You can use the OUTPUT command to specify an ASCII text file to review the output that is quickly generated.

Examples:

SUBMIT COMMANDS  (reads from file COMMANDS.CMD in current directory)
SUBMIT '\ANALYSES\NEWJOB .CMD'  (reads from named file)
SUBMIT JOB / ECHO  (reads JOB.CMD and displays commands on screen)
TRANSLATE

Purpose

The TRANSLATE command generates reports and splitting rules from a grove file. A grove file must be named by the GROVE command prior to using the TRANSLATE command, otherwise the most recently created grove file will be used.

The command syntax is:

```
TRANSLATE [ LANGUAGE = ENGLISH | SAS | C | PMML
    VLIST = <yes/no>,
    TLIST = <yes/no>,
    DETAILS = <yes/no>,
    SURROGATES = <yes/no>,
    SMI = "SAS missing value string",
    SBE = "SAS begin label",
    SDO = "SAS done label",
    SNO = "SAS node prefix",
    STN = "SAS terminal node prefix"
```

To obtain a summary of a grove file, including a listing of the trees, learn and test sample, validation sample, and makeup in the grove and the variables used to build the trees, simply issue the TRANSLATE command with no options.

Example:

```
GROVE "mygrove.grv"
TRANSLATE
```
USE

Purpose

The USE command reads data from the file you specify. You may specify the root of the filename if the file resides in the current directory (usually C:\Program Files\TreeNet 2.0\Sample Data\), or the directory specified with Utilities/Defaults/Path or the FPATH command. If you specify a path, you must provide the complete filename with the appropriate extension, and surround the whole pathname/filename with single or double quotation marks.

The command syntax is:

```plaintext
USE <file>
```

Examples:

```plaintext
USE MYDATA  (reads from MYDATA.SYS)
USE '\MONTHLY\SURVEY.SYS'
```
WEIGHT

Purpose

The WEIGHT command identifies a case-weighting variable.

The command syntax is:

   WEIGHT=<variable>

in which <variable> is a variable present in the USE dataset. The WEIGHT variable must be numeric containing any non-negative real values.
XY PLOT

Purpose

The XY PLOT command produces 2-D scatter plots, plotting one or more y variables against an x variable in separate graphs.

This command only affects the classic text output.

The command syntax is:

```
XY PLOT <yvar1> [, <yvar2>, <yvar3>] * <xvar> [ / FULL, TICKS | GRID, WEIGHTED, BIG ]
```

The plot is normally a half screen high: the FULL and BIG options will increase it to a full screen (24 lines) or a full page (60 lines).

TICKS and GRID add two kinds of horizontal and vertical gridding.

WEIGHTED requests plots weighted by the WEIGHT command variable.

NORMALIZED scales the vertical axis to 0 to 1 (or -1 to 1).

Examples:

```
XY PLOT IQ*AGE / FULL, GRID
XY PLOT LEVEL(4-7)*INCOME / NORMALIZED
XY PLOT AGE, WAGE, INDIC*DEPVAR(2) / WEIGHTED
```

Only numerical variables may be specified.

Variable groups may be used in the XY PLOT command similarly to variable names.
Appendix II

BASIC Programming Language

This chapter provides an overview of the built-in programming language available within TreeNet.
BASIC Programming Language

TreeNet, and other Salford Systems’ modules contain an integrated implementation of a complete BASIC programming language for transforming variables, creating new variables, filtering cases, and database programming. Because the programming language is directly accessible anywhere in TreeNet, you can perform a number of database management functions without invoking the data step of another program.

The BASIC transformation language allows you to modify your input files on the fly while you are in an analysis module. To save permanent copies of your changed data, please contact Salford Systems to obtain a copy of their DATA module. We expect users will find that they can accomplish almost any required data manipulation involving a single data file.

Although this integrated version of BASIC is much more powerful than the simple variable transformation functions sometimes found in other statistical procedures, it is not meant to be a replacement for more comprehensive data steps found in general use statistics packages. At present, integrated BASIC does not permit the merging or appending of multiple files, nor does it allow processing across observations. In Salford Systems’ statistical analysis packages, the programming workspace for BASIC is limited and is intended for on-the-fly data modifications of 20 to 40 lines of code (though custom large workspace versions will accommodate larger BASIC programs). For more complex or extensive data manipulation, we recommend you use the large workspace for BASIC in DATA or your preferred database management software.

The remaining BASIC help topics describe what you can do with BASIC and provide simple examples to get you started. The BASIC help topics provide formal technical definitions of the syntax.

Getting Started with BASIC Programming Language

Your BASIC program will consist of a series of statements that all begin with a “%” sign. These statements could comprise simple assignment statements that define new variables, conditional statements that delete selected cases, iterative loops that repeatedly execute a block of statements, and complex programs with the flow control provided by GOTO statements and line numbers. Thus, somewhere before a HOT! Command such as ESTIMATE or RUN in a Salford module, you might type:

```
% LET BESTMAN = WINNER
% IF MONTH=8 THEN LET GAMES = BEGIN
% ELSE IF MONTH>8 LET GAMES = ENDED
% LET ABODE= LOG (CABIN)
% DIM COLORS(10)
% FOR I= 1 TO 10 STEP 2
% LET COLORS(I) = Y * I
% NEXT
% IF SEX$="MALE" THEN DELETE
```
The % symbol appears only once at the beginning of each line of BASIC code; it should not be repeated anywhere else on the line. You can leave a space after the % symbol or you can start typing immediately; BASIC will accept your code either way.

Our programming language uses standard statements found in many dialects of BASIC.

**BASIC: Overview of BASIC Components**

**LET**
Assigns a value to a variable. The form of the statement is:

% LET variable = expression

**IF...THEN**
Evaluates a condition, and if it is true, executes the statement following the THEN. The form is:

% IF condition THEN statement

**ELSE**
Can immediately follow an IF...THEN statement to specify a statement to be executed when the preceding IF condition is false. The form is:

% IF condition THEN statement
% ELSE statement

Alternatively, ELSE may be combined with other IF–THEN statements:

% IF condition THEN statement
% ELSE IF condition THEN statement
% ELSE IF condition THEN statement
% ELSE statement

**FOR...NEXT**
Allows for the execution of the statements between the FOR statement and a subsequent NEXT statement as a block. The form of the simple FOR statement is:

% FOR
% statements
% NEXT

For example, you might execute a block of statements only if a condition is true, as in

%IF WINE=COUNTRY THEN FOR  
%LET FIRST=CABERNET  
%LET SECOND=RIESLING  
%NEXT
When an index variable is specified on the FOR statement, the statements between the
FOR and NEXT statements are looped through repeatedly while the index variable
remains between its lower and upper bounds:

% FOR [index variable and limits]
% statements
% NEXT

The index variable and limits form is:

%FOR I= start-number TO stop-number [ STEP = stepsize ]

where I is an integer index variable that is increased from start-number to stop-number in
increments of stepsize. The statements in the block are processed first with I = start-
number, then with I = start-number + stepsize, and repeated until I >=stop-number. If
STEP=stepsize is omitted, the default is to step by 1. Nested FOR–NEXT loops are not
allowed.

**DIM**

Creates an array of subscripted variables. For example, a set of five scores could be set
up with:

% DIM SCORE(5)

This creates the variables SCORE(1), SCORE(2), –, SCORE(5).

The size of the array must be specified with a literal integer up to a maximum size of 99;
variable names may not be used. You can use more than one DIM statement, but be
careful not to create so many large arrays that you exceed the maximum number of
variables allowed (currently 8019).

**DELETE**

Deletes the current case from the data set.

**Operators**

The table below lists the operators that can be used in BASIC statement expressions.
Operators are evaluated in the order they are listed in each row with one exception: a
minus sign before a number (making it a negative number) is evaluated after
exponentiation and before multiplication or division. The "<>" is the "not equal"
operator.

<table>
<thead>
<tr>
<th>Numeric Operators</th>
<th>( )</th>
<th>^</th>
<th>*</th>
<th>/</th>
<th>+</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational Operators</td>
<td>&lt;</td>
<td>&lt;=</td>
<td>&lt;&gt;</td>
<td>=</td>
<td>=&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>Logical Operators</td>
<td>AND</td>
<td>OR</td>
<td>NOT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**BASIC Special Variables**

BASIC has five built-in variables available for every data set. You can use these
variables in BASIC statements and create new variables from them. You may not
redefine them or change their values directly.
### Variable Definitions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CASE</strong></td>
<td>observation number</td>
<td>1 to maximum observation number</td>
</tr>
<tr>
<td><strong>BOF</strong></td>
<td>logical variable for beginning of file</td>
<td>1 for first record in file, 0 otherwise</td>
</tr>
<tr>
<td><strong>EOF</strong></td>
<td>logical variable for end of file</td>
<td>1 for last record in file, 0 otherwise</td>
</tr>
<tr>
<td><strong>BOG</strong></td>
<td>logical variable for beginning of BY group</td>
<td>1 for first record in BY group, 0 otherwise</td>
</tr>
<tr>
<td><strong>EOG</strong></td>
<td>logical variable for end of BY group</td>
<td>1 for last record in BY group, 0 otherwise</td>
</tr>
</tbody>
</table>

BY groups are not supported in TreeNet, so BOG and EOG are synonymous with BOF and EOF.

### BASIC Mathematical Functions

Integrated BASIC also has a number of mathematical and statistical functions. The statistical functions can take several variables as arguments and automatically adjust for missing values. Only numeric variables may be used as arguments. The general form of the function is:

```
FUNCTION(variable, variable, ....)
```

Integrated BASIC also includes a collection of probability functions that can be used to determine probabilities and confidence level critical values, and to generate random numbers.

#### Multiple-Argument Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Definition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG</td>
<td>arithmetic mean</td>
<td>%LET XMEAN=AVG(X1,X2,X3)</td>
</tr>
<tr>
<td>MAX</td>
<td>maximum</td>
<td>%LET BEST=MAX(Y1,Y2,Y3,Y4,Y5)</td>
</tr>
<tr>
<td>MIN</td>
<td>minimum</td>
<td>%LET MINCOST=MIN(PRICE1,OLDPRICE)</td>
</tr>
<tr>
<td>MIS</td>
<td>number of missing values</td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>standard deviation</td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td>summation</td>
<td></td>
</tr>
</tbody>
</table>

#### Single-Argument Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Definition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>absolute value</td>
<td>%ABSVAL=ABS(X)</td>
</tr>
<tr>
<td>ACS</td>
<td>arc cosine</td>
<td></td>
</tr>
<tr>
<td>ASN</td>
<td>arc sine</td>
<td></td>
</tr>
<tr>
<td>ATH</td>
<td>arc hyperbolic tangent</td>
<td></td>
</tr>
<tr>
<td>ATN</td>
<td>arc tangent</td>
<td></td>
</tr>
<tr>
<td>COS</td>
<td>cosine</td>
<td></td>
</tr>
<tr>
<td>EXP</td>
<td>exponential</td>
<td>%LET LOGXY=LOG(X+Y)</td>
</tr>
<tr>
<td>LOG</td>
<td>natural logarithm</td>
<td></td>
</tr>
<tr>
<td>SIN</td>
<td>sine</td>
<td>%LET PRICESR=SQR(PRICE)</td>
</tr>
<tr>
<td>SQR</td>
<td>square root</td>
<td></td>
</tr>
</tbody>
</table>
The following shows the distributions and any parameters that are needed to obtain values for either the random draw, the cumulative distribution, the density function, or the inverse density function. Every function name is composed of three letters:

**Key-Letter:**
This first letter identifies the distribution.

**Distribution-Type Letters:**
- RN (random number), CF (cumulative),
- DF (density), IF (inverse).

### BASIC Probability Functions

TreeNet BASIC also includes a collection of probability functions that can be used to determine probabilities and confidence level critical values, and to generate random numbers.

The following table shows the distributions and any parameters that are needed to obtain values for the random draw, the cumulative distribution, the density function, or the inverse density function. Every function name is composed of two parts:

The "Key" (first) letter identifies the distribution.

Remaining letters define function: RN (random number), CF (cumulative), DF (density), IF (inverse).

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Key-Letter</th>
<th>Random Draw (RN)</th>
<th>Cumulative (C) Density (D) Inverse (I)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beta</td>
<td>B</td>
<td>BRN</td>
<td>BCF(β,p,q) BDF(β,p,q) BIF(α,p,q)</td>
<td>β = beta value p,q = beta parameters</td>
</tr>
<tr>
<td>Binomial</td>
<td>N</td>
<td>NRN(n,p)</td>
<td>NCF(x,n,p) NDF(x,n,p) NIF(a,n,p)</td>
<td>n = number of trials p = prob of success in trial x = binomial count</td>
</tr>
<tr>
<td>Chi-square</td>
<td>X</td>
<td>XRN(df)</td>
<td>XCF(χ²,df) XDF(χ²,df) XIF(α,df)</td>
<td>χ² = chi-squared valued f = degrees of freedom</td>
</tr>
<tr>
<td>Exponential</td>
<td>E</td>
<td>ERN</td>
<td>ECF(x) EDF(x) EIF(a)</td>
<td>x = exponential value</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>FRN(df1,df2)</td>
<td>FCF(F,df1,df2) FDF(F,df1,df2) FIF(α,df1,df2)</td>
<td>df1, df2 = degrees of freedom F = F-value</td>
</tr>
<tr>
<td>Gamma</td>
<td>G</td>
<td>GRN(p)</td>
<td>GCF(γ,p) GDF(γ,p) GIF(α,p)</td>
<td>p = shape parameter γ = gamma value</td>
</tr>
<tr>
<td>Logistic</td>
<td>L</td>
<td>LRN</td>
<td>LCF(x) LDF(x) LIF(α)</td>
<td>x = logistic value</td>
</tr>
<tr>
<td>Distribution</td>
<td>Function</td>
<td>Arguments</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>-----------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>Normal (Standard)</td>
<td>Z</td>
<td>ZRN</td>
<td>ZCF(z)</td>
<td>( z = ) normal z-score</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ZDF(z)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ZIF(a)</td>
<td></td>
</tr>
<tr>
<td>Poisson</td>
<td>P</td>
<td>PRN(p)</td>
<td>PCF(x,p)</td>
<td>( p = ) Poisson parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PDF(x,p)</td>
<td>( x = ) Poisson value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PIF(( \alpha ),p)</td>
<td></td>
</tr>
<tr>
<td>Studentized</td>
<td>S</td>
<td>SRN(k,df)</td>
<td>SCF(s,k,df)</td>
<td>( k = ) parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SDF(s,k,df)</td>
<td>( f = ) degrees of freedom</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SIF(( \alpha ),k,df)</td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>T</td>
<td>TRN(df)</td>
<td>TCF(t,df)</td>
<td>( df = ) degrees of freedom</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TDF(t,df)</td>
<td>( t = ) t-statistic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TIF(( \alpha ),df)</td>
<td></td>
</tr>
<tr>
<td>Uniform</td>
<td>U</td>
<td>URN</td>
<td>UCF(x)</td>
<td>( x = ) uniform value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UDF(x)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UIF(( \alpha ))</td>
<td></td>
</tr>
<tr>
<td>Weibull</td>
<td>W</td>
<td>WRN(p,q)</td>
<td>WCF(x,p,q)</td>
<td>( p = ) scale parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WDF(x,p,q)</td>
<td>( q = ) shape parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WIF(( \alpha ),p,q)</td>
<td></td>
</tr>
</tbody>
</table>

These functions are invoked with either 0, 1, or 2 arguments as indicated in the table above, and return a single number, which is either a random draw, a cumulative probability, a probability density, or a critical value for the distribution.

We illustrate the use of these functions with the chi-square distribution. To generate 10 random draws from a chi-square distribution with 35 degrees of freedom for each case in your data set:

```basic
% DIM CHISQ(10)
% FOR I = 1 TO 10
% LET CHISQ(I) = XRN(35)
% NEXT
```

To evaluate the probability that a chi-square variable with 20 degrees of freedom exceeds 27.5:

```basic
%LET CHITAIL = 1 - XCF(27.5, 20)
```

The chi-square density for the same chi-square value is obtained with:

```basic
%LET CHIDEN = XDF(27.5, 20)
```

Finally, the 5% point of the chi-squared distribution with 20 degrees of freedom is calculated with:

```basic
%LET CHICRIT = XIF(.95, 20)
```

**Missing Values**

The system missing value is stored internally as the largest negative number allowed. Missing values in BASIC programs and printed output are represented with a period or
dot (\"\.",) and missing values can be generated and their values tested using standard
expressions.

Thus, you might type:

\%IF NOSE=LONG THEN LET ANSWER=.
\%IF STATUS=. THEN DELETE

Missing values are propagated so that most expressions involving variables that have
missing values will themselves yield missing values.

One important fact to note: because the missing value is technically a very large
negative number, the expression X < 0 will evaluate as true if X is missing.

BASIC statements included in your command stream are executed when a HOT! Command such as ESTIMATE, APPLY, or RUN is encountered; thus, they are
processed before any estimation or tree building is attempted. This means that any new
variables created in BASIC are available for use in MODEL and KEEP statements, and
any cases that are deleted via BASIC will not be used in the analysis.

**More Examples**

It is easy to create new variables or change old variables using BASIC.  The simplest
statements create a new variable from other variables already in the data set.  For
example:

\% LETPROFIT=PRICE *QUANTITY2* LOG(SQFTRENT), 5*SQR(QUANTITY)

BASIC allows for easy construction of Boolean variables, which take a value of 1 if true
and 0 if false.  In the following statement, the variable XYZ would have a value of 1 if
any condition on the right-hand side is true, and 0 otherwise.

\% LET XYZ = X1<.5 OR X2>17 OR X3=6

Suppose your data set contains variables for gender and age, and you want to create a
categorical variable with levels for  male-senior, female-senior, male-non-senior,
female-non-senior. You might type:

\% IF MALE = . OR AGE = . THEN LET NEWVAR = .
\% ELSE IF MALE = 1 AND AGE < 65 THEN LET NEWVAR=1
\% ELSE IF MALE = 1 AND AGE >= 65 THEN LET NEWVAR=2
\% ELSE IF MALE = 0 AND AGE < 65 THEN LET NEWVAR=3
\% ELSE LET NEWVAR = 4

If the measurement of several variables changed in the middle of the data period,
conversions can be easily made with the following:

\% IF YEAR > 1986 OR MEASTYPE$="OLD" THEN FOR
\% LET TEMP = (OLDTEMP-32)/1.80
\% LET DIST = OLDDIST / .621
\% NEXT
\% ELSE FOR
\% LET TEMP = OLDTEMP
\% LET DIST = OLDDIST
If you would like to create powers of a variable (square, cube, etc.,) as independent variables in a polynomial regression, you could type something like:

```plaintext
% DIM AGEPWR(5)
% FOR I = 1 TO 5
% LET AGEPWR(I) = AGE^I
% NEXT
```

**Filtering the Data Set or Splitting the Data Set**

Integrated BASIC can be used for flexibly filtering observations. To remove observations with SSN missing, try:

```plaintext
% IF SSN=. THEN DELETE
```

To delete the first 10 observations, type:

```plaintext
% IF CASE <= 10 THEN DELETE
```

Because you can construct complex Boolean expressions with BASIC, using programming logic combined with the DELETE statement gives you far more control than is available with the simple SELECT statement. For example:

```plaintext
% IF AGE>50 OR INCOME<15000 OR (REGION=9 AND GOLF=.) THEN DELETE
```

It is often useful to draw a random sample from a data set to fit a problem into memory or to speed up a preliminary analysis. By using the uniform random number generator in BASIC, this is easily accomplished with a one-line statement:

```plaintext
% IF URN < .5 THEN DELETE
```

The data set can be divided into an analysis portion and a separate test portion distinguished by the variable TEST:

```plaintext
% LET TEST= URN < .4
```

This sets TEST equal to 1 in approximately 40% of all cases and 0 in all other cases. The following draws a stratified random sample taking 10% of the first stratum and 50% of all other strata:

```plaintext
% IF DEPVAR = 1 AND URN < .1 THEN DELETE
% ELSE IF DEPVAR<>1 AND URN < .5 THEN DELETE
```

**Advanced Programming Features**

Integrated BASIC also allows statements to have line numbers that facilitate the use of flow control with GOTO statements. Line numbers must be integers less than 32000, and we recommend that if you use any line numbers at all, all your BASIC statements should be numbered. BASIC will execute the numbered statements in the order of the line numbers, regardless of the order in which the statements are typed, and unnumbered BASIC statements are executed before numbered statements.
Here is an example of using the GOTO:

%10 IF PARTY=GOP THEN GOTO 96
%20 LET NEWDEM=1
%30 LET VEEP$="GORE"
%40 GOTO 99
%96 LET VEEP$="KEMP"
%99 LET CAMPAIGN=1
BASIC Programming Language Commands
The following pages contain a summary of the BASIC programming language commands. They include syntax usage and examples.

DELETE Statement

Purpose

Drops the current case from the data set.

Syntax

\% DELETE
\% IF condition THEN DELETE

Examples

To keep a random sample of 75% of a data set for analysis:

\% IF URN < .25 THEN DELETE
**DIM Statement**

*Purpose*

Creates an array of subscripted variables.

*Syntax*

\[
\% \text{DIM } \text{var}(n)
\]

where \( n \) is a literal integer. Variables of the array are then referenced by variable name and subscript, such as \( \text{var}(1), \text{var}(2), \text{etc.} \).

In an expression, the subscript can be another variable, allowing these array variables to be used in \text{FOR...NEXT} loop processing. See the section on the \text{FOR...NEXT} statement for more information.

*Examples*

\[
\% \text{DIM QUARTER}(4)
\% \text{DIM MONTH}(12)
\% \text{DIM REGION}(9)
\]
**ELSE Statement**

*Purpose*

Follows an IF...THEN to specify statements to be executed when the condition following a preceding IF is false.

*Syntax*

The simplest form is:

```% IF condition THEN statement1
% ELSE statement2```

The statement2 can be another IF...THEN condition, thus allowing IF...THEN statements to be linked into more complicated structures. For more information see the section for IF...THEN.

*Examples*

```%  5 IF TRUE=1 THEN GOTO 20
% 10 ELSE GOTO 30
% IF AGE <=2 THEN LET AGEDES$ = "baby"
% ELSE IF AGE <= 18 THEN LET AGEDES$ = "child"
% ELSE IF AGE < 65 THEN LET AGEDES$ = "adult"
% ELSE LET AGEDES$ = "senior"```
FOR...NEXT Statement

Purpose

Allows the processing of steps between the FOR statement and an associated NEXT statement as a block. When an optional index variable is specified, the statements are looped through repetitively while the value of the index variable is in a specified range.

Syntax

The form is:

```
% FOR [index variable and limits]
% statements
% NEXT
```

The index variable and limits is optional, but if used, it is of the form

```
x = y TO z [STEP=s]
```

where x is an index variable that is increased from y to z in increments of s. The statements are processed first with x = y, then with x = y + s, and so on until x = z. If STEP=s is omitted, the default is to step by 1.

Remarks

Nested FOR...NEXT loops are not allowed and a GOTO which is external to the loop may not refer to a line within the FOR...NEXT loop. However, GOTOs may be used to leave a FOR...NEXT loop or to jump from one line in the loop to another within the same loop.

Examples

To have an IF...THEN statement execute more than one statement if it is true:

```
% IF X<15 THEN FOR
% LET Y=X+4
% LET Z=X-2
% NEXT
```
GOTO Statement

Purpose

Jumps to a specified numbered line in the BASIC program.

Syntax

The form for the statement is:

```plaintext
% GOTO ##
```

where ## is a line number within the BASIC program.

Remarks

This is often used with an IF...THEN statement to allow certain statements to be executed only if a condition is met.

If line numbers are used in a BASIC program, all lines of the program should have a line number. Line numbers must be positive integers less than 32000.

Examples

```plaintext
% 10 GOTO 20
% 20 STOP
% 10 IF X=. THEN GOTO 40
% 20 LET Z=X*2
% 30 GOTO 50
% 40 LET Z=0
% 50 STOP
```
**IF...THEN Statement**

*Purpose*

Evaluates a condition and, if it is true, executes the statement following the THEN.

*Syntax*

```plaintext
% IF condition THEN statement
```

An IF...THEN may be combined with an ELSE statement in two ways. First, the ELSE may be simply used to provide an alternative statement when the condition is not true:

```plaintext
% IF condition THEN statement1
% ELSE statement2
```

Second, the ELSE may be combined with an IF...THEN to link conditions:

```plaintext
% IF condition THEN statement
% ELSE IF condition2 THEN statement2
```

To allow multiple statements to be conditionally executed, combine the IF...THEN with a FOR...NEXT:

```plaintext
% IF condition THEN FOR
% statement
% statement
% NEXT
```

*Examples*

To remove outlier cases from the data set:

```plaintext
% IF ZCF(ABS((z-zmean)/zstd))>.95 THEN DELETE
```
**LET Statement**

*Purpose*

Assign a value to a variable.

*Syntax*

The form of the statement is:

```% LET  variable =  expression```

The expression can be any mathematical expression, or a logical Boolean expression. If the expression is Boolean, then the variable defined will take a value of 1 if the expression is true, or 0 if it is false. The expression may also contain logical operators such as AND, OR and NOT.

*Examples*

```% LET AGEMONTH = YEAR - BYEAR + 12*(MONTH, BMONTH)
% LET SUCCESS = (MYSPEED = MAXSPEED)
% LET COMPLETE = (OVER = 1 OR END=1)```
**STOP Statement**

*Purpose*

Stops the processing of the BASIC program on the current observation. The observation is kept but any BASIC statements following the STOP are not executed.

*Syntax*

The form of the statement is:

```
% STOP
```

*Examples*

```
%10 IF X = 10 THEN GOTO 40
%20 ELSE STOP
%40 LET X = 15
```
This chapter provides installation and usage note for our UNIX/Linux operating environments.
Installation Instructions for UNIX/Linux

To install TreeNet in a UNIX-type operating environment:

1. Extract the distribution archive. This can be done with the command
   
   "tar -xvzf <archive name>".

2. If it does not already exist, create a directory to receive files associated with Salford Systems programs. Define the environment variable SALFORD to point to this directory. The syntax to do this with the Korn, Bourne, or Bourne-again (bash) shells is:

   SALFORD=<directory path>
   export SALFORD

   These commands can be placed in $HOME/.profile for individual users, or invoked system-wide by placing them in /etc/profile. On many Linux systems, it is recommended practice to place a script in /etc/profile.d, instead of editing /etc/profile directly. This script should be made executable and have a name that ends in ".sh" (example: /etc/profile.d/salford.sh). The advantage of doing this is that it prevents your customizations from being lost if /etc/profile gets overwritten during the course of an upgrade.

   The C-shell syntax is:

   setenv SALFORD <directory path>

   This can be placed in $HOME/.login for individual users, or /etc/csh.login, to define this variable system-wide. Again, on Linux systems that use /etc/profile.d, it is recommended that a separate script be placed in this directory. The script should be made executable and be given the .csh extension (example: salford.csh).

   A suggested location for $SALFORD is /usr/local/salford.

3. Move the TreeNet executable (treenet) to a directory in the path, such as /usr/local/bin. The other files included in this distribution can be moved to $SALFORD.

4. Execute the TreeNet executable. You will see something like the following:

   =============
   No valid license found. Please contact Salford Systems to
   confirm/arrange licensing, noting the following parameters:

   HOSTID=3240823e   MODULE=Tree   VERSION=5.0

   Press ENTER to continue.
   Salford Systems: 8880 Rio San Diego Drive, #1045
   San Diego, California, 92108, USA.
   Phone: (619) 543-8880
   =============

   The information on the final line before the ENTER prompt (HOSTID, MODULE, VERSION), together with the name of the OS, the name of the person and/or organization licensing the software and a return e-mail address should be sent to unlock@salford-systems.com. Once the license is verified, a license string will be e-
mailed back. This license string must then be placed in $SALFORD/license.txt, a text file with one license string per line. There may be any number of license strings in the file. A separate license string must be obtained for each computer on which TreeNet is to execute.

Alternatively, this information may be faxed to (619)543-8888 with a return fax number. The license string can then be faxed back. The disadvantage is that the license string must then be hand-typed into license.txt.

6. To reduce the size of the distribution archive (it is often e-mailed), the TreeNet manual is not included. A soft copy of this document can be downloaded from ftp://ftp.salford-systems.com/docs/TreeNet2.pdf. It can be read in a PDF viewer such as Adobe Reader, GSView, or xpdf.


The file treenet2.unix.txt, included with this distribution, contains UNIX-specific usage notes. We recommend that it be carefully reviewed, since it deals with the issues that most often prompt technical support questions from our UNIX and Linux users.

Usage Notes for UNIX/Linux

The nature of UNIX-like operating environments affects the operation of TreeNet in non-trivial ways. In addition, there are aspects of TreeNet which are of particular concern to users in a console environment. Such issues will be discussed here.

General UNIX notes:

1. TreeNet's command interpreter is case-insensitive; in fact, commands are generally converted internally to upper-case letters. For example: "use this" is converted internally to "USE THIS", causing TreeNet to search for a file named "THIS.SYS", or if it does not exist, "THIS.CSV"; it will not search for "this.sys". The only exception to this rule is that text placed between quotation marks is not converted, remaining in its original case. For example, "use 'that.sys'" is converted to "USE 'that.sys'", causing TreeNet to open "that.sys" for reading. For this reason, among others, it is generally more convenient to give Systat datasets upper-case names.

2. The Systat file format, traditionally used by TreeNet, and other Salford Systems programs, is platform-dependent. There are three known variations on the platforms we currently support:
   a. Big-endian UNIX (SPARC/Solaris, IRIX, AIX, HP/UX)
   b. Little-endian UNIX (Alpha, Linux-i386)
   c. DOS/Windows

   The consequence of this is that Systat datasets created on Windows PCs cannot be read by TreeNet under UNIX, unless DBMS/COPY is installed (TreeNet will use the DBMS/COPY libraries to translate, if available). This is less of a problem than may seem because TreeNet will read CSV files directly. Thus, when dealing with "foreign" Systat datasets, one can simply use DBC (available from Salford Systems) on the
originating platform to convert the dataset to CSV format and then have TreeNet on the new platform read the CSV file. In addition because the CSV format is more flexible than the Systat format, we recommend its use, particularly when the analysis dataset has long field names (more than 8 bytes), or long character fields (more than 12 bytes).

3. It is always important to use binary mode when copying non-text files from a DOS/Windows environment to a UNIX environment (or vice-versa). Failure to do so *will* cause the files to be corrupted.

4. TreeNet will use the DBMS/COPY libraries to read and write in any file format supported by DBMS/COPY, if DBMS/COPY is properly installed and licensed. TreeNet looks at the following environment variables to determine whether a usable DBMS/COPY is installed:

   - `DBMSCOPY`: Location of the DBMS/COPY installation directory
   - `DBMSCOPY:LIC`: Location of the DBMS/COPY license file

To access a file through the DBMS/COPY interface, the file name should be quoted, with the DBMS/COPY pseudo-extension in square brackets, if different from the actual extension. Examples:

   ```
   rem SAS 6.x for Alpha
   use "something.ssd04 [ssd04dec]"
   
   rem DBase
   save "something.dbf"
   ```

See the DBMS/COPY documentation for further information on pseudo-extensions.

In order to read a text file through the DBMS/COPY interface, a data dictionary file (.dct) with the same base name as the data file must be present. We recommend that dictionary files be created with interactive DBMS/COPY, rather than by hand. In general, TreeNet's native CSV support is much more convenient to use than the DBMS/COPY interface when reading and writing ASCII files and we recommend its use, where possible.

Currently, UNIX versions of TreeNet can only use DBMS/COPY 7.x; Version 1.x is not supported. The DBMS/COPY interface is supported on HP Tru64 (aka Digital) UNIX, Sun Solaris, IBM AIX, and Linux, as well
as MS-Windows. It is not supported on SGI IRIX, or HP/UX.

TreeNet 1.0 Console Notes:

1. The TreeNet manual deals extensively with the TreeNet GUI, currently available only for Microsoft Windows. UNIX versions of TreeNet are text-based, command-line interpreters, designed to operate primarily in batch mode. This being the case, the portions of the manual dealing with the GUI are not applicable to console versions of TreeNet (including all UNIX versions). UNIX users will, however, want to pay particularly close attention to Chapter 8 (Working with Command Language), Appendix I (Command Reference), and Appendix II (BASIC Programming Language), which deal specifically with the command-line interface.

2. Console TreeNet will accept a the name of a command file as an argument.

   To run something.cmd from the command prompt, type:
   
   mars something.cmd

   When a command file is executed in this way, console output is normally supressed as if the "ECHO OFF" had been given (make sure there's an OUTPUT command in the file). If ECHO ON appears in the command file, console output is written to the screen in the same manner as when MARS is running interactively. Processing terminates when all statements have been executed, or if a QUIT statement is encountered. Running TreeNet without an argument causes it to run interactively. Type "quit" at the prompt to exit.

3. One can create a command file that will execute any time TreeNet or other Salford Systems programs are executed. This file is named SALFORD.CMD
and can be located in the directory where TreeNet is executing, or in the directory indicated by the SALFORD environment variable. If SALFORD.CMD exists in both directories, only the one in your current working directory will execute. Once SALFORD.CMD is executed, your command file will execute, if TreeNet is running in batch mode (see item 2), or you will get a command prompt if TreeNet is running interactively.

Generally, one should use SALFORD.CMD to specify global options that apply to all Salford Systems procedures, rather than those specific to a particular procedure such as CART, MARS, or TreeNet. Here is an example that sets default display precision to 7 decimal places and specifies the use of generic ASCII characters in the output:

```
format=7
charset generic
```

4. By default, TreeNet reads and writes Systat-format datasets.

If, when opening an input file with the USE command, unquoted file name is given without an extension, TreeNet will convert the name to uppercase and add the .SYS extension. If no such file is found, TreeNet will search for the file with each of the following extensions, in turn:

```
sys, .SYD, .syd, .CSV, .TXT, .DAT
```

The first three extensions are assumed to denote a Systat-format dataset; the others are taken as indicative of a CSV file.

If the extension is given in an unquoted filename, the entire name is converted to uppercase. Thus, the following command attempts to open DATA.CSV, in the default directory:

```
use data.csv
```

If the filename is quoted, the literal name is used unaltered. Thus, the following command attempts to open data.csv in the current directory:

```
use "data.csv"
```

When an output dataset is specified with the SAVE command, the native Systat format is assumed if no extension is given. Thus, the following command saves SCORES.SYS:

```
save scores
```

To write to a CSV file named SCORES.CSV, one can use the following:

```
save scores.csv
```

As with the USE command, quoted filenames are taken literally.
5. While it's not documented in either the published manual, nor by the help system, the FPATH command can be quite useful, when working with TreeNet in a console environment.

The FPATH command can be used to specify file path prefixes for different types of input and output files. For example, the following command will cause TreeNet to read and write files in Salford, under your home directory by default.

```
fpath "~/Salford"
```

Thereafter, if one gives an input/output command such as USE, SAVE, etc, TreeNet will look in ~/Salford unless the filename is quoted or the FPATH command is canceled by giving an FPATH command without arguments.

One can also specify different default directories for different sorts of files. To specify a default directory for input datasets, type:

```
fpath <pathname> /use
```

To specify a default directory for output datasets, use:

```
fpath <pathname> /save
```

For command files, use:

```
fpath <pathname> /submit
```

For text output files, use:

```
fpath <pathname> /output
```

FPATH without arguments restores the default, which is to use the current working directory. FPATH with an option but no pathname restores the default for the specified file type.

6. Console TreeNet can allocate arbitrary amounts of memory. The default workspace size is 25 MB, but this can be altered with either the SALFORD_M environment variable, or the -m command-line flag. We suggest that SALFORD_M be set in the system-wide startup files (/etc/profile and /etc/csh.login) as appropriate for the hardware. Since TreeNet does not use much of the static workspace traditionally employed by Salford Systems applications, such as CART or MARS, you probably will not need to change the default, unless you also are using CART, which also recognizes SALFORD_M. In this case, you may want to run TreeNet with the -m25 flag to avoid allocating unnecessary workspace.

7. TreeNet has a number of other command-line options, which can be shown by invoking TreeNet with the -h flag:

Command line syntax is:

```
treenet [options] [commandfile] [options]
```
Options are:

- **-e**  
  Echo results to console
- **-q**  
  Quiet, suppress all output including errors
- **-o<output_file>**  
  Direct text results to a file
- **-u<use_file>**  
  Attach to a dataset
- **-d<Path>**  
  Identify DBMSCOPY dll path
- **-t<Path>**  
  Identify scratch file path
- **-s<MBytes>**  
  Data amount in MB, subject to license threshold
- **-m<MBytes>**  
  Model space in MB, subject to hardware limits
- **-p<MBytes>**  
  Data paging allocation in MB (0: disk based)
- **-l<optional_logfile>**  
  Error/warnings to text logfile
- **-mt<Nnumeric,Nchar>**  
  Lookup table capacities, 0 to grow without bound

E.g.:

```
treenet -e model1.cmd
```
```
treenet /DataMining/Jobs-1/simulate.cmd -q
```
```
treenet job1.cmd -o/RESULTS/job1.txt -u/AnalysisData/sample1.sys
```
```
treenet -d/Progra~1/DBMSCopy7 -u/MyData/joint_data.xls[xls5]
```
```
treenet -s512 -p64 -m128
```

Environment variables can be used in lieu of command line switches:

```
SALFORD_S in lieu of -s
SALFORD_M in lieu of -m
SALFORD_P in lieu of -p
```
Appendix IV

Bibliography


## Index

### A
- absolute deviation · 28
- absolute error curves · 9
- accessing data · 15
- Accessing your data · 19
- additive · 34, 35, 36, 37
- Advanced Programming Features · 126
- Alternative Control Modes · 57
- APPLY command · 50
- Apply results · 52
- apply the model · 29
- applying data · 50
- Applying Models · 49
- ASCII files · 16, 17, 18, 78
  - character variables · 17
  - numeric variables · 17

### B
- BASIC Mathematical Functions · 122
- BASIC Probability Functions · 123
- BASIC programming language · 57
- BASIC Programming Language · 57, 118
- BASIC Programming Language Commands · 128
- BASIC Programming Language Overview · 119
- BASIC Special Variables · 122
- batch command file · 57
- batch processing · 56, 58
- beginning of file · 122
- beginning of group · 122
- bins · 32
- breakdown parameter · 28
- Building Classification Models · 41
- Building Regression Models · 23

### C
- cart.ini · 20
- case weights · 24, 67
- Cases in Bin · 31
- categorical predictors · 36
- categorical split storage · 72
- categorical target variable · 42, 52
  - high-level · 42
- categorical variable · 24
- CATEGORY command · 61, 63, 80
- CDF command · 81
- character variable · 17, 25, 42
- character variable names · 17, 18
- CHARSET command · 82
- class assignment · 50
- CLASS command · 83
- Classification Modeling Run · 63

### 2
- 2-D slices · 36

### 3
- 3-D plots · 13
- 3-D surface · 36
G

Gains Chart · 10, 30
interpretation of · 32
Gains Chart computing
% Pop · 31
% Target in Bin · 31
Cases in Bin · 31
Cum % Pop · 31
Cum % Target · 31
Cum Lift · 31
Lift Pop · 31
Gains Chart tab · 10, 30
gains table · 30
Getting Started · 1
GOTO command · 132
GROUP command · 91
GROVE command · 62, 65
grove file
groving · 73
saving · 73
viewing · 73
grove file · 29, 50, 51
saving · 29

H

Help · 78
HELP command · 92
HISTOGRAM command · 93
Huber-M · 28

I

ID variables · 50
IDVAR command · 94
IF...THEN command · 120, 133
increase the importance · 46
increasing costs · 46
independent testing · 25
independent variable · 24
influence · 28
input files
default directory · 5
Installation
Compact · 3
Custom · 3, 20
Permissions · 3
procedure · 3
Typical · 3
Installation Instructions · 2
interpretation of gains · 32
interpreting model · 36

K

KEEP command · 61, 63, 95

L

LABEL command · 96
language
command-line · 56
learn data · 25
learn error curve · 28, 30
learn sample · 45, 52
Least absolute deviation · 28
Least squares · 28
least squares deviation · 30
LET command · 120, 134
Lift Pop · 31
likelihood deviation curves · 39
likelihood function · 44
LIMIT command · 97
location of the DBMS/COPY · 20
Logical Operators · 121
logistic likelihood · 28, 29, 38
logistic regression · 38, 39
LOPTIONS command · 61, 63, 98
loss criterion
Huber-M · 28
least absolute deviation · 28
least squares · 28
logistic likelihood · 29
loss function · 28, 30

M

MART ATOM command · 62, 64
MART BREAKDOWN command · 62
MART command · 99
APPLY · 99
ATOM · 99
BREAKDOWN · 100
FULLREPORT · 100
GB · 100
GO · 99
INFLUENCE · 100
LEARNRATE · 100
LOSS · 100
MAXTREES · 99
MORETREES · 99
MP · 101
MV · 101
NODES · 99
PF · 101
PLOTS · 101
PP · 101
PROB · 100
SC · 100
SHRINK · 99
ST · 100
SUBSAMPLE · 100
STOP command · 135
SUBMIT command · 113
Submit Command File · 60
Submit Command File dialog · 58
Submit Current Line to End · 58
Submit Window · 58, 59
Submitting Batch Files · 58
Subsample fraction · 28
Summary Reports
  Gains Chart · 30
  Prediction Success Table · 45
  Variable Importance · 32
summary stats · 70
System Requirements
  Minimum · 2
  Recommended · 2

Target · 8, 24
target class · 75
target variable · 24, 30, 38, 42, 61
categorical · 52
continuous · 52
logistic likelihood values · 29, 38
not available for prediction · 52
temporary files
default location · 5
test data · 25
test error curve · 30
test sample · 39, 45, 52, 67
test set error · 30
Testing tab · 7, 8, 25, 26, 61, 64
  A fraction of cases selected at random… · 25
  A test sample is contained in a separate file · 25
  A variable separates learn and test samples · 25
  No independent testing · 25
text files · 16
toolbar icon
  Model Setup · 24
  Options · 69
  Score Data · 50
  View Data · 77
train sample · 39
training data · 25, 30
transforming variables · 119
TRANSLATE command · 114
tree size · 27, 30
tree storage · 72
Tree Type · 8, 24, 42
TreeNet Apply results · 52
TreeNet model · 29
TreeNet models · 50
TreeNet Notepad · 58
TreeNet Output window · 9, 24, 52
TreeNet Plot Selection · 11, 34, 52
TreeNet Reports · 10, 11, 30, 32
TreeNet Results · 9, 10, 11, 29, 30, 34, 36, 44, 74, 75
TreeNet tab · 26, 38, 40, 62, 64, 71, 72
  Influence trimming factor · 28
  Maximum nodes per tree · 27
  Maximum number of trees… · 27
  Minimum number of training observations in terminal nodes · 27
  M-Regression breakdown · 28
  Number of trees to use · 27
  Regression loss criterion · 28
  Regularization shrinkage factor · 27
  Subsample fraction · 28
  trimming factor · 28
Two-Variable Dependence Plots · 33, 35
Two-Variable Stats · 36, 37
typographical conventions · viii

UNIX platform · 56
USE command · 61, 115
Using DBMS/COPY · 16

Variable Dependence Plots · 33
variable importance · 32
variable importance · 10, 71, 100, 101
Variable Importance tab · 10
variable names · 16, 17, 18
  ASCII text · 18, 19
  character · 18
  DBMS/COPY · 18
  examples · 18
  reference to · viii
  subscripts · 18
variable selection
  categorical · 24
dependent · 24
ID · 50
independent · 24
multiple variables · 24
predictor · 24
target · 24
weight · 24, 50, 67
Variable Stats · 36
variables
  character · 17
  number of · 16
  transforming · 119
View menu
  Open Command Log… · 58
  Update Command Log · 58
  View Data · 77
viewing data · 77
Viewing Grove Files · 73
WEIGHT command · 62, 64, 116
weight variable · 24, 50, 67
Weights · 67
Working Directories · 4

XYPLOT command · 117