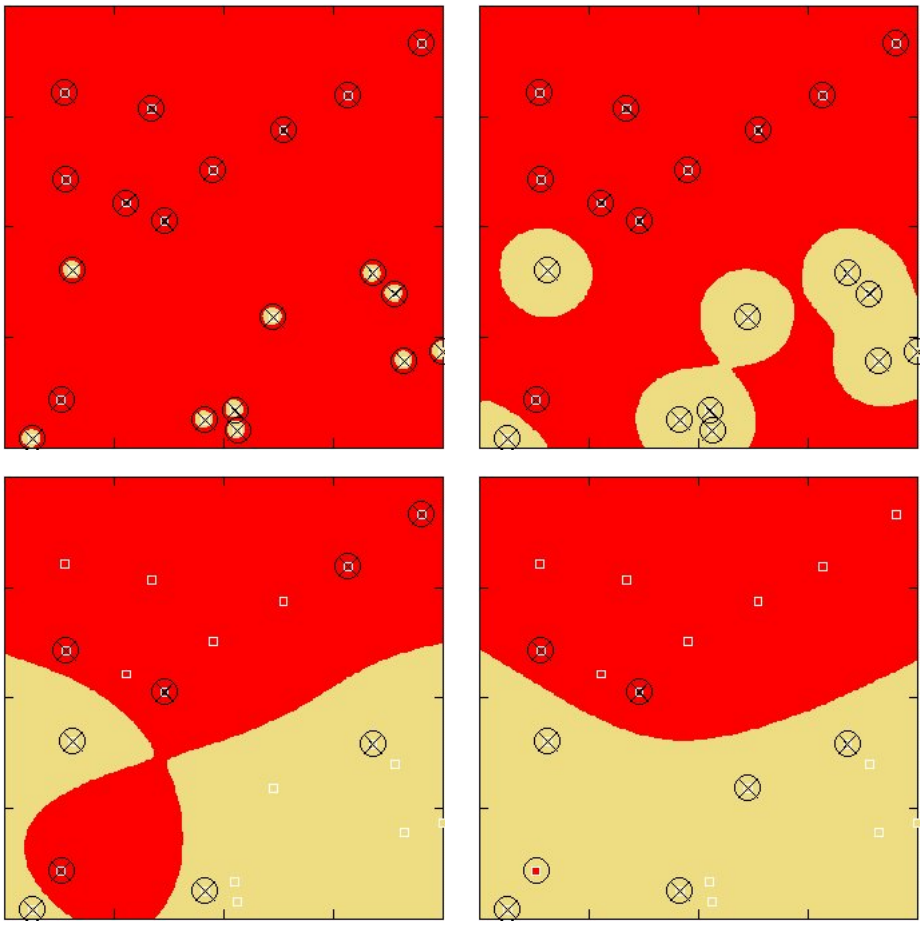
LNKnet User's Guide



Linda Kukolich
Richard Lippmann

MIT Lincoln Laboratory
10 February 2004

LNKnet User's Guide[*]

Linda Kukolich and Richard Lippmann

Revision 4, February 2004

MIT Lincoln Laboratory

*This work was sponsored by the Department of the Air Force. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Air Force.

Printed:
June 1993
Revision 1, June 1994
Revision 2, April 1995
Revision 3, May 1999
Revision 4, February 2004

Acknowledgments

Support for the development of LNKnet was provided by a number of government agencies and MIT Lincoln Laboratory. Individuals from many organizations helped support this work. They include Peter Blankenship, Jim Cupples, Laurie Fenstermacher, John Hoyt, Barbara Yoon, Tod Luginbuhl, and Roy Streit. The first user interface and some of the first LNKnet classifiers were written by Dave Nation during a six-month stay at MIT Lincoln Laboratory. Dave also contributed many ideas that helped structure LNKnet software. Many classification and feature selection algorithms were first programmed and tested by Yuchun Lee, Kenney Ng, Eric Chang, William Huang, and Charles Jankowski before they were rewritten for incorporation into LNKnet. Valuable feedback concerning the user interface and LNKnet software was provided by Mike Richard, students at the Air Force Institute of Technology, and by many members of the Speech Systems and Technology, Information Systems Technology, and Machine Intelligence Groups at MIT Lincoln Laboratory. The selection of algorithms included in LNKnet was strongly influenced by experiments performed on speech, computer intrusion detection, and other data bases at MIT Lincoln Laboratory, by results presented at the annual "Neural Information Processing Systems - Natural and Synthetic" conference, and by the results of studies performed as part of the ARPA Neural Network program directed by Barbara Yoon.

Public Domain Distribution Requirements

Table of Contents

CHAPTER 1 Introducing LNKnet

1.1 Overview

LNKnet software was developed to simplify the application of the most important statistical, neural network, and machine learning pattern classifiers. The acronym LNK stands for the first initials of three principal programmers (Richard Lippmann, Dave Nation, and Linda Kukolich). An introductory article to LNKnet, which is meant to supplement this user's guide, is available in [27]. This article reviews approaches to pattern classification and illustrates how LNKnet was applied to three different applications. LNKnet software was originally developed under Sun Microsystem's Solaris 2.5.1 (SunOS 5.5.1) UNIX operating system under Sun Open Windows. It was then ported to Solaris 2.6 (SunOS 5.6) and to Red Hat Linux. It was also recently modified to run under Microsoft Window's operating systems using the Cygwin environment. Binary versions of LNKnet are provided for Red Hat Linux, Solaris 2.6 and higher, and the Windows Cygwin environment. Source code is also provided and it is relatively easy to recompile LNKnet under other versions of Linux and Unix because the GNU auto configuration tools are used to control compilation. All illustrations and descriptions in this guide show windows and plots as they appear under Solaris 2.5.1 using Open Windows except for Support Vector Machine windows and plots which are as they appear under Red Hat Linux. Windows and plots appear slightly different under the other operating systems. LNKnet includes a graphical user interface to over 22 pattern classification, clustering, and feature selection algorithms. Decision region plots, scatter plots, histograms, structure plots, receiver operating characteristics plots, and other types of visual outputs are provided. Experiment log files and plots can be reviewed from the LNKnet graphical user interface. Classifiers can be trained on data bases with thousands of input features and millions of training patterns.

The three primary approaches to using LNKnet are shown in Figure 1.1. Experimenters can use the LNKnet point-and-click user interface, manually edit shell scripts that contain LNKnet commands to run batch jobs, or embed generated C versions of trained LNKnet classifiers in application programs. The point-and-click graphical user interface, listed on the top of Figure 1.1, can be used to rapidly and interactively experiment with classifiers on new data bases. This makes it relatively easy to explore the effectiveness of different pattern classification algorithms, to perform feature selection, and to select algorithm parameters appropriate for different problems. A new data base must first be put into a simple ASCII format that can be hand-edited using a text editor. Users then make selections on LNKnet windows using a mouse and keyboard, and run experiments by pushing buttons using the mouse. A complex series of experiments on a new moderate-sized data base (10,000's of patterns) can be completed in less than an hour.

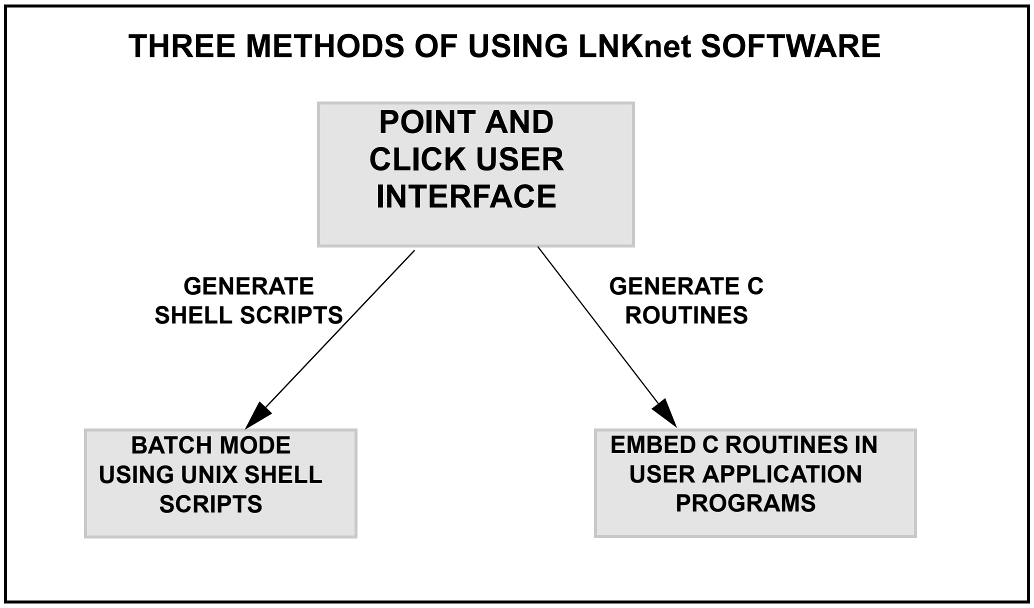

THREE METHODS OF USING LNKnet SOFTWARE
POINT AND CLICK USER INTERFACE
GENERATE SHELL SCRIPTS
GENERATE C ROUTINES
BATCH MODE USING UNIX SHELL SCRIPTS
EMBED C ROUTINES IN USER APPLICATION PROGRAMS

FIGURE 1.1 Experimenters can use the LNKnet point-and-click user interface, manually edit shell scripts that contain LNKnet commands to run batch jobs, or embed generated C versions of trained LNKnet classifiers in application programs.

Use of the point-and-click interface requires no knowledge of UNIX shell scripts, of C programming, or of how the algorithms are implemented.

Users who want to run long batch jobs can edit the shell scripts produced by the point-and-click interface and run these customized shell scripts. This simplifies repetitive application of the same algorithm to many data files and can automate the application of LNKnet when a batch mode is desirable. It requires understanding of shell scripts and of arguments to LNKnet programs. Shell scripts are almost always used for large data bases after initial explorations on smaller data subsets using the point-and-click interface.

In addition to on-line and batch control, C programmers can embed C source code that implements LNKnet subroutines and libraries in user application programs. This use of LNKnet has been simplified by providing filter programs which read in LNKnet files that define trained classifiers and create C source code subroutines to implement those classifiers. This feature of LNKnet allows classifiers to be run on any computer that has a C compiler.

This user's guide demonstrates all three approaches to using LNKnet. It primarily provides a comprehensive description of the LNKnet graphical user's interface. It also shows how shell scripts produced using the graphical user interface can be edited to create batch jobs (see Section 7.4). In addition, it describes how filter programs (mlp2c, knn2c, etc.) can be used to generate C source code to implement LNKnet classifiers and how this source code can be embedded in a user's program (see Section 7.2).

This guide assumes that the reader is familiar with the basic concepts of pattern classification. The article mentioned above [27], provides a brief introduction to LNKnet and pattern classification. Recent reviews of pattern classification techniques including neural networks and machine learning approaches are available in [2,7,14,40]. Good older discussions of pattern classification are available in [1,9,30]. Algorithmic descriptions of the classifiers included in LNKnet are included in the references listed in Table 1.1. Many of these algorithms are also described in [5,7,14].

	SUPERVISED TRAINING	COMBINED UNSUPERVISED-SUPERVISED TRAINING	UNSUPERVISED TRAINING (Clustering)
NEURAL NETWORK ALGORITHMS	Back-Propagation(BP) [21,25] Adaptive Stepsize BP [21] Cross-Entropy BP [36] Top-2-Difference BP [10,11] Hypersphere Classifier [1] Committee [7]	Radial Basis Function (RBF) [28] Incremental RBF (IRBF) [28] Top-2-Diff IRBF [10,11] Learning Vector Quantizer [21] Nearest-Cluster Classifier [7,28]	Leader Clustering [12,28]
CONVENTIONAL PATTERN CLASSIFICATION ALGORITHMS	Gaussian Linear Discriminant [7] Quadratic Gaussian [7] K-Nearest Neighbor (KNN) [7] Condensed KNN [7,28] Binary Tree [3, 21] Parzen Window [7,39] Histogram [7] Naive Bayes Classifier [14] Support Vector Machine [5]	Gaussian Mixture (GMIX) Classifier [28] Diagonal/Full Covariance GMIX Tied/Per-Class Centers GMIX	K-Means Clustering [7] E&M Clustering [28]
FEATURE SELECTION ALGORITHMS	Canonical Linear Discriminant Analysis [7,9] Forward and Backward Search using N-fold Cross Validation [6]		Principal Components Analysis [7,9]

TABLE 1.1: Current LNKnet Algorithms

1.2 Algorithms

Table 1.1 lists the static pattern classification, clustering, and feature selection algorithms that are available in LNKnet. Algorithms include classifiers trained using supervised training with labeled training data, clustering algorithms trained without supervision using unlabeled training data, and classifiers that use clustering to initialize internal parameters and then are trained further with supervised training. Canonical linear discriminant and principal components analyses are provided to reduce the number of input features using new features that are linear combinations of old features. Forward and backward searches are provided to select a small number of features from among the existing features. These searches can be performed using any LNKnet classifier with N-fold cross validation or using a nearest neighbor classifier and leave-one-out cross validation. Bracketed references after algorithm names in Table 1.1 are to references in the bibliography that provide detailed descriptions of algorithms. Overall summaries and comparisons of these algorithms are available in [2,11,14,18, 21,22,24,25,28,29,36,40].

1.3 Running a Pattern Classification Experiment

The LNKnet graphical interface is designed to simplify classification experiments. Figure 1.2 shows the sequence of operations involved in the most common classification experiment. First, a classification algorithm is selected. In addition to choosing an algorithm, parameters that affect the structure or complexity of the resulting classifier are selected. These parameters are sometimes called regularization or smoothing parameters. These hand-selected parameters must be modified to match the complexity of a classifier to the complexity of each individual classification task. They include the number of nodes and layers for MLP classifiers and trees, the training time and value of weight decay for MLP classifiers, the order of polynomial Support Vector Machines (SVMs), the width for Gaussian kernel SVMs, the number of mixture components for Gaussian mixture classifiers, the type of covariance matrix used (full or diagonal, grand average across or within classes) for Gaussian or Gaussian mixture classifiers, and parameters that affect the complexity or structure of other classifiers.

This figure assumes that a database of patterns has already been created. This database contains many labeled feature vectors where the label indicates the class the pattern belongs to and numeric feature values will be used to predict class membership in the future using generated classifiers. When a sufficient number of patterns are available (1000’s of patterns), a database can be split into three separate sets of data designated as training data, evaluation data, and test data. This split often assigns 60% of the patterns to training data, 20% to evaluation data, and 20% to test data. As shown in Figure 1.2, training data is initially used to train the internal weights or trainable parameters in a classifier. The error rate of the trained classifier is then evaluated using evaluation data. Repeated evaluations followed by retraining with different regularization parameter values are used to select a classifier structure that provides low error rate on the evaluation data. Evaluation data is necessary because it is frequently possible to design a classifier that provides a low error rate on training data but that doesn’t provide a low error rate on other data sampled from the same source. Adjusting regularization parameters and altering the classifier structure allows a user to modify the complexity of a classifier to provide good performance on the evaluation data. This approach uses training data to adjust trainable parameters and evaluation data to adjust the classifier size and complexity to provide good generalization. After all regularization parameters are adjusted, the classifier generalization error rate on unseen data is estimated using test data. The use of test data for anything but a single final estimation of generalization error on unseen data makes the error rate estimated using this data suspect. When fewer patterns are available (100’s of patterns), a database is often split into only training and test data and 10-fold cross-validation is used on the training data to select regularization parameters. In this

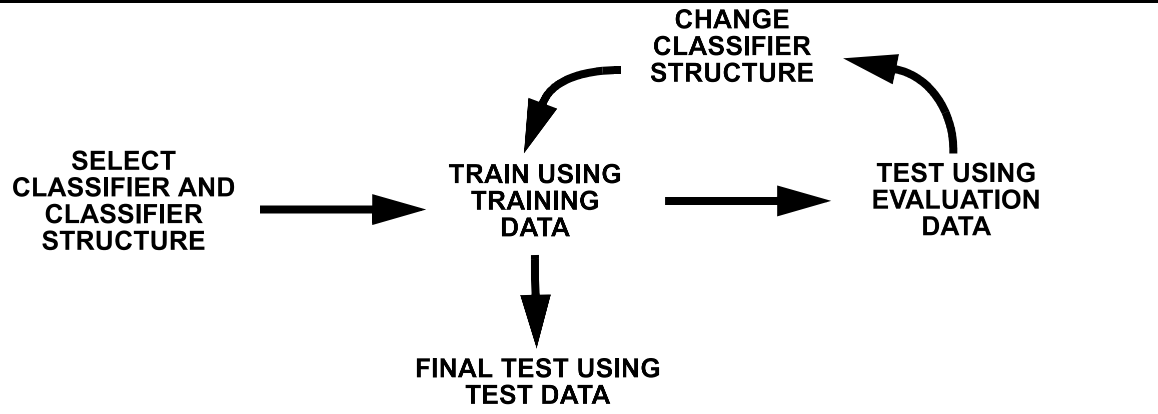

CHANGE CLASSIFIER STRUCTURE
SELECT CLASSIFIER AND CLASSIFIER STRUCTURE
TRAIN USING TRAINING DATA
TEST USING EVALUATION DATA
FINAL TEST USING TEST DATA

FIGURE 1.2 Components of a classification experiment.

case, the split often assigns 60% of the patterns to training data and 40% to test data. When only tens of patterns are available, only the training data is used with 10-fold cross validation. LNKnet automatically performs 10-fold (or more general k-fold) cross validation, but it does not partition the initial database into training, evaluation, and test sets. This partitioning must be performed prior to using LNKnet. It was not automated because the number of partitions depends on the number of patterns, partitioning is often predefined, and partitioning often depends on ancillary pattern characteristics that are not included as pattern features.

1.4 Data Normalization and Feature Selection

One of the most important features of LNKnet is the ability to normalize input data and to use a subset of input features for classification. Input feature normalization algorithms available include simple normalization (normalize each feature separately to zero mean, unit variance), Principal Components Analysis (PCA), and Linear Discriminant Analysis (LDA) [7,9]. Feature selection algorithms include forward and backward searches [9]. These searches select features one at a time based on the increase or decrease in the error rate measured using cross validation and any classifier. Once a forward or backward search, a PCA, or a LDA has been completed, a subset of features can be selected for use in classification. This subset can be the first, and presumably most important features, or a selection of unordered features.

The order in which normalization and feature selection is applied is significant because some normalization methods (PCA and LDA) rotate the input space and change the meaning of features and because feature selection can eliminate or reorder input features. LNKnet applies feature normalization and selection in the order shown in Figure 1.3. First, the full input pattern is normalized. Features are then selected, and the resulting input pattern is presented to the classifier for training or testing. Either of these steps can be skipped, allowing the classifier to use any or all features of the raw data or the normalized data.

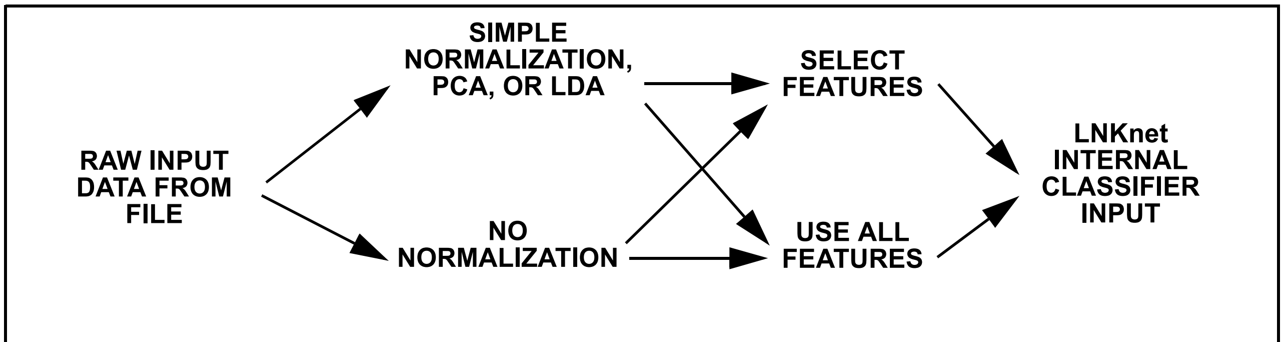


FIGURE 1.3 Feature Selection and Normalization Available in LNKnet.

1.5 Embedding LNKnet Classifiers in User Applications

All LNKnet classifiers have programs which automatically generate C code subroutine versions of the classifier testing algorithm. Parameters for the classifier, number and position of nodes, weight values, etc., are taken from a trained classifier parameter file. The C classification subroutines are self contained and can easily be included in and called from a User's program. Section 7.2 describes C code subroutine generation more fully. This feature has allowed LNKnet classifiers to be used on an extremely wide range of computers. The C-code generated only performs classification, it does not adaptively train the classifier or allow the classifier to be retrained.

1.6 What To Read Next

At the bare minimum, you should read the short "Getting Started with LNKnet" booklet [20] that should have been provided with this User's Guide. Also scan the "Common Questions and Problems" section in Appendix A. This will allow you to perform simple experiments and use the most basic LNKnet features.

If you want to use more advanced LNKnet features, read through this user's guide in the order the sections are presented. The Quick Start booklet illustrates how to perform a simple experiment. This user's guide contains a longer tutorial that walks you through a set of complex experiments. The user's guide also contains classifier and clustering algorithm descriptions, and descriptions of procedures available across classifiers including data base selection, data normalization, feature selection, a priori class probability adjustment and cross-validation. This is followed by a description of the many types of plots, the creation of movie-mode training plots, recommendations concerning including LNKnet plots in reports, a description of reviewing and printing log files and plots from LNKnet, a summary of code generation programs that generate C source code to implement trained LNKnet classifiers, a discussion concerning creating shell scripts to run LNKnet experiments in a batch mode, a review of advanced LNKnet features, and a description of input and output data formats and files. The Appendix contains a list of common problems and questions, instructions for installing LNKnet, listings of the shell scripts created during the tutorial, descriptions of installed data bases, and a short tutorial which describes how to use the mouse in Sun OpenWindows.

Detailed descriptions of LNKnet programs are available in man pages which are accessed using the UNIX man(1) command. The page LNKnet(1) lists all LNKnet programs and classifier(1) lists flags common to all classifier programs.

1.7 New LNKnet Features

LNKnet source code was converted to use the GNU auto configure tools. This combined with continued improvements in Linux and the Cygwin Linux-like environment made it relatively easy to port LNKnet to Red Hat Linux and to the Microsoft Windows OS using the Cygwin environment. LNKnet now runs on inexpensive Intel computers running Linux or Windows (under Cygwin) as well as on Sun Solaris workstations. Executables are provided for Red Hat Linux, for Windows with Cygwin, and Solaris. Others have run LNKnet on other versions of Linux and recompiled it for other versions of Linux and UNIX.

Support vector machine classifiers (SVMs) were added including linear SVMs, polynomial SVMs, and Gaussian kernel SVMs. To use SVMs on multiple-class problems (more than two classes), LNKnet includes an extension of SVMs to estimate posterior probabilities for binary classifiers. These binary classifiers are then used to estimate per-class posterior probabilities. For multiple-class problems, SVMs can be created for all pairwise combinations of classes or for each class versus the other classes. In addition to SVMs, a naive Bayes classifier was also added. The graphical user interface for the naive Bayes classifier is simplified and does not allow control of usually unimportant algorithm parameters. Additional parameters can be adjusted when running the naive Bayes classifier from a shell script.

The names of many of the LNKnet executables have been changed to avoid collision with tools provided as parts of other statistical and classifier software toolkits by adding the suffix “_lnk”. For example the k-nearest neighbor classifier executable is now called knn_lnk instead of knn. This has no effect on the GUI and only changes the names in automatically generated shell scripts.

Illustrations and descriptions in this guide for SVM and naive Bayes classifiers show windows and plots as they appear under Red Hat Linux. Other illustrations and descriptions were captured from Sun Solaris displays. Windows and plots appear slightly different under the other operating systems.

CHAPTER 2 A LNKnet tutorial

This tutorial introduces some of the general LNKnet classifiers and options. With LNKnet you will solve a speech classification problem using a Multi-Layer Perceptron and a K Nearest Neighbor algorithm. You will generate diagnostic plots. You will continue an experiment by restoring LNKnet windows using the experiment's screen file. You will use feature selection and normalization to reduce the number of input features in an experiment and you will use cross validation to experiment on a small data base. The tutorial assumes you are using the C-shell (csh) and running under the Solaris operating system.

2.1 UNIX Setup

Before you can run LNKnet, you must add the LNKnet *bin* directory to your $PATH environment variable and the LNKnet *man* path to your $MANPATH environment variable. First, find the LNKnet home directory, which is the directory in which LNKnet was installed. Assume this is `/home/rpl/lnknet`. If you are using the .cshrc shell under Solaris, then add the following three lines to the .cshrc file that can be found in your home directory.

```
setenv LNKHOME /home/rpl/lnknet
setenv PATH $LNKHOME/bin:${PATH}
setenv MANPATH $LNKHOME/man:${MANPATH}
```

The first line defines an environmental variable named $LNKHOME and sets it to the directory where LNKnet is installed. The second line uses this variable to add the LNKnet bin directory to the search path for executables and the third adds the LNKnet man directory to the search path for manual pages. If you are running under RedHat Linux and using the bash shell, then add the following three lines to the .bash-profile directory in your home directory to do the same things.

```
LNKHOME=/home/rpl/lnknet
PATH=$LNKHOME/bin:$PATH
MANPATH=$LNKHOME/man:$MANPATH
```

After making these changes, type "`source .cshrc`" under Solaris or "`source .bash-profile`" under Linux to run the shell where the modifications were made. For other shells, such as the Bourne shell, contact your system administrator for help.

2.2 Starting LNKnet

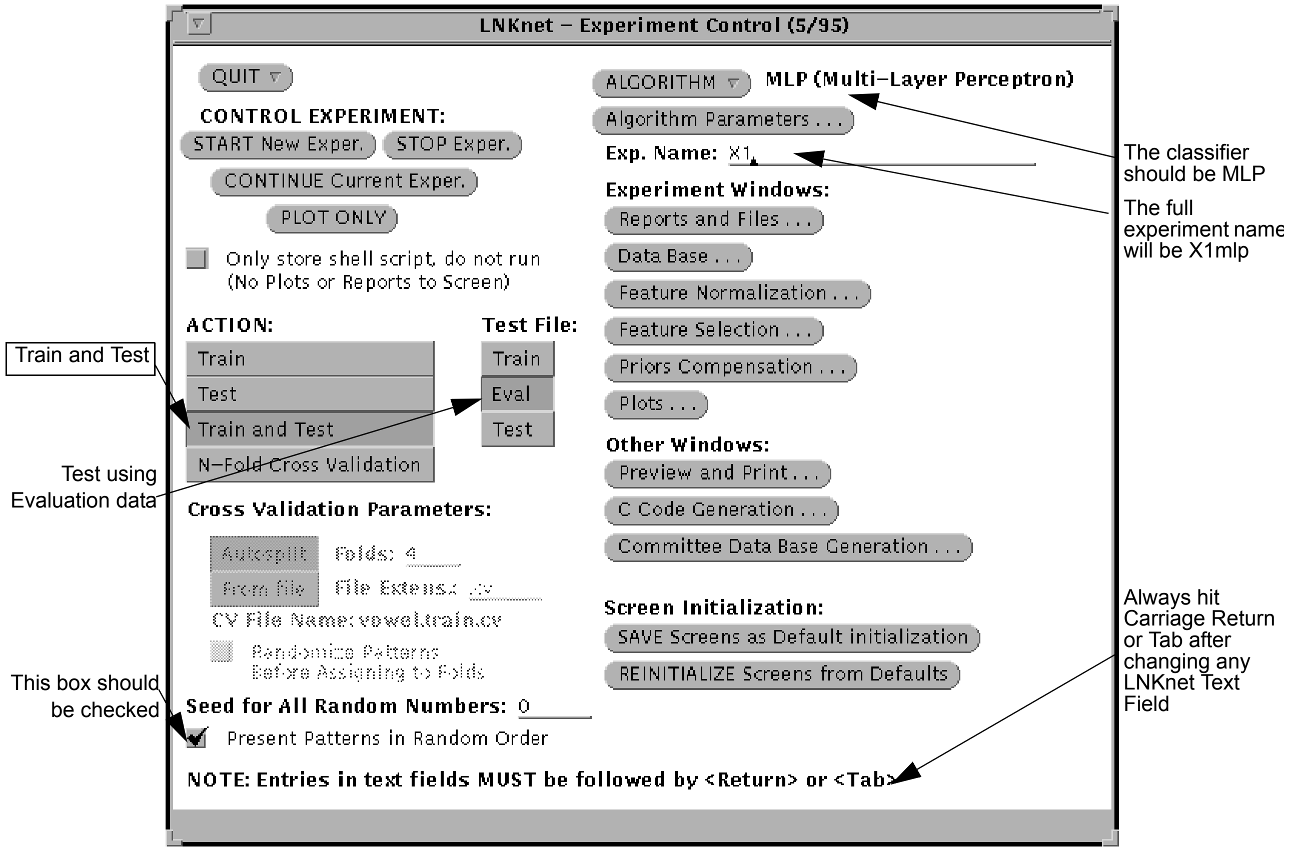
In your home directory, make an experiment directory named `Tutorial`. During the tutorial, you will generate some files in the data base directory. To insure that you have write permission for all these files, copy the data base files listed below from `$LNKHOME/data/class` into your new experiment directory. Finally, in a shell window go to your experiment directory and start the LNKnet graphical interface in the background. The following are the necessary commands for Solaris or Linux (If you are using a Bourne shell, replace ~ with $HOME):

```
> cd ~
> mkdir Tutorial
> cd $LNKHOME/data/class
> cp vowel.defaults ~/Tutorial
> cp vowel.train vowel.eval vowel.test ~/Tutorial
> cp vowel.norm.simple ~/Tutorial
> cp gnoise_var.defaults gnoise_var.train ~/Tutorial
> cp gnoise_var.eval gnoise_var.test ~/Tutorial
> cp gnoise_var.norm.simple ~/Tutorial
> cp iris.defaults iris.train ~/Tutorial
> cp iris.norm.simple ~/Tutorial
> cd ~/Tutorial
> LNKnet &
```

When LNKnet is started, the main LNKnet window should appear. If this is the first time you have used LNKnet, this window should look similar to the window in Figure 2.1. If you are unfamiliar with OpenWindows and a mouse see Appendix E. If you have used LNKnet before and have a .lnknetrc defaults file in your home directory, the parameter settings on LNKnet windows may be different than those shown in this tutorial. You can delete your defaults file and start the tutorial again, or you can change your windows to match the tutorial as you continue through it. If your .lnknetrc file is from a previous version of LNKnet, the LNKnet program may fail to start. In this case, remove the old .lnknetrc file and make a new one with the new version of LNKnet.

The left hand side of this main window shown in Figure 2.1 is a control panel used to run experiments. The right-hand side is used to select classifiers, plots, data bases, input features, and control other experimental conditions. Typically, an experiment is set up by first selecting a classifier using the top most **ALGORITHM** button and then selecting each of the buttons on the right hand side listed under **Experiment Windows** in order down to the **Plots...** button. Each button brings up a window with information to fill in or to be left in default settings. A button is highlighted if it is required and must be selected to run an experiment. Notes surrounding screen shots shown in Figure 2.1 and other figures in this tutorial show important controls that need to be set correctly to run this tutorial. Most controls are set correctly by default. When a value or selection needs

FIGURE 2.1 Main LNKnet Window


LNKnet – Experiment Control (5/95)
QUIT
CONTROL EXPERIMENT:
START New Exper.
STOP Exper.
CONTINUE Current Exper.
PLOT ONLY
ACTION:
Train
Test
Train and Test
N-Fold Cross Validation
Test File:
Train
Eval
Test
Cross Validation Parameters:
Autosplit
From file
ALGORITHM
Algorithm Parameters . . .
Experiment Windows:
Reports and Files . . .
Data Base . . .
Feature Normalization . . .
Feature Selection . . .
Priors Compensation . . .
Plots . . .
Other Windows:
Preview and Print . . .
C Code Generation . . .
Committee Data Base Generation . . .
Screen Initialization:
SAVE Screens as Default initialization
REINITIALIZE Screens from Defaults
Train and Test
Test using Evaluation data
This box should be checked
The classifier should be MLP
The full experiment name will be X1mlp
Always hit Carriage Return or Tab after changing any LNKnet Text Field

ALGORITHM MLP (Multi-Layer Perceptron)

Exp. Name: X1

Only store shell script, do not run (No Plots or Reports to Screen)

Folds: 4

File Extens.: .cv

CV File Name: vowel.train.cv

Randomize Patterns Before Assigning to Folds

Seed for All Random Numbers: 0

✓ Present Patterns in Random Order

NOTE: Entries in text fields MUST be followed by <Return> or <Tab>

to be changed from the default value, the note pointing to the control is surrounded by a box. For example, the Train and Test button on the left side of Figure 2.1 is not normally depressed by default and it must be depressed to run this tutorial.

2.3 Selecting a Classification Algorithm

The classification algorithm for the first experiment is the multi-layer perceptron. This should already be selected as the algorithm at the top of the main window. If it is not, use the left-most mouse button to display the algorithm menu and the classifier sub-menu and select MLP from it.

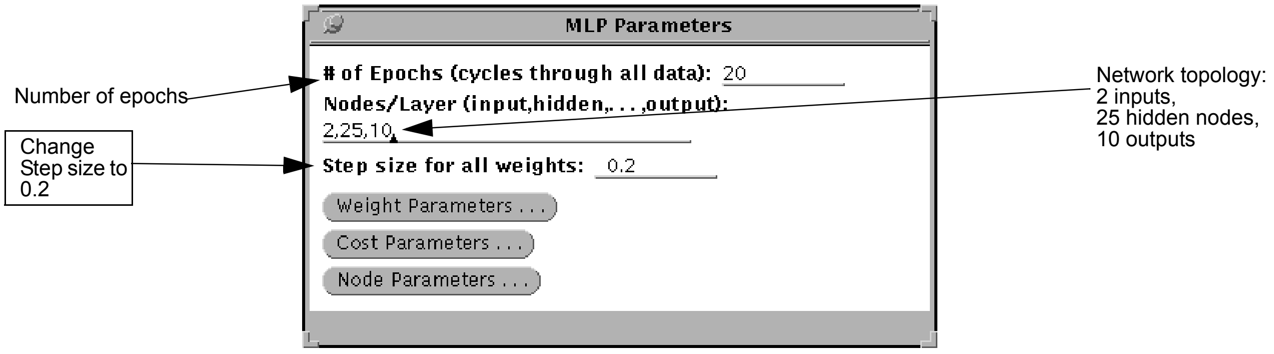
The button below the algorithm menu is the **Algorithm Params...** button. Select this button to bring up the multi-layer perceptron parameter window shown in Figure 2.2.

A Multi-Layer Perceptron trains the weights that connect each node in one layer to each node in the next layer. The network is made of an input layer and an output layer. Between them are 0, 1, or more hidden layers. For the first hidden layer, the weights can be thought of as describing hyperplanes through the input space. Sigmoid functions in the first layer of hidden nodes are used to determine whether a pattern is on one side of the plane or another, dividing the input space in half. These half spaces are combined and smoothed in upper layers to assign classes to regions of the input space.

To train network weights, training data is presented to the classifier several times. On the parameter popup shown in Figure 2.2, set the **number of epochs** to **20**. An epoch is a full pass through the training data, so each pattern will be presented 20 times over the course of training. Specify the network to have 2 inputs, 25 hidden nodes, and 10 outputs by entering **2,25,10** on the **Nodes/Layer** field on the second line in the window. Do not add spaces before or after the commas in the "2,25,10" or other comma delimited lists. The step size, which is the rate at which the weights are changed, must also be set. Change the **step size** to **0.2**, remembering to hit carriage return when you have done so. Changes to LNKnet text fields do not take effect unless carriage return is hit afterwards.

Other MLP parameters are set on three additional MLP parameter windows which can be displayed using three buttons on the main MLP window. There are explanations of the parameters on these windows in Section 3.1.1 in this User's Guide and on the **mlp(1)** manual page.

FIGURE 2.2 Main MLP Parameter window, set for first experiment


MLP Parameters

# of Epochs (cycles through all data): 20

Nodes/Layer (input,hidden,. . . ,output): 2,25,10

Step size for all weights: 0.2

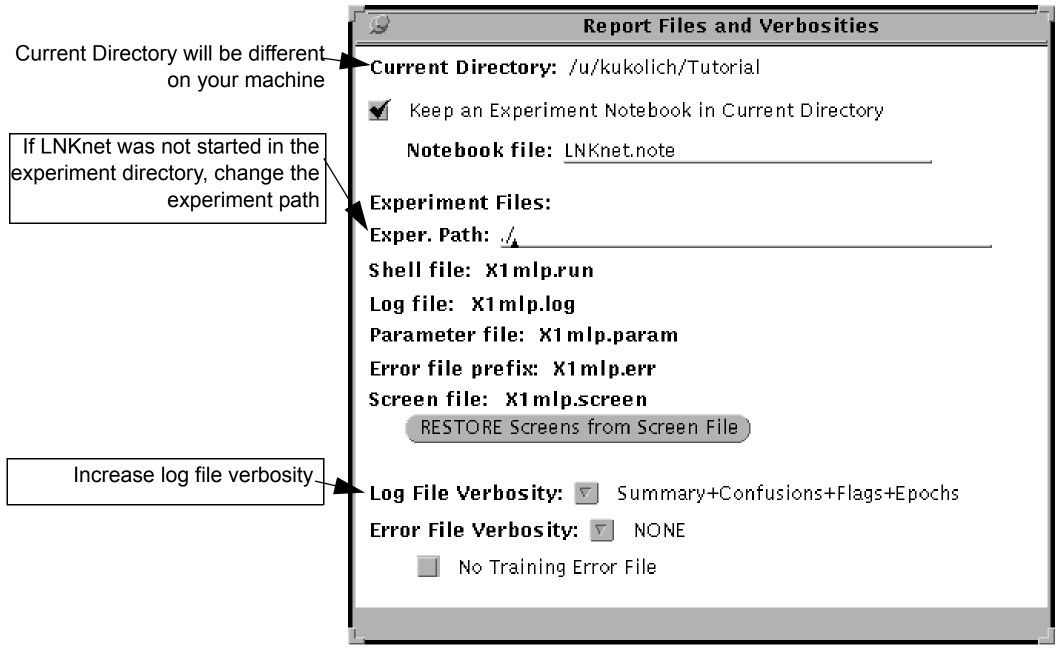
2.4 Experiment Setup

In a normal experiment, you train a classifier and evaluate it using an evaluation data set. The parts of a LNKnet experiment are set on the left side of the main window. You need to select **Train and Test** as the Action and **Eval** as the Test File, as shown in Figure 2.1.

2.4.1 File Parameters

Classification programs need files for storing experiment results. These files are listed on the **Report Files and Verbosities** popup window shown in Figure 2.3. To display this window, select the **Reports and Files...** button on the upper right of the main screen. For this experiment increase the **Error File Verbosity** to **Summary+Confusions+Flags+Epochs**. If it is necessary to change the experiment path, be sure to hit carriage return after making the change. Some of the notes which label Figure 2.3 and other figures have boxes drawn around them. As noted above, boxed notes show which features on a window are the most important or must be changed to run the tutorial.

FIGURE 2.3 Experiment Storage and Output Verbosity Window


Report Files and Verbosities
Current Directory will be different on your machine
If LNKnet was not started in the experiment directory, change the experiment path
Increase log file verbosity
RESTORE Screens from Screen File

Current Directory: /u/kukolich/Tutorial

✓ Keep an Experiment Notebook in Current Directory

Notebook file: LNKnet.note

Experiment Files:

Exper. Path: ./

Shell file: X1mlp.run

Log file: X1mlp.log

Parameter file: X1mlp.param

Error file prefix: X1mlp.err

Screen file: X1mlp.screen

Log File Verbosity: Summary+Confusions+Flags+Epochs

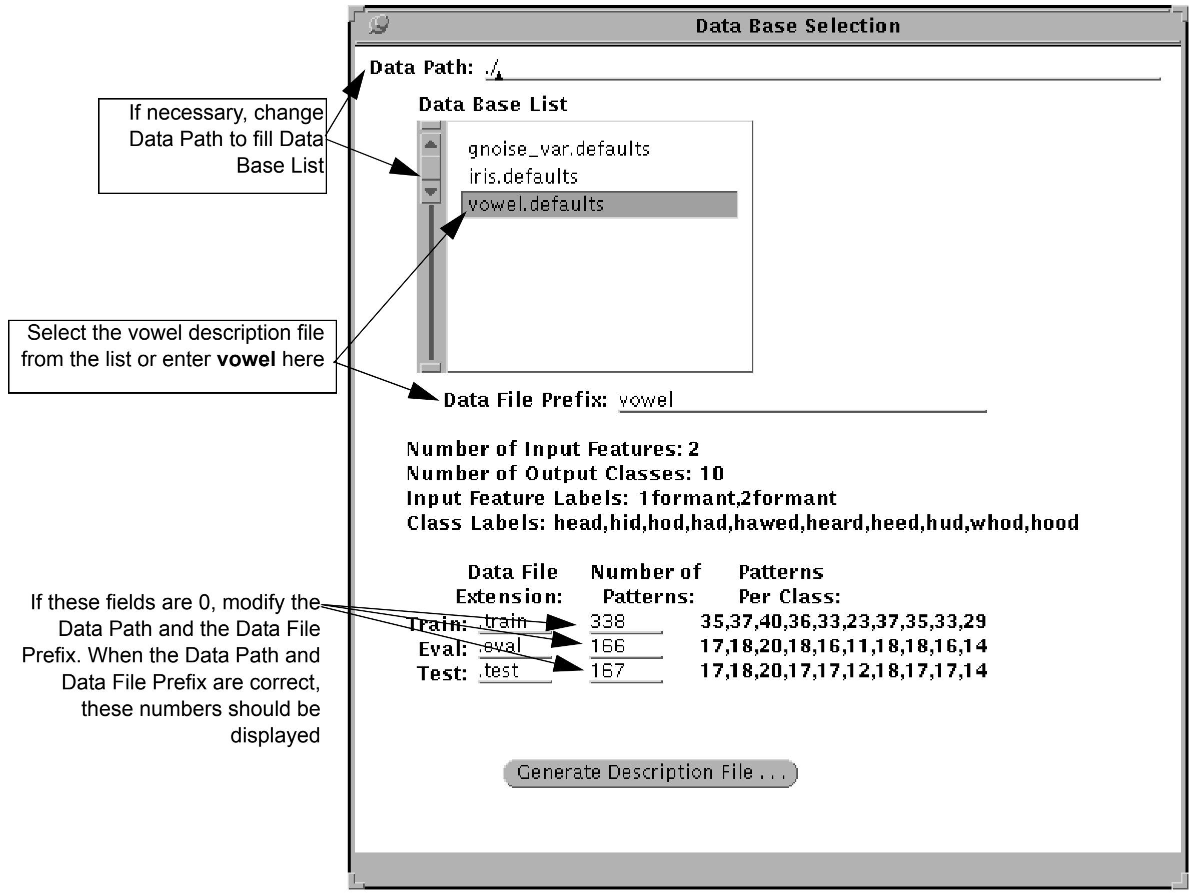
Error File Verbosity: NONE

No Training Error File

2.4.2 Data Base Selection

A data base of training and testing data is also required. Display the **Data Base Selection** window by selecting the **Data Base...** button on the main window. Figure 2.4 shows the data base selection window. The three data bases which you copied into the experiment directory should be listed in the **Data Base List** scroll box. In general, data for an experiment can be read from any directory by changing the data path and then hitting carriage return. Select vowel.defaults from this list or type "**vowel**" as the **Data File Prefix**. The other fields on the window may be left alone. The data base for this experiment has two input features and 10 classes. The classes are the 10 English vowels found in the words shown on the **Class Labels** line in the middle of the data base window. The data is a normalized version of the Peterson and Barney [32] vowel data collected in the late 50's from 67 men, women, and children. Each talker said the ten words, spectrograms were made from the waveforms, and resonant or formant frequencies for the vowels were selected. The features of the vowel data base come from the first two formant frequencies. The LNKnet data base pbvowel has the original data. Do not continue unless the bottom of the data base window appears as it does in Figure 2.4.

FIGURE 2.4 Data Base Selection Window


Data Base Selection
Data Path: ./
Data Base List
gnoise_var.defaults
iris.defaults
vowel.defaults
Data File Prefix: vowel
Generate Description File ...
If necessary, change Data Path to fill Data Base List
Select the vowel description file from the list or enter **vowel** here
If these fields are 0, modify the Data Path and the Data File Prefix. When the Data Path and Data File Prefix are correct, these numbers should be displayed

Number of Input Features: 2
Number of Output Classes: 10
Input Feature Labels: 1formant,2formant
Class Labels: head,hid,hod,had,hawed,heard,heed,hud,whod,hood

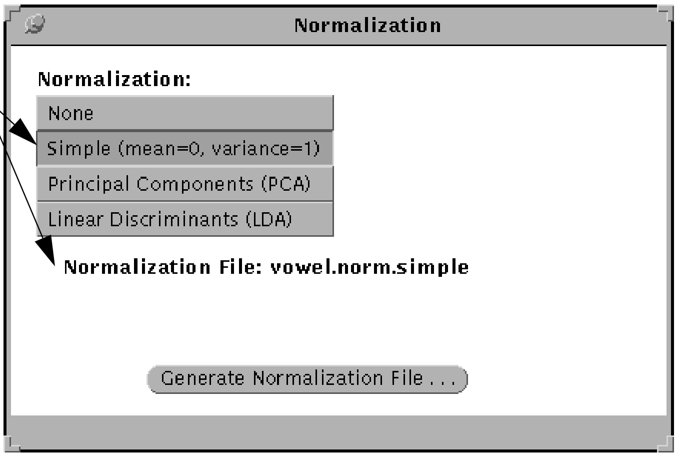
	Data File Extension:	Number of Patterns:	Patterns Per Class:
Train:	.train	338	35,37,40,36,33,23,37,35,33,29
Eval:	.eval	166	17,18,20,18,16,11,18,18,16,14
Test:	.test	167	17,18,20,17,17,12,18,17,17,14

2.4.3 Normalization

For many classifiers, classification results are improved when the data has been normalized in some way. Although this vowel data has already been normalized to range from zero to one, better results are achieved when the data is given zero mean, unit variance using simple normalization. Display the normalization window by selecting **Feature Normalization...** on the main window. The normalization window in Figure 2.5 will appear. Check that **Simple Normalization** is selected. If LNKnet cannot find the normalization file it will report an error at the bottom of the normalization window and show a small stop sign on the main window. Normalization files are stored in the data directory, so check the data path on the data base window. If the file really does not exist, Section 2.14 in this tutorial describes how a normalization file can be created using LNKnet.

FIGURE 2.5 Normalization Window

Selecting Simple normalization will set the Normalization File name. If there is an error, check the path on the data base selection window



Normalization
Normalization:
None
Simple (mean=0, variance=1)
Principal Components (PCA)
Linear Discriminants (LDA)
Generate Normalization File . . .

Normalization File: vowel.norm.simple

2.5 Plot Setup

There are several types of plots available for analyzing LNKnet experiments. To request these plots, you must bring up the **Plotting Controls** parameter window. In the column of buttons on the right side of the main LNKnet window is a button labeled **Plots...**. Select this button to bring up the **Plotting Controls** window. On this window, select the check boxes for the plots under **Decision Region Plots**, **Profile Plots**, **Structure Plots, Training Error File Graphs**, and **Testing Error File Graphs** as shown in Figure 2.6. If the classifier on the main window is not MLP, some of these plots will not be available.

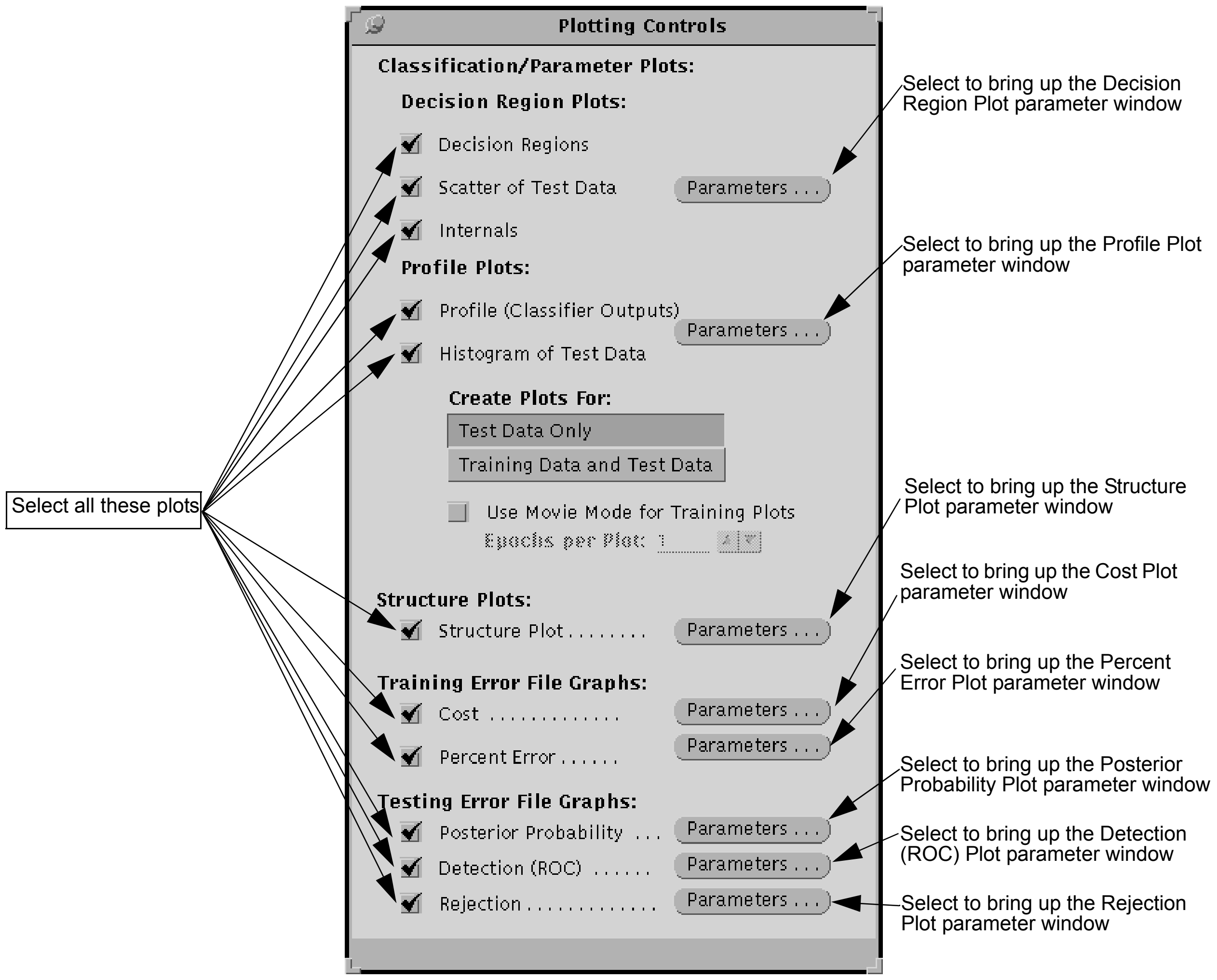
2.5.1 Setting Plot Parameters

Each plot has some parameters which should be set. Selecting each of the **Parameters...** buttons will bring up the windows for the available plots.

2.5.1.1 Decision Region Plots

Three two dimensional plots are controlled from the Decision Region Plot parameter window. They are the decision region plot, the scatter plot, and the internals plot. Push the top most **Parameters...** button to bring up the Decision Region Plot window shown in Figure 2.7. For this experiment, you should change **Number of Intervals per Dimension** from 50 to **100** on the Decision Region Plot window. This will cause the plotting program to use a finer grid for generating the decision region plot. It will take longer to generate, but the plot will look better. Figure 2.7 shows the Decision Region Plot window ready for the experiment.

FIGURE 2.6 Plotting Controls window, set for first experiment


Select all these plots
Select to bring up the Decision Region Plot parameter window
Select to bring up the Profile Plot parameter window
Select to bring up the Structure Plot parameter window
Select to bring up the Cost Plot parameter window
Select to bring up the Percent Error Plot parameter window
Select to bring up the Posterior Probability Plot parameter window
Select to bring up the Detection (ROC) Plot parameter window
Select to bring up the Rejection Plot parameter window
Parameters . . .

Plotting Controls

Classification/Parameter Plots:

Decision Region Plots:
- ✓ Decision Regions
- ✓ Scatter of Test Data
- ✓ Internals

Profile Plots:
- ✓ Profile (Classifier Outputs)
- ✓ Histogram of Test Data

Create Plots For: [✓ Test Data Only] Training Data and Test Data

- ☐ Use Movie Mode for Training Plots

Epochs per Plot: 1

Structure Plots:
- ✓ Structure Plot . . . .

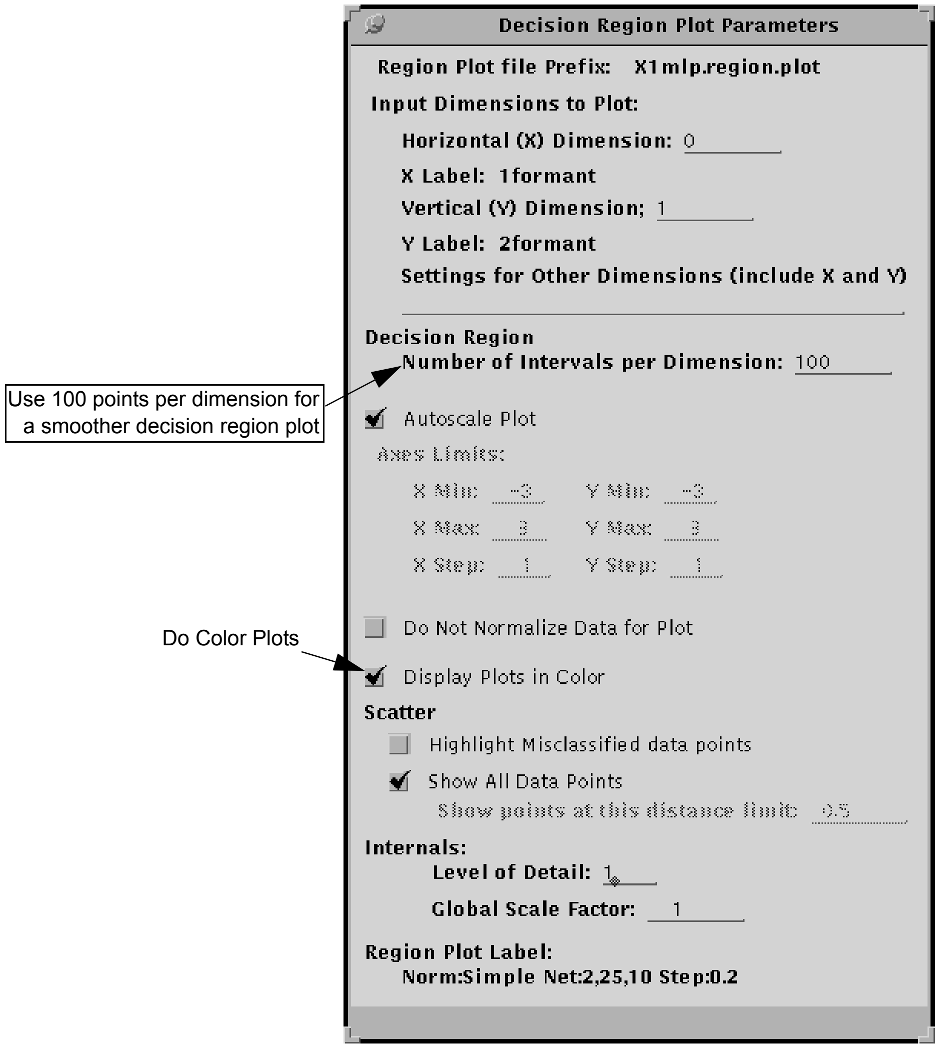
Training Error File Graphs:
- ✓ Cost . . . .
- ✓ Percent Error . . . . . .

Testing Error File Graphs:
- ✓ Posterior Probability . . .
- ✓ Detection (ROC) . . . . . .
- ✓ Rejection . . . .

2.5.1.2 Profile Plots

Two one-dimensional plots are controlled from the profile plot parameter window. They are the profile plot and the histogram plot. Push the second **Parameters...** button to bring up the Profile Plot Parameters window shown in Figure 2.8. The one dimensional plots are available for classifiers with continuous outputs including the MLP classifier. No profile plot parameters need to be changed from their default settings.

FIGURE 2.7 Decision Region Plot Parameters


Use 100 points per dimension for a smoother decision region plot
Do Color Plots

Decision Region Plot Parameters

Region Plot file Prefix: X1mlp.region.plot

Input Dimensions to Plot:

Horizontal (X) Dimension: 0

X Label: 1formant

Vertical (Y) Dimension; 1

Y Label: 2formant

Settings for Other Dimensions (include X and Y)

Decision Region

Number of Intervals per Dimension: 100

✓ Autoscale Plot

Axes Limits:

X Min: -3 Y Min: -3

X Max: 3 Y Max: 3

X Step: 1 Y Step: 1

☐ Do Not Normalize Data for Plot

✓ Display Plots in Color

Scatter

☐ Highlight Misclassified data points

✓ Show All Data Points

Show points at this distance limit: 0.5

Internals:

Level of Detail: 1

Global Scale Factor: 1

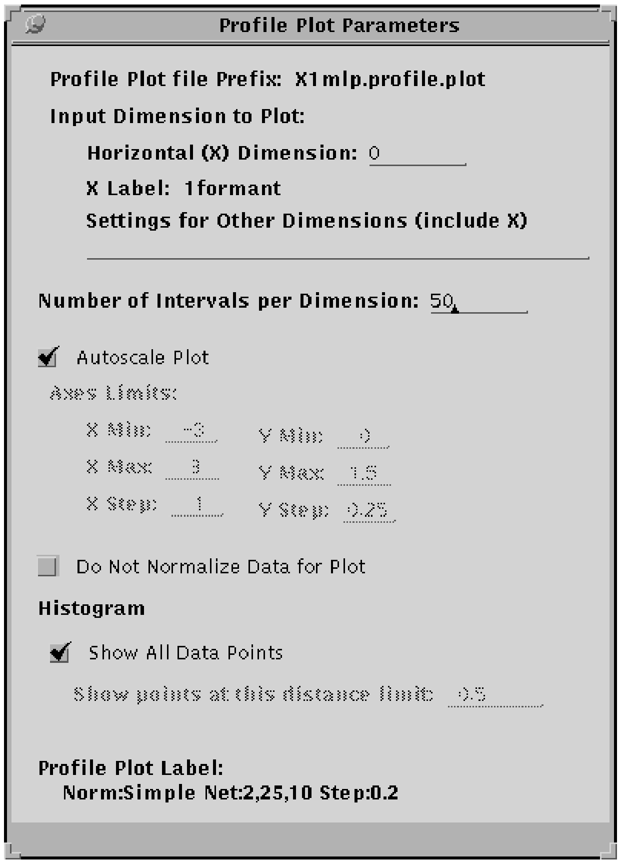
Region Plot Label:

Norm:Simple Net:2,25,10 Step:0.2

2.5.1.3 Structure Plots

It can be informative to see the node structure of a trained classifier. Each of those LNK-net classifiers for which it is appropriate has a structure plot. Depending on the classification algorithm, the structure plots show the input and output nodes of the classifier and the connections between them. If these connections have weights, the weights can be displayed. Explanations of the structure plots can be found on the manual pages for each one and in Section 6.3 of this User's Guide. Select the third **Parameters...** button to bring up the structure plot window. Select **Autoscale Plot To Fit on Screen**, **Show Weight Magnitudes**, and **Display Bias Nodes,** as shown in Figure 2.9.

FIGURE 2.8 Profile Plot Parameters



Profile Plot Parameters

Profile Plot file Prefix: X1mlp.profile.plot

Input Dimension to Plot:

Horizontal (X) Dimension: 0

X Label: 1formant

Settings for Other Dimensions (include X)

Number of Intervals per Dimension: 50

✓ Autoscale Plot

Axes Limits:

X Min: -3 Y Min: 0

X Max: 3 Y Max: 1.5

X Step: 1 Y Step: 0.25

☐ Do Not Normalize Data for Plot

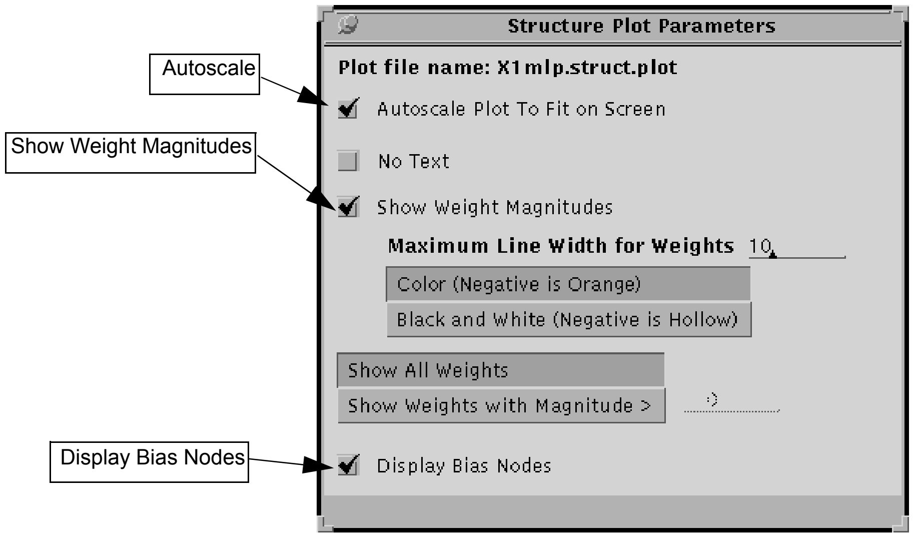
Histogram

✓ Show All Data Points

Show points at this distance limit: 0.5

Profile Plot Label:
Norm:Simple Net:2,25,10 Step:0.2

FIGURE 2.9 Structure Plot Parameters



Structure Plot Parameters

Plot file name: X1mlp.struct.plot

✓ Autoscale Plot To Fit on Screen

☐ No Text

✓ Show Weight Magnitudes

Maximum Line Width for Weights 10

Color (Negative is Orange)

Black and White (Negative is Hollow)

Show All Weights

Show Weights with Magnitude > 0

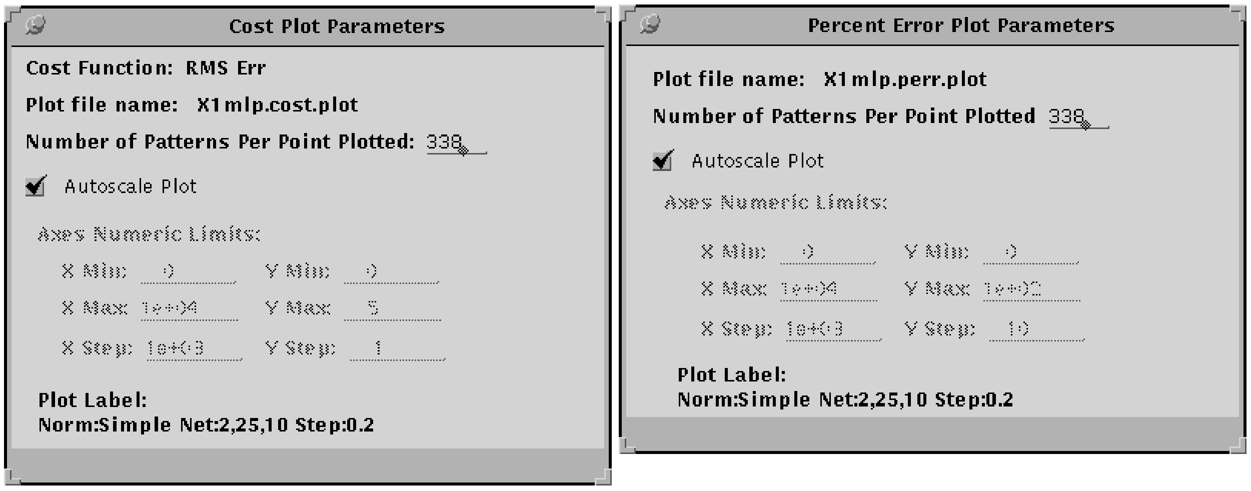
✓ Display Bias Nodes

2.5.1.4 Training Error File Plots

While training a classifier by cycling through the data, a classification error file is created. The accuracy of the classifier during training can be plotted in two ways, using a cost plot or a percent error plot. The cost is the function being minimized by the classifier. These plots are not available for algorithms that train in a single pass through the data.

If this is the first time you have run LNKnet, the **Cost Plot** and **Percent Error Plot** parameter windows should be ready for the experiment. They should appear as in Figure 2.10.

FIGURE 2.10 Cost Plot and Percent Error Plot Parameters



Cost Plot Parameters

Cost Function: RMS Err

Plot file name: X1mlp.cost.plot

Number of Patterns Per Point Plotted: 338

✓ Autoscale Plot

Axes Numeric Limits:

X Min: 0 Y Min: 0

X Max: 1e+04 Y Max: 5

X Step: 1e+03 Y Step: 1

Plot Label:
Norm:Simple Net:2,25,10 Step:0.2

Percent Error Plot Parameters

Plot file name: X1mlp.perr.plot

Number of Patterns Per Point Plotted 338

✓ Autoscale Plot

Axes Numeric Limits:

X Min: 0 Y Min: 0

X Max: 1e+04 Y Max: 1e+02

X Step: 1e+03 Y Step: 10

Plot Label:
Norm:Simple Net:2,25,10 Step:0.2

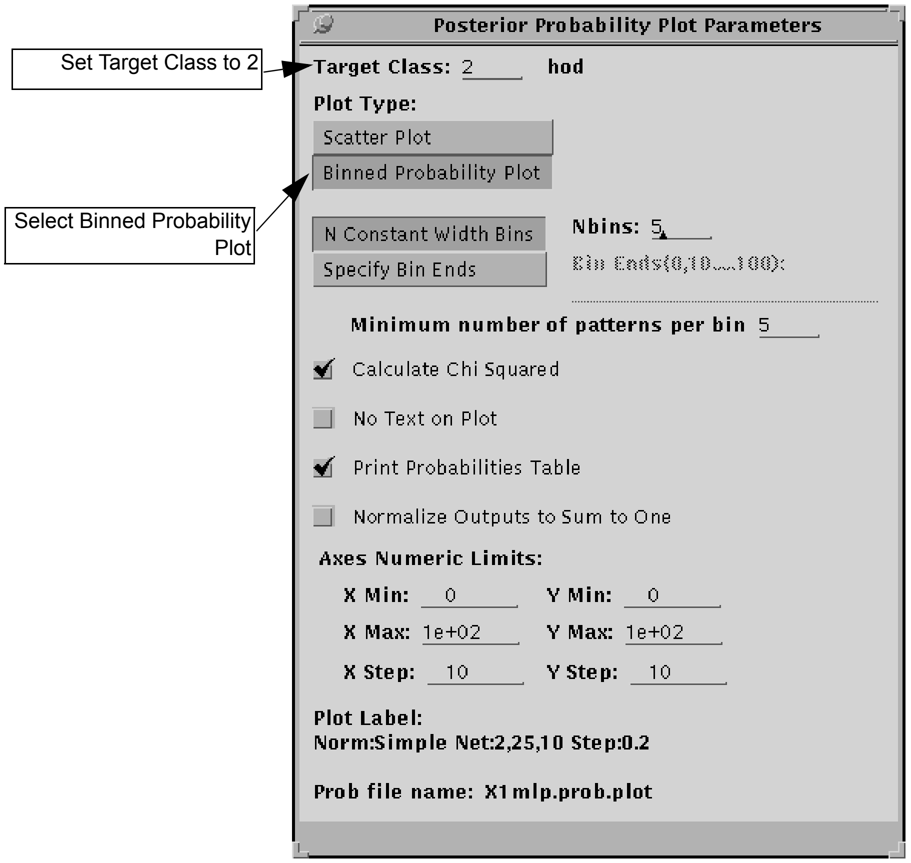
2.5.1.5 Testing Error File Plots

During the testing portion of the experiment, a testing error file is produced. If the classification algorithm produces continuous outputs, as the MLP algorithm does, this test error file can be used to produce several plots. Some plot parameters need to be set for this experiment. On the Posterior Probability plot window, shown in Figure 2.11, set the **target class** to **2 (hod)** and select **Binned Probability Plot**. On the ROC plot window shown in Figure 2.12, set the **target class** to **2 (hod).** On the rejection plot window, shown in Figure 2.13, set the **table step** to **10**.

2.6 Saving Defaults

It is inconvenient to have to set these general parameters each time LNKnet is started. To save the settings, select **Save Screens as Default initialization** on the lower left of main LNKnet window. A file, `.lnknetrc`, will be created in your home directory. The next time you start LNKnet, this file will be read and the settings on all of the LNKnet screens will be as they are now.

FIGURE 2.11 Posterior Probability Plot



Target Class: 2 hod

Plot Type: Binned Probability Plot

Nbins: 5

Bin Ends(0,10,...,100):

Minimum number of patterns per bin 5

✓ Calculate Chi Squared

No Text on Plot

✓ Print Probabilities Table

Normalize Outputs to Sum to One

Axes Numeric Limits:

X Min: 0 Y Min: 0

X Max: 1e+02 Y Max: 1e+02

X Step: 10 Y Step: 10

Plot Label:
Norm:Simple Net:2,25,10 Step:0.2

Prob file name: X1mlp.prob.plot

FIGURE 2.13 Rejection Plot Parameters

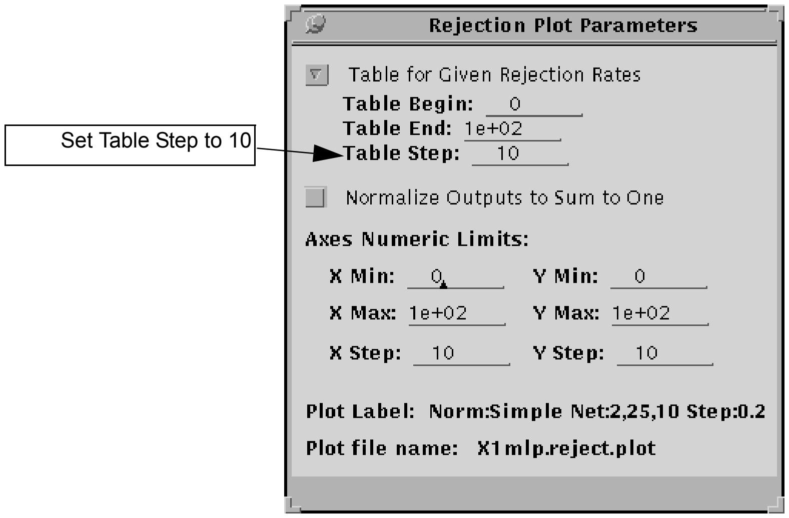


Table for Given Rejection Rates

Table Begin: 0

Table End: 1e+02

Table Step: 10

Normalize Outputs to Sum to One

Axes Numeric Limits:

X Min: 0 Y Min: 0

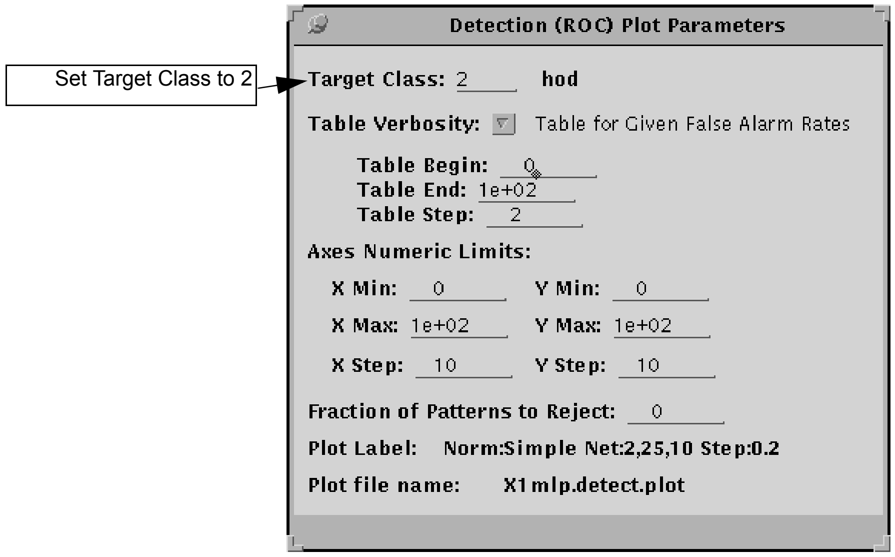
X Max: 1e+02 Y Max: 1e+02

X Step: 10 Y Step: 10

Plot Label: Norm:Simple Net:2,25,10 Step:0.2

Plot file name: X1mlp.reject.plot

FIGURE 2.12 ROC (Detection) Plot Parameters



Target Class: 2 hod

Table Verbosity: Table for Given False Alarm Rates

Table Begin: 0
Table End: 1e+02
Table Step: 2

Axes Numeric Limits:

X Min: 0 Y Min: 0
X Max: 1e+02 Y Max: 1e+02
X Step: 10 Y Step: 10

Fraction of Patterns to Reject: 0

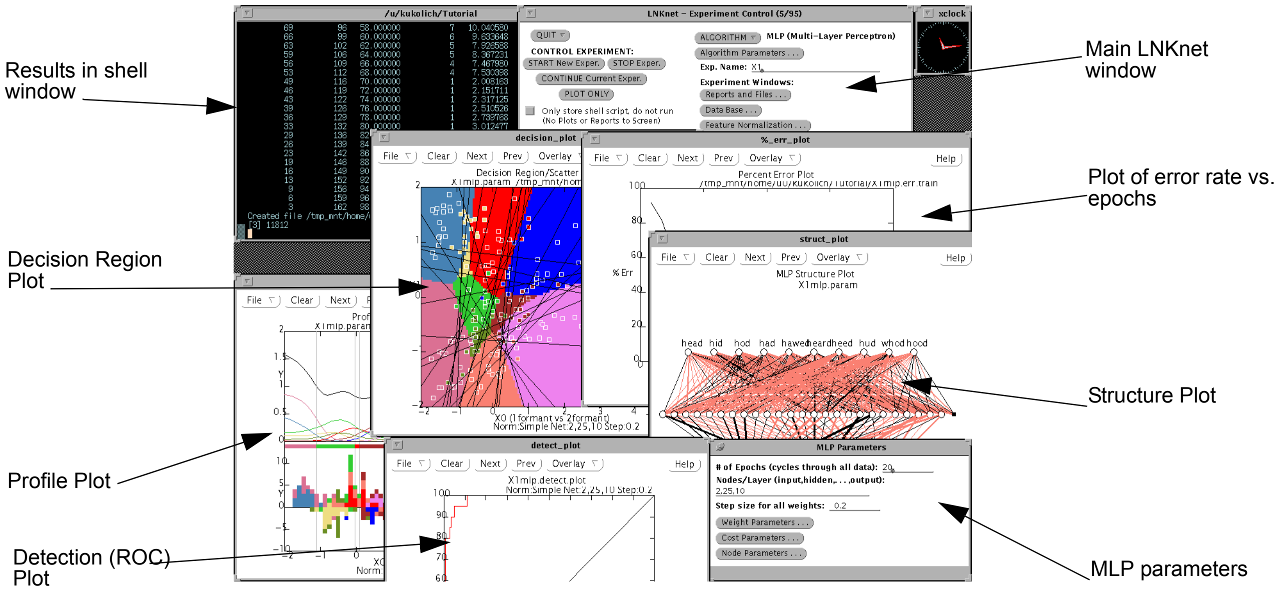
Plot Label: Norm:Simple Net:2,25,10 Step:0.2

Plot file name: X1mlp.detect.plot

2.7 Starting an Experiment

You can now start this experiment by selecting **START New Exper.** on the main window. LNKnet writes an entry in the notebook file describing this experiment and writes a shell script in the experiment directory `~/Tutorial`. That shell script is run and the results of the experiment are printed to your shell window and to a log file. A one line experiment results entry is added to the experiment notebook file by the shell script. The notebook, shell script, and the log file are included in Appendix C. The shell script first makes a call to the mlp program which trains the classifier. After each training epoch, that program prints the current classification error rate and the current average value of the function being minimized by the classifier. After the 20 epochs are over, a summary of the training errors is printed. The shell script then calls the mlp program to test the classifier using the evaluation data. The results of that test are below. Finally, the shell script displays the requested plots. Each plot is displayed in its own plotting window. Figure 2.14 shows the screen of a workstation after running this experiment.

FIGURE 2.14 Workstation Screen During MLP Classification Experiment



```
   69      96   58.000000      7   10.040580
   66      99   60.000000      6    9.633648
   63     102   62.000000      5    7.926588
   59     106   64.000000      5    8.367231
   56     109   66.000000      4    7.467980
   53     112   68.000000      4    7.530398
   49     116   70.000000      1    2.008163
   46     119   72.000000      1    2.151711
   43     122   74.000000      1    2.317125
   39     126   76.000000      1    2.510526
   36     129   78.000000      1    2.739768
   33     132   80.000000      1    3.012477
   29     136   82
   26     139   84
   23     142   86
   19     146   88
   16     149   90
   13     152   92
    9     156   94
    6     159   96
    3     162   98
 Created file /tmp_mnt/home/
 [3] 11812
```

MLP (Multi–Layer Perceptron)

Exp. Name: X1

Only store shell script, do not run (No Plots or Reports to Screen)

# of Epochs (cycles through all data): 20

Nodes/Layer (input,hidden,. . . ,output): 2,25,10

Step size for all weights: 0.2

2.8 MLP Results

Table 2.1 shows the files created during the experiment by LNKnet, the MLP program and the plot programs.

TABLE 2.1: Files Created During MLP Experiment

Files created by LNKnet	
`LNKnet.note`	Notebook file with results for all experiments
`X1mlp.run`	Shell script
`X1mlp.screen`	Settings for all LNKnet windows
`LNKnet.note.screen`	Backup of screen file for comparisons
Files Created by MLP	
`X1mlp.param`	Parameters for trained MLP classifier
`X1mlp.log`	Copy of results printed to screen
`X1mlp.err.train`	Trial-by-trial results during training

TABLE 2.1: Files Created During MLP Experiment

`X1mlp.err.eval`	Results during testing
Files Created by Plot Programs	
`X1mlp.region.plot.eval`	2-Dimensional plots
`X1mlp.profile.plot.eval`	1-Dimensional plots
`X1mlp.struct.plot`	Structure plot
`X1mlp.cost.plot`	Cost plot
`X1mlp.perr.plot`	Percent Error plot
`X1mlp.prob.plot`	Posterior Probability plot
`X1mlp.detect.plot`	Detection (ROC) plot
`X1mlp.reject.plot`	Rejection plot

2.8.1 MLP Eval Results

These are the classification results on the evaluation data, shown in the log file and on the window used to start LNKnet. First there is a confusion matrix which shows the classification results for all patterns from each class. Numbers on the diagonal of this matrix represent the patterns classified correctly. Other numbers represent the number and distribution of errors. Below the confusion matrix is an error summary giving the number of errors for patterns in each class. Finally, there is an overall error rate, which is 32.53%, ±3.6% for this experiment.

```
Classification Confusion Matrix - X1mlp.err.eval
------------------------------------------------------------------------------
------------------------------------------------------------------------------

Desired                          Computed Class
 Class     0     1     2     3     4     5     6     7     8     9   Total
 -----   ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
   0      14                 3                                        17
   1       5     1                             12                     18
   2                  19     1                                        20
   3       5                11           1           1                18
   4                   2          14                                  16
   5       1                             9                 1          11
   6                                          18                      18
   7                   6     3     2                 7                18
   8                                                      15     1    16
   9                                     6           1     3     4    14
 -----   ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
 Total    25     1    27    18    16    16    30     9    19     5   166

------------------------------------------------------------------------------
------------------------------------------------------------------------------
```

```
Error Report - X1mlp.err.eval
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
Class     Patterns   # Errors     % Errors StdDev RMS Err  Label
  0          17          3        17.65  ( 9.2)      0.173  head
  1          18         17        94.44  ( 5.4)      0.304  hid
  2          20          1         5.00  ( 4.9)      0.127  hod
  3          18          7        38.89  (11.5)      0.197  had
  4          16          2        12.50  ( 8.3)      0.182  hawed
  5          11          2        18.18  (11.6)      0.249  heard
  6          18          0         0.00  ( 0.0)      0.051  heed
  7          18         11        61.11  (11.5)      0.268  hud
  8          16          1         6.25  ( 6.1)      0.121  whod
  9          14         10        71.43  (12.1)      0.289  hood
          --------    --------    ------- -------------

Overall      166         54       32.53  ( 3.6)     0.207
```

2.8.2 MLP Plots

These are the plots generated during the MLP experiment. They should have been displayed in new windows on your screen.

FIGURE 2.15 MLP Decision Region Plot after 20 Epochs

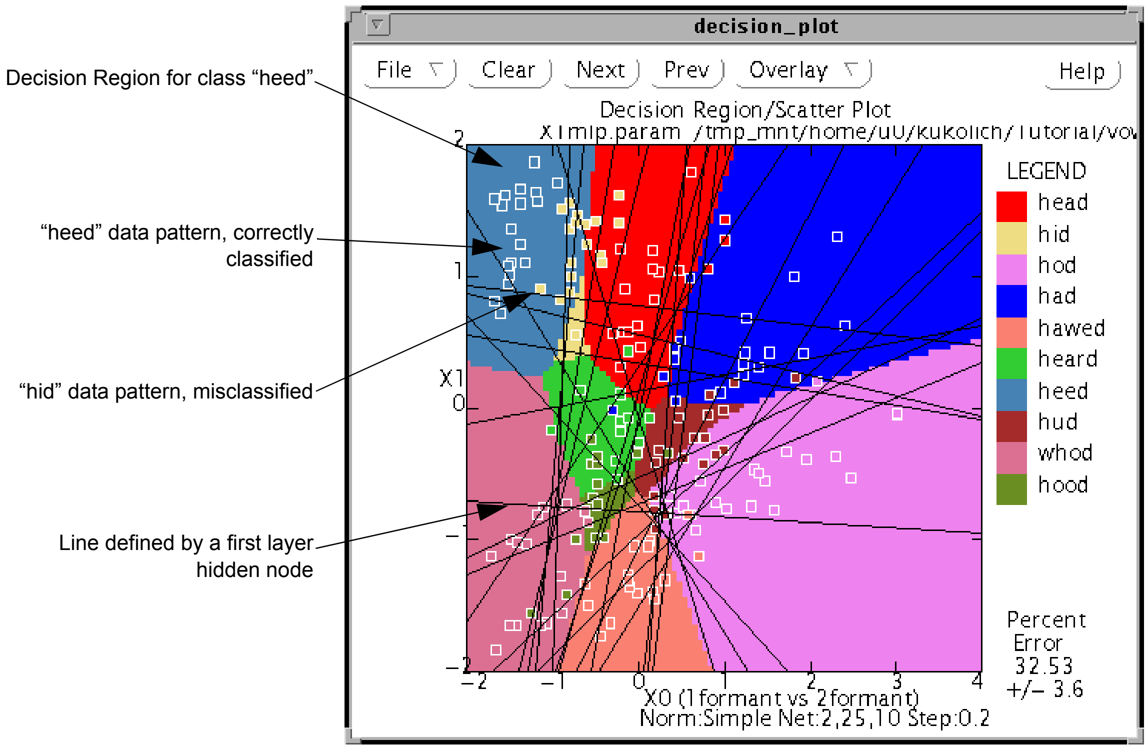

Decision Region for class “heed”
“heed” data pattern, correctly classified
“hid” data pattern, misclassified
Line defined by a first layer hidden node
decision_plot
File
Clear
Next
Prev
Overlay
Help
Decision Region/Scatter Plot
X1mlp.param /tmp_mnt/home/u0/kukolich/Tutorial/vow
LEGEND
head
hid
hod
had
hawed
heard
heed
hud
whod
hood
Percent Error 32.53 +/− 3.6
X1
X0 (1 formant vs 2 formant)
Norm:Simple Net:2,25,10 Step:0.2
2
1
0
−2
−2
−1
0
1
2
3
4

Figure 2.15 shows the set of three overlaid 2D plots. There is a decision region plot (the solid regions), a scatter plot of the evaluation data (the small white rimmed squares), and an internals plot (the black lines). The decision region plot shows the class that would be selected at each point on the plot. The values for the two selected input dimensions are as you see them. Any other input dimensions are held constant either to 0 or to values specified on the decision region plots window. The scatter plot shows the evaluation data, color coded to show the class. All patterns within a set distance of the decision region plot plane are shown. Classification errors can be found by looking for those patterns whose color does not match the background color from the decision region plot. The form of the internals plot depends on the type of algorithm being used for classification. In this case, the multi-layer perceptron, the lines represent hyperplanes defined by the nodes of the first hidden layer. The hidden nodes which generate particular borders between decision regions can often be identified using this plot.

FIGURE 2.16 MLP Profile Plot after 20 Epochs

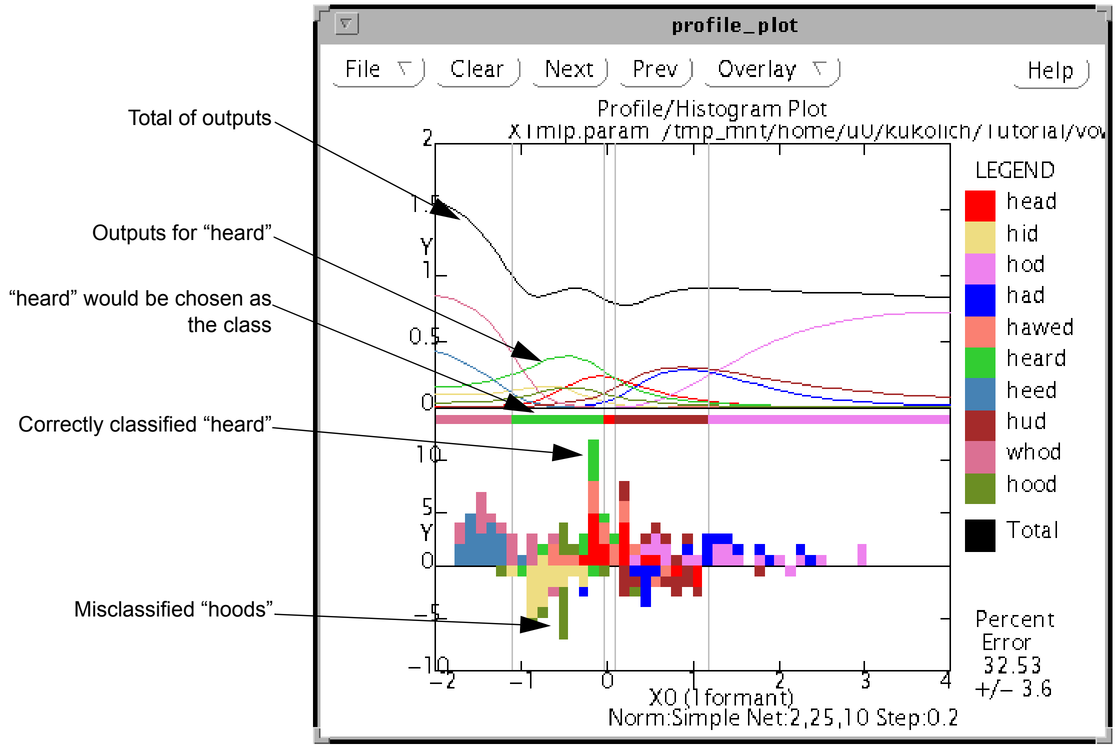


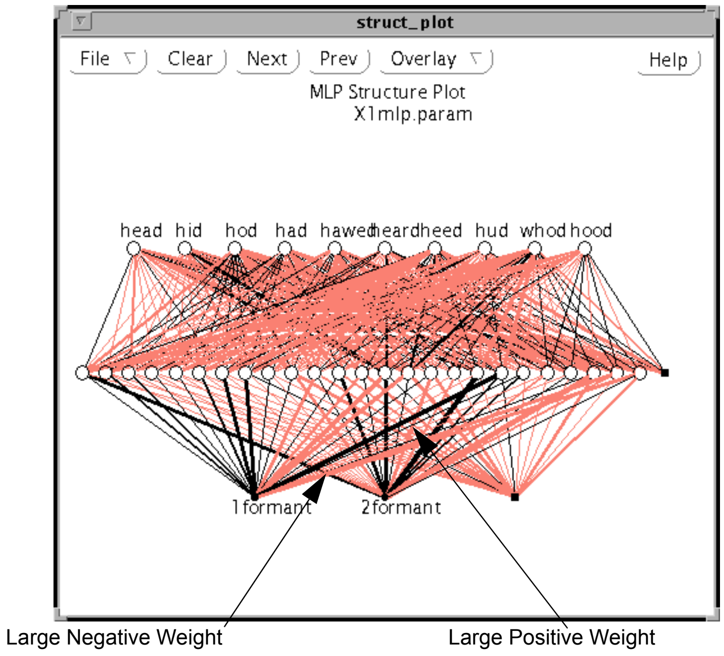
Figure 2.16 shows the two 1D plots. There is a profile plot (the black and colored lines and the solid bars below them), and a histogram of the evaluation data (the squares at the bottom).

For the profile plot, all of the input features but one are held constant while one feature is varied. The output levels for each class are plotted. These output level lines are the colored lines. The total of these output levels is plotted as a black line. This line should be close to 1.0 for a well trained classifier that estimates posterior class probabilities, like the MLP classifier. Below the output level lines is something like a one dimensional decision region plot. It shows the class which would be chosen for a pattern with the given generated inputs. Where the class changes, a dotted vertical line is drawn.

The histogram plot at the bottom of Figure 2.16 is in two parts. The points shown are either all the patterns in the evaluation data set, or those which are within some distance of the line being sampled for the profile plot. The squares above the line represent those patterns which are correctly classified by the current model. The squares below represent misclassified patterns. These squares are color coded by class, as in the scatter plot.

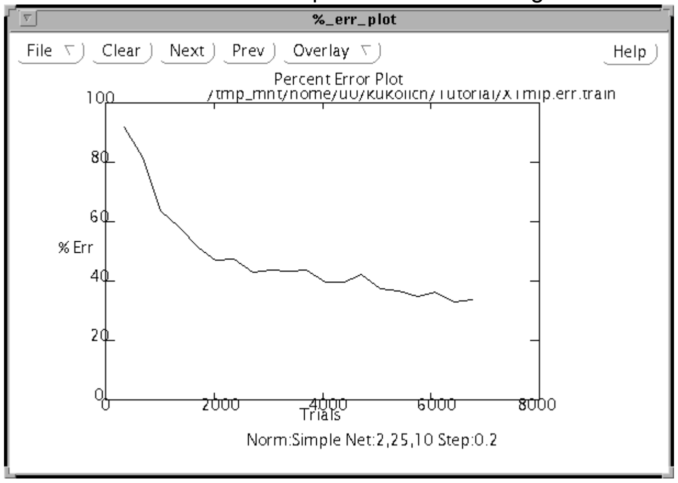
Figure 2.17 shows a structure plot for the trained multi-layer perceptron. At the bottom of the plot there are two small black circles representing input nodes and a small black square representing the input bias node. Below each input node is the input label for that node. From the bottom to the middle is a set of lines of varying thicknesses. These lines represent the weighted connections from the input layer to the hidden layer. The thickness of these and other lines is proportional to the magnitude of the connecting weight. Some of the lines are orange, indicating that connecting weights are negative. The large white circles represent the hidden nodes where weighted sums of the inputs are calculated and passed through a sigmoid function. The hidden layer also has a bias node. At the top of the plot are large white circles representing the output nodes. Another set of lines shows the weighted connections from the hidden layer to the output layer. Above the output nodes are the class labels for the output classes.

FIGURE 2.17 MLP Structure Plot after 20 epochs



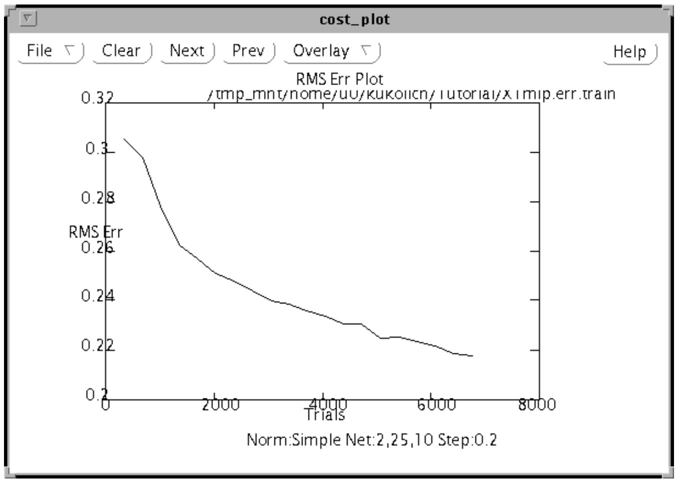
During training, each pattern is tested by the MLP classifier. The classification results from these tests are stored in the file `X1mlp.err.train`. The average percent error in classification during each epoch of training is shown in Figure 2.18.

FIGURE 2.18 Percent Error Plot after 20 epochs of MLP training



The cost of each pattern tested during training is also stored in the file `X1mlp.err.train`. The average of these values for each epoch is shown in Figure 2.19.

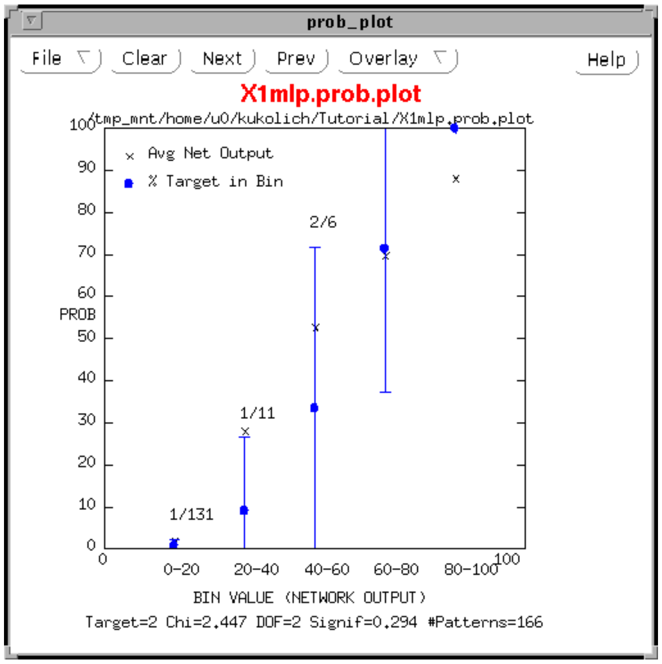
FIGURE 2.19 Cost Plot after 20 epochs of MLP training



The cost here is the square root of the mean squared error of the outputs normalized by the number of classes. A desired output of 1 for the correct class and 0 for the other classes is subtracted from the actual outputs for a pattern. These values are then squared, averaged over the number of classes, and stored as the cost. This plot, then, averages the costs for each epoch and takes the square root to get each point on the plot.

Figure 2.20 shows a posterior probability plot for class 2, hod. To generate the plot, each evaluation pattern is binned according to its output for class 2. Five bins were used to create this plot. They represent class 2 outputs of 0.0 to 0.2, 0.2 to 0.4, 0.4 to 0.6, 0.6 to 0.8, and 0.8 to 1.0. The average class 2 output values for the patterns in each bin are shown as X's. The actual percentage of class 2 patterns in a bin is drawn as a filled circle. Lines above and below the circle represent two standard deviations about the actual percentages. The total number of patterns and the number of class 2 patterns in each bin are displayed above the upper limit mark. For example the numbers "2/6" over the 40-60 bin mean two patterns in this bin were from class 2 and there were six patterns in this bin. If all the X's are within the ±2 standard deviation limits, the classifier provides accurate posterior probability estimates. A table of the values in the plot is printed in the log file and a Chi Squared test and significance values are printed in the experiment notebook.

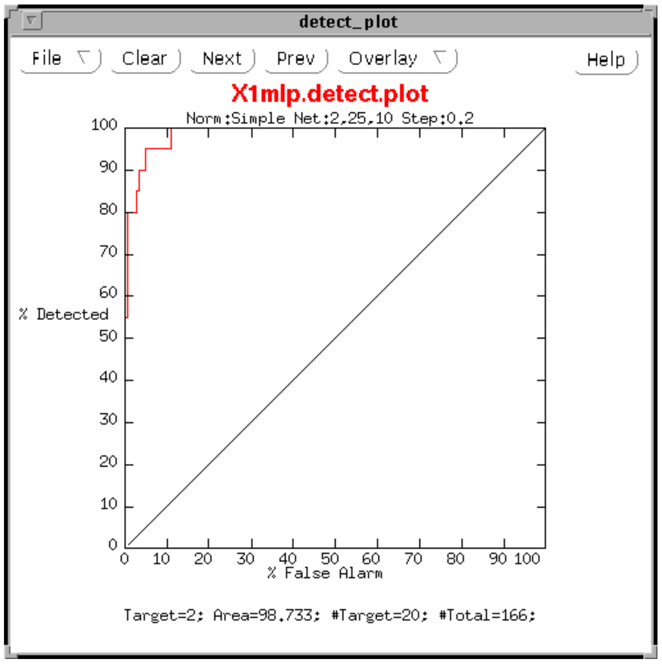
FIGURE 2.20 Posterior Probability Plot after 20 epochs of MLP training



Target=2 Chi=2.447 DOF=2 Signif=0.294 #Patterns=166

Figure 2.21 shows a receiver operating characteristics or ROC curve for class 2, hod. This plot shows the detection rate (hod patterns labeled as hod) *versus* the false alarm rate (other patterns labeled as hod) for a varying threshold value on the classifier output for the "hod" class. To generate the plot, the evaluation patterns are sorted by their class 2 output values. For each point on the plot, a threshold value is set. All patterns which have a class 2 output greater than the threshold are labeled as belonging to the class and all other patterns are labeled as not in the target class. The detection rate and false alarm rate that result from this labeling give the position of the plotted point. The plot in Figure 2.21 shows that the detection accuracy for the "hod" class is higher than 95% correct when 10% false alarms are allowed. The quality of the ROC curve can sometimes be judged by the area under the curve. In this case it is 98.7%, which is good. A perfect area of 100% is achieved if there is a threshold value such that all patterns above the threshold are in the target class and all patterns below the threshold are not. If the classifier output is random and contains no information, the ROC area is near 50% and the ROC is close to the diagonal line "%Detect = %False Alarms". A table of the values in the ROC plot curve is printed in the log file and the area under the curve is printed in the experiment notebook.

FIGURE 2.21 Detection (Receiver Operating Characteristic) Plot after 20 epochs of MLP training



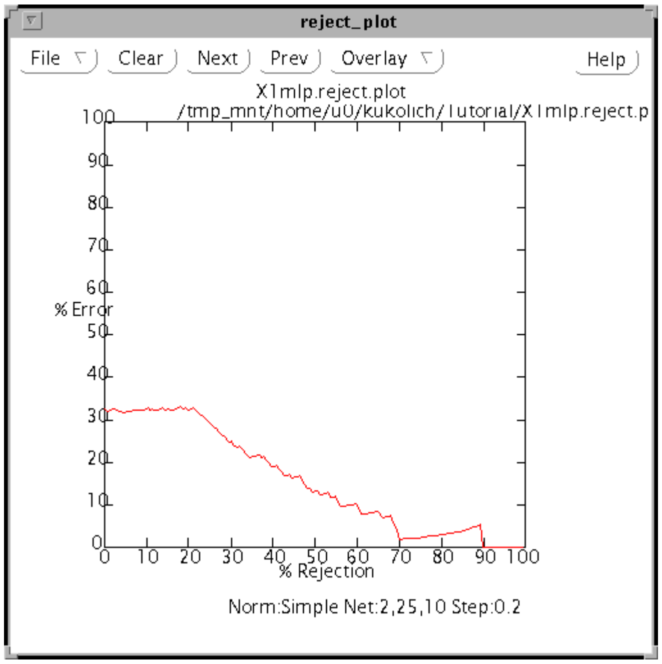
Target=2; Area=98.733; #Target=20; #Total=166;

Figure 2.22 shows a rejection plot. To generate the plot, all evaluation patterns are sorted by their highest output value across all classifier outputs. Patterns whose highest outputs are below a rejection threshold are rejected and not classified. The error rate of the classifier on the non-rejected patterns is plotted *versus* the percentage of the patterns rejected. If all the patterns which cause errors have low maximum output values, the percent error will drop until all the incorrectly classified patterns have been rejected. For the current experiment, rejecting more than 20% of the patterns substantially reduces the error rate on remaining patterns. The curve is erratic above 70% rejection because so few patterns remain.

2.9 Classification with Other Algorithms

There are many other classification algorithms available from LNKnet. One of the simplest ones to try on any problem is a K Nearest Neighbor classifier. To select the KNN algorithm, first use the Menu mouse button to display the menu attached to the **ALGORITHM** button on the main window. Move the mouse down and right to show the **Classifier** menu. Stop the mouse over **KNN (K-Nearest Neighbor)** and let go of the mouse button to select the KNN algorithm as shown in Figure 2.23.

FIGURE 2.22 Rejection Plot after 20 epochs of MLP training

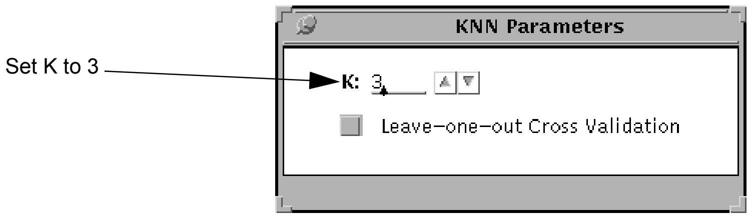


2.10 K Nearest Neighbor

A K-Nearest Neighbor classifier finds the K training patterns which are closest in Euclidean distance to a test pattern. It then assigns that test pattern to the most common class among the K neighbors. Ties are broken randomly.

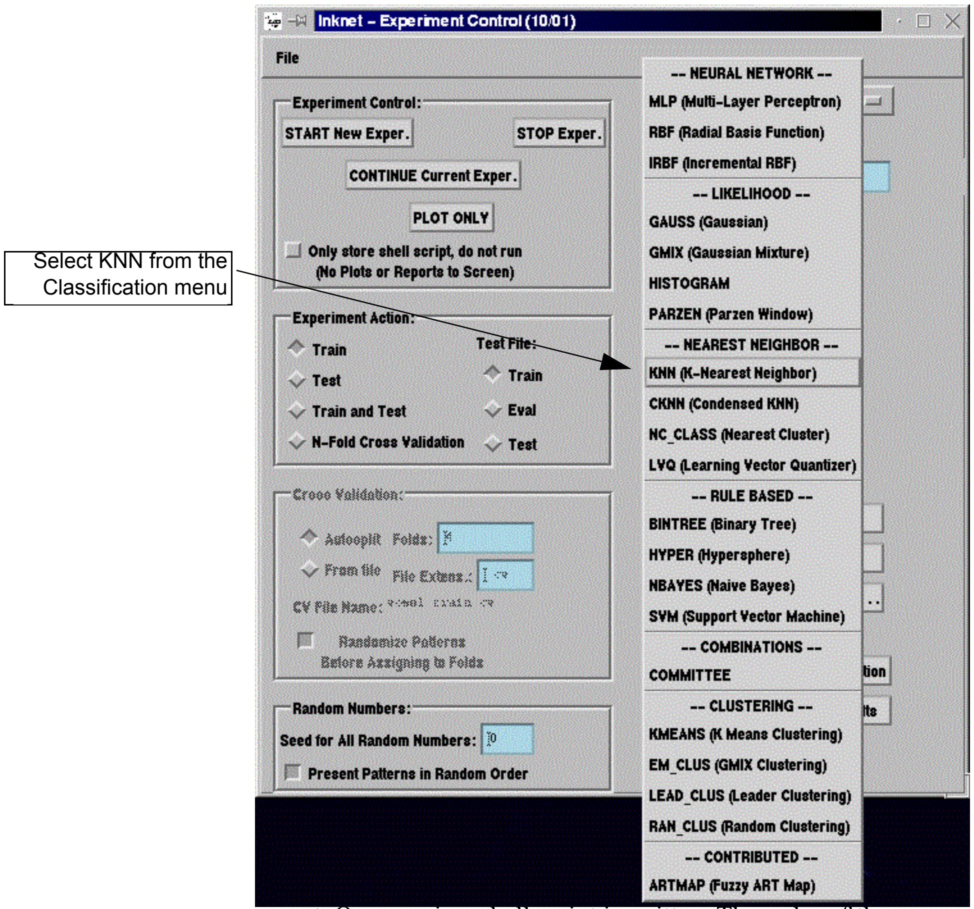
The only parameters you should have to set now are those on the **KNN Parameter** window shown in Figure 2.24. To display this window, select the **Algorithm Params...** button on the main window, as you did with the MLP classifier.

FIGURE 2.24 KNN Parameters



On the KNN parameter window set **K** to **3**. If you type 3, don't forget to hit carriage return to enter the new value. Select **START** on the main window to start the experi-

FIGURE 2.23 Selecting the Algorithm KNN



ment. Once again a shell script is written. The order of the commands is again train, evaluate, and plot. The files created during this experiment are shown in Table 2.2.

TABLE 2.2 Files Created During KNN Experiment

Files created by LNKnet	
`LNKnet.note`	Added X1knn parameters and results
`X1knn.run`	Shell script
`X1knn.screen`	Setting for all LNKnet windows
`LNKnet.note.screen`	Stored backup copy of new screen file
Files Created by KNN	
`X1knn.param`	Parameters for trained KNN classifier
`X1knn.log`	Copy of results printed to screen
`X1knn.err.eval`	Trial-by-trial results from testing

TABLE 2.2 Files Created During KNN Experiment

Files created by LNKnet	
`LNKnet.note`	Added X1knn parameters and results
Files created by Plot Programs	
`X1knn.region.plot.eval`	2-Dimensional plots

2.10.1 KNN Eval Results

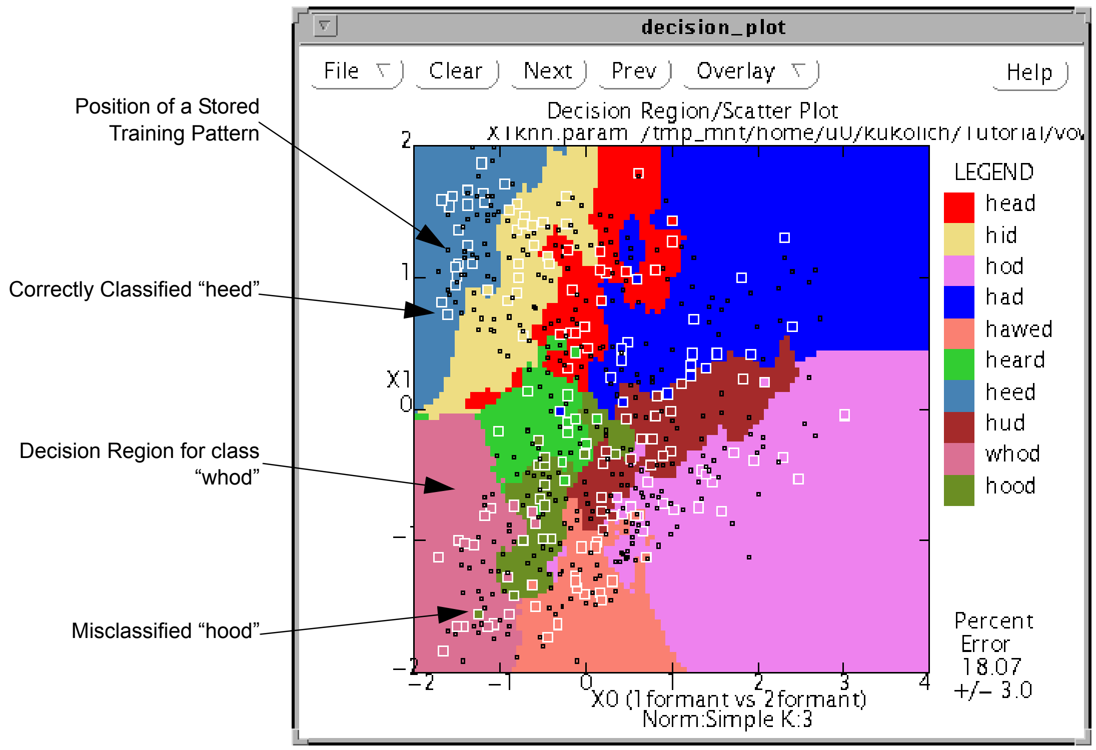
These are the classification results on the evaluation data, taken from the log file. The overall error rate is 18.07%, ±3%.

```
Classification Confusion Matrix - X1knn.err.eval
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Desired                          Computed Class
 Class     0     1     2     3     4     5     6     7     8     9   Total
 -----   ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
   0      15     1           1                                         17
   1            15                             3                       18
   2                  17           1                 2                 20
   3       1               15           1           1                 18
   4                   1          14                             1     16
   5       1                            8                        2     11
   6                                          18                       18
   7                         2                      15           1     18
   8                                                      12     4     16
   9                                    4           1     2     7     14
 -----   ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
 Total    17    16    18    18    15    13    21    19    14    15    166
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Error Report - X1knn.err.eval
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Class     Patterns   # Errors     % Errors StdDev RMS Err  Label
  0           17          2        11.76   ( 7.8)     0.149  head
  1           18          3        16.67   ( 8.8)     0.136  hid
  2           20          3        15.00   ( 8.0)     0.145  hod
  3           18          3        16.67   ( 8.8)     0.169  had
  4           16          2        12.50   ( 8.3)     0.149  hawed
  5           11          3        27.27   (13.4)     0.234  heard
  6           18          0         0.00   ( 0.0)     0.000  heed
  7           18          3        16.67   ( 8.8)     0.153  hud
  8           16          4        25.00   (10.8)     0.154  whod
  9           14          7        50.00   (13.4)     0.270  hood
          --------     --------    ------- -------------
Overall      166         30        18.07   ( 3.0)    0.163
```

2.10.2 Plots

Because KNN trains in a single pass, the cost plot and percent error plot are not available. KNN takes a vote amongst a pattern's nearest neighbors to determine the class of that pattern. This does not produce continuous outputs so there is no profile plot, posterior probability plot, detection plot, or rejection plot. There are no connections between the stored KNN parameters, so little information would be gained from a KNN structure plot. This leaves us with only the decision region plot displayed in Figure 2.25.

FIGURE 2.25 KNN Decision Region Plot


Position of a Stored Training Pattern
Correctly Classified “heed”
Decision Region for class “whod”
Misclassified “hood”
decision_plot
File
Clear
Next
Prev
Overlay
Help
Decision Region/Scatter Plot
X1knn.param /tmp_mnt/home/u0/kukolich/Tutorial/vow
LEGEND
head
hid
hod
had
hawed
heard
heed
hud
whod
hood
X1
X0 (1 formant vs 2 formant)
Norm:Simple K:3
Percent Error 18.07 +/- 3.0

The KNN decision region plot is generated in the same way as the MLP plot. The classifier is tested at every point in a 100 by 100 grid. The classification results are shown by drawing color coded regions for the class returned for each tested grid point. The overlaid scatter plot is identical to that in the MLP plot. Because the classification algorithm is different, the overlaid internals plot is different for the KNN classifier. Small black squares are drawn which show the positions of the stored training patterns.

2.11 Continuing Training

Looking at the results of the KNN classifier, the MLP results do not seem to be as good as they could be. The MLP classifier misclassified 32.5% of the evaluation data while the KNN classifier misclassified only 18%. Perhaps if the MLP classifier is trained more, it will perform as well as the KNN classifier.

2.11.1 Restoring Previous Experiment

To continue training the MLP classifier, LNKnet will first be restored to the state it was in for the MLP experiment.

Select the **MLP** classifier from the algorithm menu. Return to the **Report Files** window and select **RESTORE Screens from Screen File**. All of the windows in LNKnet should now be as they were when the MLP was trained.

On the MLP parameter window, the number of epochs was set to 20. Selecting **CONTINUE Current Exper.** will create a shell script that trains the previous MLP model for 20 more epochs. A notifier window will appear which says "**Shell file exists: OK to overwrite?**". LNKnet by default will use the same name for the shell script that continues the experiment. Either select **Overwrite** or hit **Return** to replace the old contents of `X1mlp.run` with the new script. The new shell script differs from the old one on only two lines. The create flag is not included in the new training call and the new training results are appended to `X1mlp.log` and `X1mlp.err.train.` When training is complete, the new model parameters will be stored in `X1mlp.param`, replacing the old ones. New versions of the plots will be created which will overwrite the existing plot files. An entry with the new experiment results will be added to the experiment notebook file, `LNKnet.note`.

2.11.2 MLP Eval Results

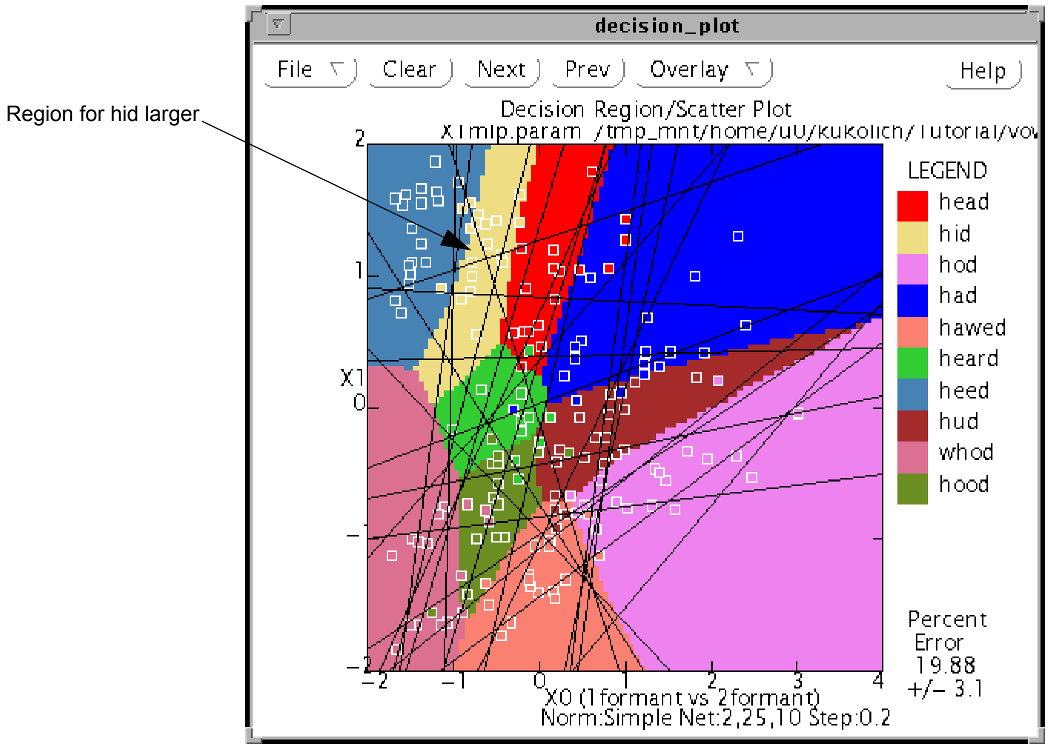
These are the classification results on the evaluation data after a total of 40 epochs of training. The MLP classifier now provides an error rate of 19.88%, ±3.1%. With the given standard deviation of 3.1% the new error rate is about the same as KNN's. The error rate might be improved with more training, but the amount of improvement for each epoch of training becomes increasingly small.

```
Classification Confusion Matrix - X1mlp.err.eval
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

Desired                          Computed Class
 Class     0     1     2     3     4     5     6     7     8     9   Total
 -----    ----  ----  ----  ----  ----  ----  ----  ----  ----  ---- -----
   0      13                 3           1                             17
   1            12                             6                       18
   2                  18                             2                 20
   3                        15           1           2                 18
   4                   2          13                             1     16
   5       1                             6           1           3     11
   6                                          18                       18
   7                               3                15                 18
   8                                                      13     3     16
   9                                     2           1     1    10     14
 -----    ----  ----  ----  ----  ----  ----  ----  ----  ----  ---- -----
 Total    14    12    20    18    16    10    24    21    14    17    166

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Error Report - X1mlp.err.eval
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Class     Patterns   # Errors     % Errors StdDev RMS Err  Label
  0           17          4        23.53   (10.3)     0.213  head
  1           18          6        33.33   (11.1)     0.227  hid
  2           20          2        10.00   ( 6.7)     0.152  hod
  3           18          3        16.67   ( 8.8)     0.172  had
  4           16          3        18.75   ( 9.8)     0.145  hawed
  5           11          5        45.45   (15.0)     0.231  heard
  6           18          0         0.00   ( 0.0)     0.038  heed
  7           18          3        16.67   ( 8.8)     0.187  hud
  8           16          3        18.75   ( 9.8)     0.171  whod
  9           14          4        28.57   (12.1)     0.224  hood
          --------     --------    ------- ------------

Overall      166         33        19.88   ( 3.1)    0.181
```

2.11.3 MLP Plots

The plots generated after the second twenty epochs of training are very similar to the first set of plots. The decision region plot is shown in Figure 2.26. The decision region boundaries and internals plot lines have moved. Naturally, the scatter plot has remained the same.

FIGURE 2.26 MLP Decision Region Plot after 40 Epochs



The profile plot is shown in Figure 2.27. The new profile plot shows that the outputs for the dominant class in each region of the input space have gotten closer to 1. The outputs for the other classes in those regions are closer to 0, making the transitions between classes sharper. The total line is less smooth but is still near one.

Because the weight magnitudes have not changed much, the new structure plot is almost identical to the old one. The structure plot is in Figure 2.28. Because the pattern by pattern results from the continuation training are appended to `X1mlp.err.train`, the error rate from all training is shown in the new cost plot and the new percent error plot, not just the rate from the twenty new epochs. These new plots are in Figure 2.29 and Figure 2.30 on page 37. On the probability plot, shown in Figure 2.31, the value in the second bin (output values of 0.2 to 0.4) has improved. The bin for values from 0.4 to 0.6 has been combined with the 0.6 to 0.8 bin. The bins which cover the middle of the range have been eliminated because they contain too few patterns. Figure 2.32 shows the new ROC plot which has changed very little. The ROC area has increased to 99.0%. Figure 2.33 shows the new rejection plot. There are more high scoring correctly classified pat-

FIGURE 2.27 MLP Profile Plot after 40 Epochs

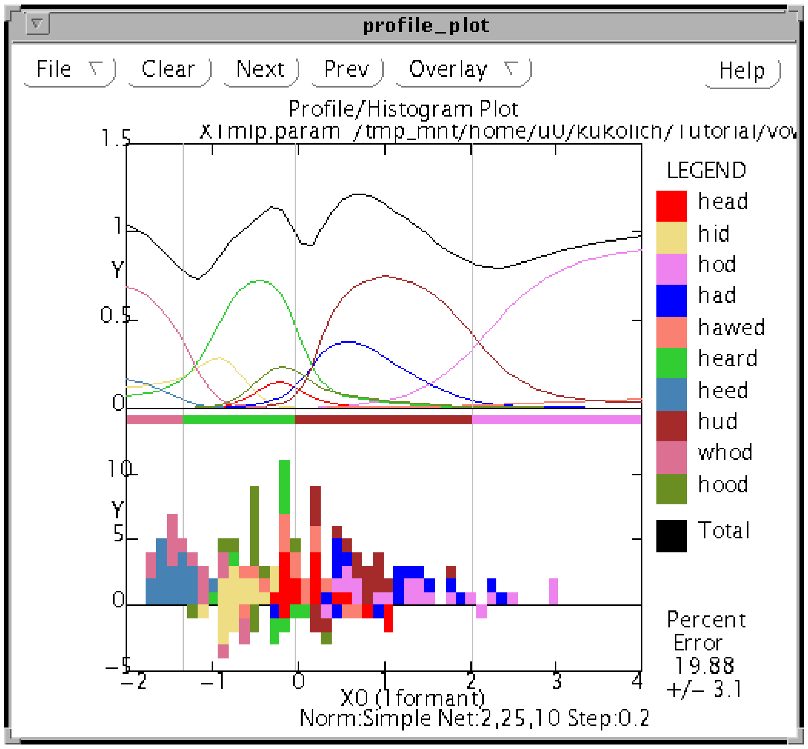
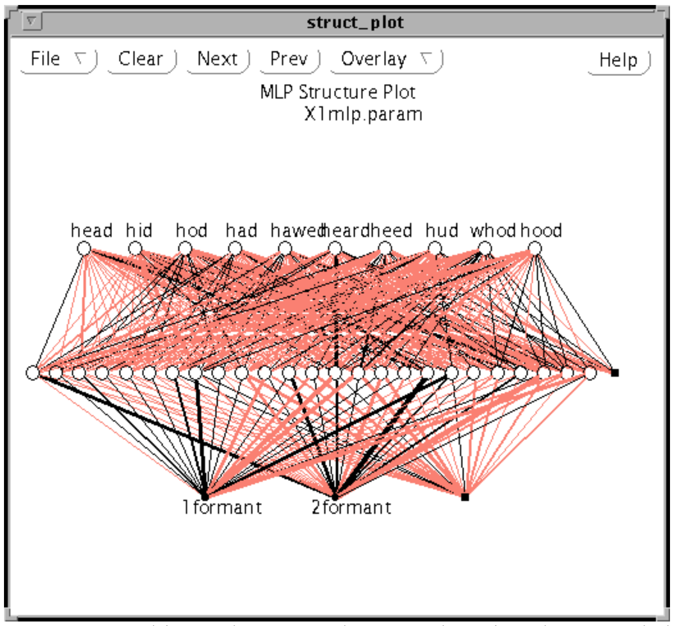


FIGURE 2.28 Structure Plot for 40 Epochs of MLP training



terns now. This can been seen because there is a downward slope in this curve with few rejections.

FIGURE 2.29 Percent Error Plot for 40 Epochs of MLP training

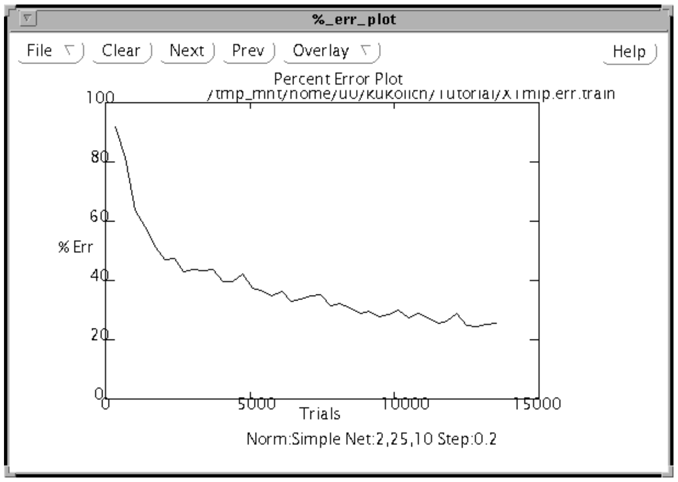
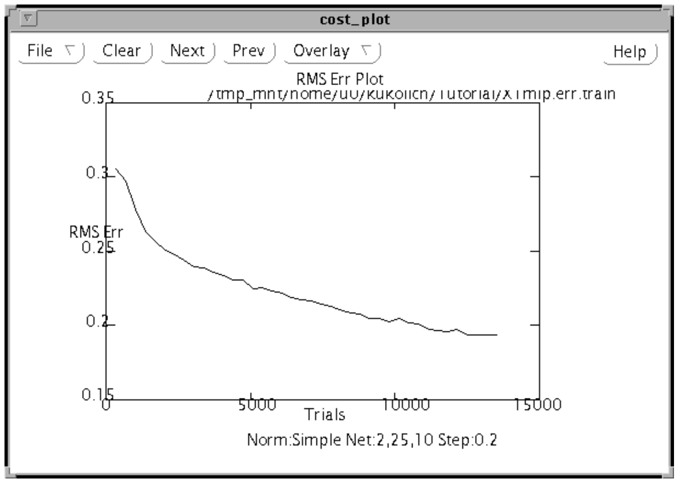


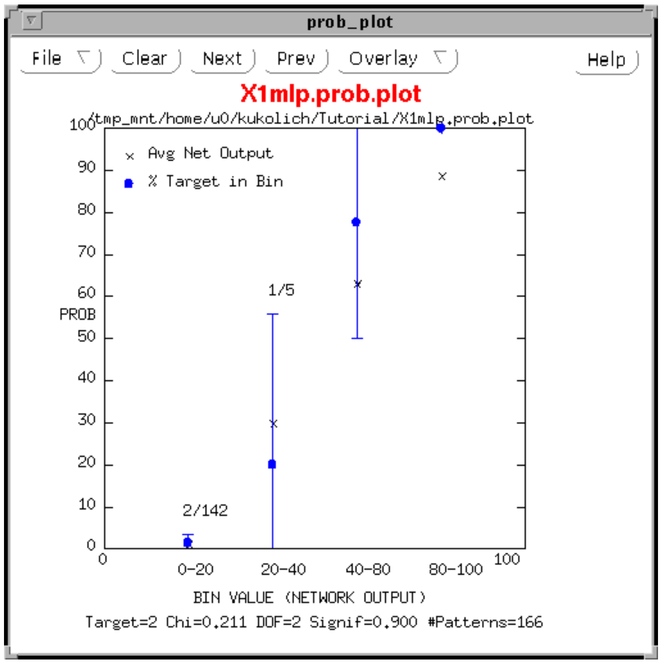
FIGURE 2.30 Cost Plot for 40 epochs of MLP training



2.12 Cleaning up Windows and Files

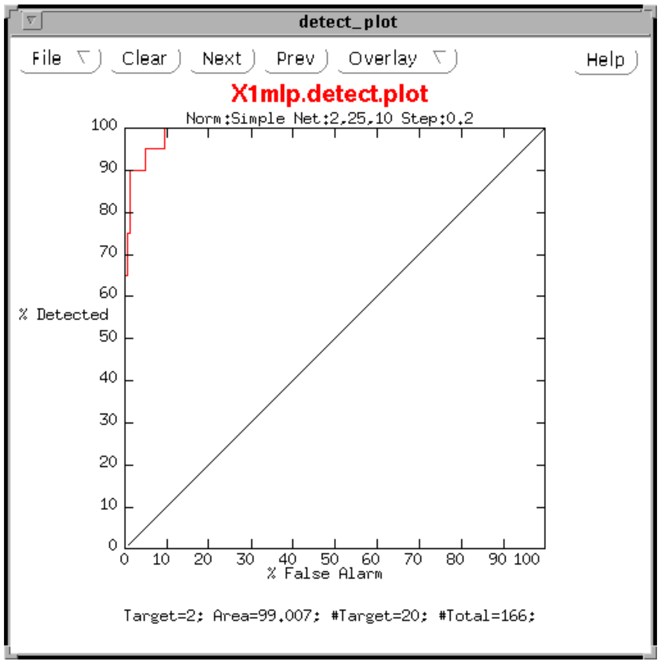
By now your screen is cluttered with many small LNKnet popup windows and plot windows. The popup windows can be closed by closing the windows using the mechanism provided in the window manager you are using. The many plots can be removed by moving the mouse into the plotting area of the window and typing ‘q’ or by selecting **quit** from the file menu at the top of each plot window. The experiment directory is also cluttered with shell scripts, plot files, and error files. Removing the error files regularly

FIGURE 2.31 Posterior Probability Plot after 40 epochs



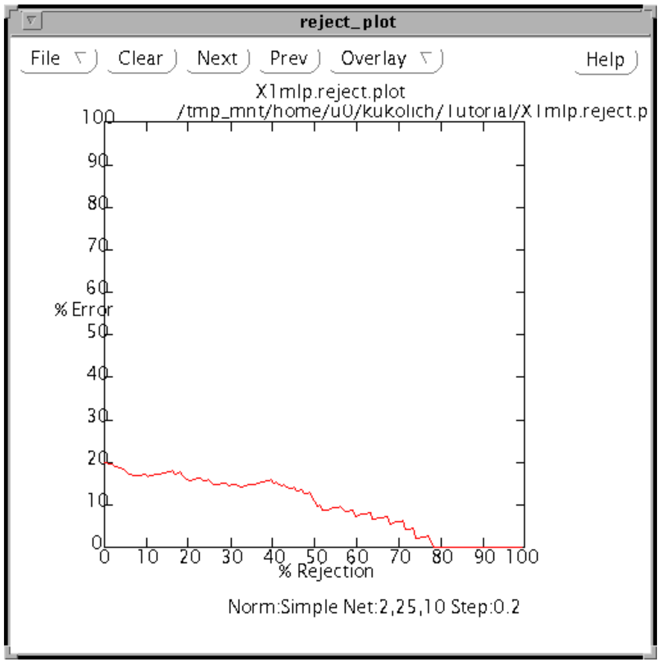
```
Target=2 Chi=0.211 DOF=2 Signif=0.900 #Patterns=166
```

FIGURE 2.32 ROC (Detection) Plot after 40 epochs



```
Target=2; Area=99.007; #Target=20; #Total=166;
```

FIGURE 2.33 Rejection Plot after 40 epochs



is important because they can be very large. The command `rm X1*.err*` will remove the error files generated during the previous experiments in this tutorial. These files can be recreated by re-running a LNKnet experiment and are only necessary if you want to continue training an incrementally trained classifier or if you want to generate additional plots that depend on these files such as the error rate versus training time plot.

2.13 Feature Selection

LNKnet allows you to select features and thus reduce the number of input features as an approach to improve generalization. There are two ways to do this, feature selection of the raw input data, or rotation of the input space using normalization followed by feature selection. This first series of experiments will use feature selection alone.

You will run several experiments, trying to find the best set of features to use to solve a multi-dimensional problem. Because the classification algorithm will not change, you will need to change the experiment name to prevent new experiments from overwriting the old ones. The first experiment will use all input features so change the **Experiment Name** on the main window to **all**.

For this series of experiments you must use a data base with more than two features. Select the data base **gnoise_var** from the **Data Base List** scrolling list on the data base window. This data base is provided with LNKnet and should have been copied into the experiment directory at the start of this tutorial. There are 8 input features and 10 classes. The classes are clusters each centered along the line $x_0 = x_1 = \ldots = x_7$ where x_d is input dimension d $(0 \le d \le 7)$. Class 0 is centered on (0,0,0,0,0,0,0,0), class 1 on

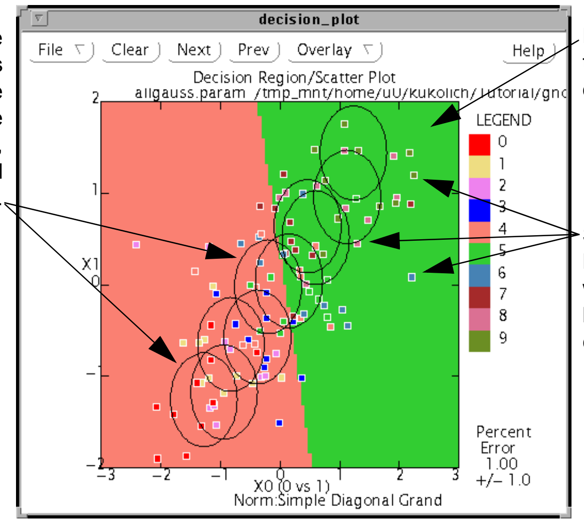
(1,...,1), class 2 on (2,...,2), and so on. Gaussian noise is added to the centers to generate the patterns for each class. The variance of the noise depends on the input feature number. It is lowest for feature number 7 and highest for feature number 0. The variance of the data in the eighth input dimension is 0.25. The variance in the lower dimensions increases by 0.25 every dimension giving a variance of 2 for the first dimension. The high numbered features thus provide more information than the lower numbered features because they have less variance.

The **gnoise_var** data base has 8 input dimensions. If the current plot parameter settings are used, scatter plots may not show all of the data. Go to the **Plot Selection** window and bring up the **Decision Region Plots** window. On the decision region plot window, make sure that **Show All Data Points** is selected. For the remainder of this tutorial, only the decision region plots will be shown. Select the check boxes for the other plots again to remove the checks and deselect the plots.

Because this data base was generated using a Gaussian distribution for each class, a Gaussian classifier should be used to solve this problem. On the main window, select the classification algorithm **Gauss**. Go to the algorithm parameter window to check the variables for the Gaussian classifier. Each class has the same variance, so make sure that **Same for All Classes (Grand)** is selected. The variance in each direction is independent, so make sure that **Diagonal Covariance Matrix** is selected.

Your first experiment uses all of the input features to obtain a base error rate. Select **START** to run the first feature selection experiment. The classifier should make one error classifying the evaluation data. Figure 2.34 shows the decision region plot for the first experiment. Because all of the evaluation data is displayed, the decision regions do

FIGURE 2.34 Decision Region Plot using all 8 inputs


decision_plot
File
Clear
Next
Prev
Overlay
Help
Decision Region/Scatter Plot
allgauss.param /tmp_mnt/home/u0/kukolich/tutorial/gnc
LEGEND
0
1
2
3
4
5
6
7
8
9
X1
X0 (0 vs 1)
Norm:Simple Diagonal Grand
Percent
Error
1.00
+/- 1.0
-3
-2
-1
0
1
2
3
2
1
0
-1
-2
Internals plot. Because these are grand variances, all of the ellipses representing the Gaussians have the same shape. Because the covariance matrices are diagonal, the axes of the ellipses are parallel to the input dimensions.
Decision Regions looking at the first two dimensions. The values for other dimensions are set to zero.
Scatter plot of all of the data. Because there are 8 dimensions, whether the color matches no longer indicates correct classifications.

not seem to match the scatter plot data. The internals plot for the Gaussian classifier is the set of ellipses shown over the scatter plot. These ellipses represent the Gaussian functions that model each class. The length and width of the ellipses are proportional to the variances of the Gaussians. More plots could be generated showing the other dimensions by changing the **Input Dimensions to Plot** on the **Decision Region Plot** window.

Select **PLOT ONLY** on the main screen to write a shell script that generates the requested plots without retraining or retesting the classifier first.

Each feature in this data base is a noisy estimate of the class number. All eight features may not be necessary to get the right answers. You can try using only the first feature. On the main window, change **Experiment Name** to **N1** . Bring up the **Feature Selection** window by selecting **Feature Selection...** on the main window. On the feature selection window, select **First N** and change **N** to **1.** Select **START** to run the experiment. The error rate using just the first feature should be 78% on the evaluation data.

FIGURE 2.35 Decision Region Plots using first input

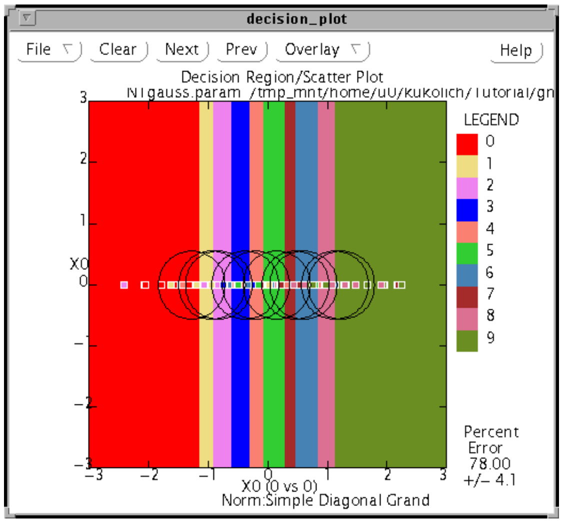
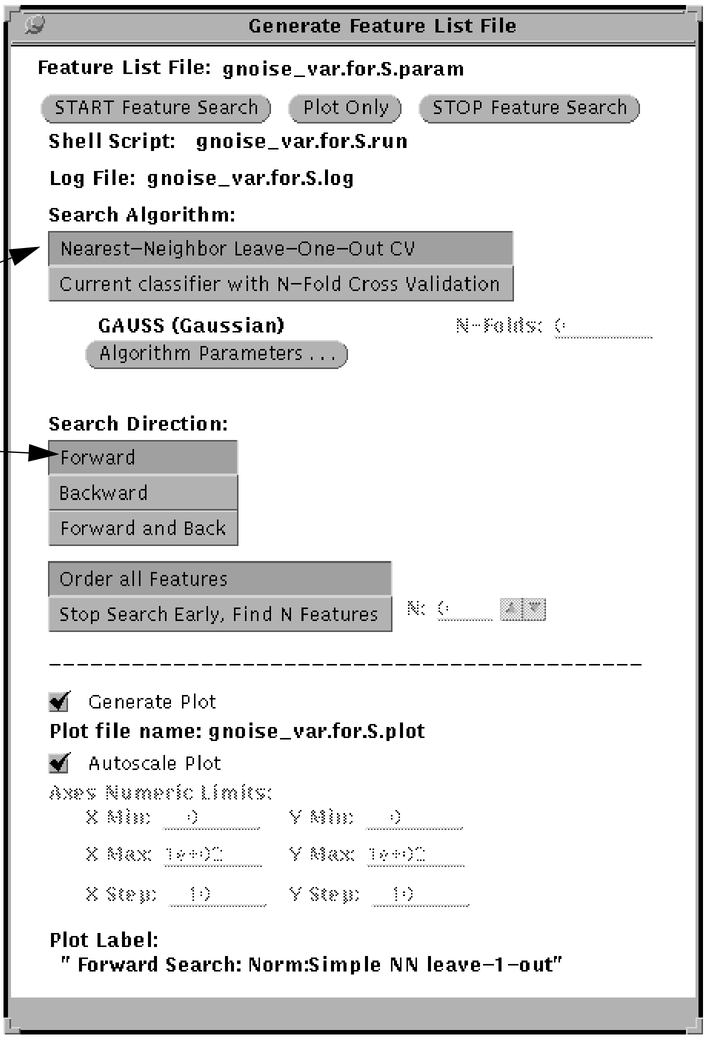

decision_plot
File
Clear
Next
Prev
Overlay
Help
Decision Region/Scatter Plot
N1gauss.param /tmp_mnt/home/u0/kukolich/Tutorial/gn
LEGEND
0
1
2
3
4
5
6
7
8
9
Percent
Error
78.00
+/– 4.1
X0
X0 (0 vs 0)
Norm:Simple Diagonal Grand
3
2
1
0
–1
–2
–3

Figure 2.35 shows the decision region plot for the classifier using only the first input feature. Because there is really only one dimension being plotted there is no variation along the Y direction of the decision region plot, the scatter plot data is all shown on the line y=0 and the internals plot uses circles to represent the variance of the Gaussians.

For the next experiment, change the experiment name to **N2**. On the feature selection window change **N** to **2**. Now repeat the experiment using the first two inputs. The error rate should be 62%. Finally, change the experiment name to **N4** and change **N** to **4** on the feature selection window. Repeat the experiment using the first 4 inputs. The error rate should be 48%. The 2D plots for these experiments will still be generated, but they are not shown.

The variance of the data in the first few features is too high for these features to be useful in discriminating the classes. Perhaps the error rate can be reduced by picking out particular features rather than just taking them in order. One approach to feature selection is to create a list of features in order of presumed importance. Any of the feature search algorithms can be used to create such a list. On the Feature Selection window select **Read Feature List from File.** Because the feature list file has not been created yet, an error sign should appear on the feature selection window and beside the Feature Selection button on the main window. Select the **Generate Feature List File...** button at the bottom of the window. This brings up the **Generate Feature List File** window shown in Figure 2.36. Select **Nearest-Neighbor Leave-One-Out CV** as the search algorithm and **Forward** as the search direction. Select **Start Feature Search** to start the search for the best set of input features to use.

FIGURE 2.36 Feature List File Generation Window


Use nearest neighbor classifier to test feature lists
Search forward, selecting one feature to add at a time
Generate Feature List File
START Feature Search
Plot Only
STOP Feature Search
Nearest–Neighbor Leave–One–Out CV
Current classifier with N–Fold Cross Validation
Algorithm Parameters . . .
Forward
Backward
Forward and Back
Order all Features
Stop Search Early, Find N Features

Feature List File: gnoise_var.for.S.param

Shell Script: gnoise_var.for.S.run

Log File: gnoise_var.for.S.log

Search Algorithm: Nearest–Neighbor Leave–One–Out CV

GAUSS (Gaussian) N-Folds: 0

Search Direction: Forward

N: 0

✓ Generate Plot

Plot file name: gnoise_var.for.S.plot

✓ Autoscale Plot

Axes Numeric Limits:

X Min: 0 Y Min: 0

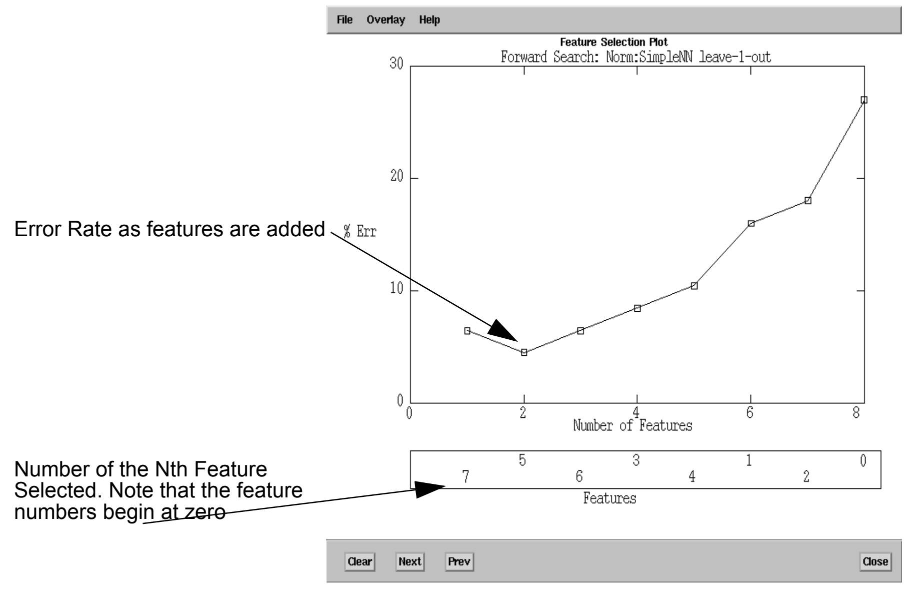
X Max: 1e+02 Y Max: 1e+02

X Step: 10 Y Step: 10

Plot Label:
" Forward Search: Norm:Simple NN leave–1–out"

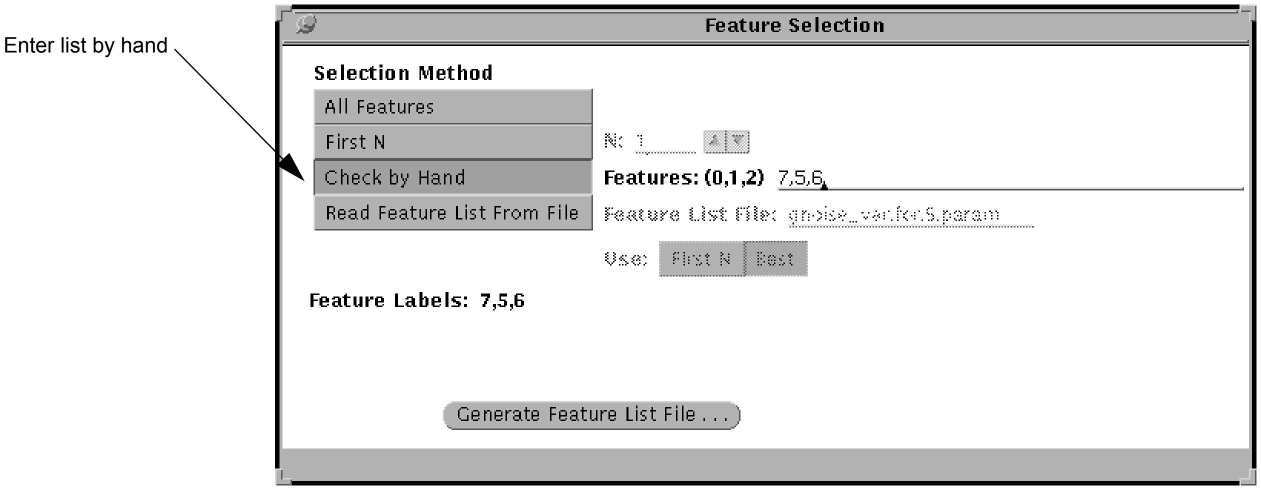
In this search, the program feat_sel tests each feature to find the one which is most effective in classifying the data by itself. The remaining features are then paired with the first and the best is selected as the second feature. Features are added this way until there are no more left. The feature sets are tested using a nearest neighbor classifier using leave-one-out cross validation. They could also have been tested using the current Gaussian classifier with ten-fold cross validation. The results of each step of the search and the best set of features is printed to the screen and to a log file. The plot in Figure 2.37 shows the cross validation error rate achieved as each feature is added. We can see that most of the features actually increase the error rate and that using features 7, 5, and 6 achieves a good error rate.

FIGURE 2.37 Feature Search on gnoise_var data base


File
Overlay
Help
Feature Selection Plot
Forward Search: Norm:SimpleNN leave-1-out
% Err
Number of Features
Features
5 3 1 0
7 6 4 2
Clear
Next
Prev
Close
Error Rate as features are added
Number of the Nth Feature Selected. Note that the feature numbers begin at zero

To use a subset of these features in the selected order, change the experiment name on the main window to **last3.** Return to the feature selection window. Set **Use** to **First N** and **N** to **3** so that you are using the first 3 features from the list in the new feature list file. If the feature list file did not previously exist, there will still be an error message saying so. Click the mouse on the error message or select **Read Feature List From File** again to erase the message. An alternate way to use these features is to select **Check by Hand** as the selection method. Then type in the following list: **7,5,6.** These comma delimited lists are used in many places in LNKnet. The list is made from integers separated by commas with no spaces or tabs. See Problem 2.12 on page 131 for more information about comma delimited lists. Figure 2.38 shows the feature selection window with the last three features selected. This experiment should produce an error rate of 3% on the evaluation data. The shell script and log file for this experiment, last3gauss, can be found in Appendix C. Table 2.3 shows the results of the feature selection experi-

FIGURE 2.38 Hand picking features



Features: (0,1,2) 7,5,6,

Feature List File: gnoise_var.fc.S.param

Feature Labels: 7,5,6

ments. These results can also be found in the notebook file, LNKnet.note. A copy of the notebook file is found in Appendix C.

TABLE 2.3 Error rate of Gaussian Classifier on evaluation data using different features of gnoise_var data

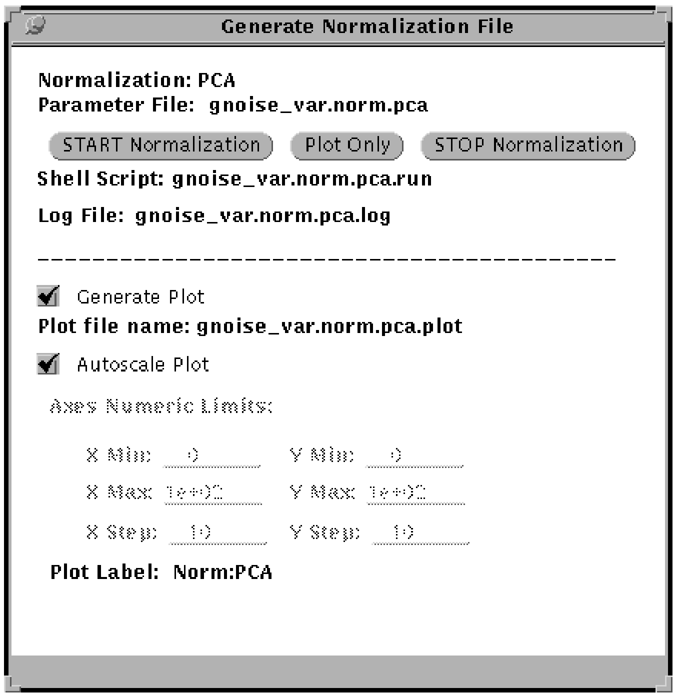
Experiment Name	Number of features	Eval Error Rate
`all`	`All 8`	1%
`N1`	`first 1`	78%
`N2`	`first 2`	62%
`N4`	`first 4`	48%
`last3`	`3 picked`	3%

2.14 Feature Reduction using Normalization

Another way to reduce the number of features is to project the input space onto a different space that either rotates the original features or reduces the number of input features. Two projection algorithms are provided. They are principal components analysis (PCA) and linear discriminant analysis (LDA). PCA rotates the space so that the first dimension is in the direction of greatest variance in the input space. The other dimensions follow in decreasing order of variance. LDA rotates the space so that the lowest numbered dimensions are in directions which best discriminate between classes. LDA assumes classes and class means have Gaussian distributions. PCA produces as many output features as there are original input features and ignores class labels. LDA uses class labels in training and produces a smaller number of output features when there are fewer classes than input features. The number of output features with LDA is the minimum of M-1 or D where M is the number of classes and D is the number of original input features. Because PCA and LDA are applied to raw data before the input vectors are handed to the classifiers, they are included as normalization methods.

To try PCA and LDA on the gnoise_var problem, display the normalization window by selecting **Feature Normalization...** on the main window. Select Principal Components as the normalization. Because the normalization file has not been created yet, an error will appear on this window and beside the normalization button on the main window. Select **Generate Normalization File**... to bring up the window which creates normalization files. The **Generate Normalization File** window is shown in Figure 2.39. Select **Run** on this window to calculate the PCA parameters. Now select Linear Discriminant on the normalization window. Select **Run** again on the normalization file generation window to calculate the LDA parameters.

FIGURE 2.39 Generate Normalization Window


Generate Normalization File

Normalization: PCA
Parameter File: gnoise_var.norm.pca

Shell Script: gnoise_var.norm.pca.run

Log File: gnoise_var.norm.pca.log

✓ Generate Plot
Plot file name: gnoise_var.norm.pca.plot

✓ Autoscale Plot

Axes Numeric Limits:

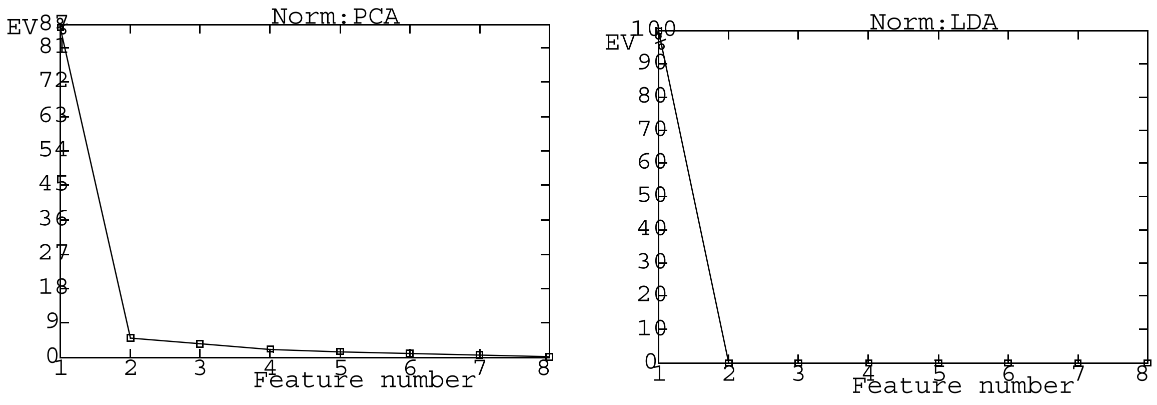
X Min: 0 Y Min: 0
X Max: 1e+02 Y Max: 1e+02
X Step: 10 Y Step: 10

Plot Label: Norm:PCA

A plot is generated for each normalization method. The plots show the relative sizes of the eigen values for each of the features in the rotated space. This can be taken as a measure of the importance of each feature. The plots in Figure 2.40, show that with both

PCA and LDA the eight input features can be replaced by one feature that accounts for most of the variance.

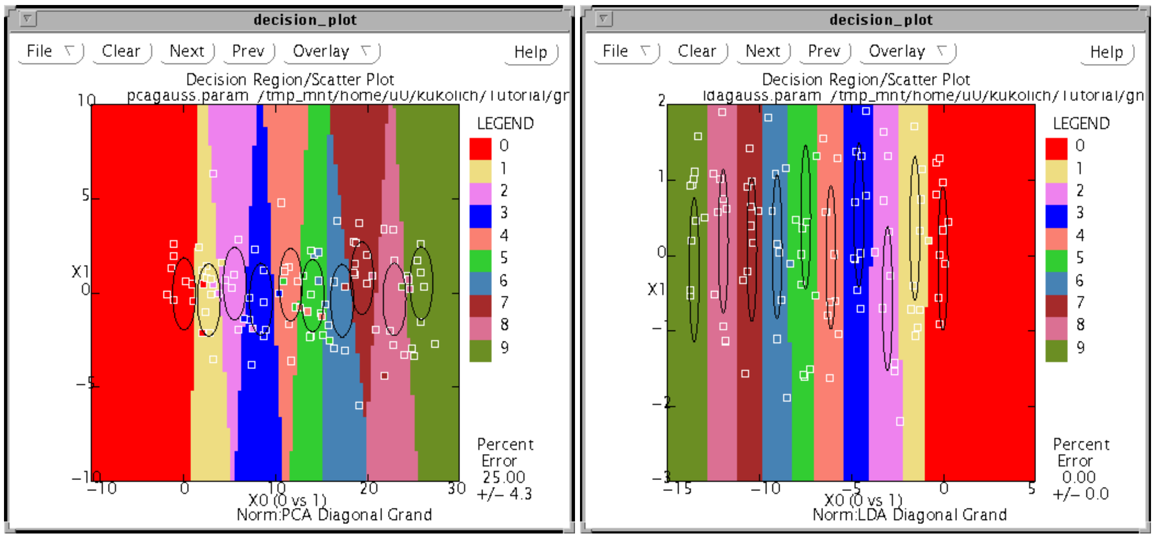
FIGURE 2.40 Eigenvalue Plots for PCA and LDA



To continue the feature reduction experiments, change the experiment name to **pca**. Go to the **Feature Selection** window and use only the first two features by selecting **First N** as the selection method and changing **N** to **2**. Select **Principal Components** as the normalization method on the normalization window. There should be an error rate of 25% on the evaluation data when you run the experiment with PCA. Change the experiment name to **lda**, select **Linear Discriminant** as the normalization method and run the experiment one final time. You should get no errors on the evaluation data when normalizing with LDA. Figure 2.41 shows the decision region plots for these two experiments. The dimensions being plotted are the first two input dimensions after normalization. It is possible to plot using the original dimensions by selecting **Do Not Normalize Data for Plot** on the **Decision Region Plot** window.

FIGURE 2.41 Decision Region Plots when First Two Rotated Dimensions are Used (PCA and LDA)

Note that the scatter plot data, which was scattered along the line X1=X0 before, is found along the line X1=0.


decision_plot
File
Clear
Next
Prev
Overlay
Help
Decision Region/Scatter Plot
pcagauss.param /tmp_mnt/home/u0/kukolich/Tutorial/gn
LEGEND
0
1
2
3
4
5
6
7
8
9
Percent Error 25.00 +/- 4.3
X0 (0 vs 1)
X1
Norm:PCA Diagonal Grand
decision_plot
File
Clear
Next
Prev
Overlay
Help
Decision Region/Scatter Plot
ldagauss.param /tmp_mnt/home/u0/kukolich/Tutorial/gn
LEGEND
0
1
2
3
4
5
6
7
8
9
Percent Error 0.00 +/- 0.0
X0 (0 vs 1)
X1
Norm:LDA Diagonal Grand

The gnoise_var data base is unusual in that many features are noisy and contribute little to discriminating the classes. Because principal components analysis looks for the greatest variance, it favored the lower dimensions and rotated the space to accentuate them. An interesting exercise is to do a feature search on the gnoise_var data base with PCA set. This will find the best set of rotated features. When this search is run, the best set achieves an error rate of 21%. Although this is not as good as using the unrotated dimensions with smaller variances, it is still better than using the original large variance dimensions alone.

LDA assumes that the classes and their means can be modelled by unimodal Gaussian distributions. Because this is correct in the case of gnoise_var, normalizing the data with LDA produces a good classification result.

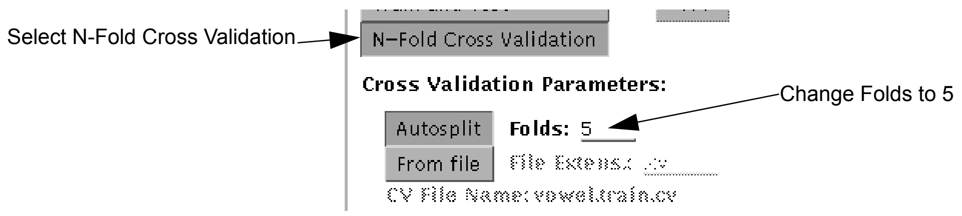
2.15 Cross Validation

Sometimes there is not enough data available to divide it into separate training, testing, and evaluation partitions. In such a case, N-fold cross validation can be used to estimate the future classification error rate on new data. The idea of cross validation is to split the data in to N equal-sized folds and test each fold against a classifier trained on the data in the other folds. The cross-validation error rate is obtained by summing the errors from the tests. The draw-back of cross-validation is the time it takes to run N experiments. It is thus primarily used only with a small N (between 4 and 10 folds) and when the number of patterns is low (e.g. tens to thousands of patterns).

Before starting cross validation experiments, reset the selections made while exploring feature selection and normalization. On the main window, set the experiment name to **all**. On the Reports window, select **Restore Experiment Screens** to bring all of the LNKnet and algorithm parameters back to their states at the start of Section 2.13.

Now set some of the general parameters. You will be doing a 5-fold cross validation experiment. This is handled automatically by the classification program, so you do not need to separately train and evaluate the five classifiers. Select **N-Fold Cross-Validation** as the action on the main window and change **Folds** to **5**. Change the experiment name to **cv**. Figure 2.42 shows the section of the main window which holds the cross validation parameters.

FIGURE 2.42 Cross Validation Selection



Cross Validation Parameters:

Folds: 5

File Extens.: .cv

CV File Name: vowel.train.cv

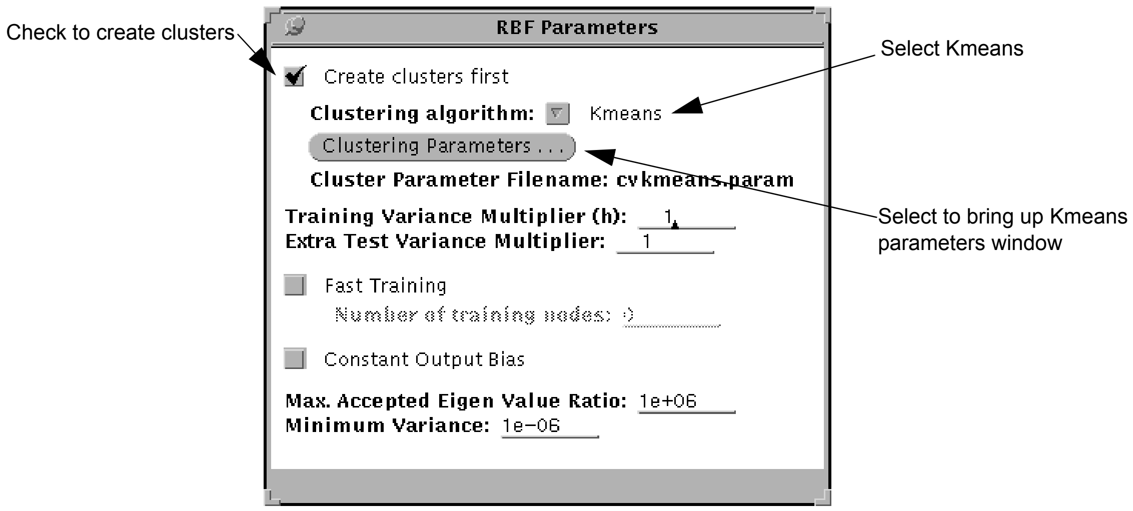
You need to select a small data base for the cross validation experiment. The **iris** data base has the fewest patterns of any of the “real” data bases provided with LNKnet. Select it on the **Data Base** window. The classes in the iris data base are three kinds of iris flowers. The inputs are the sepal length and width and the petal length and width. This data base was collected by R.A. Fisher [8] in the 1930's.

For the cross validation experiment, use the **Radial Basis Function (RBF)** classifier. The RBF classifier uses a set of Gaussian basis functions to map the input space into data clusters. In assigning a class to a pattern, the output for each class is a weighted sum of the basis function outputs for the pattern. The RBF program trains the weights connecting the basis functions to the outputs. LNKnet has another RBF program, IRBF or incremental RBF, that also trains the means and variances of the basis functions. Both programs use clustering algorithms to specify initial basis function locations.

This experiment will use the K-Means algorithm for clustering. The K-Means algorithm generates a set of K cluster centers and assigns training patterns to these centers. It uses these sets of patterns to iteratively improve the positions of the centers and to calculate the final variances of the clusters.

Select **RBF** as the current classification algorithm. The K-Means program will be run automatically before the RBF program, if desired. On the **RBF Parameter** window, shown in Figure 2.43, select **Create clusters first** and select **Kmeans** as the clustering algorithm.

FIGURE 2.43 RBF Parameter Window


Check to create clusters
Select Kmeans
Select to bring up Kmeans parameters window
RBF Parameters
Clustering Parameters . . .

Create clusters first ✓

Clustering algorithm: Kmeans

Cluster Parameter Filename: cvkmeans.param

Training Variance Multiplier (h): 1

Extra Test Variance Multiplier: 1

Fast Training

Number of training nodes: 0

Constant Output Bias

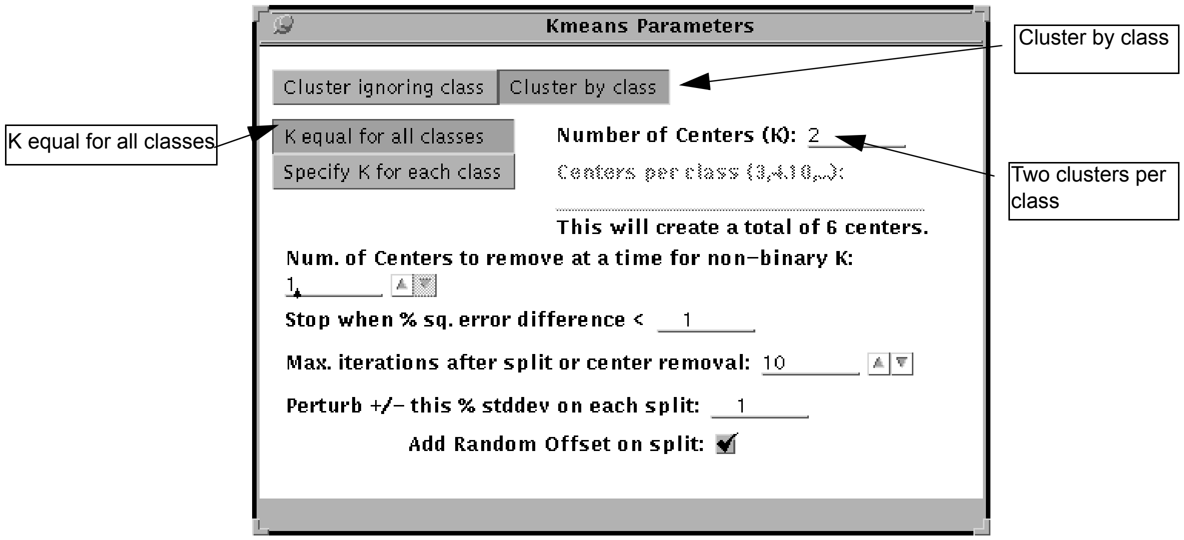
Max. Accepted Eigen Value Ratio: 1e+06

Minimum Variance: 1e-06

Now check the K-Means parameters. On the RBF window, select **Clustering Parameters...** to bring up the **Kmeans Parameters** window shown in Figure 2.44. On the

Kmeans window select **Cluster by class**, **K equal for all classes**, and set **K** to **2** centers per class.

FIGURE 2.44 KMEANS Parameter Window



Number of Centers (K): 2

Centers per class (3,4,10,...):

This will create a total of 6 centers.

Num. of Centers to remove at a time for non-binary K: 1

Stop when % sq. error difference < 1

Max. iterations after split or center removal: 10

Perturb +/- this % stddev on each split: 1

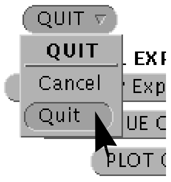
Add Random Offset on split: ✓

Select **START** to write the cross validation shell script. This script first generates five sets of K-means clusters. The training data for these clusters is the same as will be used to train the classifier. The clustering program will generate two clusters for each of the three classes, for a total of six for each of the five cross validation folds. After the clustering is finished, the RBF program is called to train and test the five classifiers. A confusion matrix and error summary is generated for each of the classifiers. At the end, a confusion matrix and error summary is displayed for the results of all of the testing. The combined error rate for the cross validation experiment is 4%. This result is also appended to the notebook file which is in Appendix C. The shell script and log file from the cross validation experiment are also in Appendix C.

2.16 Exiting LNKnet

Congratulations, you have completed the LNKnet tutorial! To quit LNKnet, use the mouse menu button to select **Quit** from the Quit menu button in the top left corner of the LNKnet main window as shown in Figure 2.45. Further details on classifiers, plots, and other features in LNKnet can be found in the following chapters of this user's guide and on the LNKnet manual pages.

FIGURE 2.45 Exiting LNKnet



CHAPTER 3 Classifiers

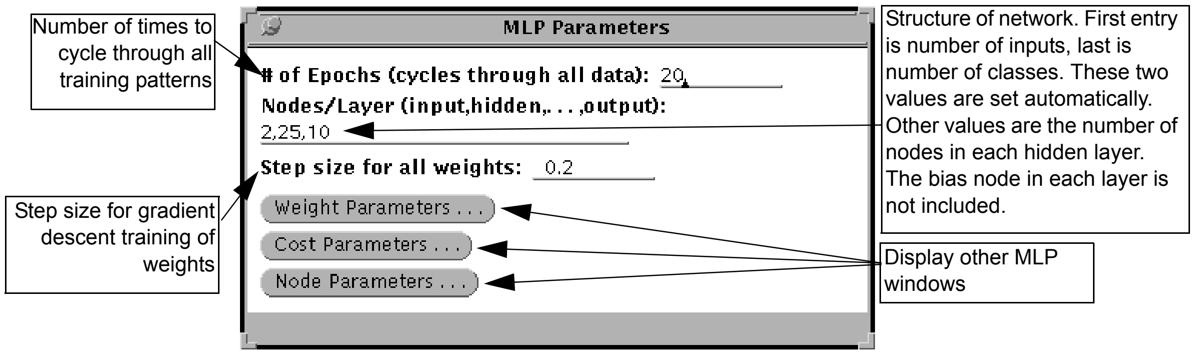
3.1 Neural Network Classifiers

Neural network classifiers are motivated by biological nervous systems and use many simple processing elements to estimate posterior class probabilities of input patterns. That is, they estimate $p(A|X)$ where A represents a particular class, X represents the input pattern, and $p(A|X)$ is the posterior probability for class A. LNKnet includes important types of neural network algorithms that can be applied to classification problems.

3.1.1 Multi-Layer Perceptron (MLP)

Multi-Layer Perceptron classifiers[21] are the most widely used neural network classifiers. They provide good performance on many problems and create decision regions by positioning smooth plateau-like functions produced by sigmoids in the input space. In the limits where connections weights are “high” they create hyperplanes which define “half-spaces”. These are combined to form class decision regions. The hyperplanes and their combinations are specified by weighted connections between the layers of the multi-layer perceptron. The weights are trained using a back propagation algorithm to perform a gradient descent which minimizes the error of the outputs according to the selected cost function.

FIGURE 3.1 MLP Parameters

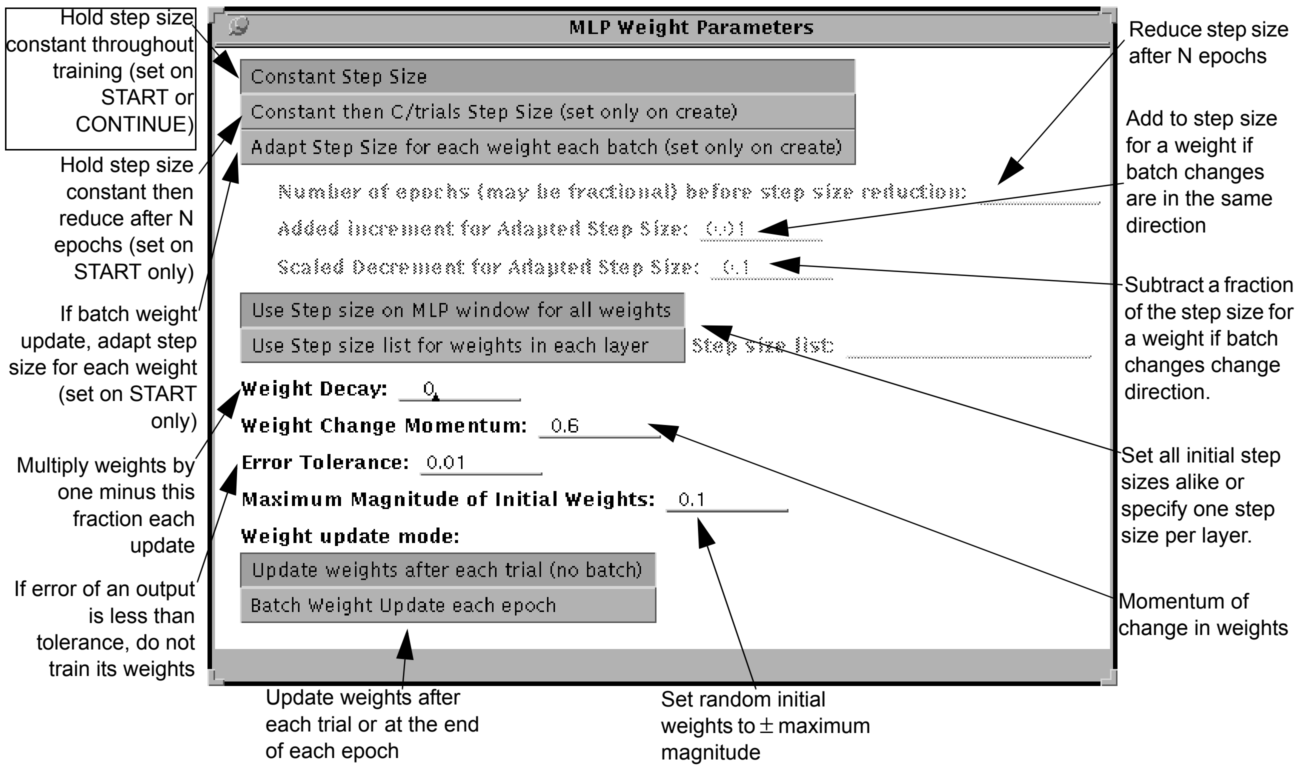


MLP Parameters
Weight Parameters . . .
Cost Parameters . . .
Node Parameters . . .
Number of times to cycle through all training patterns
Structure of network. First entry is number of inputs, last is number of classes. These two values are set automatically. Other values are the number of nodes in each hidden layer. The bias node in each layer is not included.
Step size for gradient descent training of weights
Display other MLP windows

# of Epochs (cycles through all data): 20
Nodes/Layer (input,hidden,. . . ,output): 2,25,10
Step size for all weights: 0.2

This algorithm has the most options of any LNKnet program. MLP classifiers examine all the data many times in training. The first option to set is the number of times to examine the data. The next is the structure of the network which is contained in a comma delimited list with the number of nodes in each layer of the network. The first and last entries are the number of inputs and the number of classes. They are set auto-

FIGURE 3.2 MLP Weight Parameters


Hold step size constant throughout training (set on START or CONTINUE)
Hold step size constant then reduce after N epochs (set on START only)
If batch weight update, adapt step size for each weight (set on START only)
Multiply weights by one minus this fraction each update
If error of an output is less than tolerance, do not train its weights
MLP Weight Parameters
Constant Step Size
Constant then C/trials Step Size (set only on create)
Adapt Step Size for each weight each batch (set only on create)
Use Step size on MLP window for all weights
Use Step size list for weights in each layer
Update weights after each trial (no batch)
Batch Weight Update each epoch
Reduce step size after N epochs
Add to step size for a weight if batch changes are in the same direction
Subtract a fraction of the step size for a weight if batch changes change direction.
Set all initial step sizes alike or specify one step size per layer.
Momentum of change in weights
Update weights after each trial or at the end of each epoch
Set random initial weights to ± maximum magnitude

Number of epochs (may be fractional) before step size reduction:
Added Increment for Adapted Step Size: 0.01
Scaled Decrement for Adapted Step Size: 0.1
Step size list:
Weight Decay: 0
Weight Change Momentum: 0.6
Error Tolerance: 0.01
Maximum Magnitude of Initial Weights: 0.1
Weight update mode:

matically when a data base is chosen. Any other entries are the number of nodes in each hidden layer. There is a constant bias node in each layer of the network. It is not included in the list for the network structure. A gradient descent algorithm needs a step size, which is a multiplier applied to the gradient when the weights are updated. The main MLP parameter window is shown in Figure 3.1. Other LNKnet parameters are set on three additional parameter windows. These are displayed by selecting the three buttons on the main MLP window. These other options do not normally need to be changed and are included primarily for pedagogical purposes.

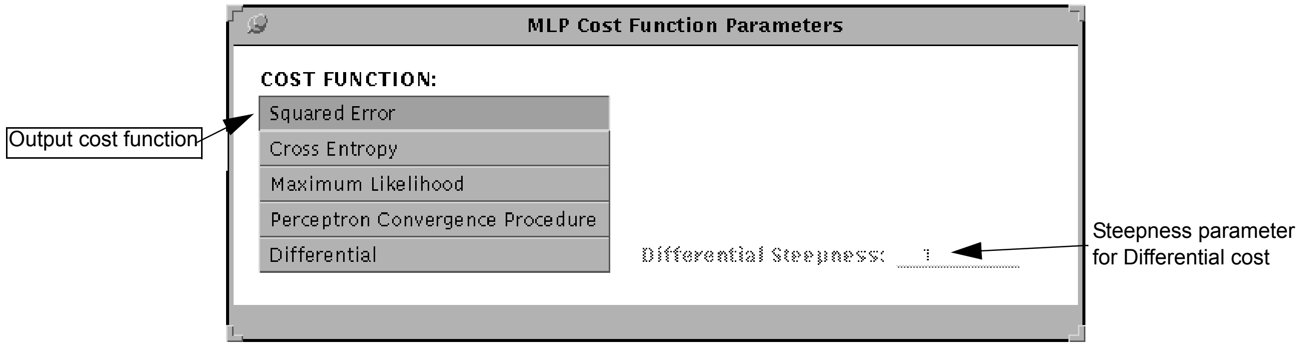
Parameters associated with training the weights are found on the MLP Weight parameters window shown in Figure 3.2. For most problems, the default settings for these parameters are appropriate. In our MLP classifier, there are three options for changing the step size during training. The step size for all weights can be held constant throughout each training run, the step size for all weights can be automatically reduced after a set number of training epochs, or the step size of each weight can be adapted automatically. The step size change type selection must be coordinated with the weight update mode, as described in the paragraph below. The initial step sizes can be the same for all the weights in the network or a different initial step size can be set for each layer. In the first case the initial step size is the one on the main MLP window. In the second, step sizes for each layer are taken from the step size list on the MLP weight parameter window. Using this list, you can initialize the input weights of a network and then prevent training of those weights by setting their step size to zero. There is a momentum term which often reduces training time by moving weights in the direction of previous changes. The weights can be systematically reduced by setting a weight decay parameter. This has the effect of pruning small weights. All weights are multiplied by one minus the decay parameter on every trial. This is equivalent to adding a penalty term to the cost function that penalizes large weights. There is a tolerance parameter in the error, which turns off back-propagation if the output is within the tolerance limit of the desired output. Finally, the magnitude of the random initial weights can be set.

Weight updates can be performed after each trial or in batches at the end of each epoch. Fastest training typically is obtained by updating weights every trial after each training pattern is presented. To automatically reduce the step sizes for all weights after a set number of epochs of training, weight updates must be performed after each trial. If a batch update is being used, it is possible to automatically set a step size for each network weight using the multiple adaptive step size algorithm. When the total correction for a weight in one batch is in the same direction as in the previous batch, the step size for that weight is increased. If the direction changes, the step size is reduced by a set factor. Another factor in the speed of weight training is the order in which training patterns are presented. Remember to randomize the order of the patterns when using the MLP classifier. The random order flag is set on the main LNKnet window.

Several different versions of back propagation are available in this version of the MLP classifier. Most differ in the cost function used to determine the error of the outputs. The squared-error, maximum likelihood and cross-entropy cost functions are described in [36]. Cross-entropy and maximum likelihood cost functions sometimes provide better posterior probability estimates than a squared error cost function. The top-two difference cost function has been called the classification figure of merit by Hampshire [10]. It attempts to minimize the number of errors on training data and can be used with all networks. It should normally be used with linear output nodes. A steepness of 1 uses a maximally sharp sigmoid with the difference term and a steepness of 0 uses a maximally smooth sigmoid. The perceptron convergence procedure, which is an implementation of Rosenblatt's original single layer perceptron, differs from the other cost functions. It trains a single plane for each class which separates that class from all others. All of the patterns on one side of the plane for a class are considered to be in one class and all patterns on the other side are in the other class. The perceptron convergence procedure can only be used when there are no hidden layers. This procedure is normally only defined for two-class problems. In the LNKnet implementation, if there are more than two classes, multiple perceptrons are trained simultaneously to discriminate each class from the others. The classification decision is made by determining the perceptron with the highest unclipped output. The MLP Cost Function parameter window is shown in Figure 3.3.

When a squared error or top two difference cost function is being used, there are three choices of output function. This output function is applied to the weighted sum calculated for the output layer. The output functions are a standard sigmoid, which goes from 0 to 1 with an output of 0.5 for an input of 0, a symmetric sigmoid which goes from -1 to 1, and a linear output, which simply gives the weighted sums as the final outputs of the network. The hidden node sigmoid functions can be either standard or symmetric. There is a steepness parameter for these node functions. This steepness parameter can be the same for all nodes in the network or it can be set for each layer. A higher steepness

FIGURE 3.3 MLP Cost Parameters


MLP Cost Function Parameters

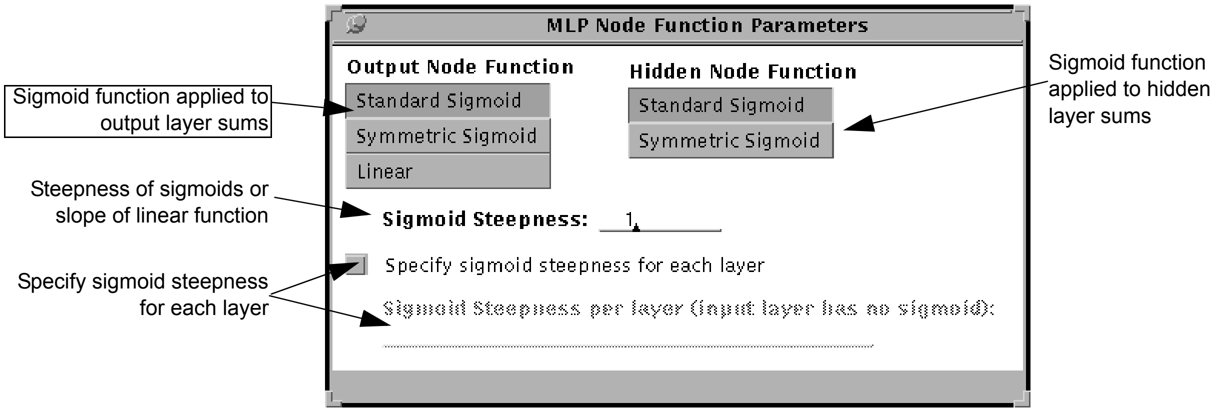
COST FUNCTION:

- Squared Error
- Cross Entropy
- Maximum Likelihood
- Perceptron Convergence Procedure
- Differential

Differential Steepness: 1

value for the first hidden layer can sharpen the decision region boundaries for an MLP classifier that has been initialized using bintree2mlp, which is explained in Chapter 7. The MLP node function parameter window is shown in Figure 3.4.

FIGURE 3.4 MLP Node Parameters


MLP Node Function Parameters

Output Node Function: Standard Sigmoid, Symmetric Sigmoid, Linear

Hidden Node Function: Standard Sigmoid, Symmetric Sigmoid

Sigmoid Steepness: 1

Specify sigmoid steepness for each layer

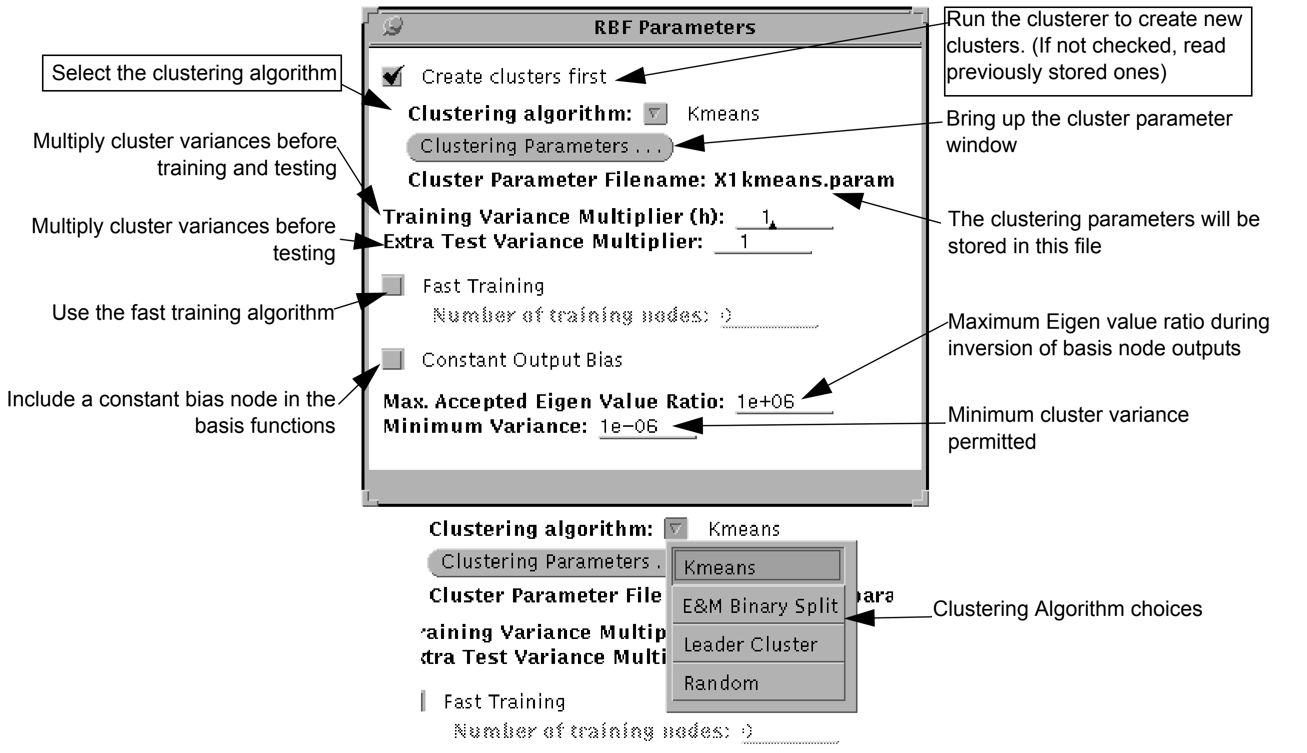
Sigmoid Steepness per layer (input layer has no sigmoid):

Sigmoid function applied to output layer sums
Steepness of sigmoids or slope of linear function
Specify sigmoid steepness for each layer
Sigmoid function applied to hidden layer sums

3.1.2 Radial Basis Function (RBF)

Radial Basis Function classifiers[28] calculate discriminant functions using local Gaussian functions instead of sigmoids of hidden node sums. They may perform better than MLP classifiers if input features are normalized so a Euclidean distance is meaningful and if class distributions exhibit radial symmetries. Network outputs are weighted sums of the outputs of Gaussian hidden nodes or basis functions. Hidden node outputs are normalized to sum to one. Weights are trained using least-squares matrix inversion to minimize the squared error of the output sums given the basis function outputs for the training patterns. These basis functions can include a constant bias node. The variances given by the clustering algorithm can be increased during training if they are too small to provide good coverage of the data. The variances can be further increased during testing. The variances used by RBF hidden nodes are diagonal (one variance per input dimension for each basis function). For problems that require a very large number of hidden nodes (>200), training time can be reduced using the fast train option. In fast training, rather than update each of the Nouputs*Nnodes connection weights for each pattern, only the weights connecting the hidden nodes with the highest outputs are updated. If enough hidden nodes are used training each pattern, the classification results are equivalent for RBF classifiers using fast training. Fast training is not normally required and should not be used.

FIGURE 3.5 RBF Parameters

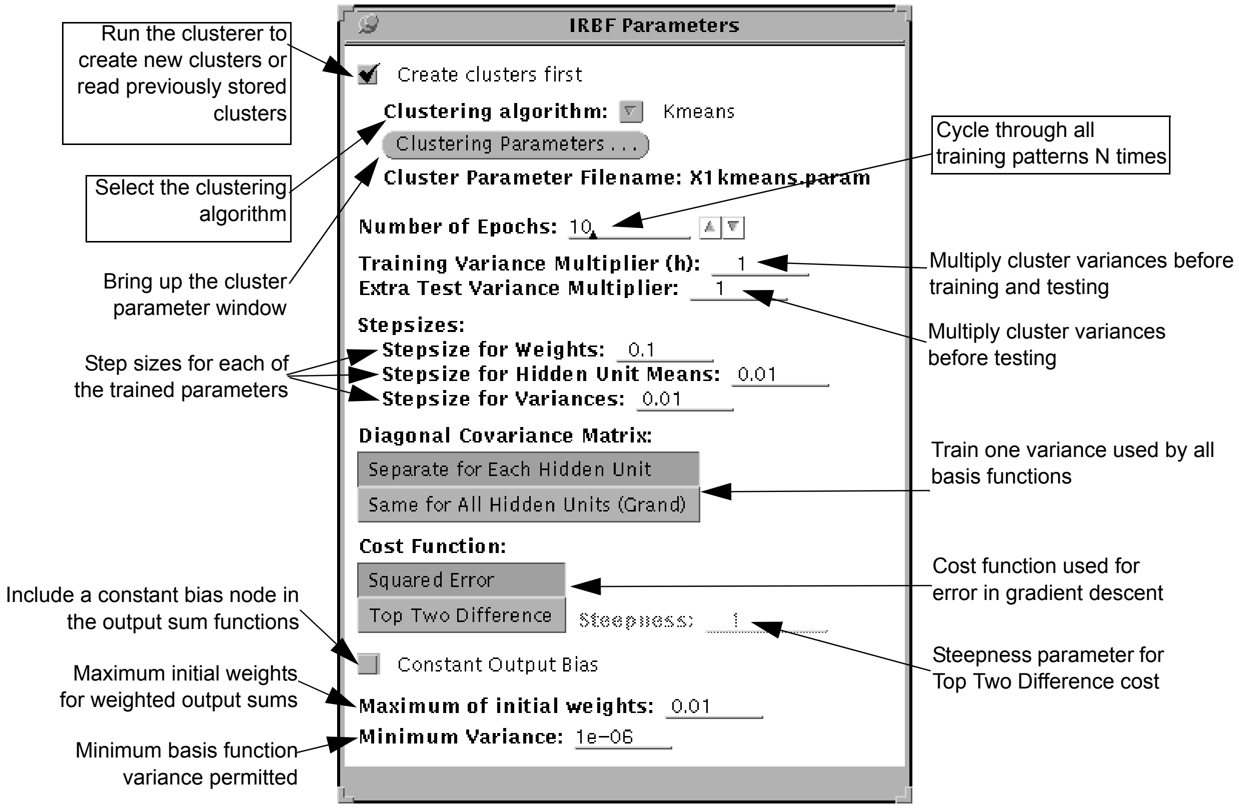

RBF Parameters
Select the clustering algorithm
Multiply cluster variances before training and testing
Multiply cluster variances before testing
Use the fast training algorithm
Include a constant bias node in the basis functions
Run the clusterer to create new clusters. (If not checked, read previously stored ones)
Bring up the cluster parameter window
The clustering parameters will be stored in this file
Maximum Eigen value ratio during inversion of basis node outputs
Minimum cluster variance permitted
Clustering Parameters . . .
Kmeans
E&M Binary Split
Leader Cluster
Random
Clustering Algorithm choices

[✓] Create clusters first
Clustering algorithm: Kmeans
Cluster Parameter Filename: X1kmeans.param
Training Variance Multiplier (h): 1
Extra Test Variance Multiplier: 1
[ ] Fast Training
Number of training nodes: 0
[ ] Constant Output Bias
Max. Accepted Eigen Value Ratio: 1e+06
Minimum Variance: 1e-06

3.1.3 Incremental Radial Basis Function (IRBF)

The RBF classifier is limited because hidden node means and variances are fixed during training. The Incremental Radial Basis Function classifier (IRBF) [28] can sometimes provide better performance by training these parameters. In testing, the IRBF Classifier is identical to the RBF Classifier. In training, the means and variances of the Gaussian basis functions are trained in addition to the weights. All of the parameters are trained using gradient descent which tries to minimize the squared error in the final outputs. Each of the three variables being trained, the weights, means, and variances, has its own step size. There is one other difference between the LNKnet RBF and IRBF classifiers. In the IRBF classifier, the variances in each dimension can be averaged, as they are in the Gaussian classifier using grand variances.

FIGURE 3.6 IRBF Parameters


Run the clusterer to create new clusters or read previously stored clusters
Select the clustering algorithm
Bring up the cluster parameter window
Step sizes for each of the trained parameters
Include a constant bias node in the output sum functions
Maximum initial weights for weighted output sums
Minimum basis function variance permitted
Cycle through all training patterns N times
Multiply cluster variances before training and testing
Multiply cluster variances before testing
Train one variance used by all basis functions
Cost function used for error in gradient descent
Steepness parameter for Top Two Difference cost
Clustering Parameters . . .
Separate for Each Hidden Unit
Same for All Hidden Units (Grand)
Squared Error
Top Two Difference

IRBF Parameters

[✓] Create clusters first

Clustering algorithm: Kmeans

Cluster Parameter Filename: X1kmeans.param

Number of Epochs: 10

Training Variance Multiplier (h): 1

Extra Test Variance Multiplier: 1

Stepsizes:

Stepsize for Weights: 0.1

Stepsize for Hidden Unit Means: 0.01

Stepsize for Variances: 0.01

Diagonal Covariance Matrix:

Cost Function:

Steepness: 1

[ ] Constant Output Bias

Maximum of initial weights: 0.01

Minimum Variance: 1e-06

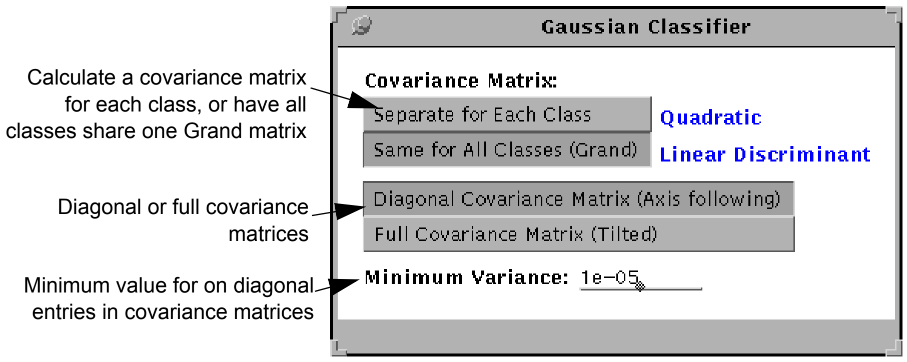
3.2 Likelihood Classifiers

Likelihood classifiers estimate a scaled probability density function or likelihood for each class, $p(X|A)P(A)$ where A again represents a class label, X is the input feature vector for a pattern, $p(X|A)$ is the likelihood of the input data for class A, and $P(A)$ is the prior probability for class A. For a given test pattern, the class which has the highest likelihood times the class prior probability is selected as the class of the pattern. Because the output values are continuous, they can be used for further analysis of sequences of input patterns. For example, Gaussian mixtures are widely used in speech recognizers as low-level probability estimators in hidden Markov models.

3.2.1 Gaussian (GAUSS)

Gaussian classifiers [7] and especially linear discriminant classifiers are the most common and simplest classifiers. They should always be tried first on new problems. A Gaussian classifier models each class with a Gaussian distribution centered on the mean of that class. There are four choices in the calculation of the variances for these Gaussians. First, the variance of each class can be found or those class variances can be averaged to give a single grand variance used for all classes. Second, the variance calculated can be diagonal, one variance for each input dimension, or full covariance matrices can be calculated. When there are many input features, full-covariance Gaussian classifiers have many more parameters than diagonal-covariance classifiers and may perform worse with limited training data. In addition the variance can be limited to be above a minimum value to prevent numerical problems when input features are unchanged across training patterns. A **linear discriminant classifier** is a Gaussian classifier with grand variances, where variances are the same for all classes. The simplest linear discriminant classifier uses the same diagonal covariance matrix for each class. A **quadratic classifier** is a Gaussian classifier with separate variances for each class.

FIGURE 3.7 Gaussian Classifier Parameters


Calculate a covariance matrix for each class, or have all classes share one Grand matrix
Diagonal or full covariance matrices
Minimum value for on diagonal entries in covariance matrices
Gaussian Classifier
Separate for Each Class
Same for All Classes (Grand)
Quadratic
Linear Discriminant
Diagonal Covariance Matrix (Axis following)
Full Covariance Matrix (Tilted)

Covariance Matrix:

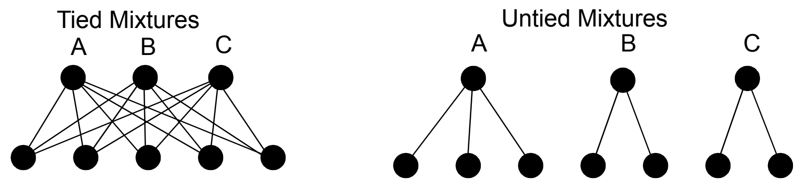
Minimum Variance: 1e-05

3.2.2 Gaussian Mixture (GMIX)

The Gaussian Mixture classifier [28] can perform better than a Gaussian classifier when classifier distributions are not unimodal Gaussian. It models each class distribution with one or more Gaussian mixture components. The outputs of the classifier are weighted sums of the outputs of Gaussian mixture components. In training, the classifier changes the Gaussian means and variances and the connection weights for the outputs using the Estimate-Maximize algorithm to maximize the likelihood of the training patterns. For problems in high dimensions, a savings in the number of classifier parameters can be gained by switching from a full covariance Gaussian classifier to a diagonal covariance Gaussian mixture classifier with several Gaussians per class.

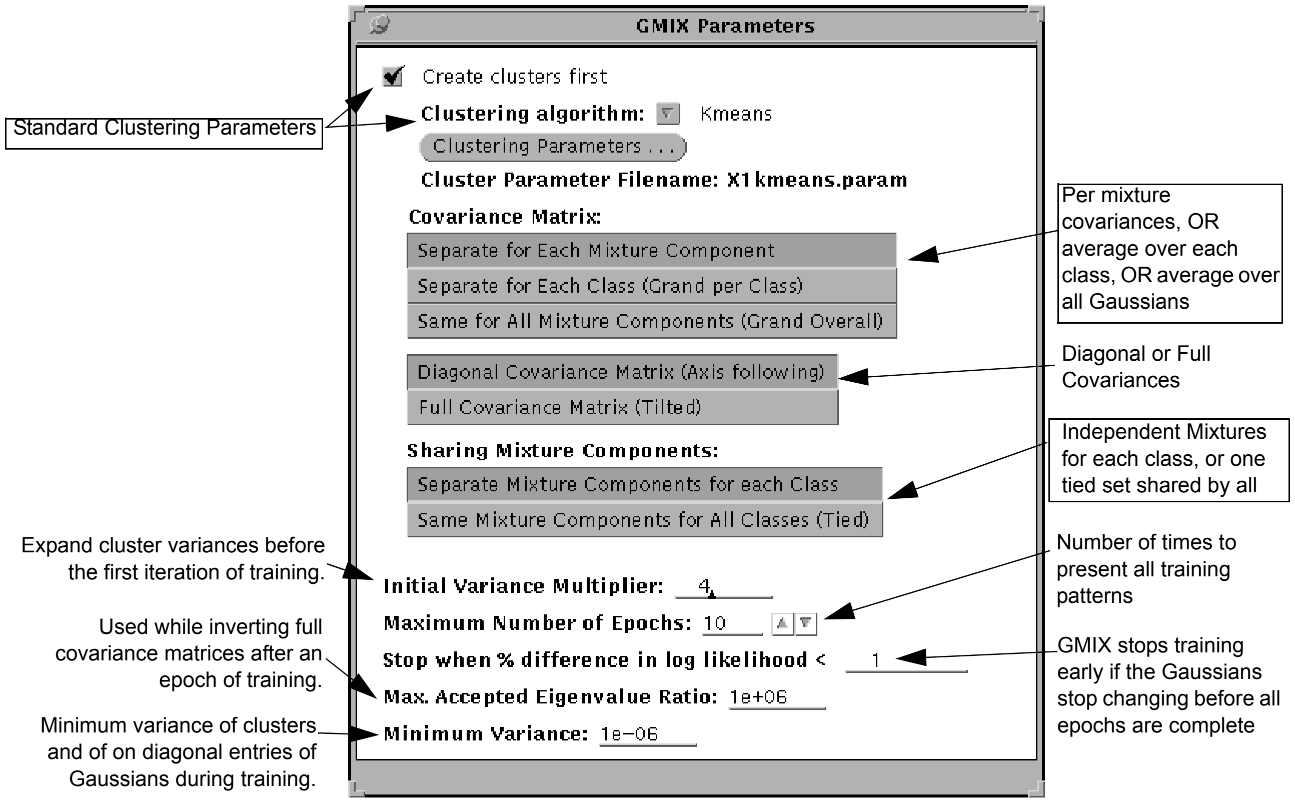
Many of the options available in the Gaussian Mixture program deal with the type of Gaussians to be used. The first choice is whether each class has its own Gaussian mix-

FIGURE 3.8 Tied versus Untied Gaussian Mixtures



ture or if all of the classes share a single set of tied Gaussian mixtures. Figure 3.8 illustrates the two types of Gaussian Mixtures. The lower dots in this figure represent the Gaussian components and the upper dots represent outputs for each class. As with the Gaussian classifier, the Gaussians in a mixture can have either diagonal or full covariance matrices. Similarly, there can be a separate variance for each Gaussian in the classifier model or the variances can be averaged giving a grand variance. The averaging can be done over all of the Gaussians, so only one is estimated, or the Gaussians in each mixture can be average separately, giving one variance per class.

FIGURE 3.9 Gaussian Mixture Parameters


Standard Clustering Parameters
Per mixture covariances, OR average over each class, OR average over all Gaussians
Diagonal or Full Covariances
Independent Mixtures for each class, or one tied set shared by all
Expand cluster variances before the first iteration of training.
Number of times to present all training patterns
Used while inverting full covariance matrices after an epoch of training.
GMIX stops training early if the Gaussians stop changing before all epochs are complete
Minimum variance of clusters and of on diagonal entries of Gaussians during training.

GMIX Parameters

✓ Create clusters first

Clustering algorithm: Kmeans

Clustering Parameters . . .

Cluster Parameter Filename: X1kmeans.param

Covariance Matrix:

- [✓ Separate for Each Mixture Component]
- Separate for Each Class (Grand per Class)
- Same for All Mixture Components (Grand Overall)

- [✓ Diagonal Covariance Matrix (Axis following)]
- Full Covariance Matrix (Tilted)

Sharing Mixture Components:

- [✓ Separate Mixture Components for each Class]
- Same Mixture Components for All Classes (Tied)

Initial Variance Multiplier: 4

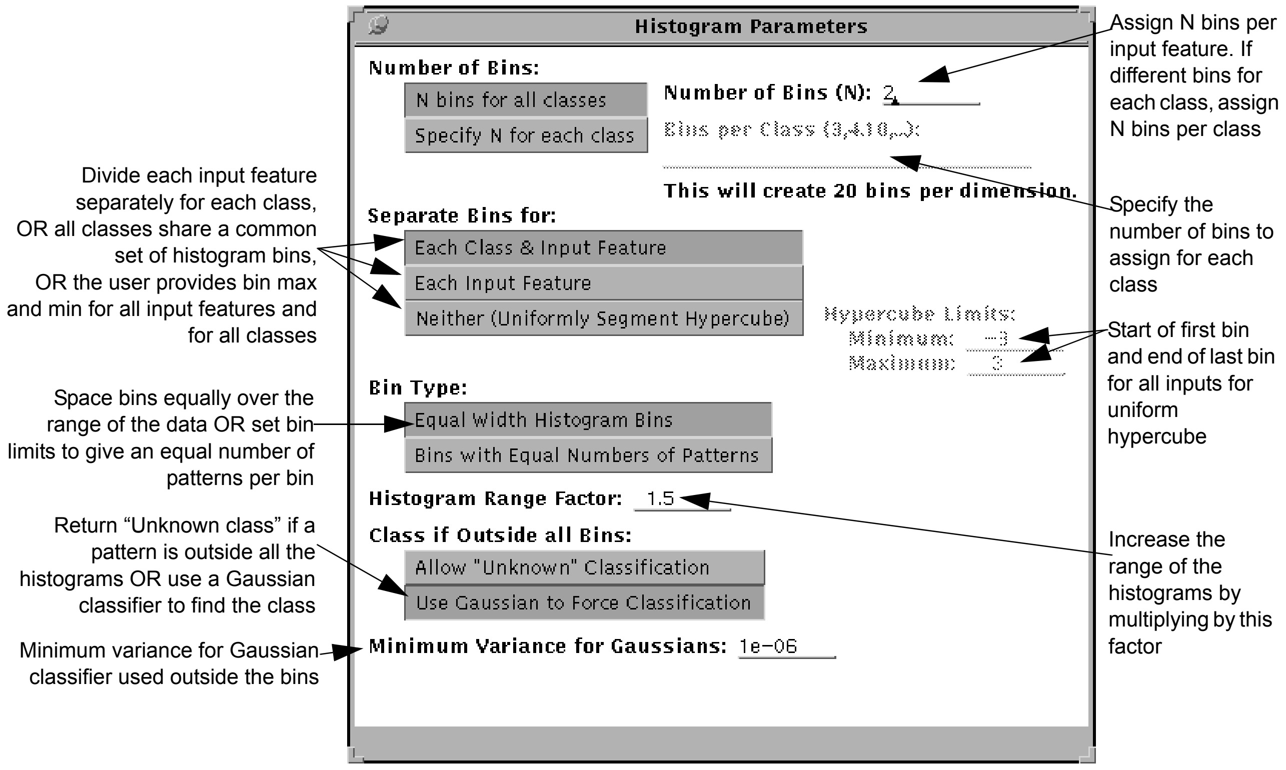
Maximum Number of Epochs: 10

Stop when % difference in log likelihood < 1

Max. Accepted Eigenvalue Ratio: 1e+06

Minimum Variance: 1e-06

FIGURE 3.10 Histogram Parameters



Histogram Parameters

Number of Bins:

- N bins for all classes
- Specify N for each class

Number of Bins (N): 2

Bins per Class (3,4,10,..):

This will create 20 bins per dimension.

Separate Bins for:

- Each Class & Input Feature
- Each Input Feature
- Neither (Uniformly Segment Hypercube)

Hypercube Limits:
Minimum: -3
Maximum: 3

Bin Type:

- Equal Width Histogram Bins
- Bins with Equal Numbers of Patterns

Histogram Range Factor: 1.5

Class if Outside all Bins:

- Allow "Unknown" Classification
- Use Gaussian to Force Classification

Minimum Variance for Gaussians: 1e-06

Assign N bins per input feature. If different bins for each class, assign N bins per class

Specify the number of bins to assign for each class

Divide each input feature separately for each class, OR all classes share a common set of histogram bins, OR the user provides bin max and min for all input features and for all classes

Start of first bin and end of last bin for all inputs for uniform hypercube

Space bins equally over the range of the data OR set bin limits to give an equal number of patterns per bin

Increase the range of the histograms by multiplying by this factor

Return “Unknown class” if a pattern is outside all the histograms OR use a Gaussian classifier to find the class

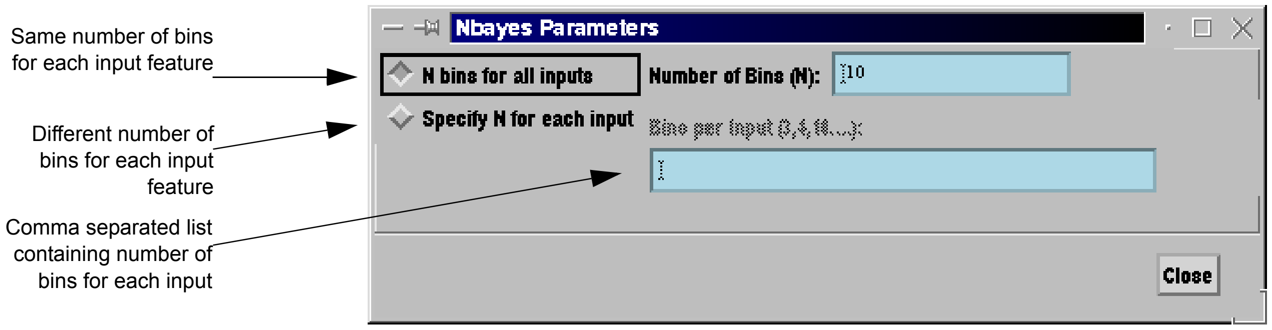
Minimum variance for Gaussian classifier used outside the bins

3.2.3 Histogram

A histogram classifier[7] estimates the likelihood of each class by creating a set of histograms for each input feature. Input features are continuous-values and each input feature is divided into a number of bins. The likelihood assigned to each bin is proportional to the number of training patterns that fall in that bin divided by the bin width. In testing, the likelihoods for each input dimension are multiplied to give an overall likelihood for each class. An optional per class diagonal Gaussian classifier can be used to determine the class of all patterns that fall outside histogram bins. Unlike the naive Bayes classifier, the histogram classifier is designed for continuous valued inputs and provides many alternative approaches to categorize continuous data by forming bins.

The LNKnet histogram classifier provides several options for dividing the input space into bins. A fixed set of bins can be defined which evenly divides the space into smaller hypercubes **(uniformly segmented hypercube)**. This works best when all the input features have been normalized to have the same ranges. The bins can be autoscaled by calculating one set for each input feature **(Separate bins for each input feature)**. This allows for more variability across input dimensions. Finally, separate bins can be found for each class. This allows the greatest flexibility in the histogram parameters, binning each class and input feature. The range covered by the histogram can be multiplied by

FIGURE 3.11 Naive Bayes Classifier Parameters


Same number of bins for each input feature
Different number of bins for each input feature
Comma separated list containing number of bins for each input
Nbayes Parameters
Close

[✓ N bins for all inputs] Number of Bins (N): 10
Specify N for each input Bins per input (3,4,10,...):

the **histogram range factor** to classify test patterns found near the edges of the range seen during training. Two methods are used for finding the edges of histogram bins. In the first, the bins uniformly segment the covered range. This is usually better for classification. In the second method, the bins segment the space to give uniform numbers of patterns in each bin. Where the data is denser the bins are thinner. This is usually better for likelihood estimation.

3.2.4 Naive Bayes Classifier.

Unlike the histogram classifier, the naive Bayes classifier is explicitly designed for categorical data. It has become a popular classifier for processing large amounts of data typical of "data mining" applications and is not necessarily naive or simple. A straightforward approach is used, but good performance, that rivals that of more complex classifiers, is often provided. The LNKnet graphical interface, shown in Figure 3.11, makes it possible to change the number of bins or values for each input feature. This can be the same for all input features or it can be specified for each feature. Other parameters, described below, can be edited by hand in the shell script produced by LNKnet.

This classifier is designed for use with only categorical features and the categories must be indicated by input features that take on integers ranging from zero to *nvalues*-1, where *nvalues* is the number of different values for the input feature. Categorical features take on values that are not ordered in a meaningful way. An example would be an input feature used to classify Internet web servers that was the name of the web server host computer operating system. If there are 12 different types of operating systems, then this input feature would take on 12 values. For use with LNKnet, the operating system input feature values must range from 0 to 11. Note that input features must be preprocessed to take on these integer values and they should not be further normalized within LNKnet. For example, "simple normalization" as assigned in the LNKnet "Feature Normalization" window should not be used. This will change input feature values to be non-integers that do not range from 0 to *nvalues*-1. Likewise, other forms of normalization should not be used.

Every implementation of naive Bayes classifiers must address three subtle issues. The first is how to assign probabilities to bins containing values not seen in any training patterns. For example, if a feature can take values from 0 to 11, but the value 3 is never seen during training, a non-zero probability must be assigned to the value 3 seen during testing. The Laplace correction is used in this program because it often works well [19]. A less common variant can be selected by adding the -unity_laplace flag for the nbayes_lnk command in the shell script that LNKnet produces. The second issue is how to assign bin probabilities for categorical features when training patterns take on values that are outside the expected range. For example if the number of bins for a feature is set to 12, then feature values should range from 0 to 11. Other input feature values such 12 or 21 are outside this range. This program creates an extra "unseen" bin for any feature where this occurs. All patterns that fall outside the expected range are counted as falling in this bin. These patterns can be ignored by adding the -ignore_unseen flag to the nbayes_lnk command in the shell script that LNKnet produces. The third issue is how to treat features in testing that take on values that are outside the expected range. This program ignores such features. If there are many features, and at least one takes on an expected value, but all others take on unexpected values, then classification is still possible and will be based on the one feature. If all features take on unexpected values, then no class will be selected and no classification decision will be made.

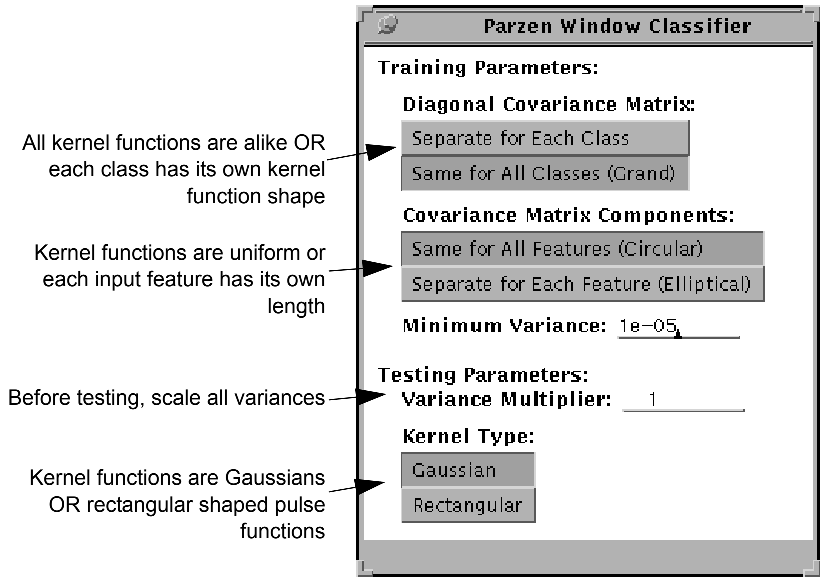
3.2.5 Parzen Window

For a Parzen window classifier [7,39], kernel functions are placed over each training pattern. Kernel functions can be Gaussians or rectangular pulse functions. Kernel functions can be uniform, that is circular or square functions, or the length of each side can be proportional to the variance of each input feature, that is elliptical or rectangular. All kernel functions can have the same shape or there can be separate kernel function shapes for each class. The class likelihood of an input pattern is the sum of the likelihoods for each kernel function in the class normalized by the number of training patterns in the class. The Parzen window classifier can map very complicated likelihood functions with little training. The variance of all kernel functions is initially set equal to the variance of the training data. This variance can be reduced or increased using the variance multiplier.

3.3 Nearest Neighbor Classifiers

Nearest Neighbor classifiers work on the principle that a pattern is probably of the same class as those patterns nearest to it. The simplest algorithm is to store all the training patterns and to find distances to them all for each testing pattern. The computation necessary for testing can be prohibitive for large databases. Most enhancements to the algorithm involve reducing the number of patterns stored and used for testing. Nearest neighbor classifiers are simple and easily understood, but do not produce continuous outputs for later analysis and do not generalize well where training and test data differ.

FIGURE 3.12 Parzen Window Parameters



Parzen Window Classifier

Training Parameters:

Diagonal Covariance Matrix:

Separate for Each Class

Same for All Classes (Grand)

Covariance Matrix Components:

Same for All Features (Circular)

Separate for Each Feature (Elliptical)

Minimum Variance: 1e-05

Testing Parameters:

Variance Multiplier: 1

Kernel Type:

Gaussian

Rectangular

All kernel functions are alike OR each class has its own kernel function shape

Kernel functions are uniform or each input feature has its own length

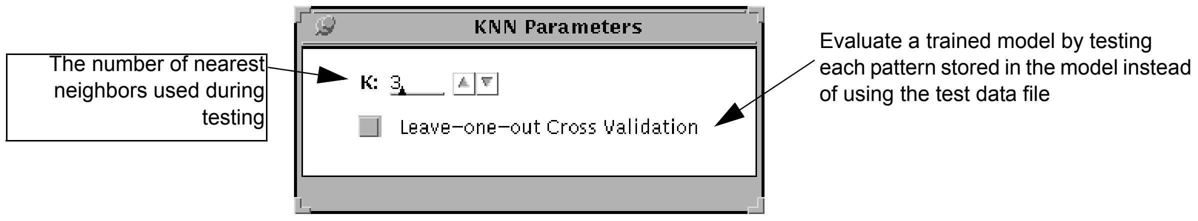
Before testing, scale all variances

Kernel functions are Gaussians OR rectangular shaped pulse functions

3.3.1 K Nearest Neighbor (KNN)

A K-Nearest Neighbor classifier [7] can be used to obtain a rough estimate of the difficulty of a new problem. It can form complex decision regions but stores all training data and must compute distances to all training patterns during testing. A K-Nearest Neighbor classifier trains by storing all training patterns presented to it. During testing, the K stored patterns closest to the test pattern are found using a Euclidean distance measure. A vote is taken amongst the K neighbors and the class that occurs the most is assigned to the test pattern. In leave-one-out cross validation, the stored training patterns are tested one at a time against a KNN model containing all but the single test pattern.

FIGURE 3.13 K Nearest Neighbor Parameters



KNN Parameters

K: 3

Leave-one-out Cross Validation

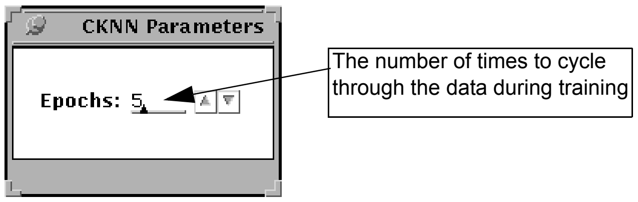
The number of nearest neighbors used during testing

Evaluate a trained model by testing each pattern stored in the model instead of using the test data file

3.3.2 Condensed Nearest Neighbor (CKNN)

The Condensed K Nearest Neighbor classifier (CKNN) [7,28] can sometimes provide performance that is similar to that of a K Nearest Neighbor classifier, but with fewer stored patterns. In testing, the CKNN classifier is a nearest neighbor classifier. It assigns the class of the nearest stored pattern to the test pattern. To train, the Condensed Nearest Neighbor classifier examines the training patterns successively and stores any that, when tested, are assigned the wrong class.

FIGURE 3.14 Condensed Nearest Neighbor Parameters

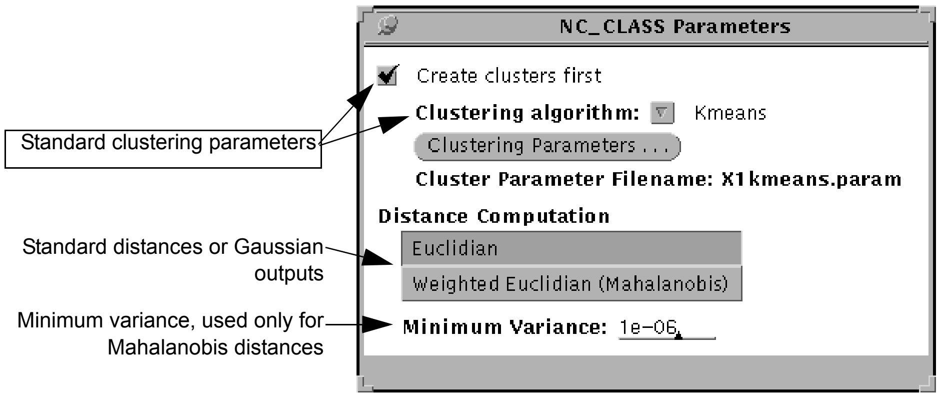

CKNN Parameters
The number of times to cycle through the data during training

Epochs: 5

3.3.3 Nearest Cluster (NC_CLASS)

The Nearest Cluster classifier [7] can sometimes provide error rates similar to a KNN classifier but with many fewer stored parameters. It is a nearest neighbor classifier which uses the centers of clusters as its stored patterns. During training, a class is assigned to each cluster center using a nearest neighbor search over the training data. When determining the nearest neighbors in either training or testing, either a Euclidean or Mahalanobis distance can be used. That is, a distance based on the squared difference in the inputs or one based on the output of the Gaussian represented by the cluster. The Mahalanobis distance metric is intended for use when the Estimate-Maximize clustering algorithm, **em_clus**, is used for clustering.

FIGURE 3.15 Nearest Cluster Classifier Parameters

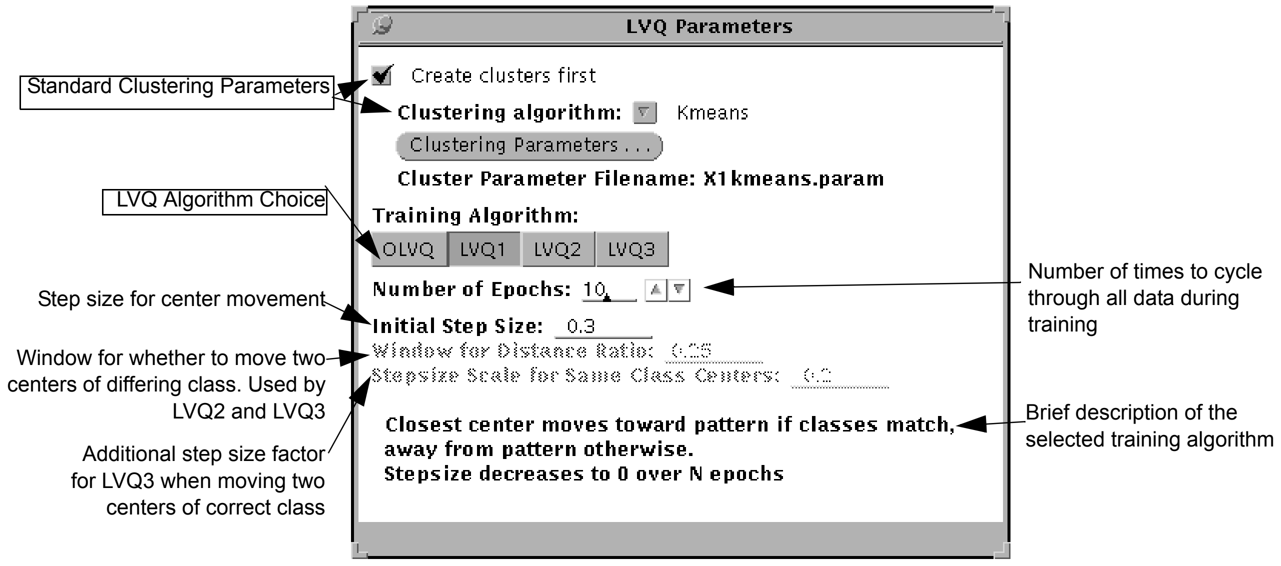

NC_CLASS Parameters
Standard clustering parameters
Clustering Parameters . . .
Distance Computation
Euclidian
Weighted Euclidian (Mahalanobis)
Standard distances or Gaussian outputs
Minimum variance, used only for Mahalanobis distances

✓ Create clusters first
Clustering algorithm: Kmeans
Cluster Parameter Filename: X1kmeans.param
Minimum Variance: 1e-06

3.3.4 Learning Vector Quantizer (LVQ)

The learning vector quantizer (LVQ) training algorithm[21] can sometimes improve the performance of a nearest cluster classifier by moving cluster centers. This is an implementation of four of Kohonen's Learning Vector Quantizer algorithms. In testing, the LVQ is a nearest cluster classifier. In training, cluster centers are assigned classes as they are in the nearest cluster classifier using Euclidean distances. Training then moves centers to improve classification performance on the training data. In LVQ1, the center closest to the training pattern is moved towards or away from it, depending on whether it is of the same class. OLVQ is an optimized version of LVQ1. Each center has its own step size, which is modified when the center is the nearest neighbor. In LVQ2 the two closest centers are moved if one is of the correct class and one is of some other class. Further, the pattern must fall in a window between the two closest centers. Finally, LVQ3 is the same as LVQ2 with one exception. When the two closest centers are both of the correct class, they are both moved closer to the training pattern. The step size for this "correct" movement is smaller than the normal stepsize by a factor of epsilon.

FIGURE 3.16 Learning Vector Quantizer Parameters


LVQ Parameters
Standard Clustering Parameters
LVQ Algorithm Choice
Step size for center movement
Window for whether to move two centers of differing class. Used by LVQ2 and LVQ3
Additional step size factor for LVQ3 when moving two centers of correct class
Number of times to cycle through all data during training
Brief description of the selected training algorithm
Clustering Parameters . . .
OLVQ
LVQ1
LVQ2
LVQ3

✓ Create clusters first

Clustering algorithm: Kmeans

Cluster Parameter Filename: X1kmeans.param

Training Algorithm: LVQ1

Number of Epochs: 10

Initial Step Size: 0.3

Window for Distance Ratio: 0.25

Stepsize Scale for Same Class Centers: 0.1

Closest center moves toward pattern if classes match,
away from pattern otherwise.
Stepsize decreases to 0 over N epochs

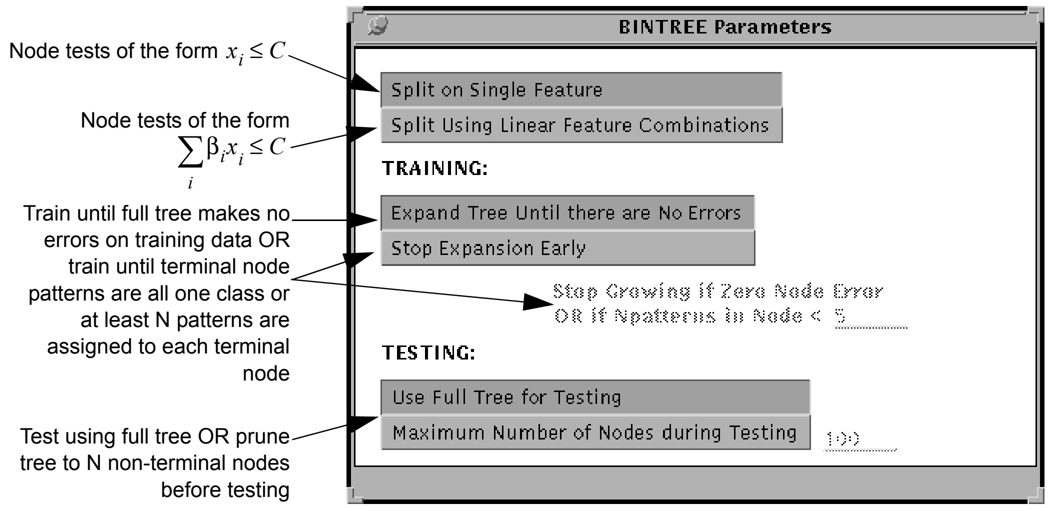
3.4 Rule Based Classifiers

Rule based classifiers partition the input space into decision regions using threshold logic nodes or rules. They can often be easily implemented in hardware applications.

3.4.1 Binary Tree Classifier (BINTREE)

The binary decision tree classifier trains and tests very quickly and is similar to the CART algorithm described in [3]. It can also be used to identify the input features which are most important for classification because feature selection is part of the tree-building process. BINTREE is well suited to problems with categorical input features or with uncorrelated continuous input features. During training, BINTREE builds trees using tests of the form $x_i \le C$ at each node to divide training patterns for classification. Patterns which pass the test are assigned to one node and those which fail are assigned to another. Tests for the two new nodes are found and training continues until there are no nodes that have training patterns from more than one class. Before testing, the tree can be pruned to a set number of non-terminal nodes. This reduces the size of the tree and can improve classification error rates in testing. To prune, the non-terminal node which least affects the error rate on all the training data is found. It is made into a terminal node and its children are removed from the tree. Nodes are cut until the desired number of non-terminal nodes in the tree is reached. The BINTREE parameter window is shown in Figure 3.17. The “Split Using Linear Feature Combinations” option should not be used except for pedagogical purposes because the power of the BINTREE classifier comes from the simpler single-feature splits performed by default.

FIGURE 3.17 Binary Tree Parameters

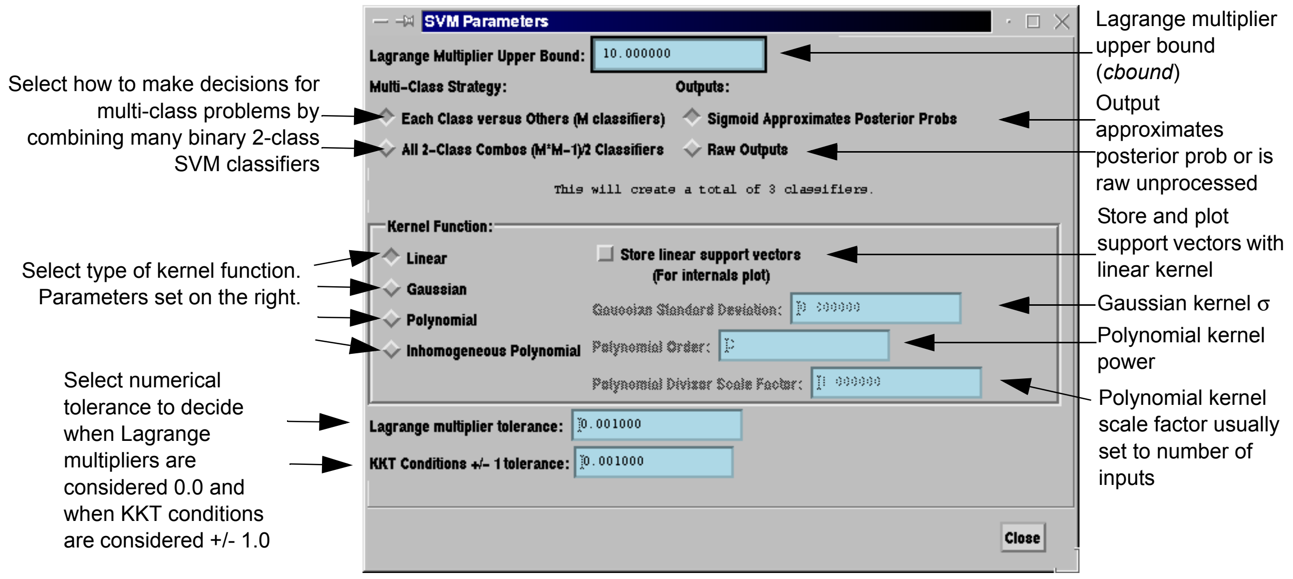

Node tests of the form $x_i \le C$
Node tests of the form $\sum_i \beta_i x_i \le C$
Train until full tree makes no errors on training data OR train until terminal node patterns are all one class or at least N patterns are assigned to each terminal node
Test using full tree OR prune tree to N non-terminal nodes before testing
BINTREE Parameters
Split on Single Feature
Split Using Linear Feature Combinations
TRAINING:
Expand Tree Until there are No Errors
Stop Expansion Early
Stop Growing if Zero Node Error
OR if Npatterns in Node < 5
TESTING:
Use Full Tree for Testing
Maximum Number of Nodes during Testing
100

3.4.2 Support Vector Machine (SVM)

The support vector machine (SVM) is a modern highly flexible classifier [5]. The LNK-net implementation classifies two or more classes using one or more linear or nonlinear two-class support vector machine classifiers. SVM classifiers are similar to perceptrons. They separate patterns in two classes using a hyperplane. SVM’s, however, position the separating hyperplane to maximize the margin, where the margin is the minimum distance from the separating hyperplane to patterns in the two classes. Training involves attempting to satisfy Karush-Kuhn-Tucker (KKT) conditions that specify the quadratic minimization problem that defines the SVM. A linear SVM performs this minimization in the space of the original input features while a nonlinear SVM performs this minimization in an higher-order space implicitly generated using nonlinear kernels [5].

In practice, it is often impossible to satisfy strict KKT conditions and separate all patterns. An upper bound (*cbound*) is used to set the maximum cost incurred for violation of the KKT conditions. This allows the output for support vectors with non-zero Lagrange multipliers to deviate from *+/- 1.0*. Higher *cbound* values lead to more complex classifiers that try to correctly classify each training pattern. Lower *cbound* values lead to simpler classifiers that allow misclassifications and violations of strict KKT conditions. The value of ***cbound*** for a particular problem must be selected empirically using cross-validation. Figure 3.18 shows the SVM LNKnet window. The value for cbound is set using the upper box labeled “Lagrange Multiplier Upper Bound.”

FIGURE 3.18 Support Vector Machine Parameters



SVM Parameters

Lagrange Multiplier Upper Bound: 10.000000

Multi-Class Strategy:
- Each Class versus Others (M classifiers)
- All 2-Class Combos (M*M-1)/2 Classifiers

Outputs:
- Sigmoid Approximates Posterior Probs
- Raw Outputs

This will create a total of 3 classifiers.

Kernel Function:
- Linear
- Gaussian
- Polynomial
- Inhomogeneous Polynomial

Store linear support vectors (For internals plot)

Gaussian Standard Deviation: 1.000000

Polynomial Order: 2

Polynomial Divisor Scale Factor: 1.000000

Lagrange multiplier tolerance: 0.001000

KKT Conditions +/- 1 tolerance: 0.001000

Close

Lagrange multiplier upper bound (*cbound*)

Output approximates posterior prob or is raw unprocessed

Store and plot support vectors with linear kernel

Gaussian kernel σ

Polynomial kernel power

Polynomial kernel scale factor usually set to number of inputs

Select how to make decisions for multi-class problems by combining many binary 2-class SVM classifiers

Select type of kernel function. Parameters set on the right.

Select numerical tolerance to decide when Lagrange multipliers are considered 0.0 and when KKT conditions are considered +/- 1.0

The kernel type determines whether an SVM is a simple linear discriminator or whether it maps the inputs to a higher-order space. Kernel types are selected in the left middle of the SVM window shown in Figure 3.18. It is possible to use linear kernels, Gaussian kernels, polynomial kernels $(xy)^n$, and inhomogeneous polynomial kernels $(xy + 1)^n$. Some kernels have free parameters these are selected on the right middle of the SVM window. The standard deviation has to be selected for the Gaussian kernel and the order has to be selected for the polynomial kernels. In addition, the inner terms in the polynomial and inhomogeneous kernels can be divided by a scale factor before being raised to power. This improves numerical stability if there are many input features. For example you could divide by 256 if there were 256 input features and the data was normalized to a mean of zero and standard deviation of one. This scale factor is entered in the bottom right box shown in Figure 3.18. Kernel locations are normally not stored for linear SVM classifiers because they are not required for classification. To force storage of linear SVM kernels to plot them with the “internals” plot check the box in the bottom middle of the SVM window.

SVM classifiers only discriminate between two classes and extensions are required for multi-class problems. Two approaches can be selected using check boxes in the upper left of the SVM window. The upper check box constructs M component binary classifiers which separate each class from all the remaining classes. During testing, the classification decision corresponds to the class of the component classifier with the highest output (before the clipping nonlinearity). The lower check box constructs many more simpler binary classifiers that separate all possible combinations of classes taken two at a time. This results in M*(M-1)/2 simple classifiers. During testing, the class with the most votes across all binary component classifiers is selected. In the case of ties, outputs (before the clipping nonlinearity) for each class are scanned across all pairwise classifiers that include that class, and the minimum is found. These minimum values are compared to find the class with the highest minimum value. The final classification decision corresponds to that class. The second pairwise approach sometimes provides better performance. Although it requires many more classifiers, they are simpler, and overall training time is often similar across both approaches. For reference, the total number of classifiers that will be created is printed in the upper middle of the SVM window.

Classical SVM classifiers provide zero/one outputs that indicate only whether the input pattern belongs to class A or B. They do not provide posterior probabilities that can be used to adjust differences in prior probabilities between training and testing, assign costs to different types of errors, reject patterns, and form complete ROC curves. LNKnet software approximates posterior probabilities using an approach motivated by [4] but simplified to use only training data. The output of each component SVM (before the clipping nonlinearity) is fed into a sigmoid function with an output ranging from 0 to 1.0 and constrained to produce an output of 0.5 when the input is at the decision region boundary (input = 0.0). This constraint preserves the error rate for component binary classifiers when errors have equal costs. The slope of the sigmoid is selected during training to minimize the mean squared error between the sigmoid output and desired outputs of zero and one for the two classes. Training patterns with unclipped outputs near +/- 1 (mainly support vectors) are weighted much less in this minimization because internal parameters in the classifier have been tuned to produce outputs of +/- 1 for these patterns. For multi-class problems, posterior probabilities are computed from the component classifiers. When M classifiers are generated for an M-class problem, posterior probabilities are the M outputs for each class from the M component classifiers. When M*(M-1)/2 pairwise classifiers are generated, the posterior probability for each class is the minimum posterior probability output for that class across all pairwise classifiers. A sigmoid is always fit to the output of every component classifier. This fit can be used or ignored depending on the -sigmoid_fit flag. This flag should normally be used to provide an output that approximates posterior probabilities.

This implementation of SVM's uses an efficient, fast algorithm that scales well to problems with many features and many training patterns. It uses John Platt's Sequential Minimal Optimization (SMO) algorithm [34] as improved by Keerthi and Shevade [17]. The core algorithm examines pairs of patterns (one from each class) and modifies Lagrange multipliers using an analytic solution when patterns violate KKT conditions. Training involves two-pass sweeps. In the first pass of a sweep, all patterns are examined one at a time to find violations of KKT conditions. Lagrange multipliers are adapted when a violation is found. In the second pass, the subset of patterns found in the first pass that violate KTT conditions are examined repetitively and their Lagrange multipliers are adjusted until such patterns satisfy KKT conditions. Adjustments always involve pairs of patterns that do not satisfy KKT conditions. Another examination of all patterns to find KKT condition violations begins the next sweep. Training stops when all patterns satisfy KKT conditions. During training, two bias values (the bias for the hyperplane) are maintained and used by the algorithm. These high and low bias values initially differ, and then converge to be similar after convergence. After the algorithm completes, a final independent check is made to make sure the solution satisfies all KKT conditions.

A warning is printed along with diagnostics if the solution does not satisfy KKT conditions.

During training, information is printed out during each pass of every sweep when the log file verbosity set in the “Report Files and Verbosity” window shown in Figure 2.3 is greater than the lowest “Overall Error Rate” setting. The following is an example of a table for a component classifier which separates classes 9 and 8 (digits “9” versus “8”) for the ocrdigit data base. A linear kernel was used, cbound was 1.0, and there were 120 training patterns.

TABLE 3.1 Example training sweeps printout for a SVM classifier.

```
NSweeps  Changed  TChanged  KernelEvals  UnBounded  AtUpper  HiBias  LoBias  DeltaBias
1        58       58        3140         53         0        1.73    0.07    1.655375
1        2        60        3357         50         0        1.77    0.29    1.478128
2        64       124       9528         53         0        1.45    0.80    0.644528
2        0        124       9528         53         0        1.45    0.80    0.644528
3        62       186       16061        47         0        1.45    1.07    0.379679
3        18       204       17594        37         0        1.36    1.19    0.175545
4        42       246       20861        40         0        1.36    1.24    0.122306
4        409      655       50742        33         0        1.27    1.27    0.001997
5        0        655       50742        33         0        1.27    1.27    0.001997
```

The first column indicates the sweep number. As noted above, there are two passes per sweep. The first pass examines all training patterns and the second examines only the subset of patterns found in the first pass that violates the KKT conditions. The second column indicates the number of patterns in a pass that violate KKT conditions. For example, on the first pass through all 120 training patterns, 58 patterns violated the KKT conditions. Lagrange multipliers for these patterns are all updated or “changed”. The third column indicates the cumulative number of patterns where Lagrange multipliers were adapted or the total patterns with “changed” Lagrange multipliers. For example, on the first past, 58 adaptations occurred. A total of 655 adaptations were required to complete training. The Fourth column shows the cumulative number of kernel evaluations required during training. When the number of input features is large, most of the computation in this algorithm involves kernel evaluations. For this problem, more than 50,000 kernel evaluations were required to complete training. The fifth column shows the number of support vectors that have non-zero Lagrange multipliers that are below cbound. After training is complete, there are 33 non-zero support vectors below cbound and none at the upper bound. All support vectors (unbounded and at the upper bound) must be stored and used for classification. Support vectors at the upper bound correspond to patterns where outputs are not +/- 1.0. These patterns may or may not be misclassified. The final three columns show the lower bias bound, the upper bias bound, and the difference between these bounds. See [17] for a descriptions of these bounds and how they are computed. After training is complete, the difference between these bias bounds should be small and less than the KKT tolerance.

Any implementation of SVMs must address numerical precision limitations and the desired accuracy of fit to KKT conditions. LNKnet software is designed for input features that have been normalized to have zero mean and unit variance. This is achieved in LNKnet using simple normalization in the “Feature Normalization” window. In addition, the accuracy desired for KKT conditions can be adjusted. KKT conditions specify that the unclipped component classifier output for non-zero support vectors below cbound must be +/- 1. In practice, exactly producing outputs of +/- 1 may take excessively long and have little effect on classification performance. The tolerance (absolute difference between actual and desired outputs) allowed around desired outputs of +/- 1 can be set on the bottom left of the SVM window. This value defaults to 0.001. It can be increased, for example to 0.01, to reduce convergence time. It is also possible to set a lower limit on Lagrange multipliers in the lower left of the SVM window. Lagrange multipliers below this limit are set to zero. This defaults to 0.001. It can be lowered when Lagrange multiplier adjustments are small and below the threshold. Evidence of small Lagrange multiplier adjustments below this limit is that the algorithm converges rapidly to a bad solution that doesn't satisfy KKT conditions without changing Lagrange multipliers on any training patterns. A warning will be printed with recommended changes if this occurs. This tolerance can also be increased if there are too many small Lagrange multipliers.

This algorithm converges (usually rapidly) to a good solution. Good solutions are found for a wide range of parameter values. Extensive error checking is performed to verify the final solution and warnings and corrective suggestions are provided if KKT conditions are not satisfied. This only occurs if numerical precision problems occur. Such problems usually don't occur if (1) The data is normalized to zero mean unit variance using simple normalization, (2) If the Gaussian kernel standard deviation isn't too large compared to the number of input features, (3) The polynomial kernel divisor is roughly equal to the number of input features, and (4) There are no severe outlier data patterns that are far away from other patters of the same class but among patterns of some other class. When KKT conditions can't be satisfied, the algorithm will still converge and warnings will be printed out stating why KKT conditions weren't met and how serious this is. These warnings can sometimes be ignored because when they occur, classifiers are created and they typically work reasonably well. The extent of KKT violation is printed out and small violations of KKT conditions don't affect classification performance significantly. If warnings are printed out and KKT violations are substantial, try a different kernel (e.g. Gaussian or polynomial instead of linear), try increasing cbound, try a different approach to building multi-class classifiers, increase the polynomial divisor scale factor, or decrease the standard deviation of Gaussian kernels. In addition try searching for obvious extreme outlier patterns that might be due to mislabeled data. For high-order polynomial kernels and Gaussian kernels with large standard deviations, lowering the Lagrange multiplier tolerance may help. For multi-class problems, warnings are printed out for each component classifier and the total number of warnings is printed out when training is complete. Search for the string "WARNING" in the training log file. This software has been successfully applied to large problems with many input features and many training patterns. Memory requirements increase roughly linearly in the number of input features and number of training patterns.

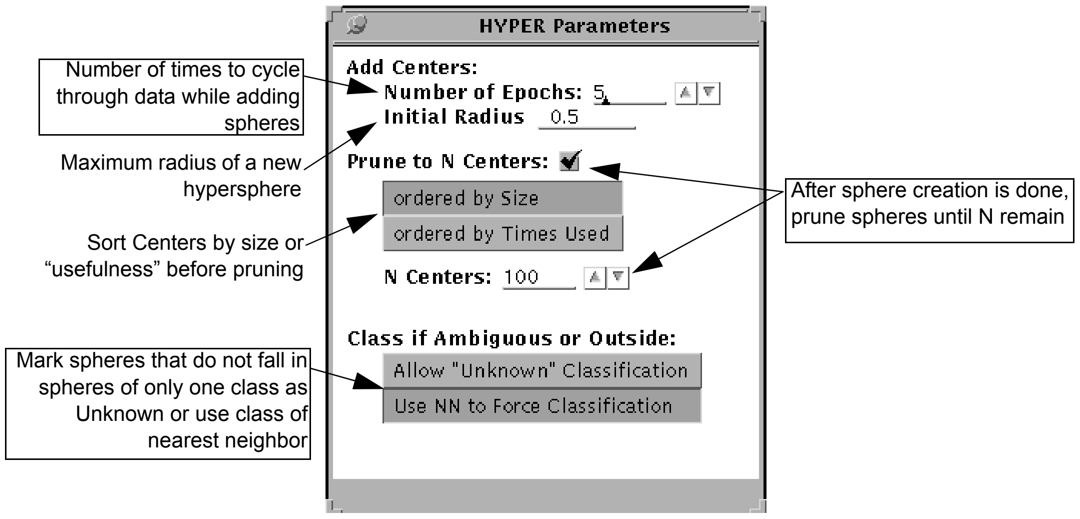
3.4.3 Hypersphere (HYPER)

The hypersphere classifier [1,21] forms decision regions using hyperspheres and can require far fewer hyperspheres than there are training patterns. It covers the input space with hyperspheres using a predefined initial radius. A new sphere is added whenever a classification error is made on a training pattern. The radii of overlapping spheres are reduced when a new sphere is added such that no sphere covers the center of a sphere of another class.

At the end of training, spheres can be pruned. During pruning, the spheres are sorted either by size or by the number of times each sphere was used in a correct classification of a training pattern. Then, the spheres are pruned until a certain number remains.

Classification is performed by first testing to see whether a test pattern falls inside a sphere. The “rules” used by this classifier are then distance tests to determine which sphere or spheres the pattern falls in. If a pattern falls outside all spheres, the “unknown” class is assigned to it. An “unknown” response class can be prevented by responding with the class label of the nearest hypersphere center. An answer of “unknown” can also be returned if the pattern falls inside two or more spheres of differing classes. In this case, a nearest neighbor search can be performed over the centers of the spheres the pattern falls in.

FIGURE 3.19 Hypersphere Classifier Parameters


Number of times to cycle through data while adding spheres
Maximum radius of a new hypersphere
Sort Centers by size or “usefulness” before pruning
Mark spheres that do not fall in spheres of only one class as Unknown or use class of nearest neighbor
After sphere creation is done, prune spheres until N remain
ordered by Size
ordered by Times Used
Allow "Unknown" Classification
Use NN to Force Classification

HYPER Parameters

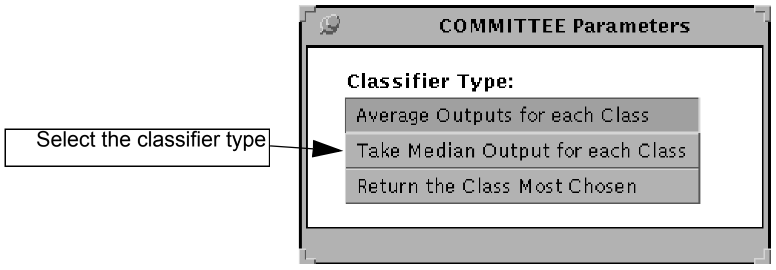
Add Centers:
Number of Epochs: 5
Initial Radius 0.5
Prune to N Centers: ✓
N Centers: 100
Class if Ambiguous or Outside:

3.5 Committee Classifier

One single classifier used alone often does not provide the best performance. In many cases, better performance is provided by a committee made up of many different classifiers either of different types or trained using different samplings of the same training data [14]. A committee classifier combines the outputs of several trained classifiers to return a final combined classification decision. LNKnet makes it possible to evaluate committee classifiers using a three step process. First all classifiers are trained and tested independently as described above. Second, a committee database is created using the "Committee Data Base Generation..." button on the main LNKnet window. This takes the outputs from all classifiers trained independently, concatenates them, and creates a new committee data base where these outputs can be used as inputs to another classifier. Finally, the committee classifier or any other classifier can be used to combine the outputs to make a final decision. The committee classifier combines other classifier outputs by averaging, forming the median output, or taking a majority vote. If average or median outputs are used, it is important that all the members of the committee be of the same type. That is they should all estimate posterior probabilities or they should estimate likelihoods. Because the nearest neighbor classifiers do not produce continuous outputs, only the third classifier type, voting for the most chosen class, is appropriate for them. Note that decision region and profile plots can not be produce with committee classifiers because it is not easy to determine the output for all possible inputs.

The input data base for a LNKnet committee classifier must be a committee data base which contains the outputs for all the classifiers in the committee. Section 7.3 describes how committee data bases are created. It is up to users who are testing committee classifiers to make sure the same training data, features, and output classes are used with all classifiers that are committee members. These characteristics of committee members are not verified when the committee classifier is run.

FIGURE 3.20 Committee Classifier Parameters


COMMITTEE Parameters
Classifier Type:
Average Outputs for each Class
Take Median Output for each Class
Return the Class Most Chosen
Select the classifier type

CHAPTER 4 Clustering

Several of the LNKnet classifiers initialize hidden nodes or other parameters using pre-trained clusters. Each cluster has a mean and a diagonal covariance matrix. The clusters can be trained on labeled or unlabeled training data. That is, a separate set of clusters can be trained for each class, or a single set of clusters can be trained for all of the training data. When clustering labeled data, a different number of clusters can be generated for each class.

4.1 K-Means

The K-Means clustering algorithm [7] positions a set of K centers in order to minimize the total squared error distance between each training pattern and its nearest center. It is trained using multiple passes through all training patterns. During a single training epoch each training pattern is assigned to its nearest center. The position of that center is then moved to the mean of the patterns assigned to it.

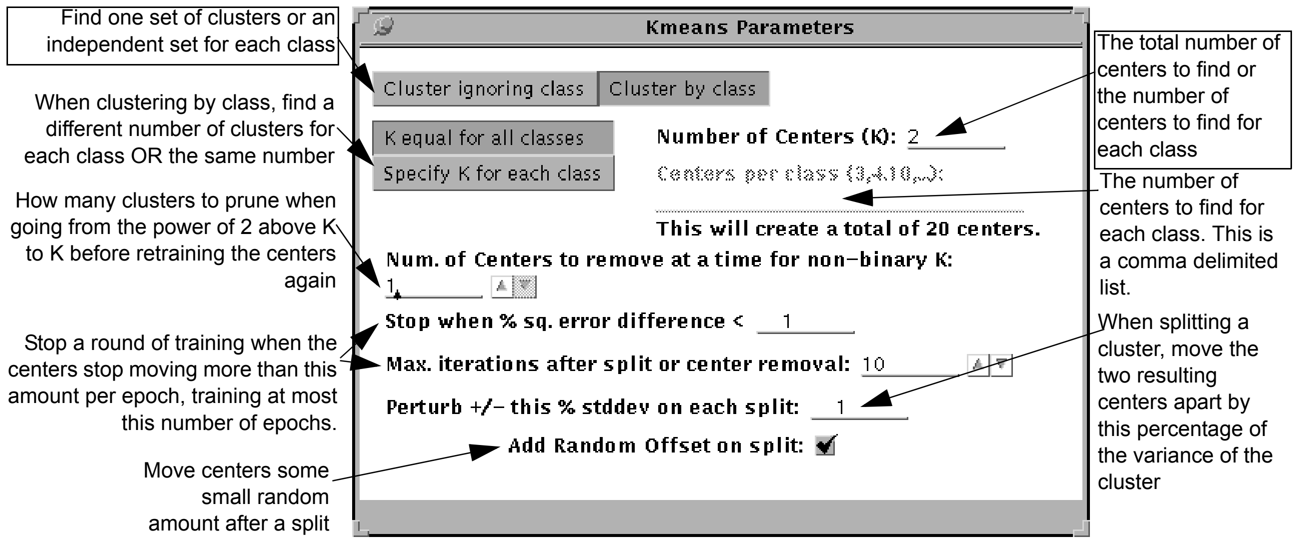
In this implementation, the K centers are initialized using a binary splitting algorithm first described in [4]. The program first places a single center at the mean of all of the training data. This center is then split in two, with the resulting centers being moved slightly away from the original center's position. These centers are then trained for a set number of epochs or until the total error goes below a threshold. The algorithm then splits the existing centers and proceeds as before. If, during training, a center ever has no patterns assigned to it, that center is moved near the center which accounts for the largest amount of the total error and training proceeds as before. When a non-binary number of centers is requested, the algorithm finds $2^{\lceil \log 2(K) \rceil}$ centers, the power of two above the requested number of centers, K. This set of centers is then pruned to bring the number back down to K. Pruning eliminates first those clusters which account for the least total variance.

4.2 Estimate-Maximize (EM_CLUS)

EM_CLUS uses the Estimate-Maximize algorithm (EM) [28] to maximize the likelihood of the training patterns while training the means, variances, and mixture weights of Gaussian mixture cluster centers. The algorithm is the same one as is used by the Gaussian Mixture Classifier, **GMIX**, except that mixture weights are ignored.

This implementation uses binary splitting to generate the requested number of clusters. When finding a non-binary number of centers, this algorithm goes to the power of two

FIGURE 4.1 K Means Parameters



Find one set of clusters or an independent set for each class

When clustering by class, find a different number of clusters for each class OR the same number

How many clusters to prune when going from the power of 2 above K to K before retraining the centers again

Stop a round of training when the centers stop moving more than this amount per epoch, training at most this number of epochs.

Move centers some small random amount after a split

The total number of centers to find or the number of centers to find for each class

The number of centers to find for each class. This is a comma delimited list.

When splitting a cluster, move the two resulting centers apart by this percentage of the variance of the cluster

Kmeans Parameters

Cluster ignoring class | Cluster by class

K equal for all classes | Specify K for each class

Number of Centers (K): 2

Centers per class (3,4,10,..):

This will create a total of 20 centers.

Num. of Centers to remove at a time for non-binary K: 1

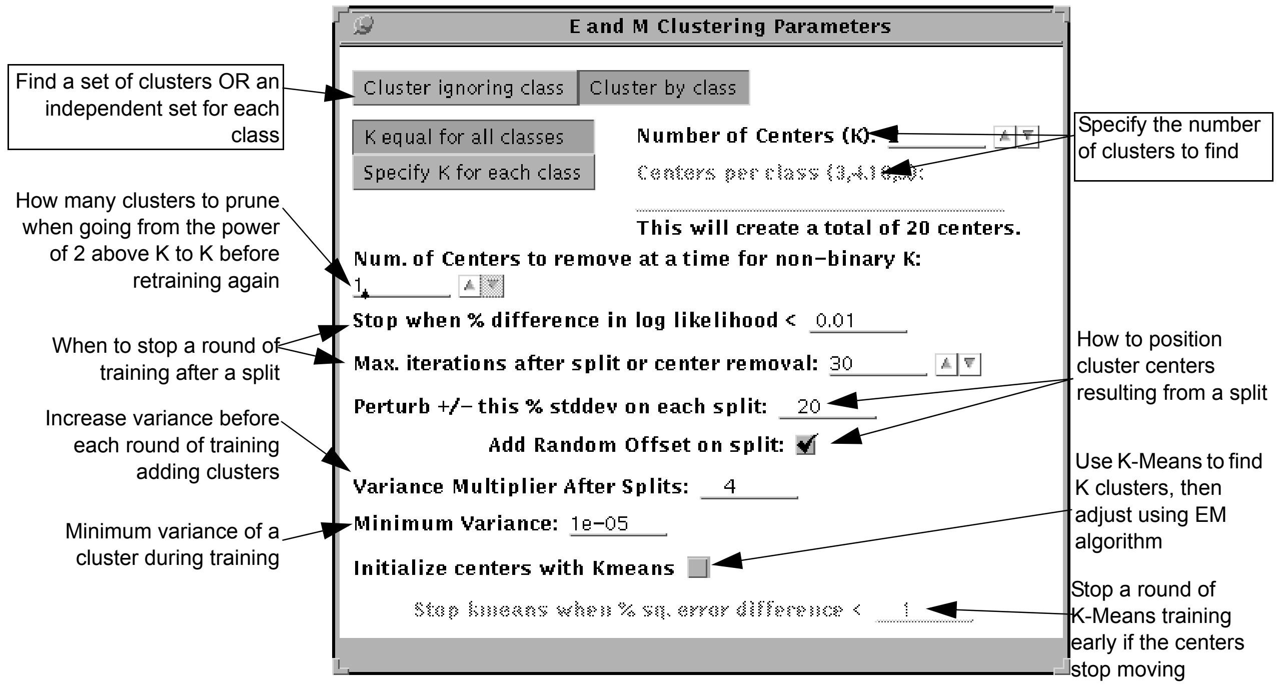
Stop when % sq. error difference < 1

Max. iterations after split or center removal: 10

Perturb +/- this % stddev on each split: 1

Add Random Offset on split: ✓

FIGURE 4.2 Estimate-Maximize Likelihood Parameters



Find a set of clusters OR an independent set for each class

How many clusters to prune when going from the power of 2 above K to K before retraining again

When to stop a round of training after a split

Increase variance before each round of training adding clusters

Minimum variance of a cluster during training

Specify the number of clusters to find

How to position cluster centers resulting from a split

Use K-Means to find K clusters, then adjust using EM algorithm

Stop a round of K-Means training early if the centers stop moving

E and M Clustering Parameters

Cluster ignoring class | Cluster by class

K equal for all classes | Specify K for each class

Number of Centers (K): 1

Centers per class (3,4,10,..):

This will create a total of 20 centers.

Num. of Centers to remove at a time for non-binary K: 1

Stop when % difference in log likelihood < 0.01

Max. iterations after split or center removal: 30

Perturb +/- this % stddev on each split: 20

Add Random Offset on split: ✓

Variance Multiplier After Splits: 4

Minimum Variance: 1e-05

Initialize centers with Kmeans ☐

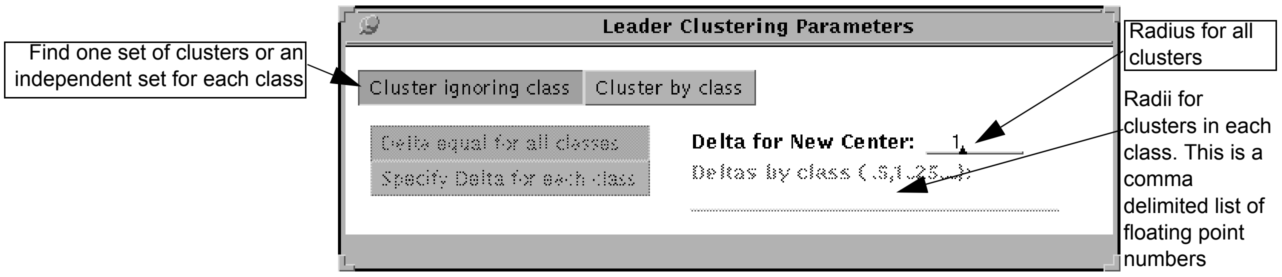
Stop kmeans when % sq. error difference < 1

above K and then prunes, just as KMEANS does. This program can also use **KMEANS** to initialize the clusters. In this case, the EM algorithm is only used at the end to adjust clusters found by **kmeans**.

4.3 Leader Clustering (LEAD_CLUS)

Leader clustering [12] is a simple fast sequential clustering algorithm. Training patterns are presented one at a time. The first pattern is the first cluster center. Any other pattern that is farther away than delta from an existing cluster center is stored as a new cluster center. Larger values of delta result in few clusters while small values of delta result in many clusters. When clustering by class, a different delta may be set for each class. When these clusters are used by a classification algorithm, delta is used as the cluster variance in all directions.

FIGURE 4.3 Leader Clustering Parameters

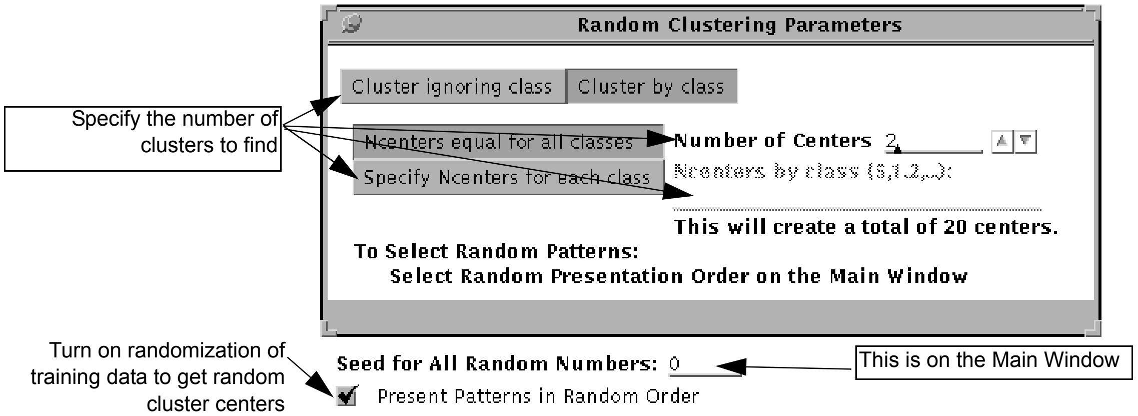

Find one set of clusters or an independent set for each class
Radius for all clusters
Radii for clusters in each class. This is a comma delimited list of floating point numbers
Leader Clustering Parameters
Cluster ignoring class
Cluster by class
Delta equal for all classes
Specify Delta for each class

Delta for New Center: 1
Deltas by class (.5,1.25,..):

4.4 Random (RAN_CLUS)

This clusterer selects K training patterns to use as the cluster centers. These centers will be the first K patterns presented to this clusterer. To get a random set of centers, the data must be presented in a random order by clicking the **Present Patterns in Random Order** box on the main LNKnet window. After centers have been selected, cluster variances are calculated. As with kmeans, each training pattern is assigned to the cluster center closest to it. The cluster is then assigned the variance of its patterns.

FIGURE 4.4 Random Clustering Parameters


Specify the number of clusters to find
Turn on randomization of training data to get random cluster centers
This is on the Main Window
Random Clustering Parameters
Cluster ignoring class
Cluster by class
Ncenters equal for all classes
Specify Ncenters for each class

Number of Centers 2
Ncenters by class (5,1,2,..):
This will create a total of 20 centers.
To Select Random Patterns:
Select Random Presentation Order on the Main Window
Seed for All Random Numbers: 0
✓ Present Patterns in Random Order

CHAPTER 5 General LNKnet Parameters

The design of LNKnet separates those parameters which are algorithm specific from those which are general across most classification algorithms. This chapter discusses those general LNKnet features which are available to most classification programs.

5.1 LNKnet Main Window

The left side of the main LNKnet window, shown in Figure 5.1, is a control panel which runs classification and clustering experiments. The right side, shown in Figure 5.2, sets the classification algorithm and experiment name, and displays the other LNKnet parameter windows using the many buttons whose names end in "...".

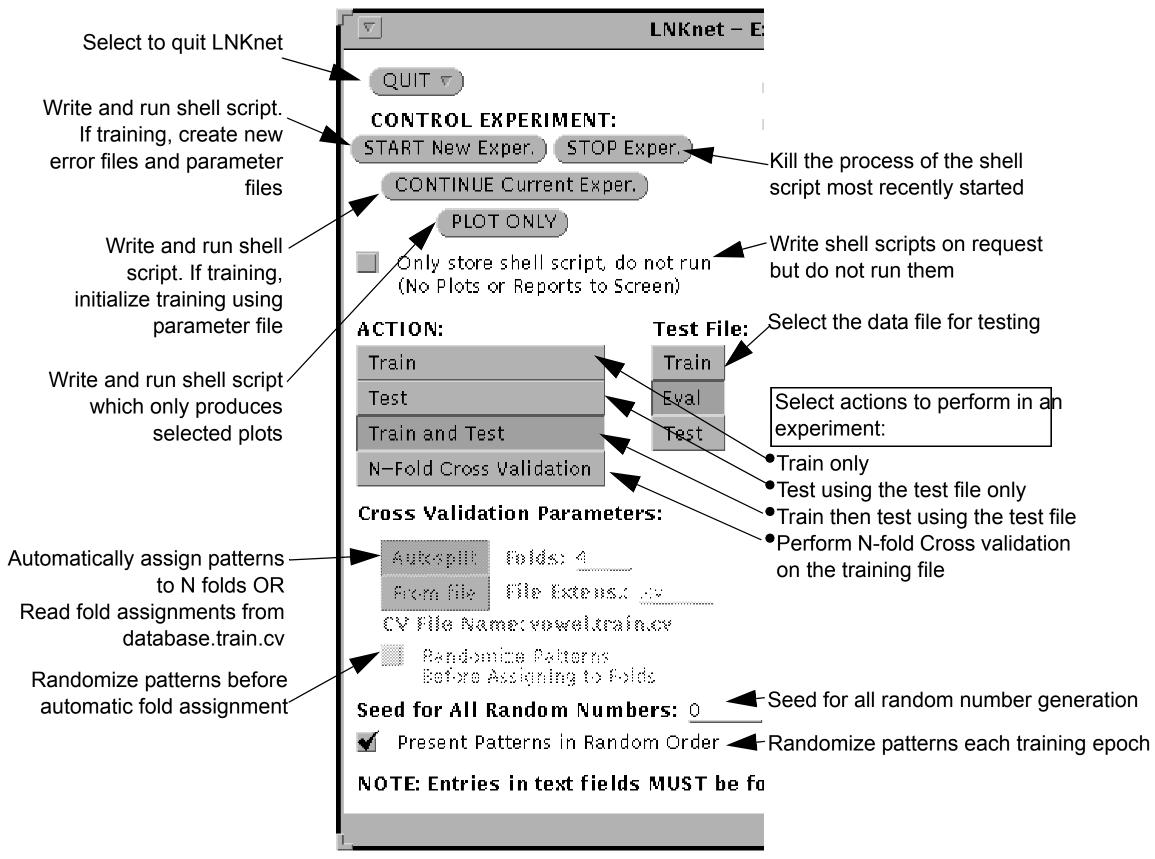
At the top of the left side of the main window is the **QUIT** menu. When Quit is selected from this menu, LNKnet quits. The experiments started by LNKnet still continue, however. These experiments are started by selecting the **START New Exper.** and **CONTINUE Current Exper.** buttons below the **CONTROL EXPERIMENT** label. Plots are usually generated as part of an experiment when START or CONTINUE are selected. To generate plots without repeating an experiment, you can select **PLOT ONLY** below the CONTINUE button. The most recently started experiment or plot can be stopped by selecting the **STOP Exper.** button to the right of the START button. Below these control buttons is a check box labeled **Only Store shell script, do not run**. As stated, when this box is checked, shell scripts and screen files are stored when START, CONTINUE, or PLOT are selected but the scripts are not automatically run by LNKnet. When a shell script stored this way is run, the plot files will be generated but not displayed and text output will go to the log file but not to any shell window. This also affects shell scripts created on other windows such as the **Normalization File Generation** window or the **Committee Data Base Generation** window.

Below the control area is a selection list for choosing the experiment action. The action can be training a classifier on a training file, testing a trained classifier on a testing file, doing training immediately followed by testing, or performing an N-fold cross validation experiment on the training file. The data file to use as the testing file is selected from the list to the right of the action selection list. The user can test a classifier on training, evaluation, or test data. The files themselves are specified on the **Data Base Selection** window shown in Figure 5.4.

When the experiment action is N-fold cross validation, cross validation parameters can be set below the action list. The user chooses whether to automatically divide the data into training and testing folds or to read those fold assignments from a file. The format of that file is described in Section 5.7 on page 89. The patterns can be randomized before assignment to training and testing folds. Selecting **Randomize patterns before assigning to folds** and changing the random number **seed** lets the user perform a series of cross validation experiments to find an average classification error rate on the data.

Finally, at the bottom of the left side of the main window, the user can request to present training patterns to classifiers in a **random order**. The user can also set the random

FIGURE 5.1 The LNKnet Main Window (left side)


Select to quit LNKnet
Write and run shell script. If training, create new error files and parameter files
Write and run shell script. If training, initialize training using parameter file
Write and run shell script which only produces selected plots
Automatically assign patterns to N folds OR Read fold assignments from database.train.cv
Randomize patterns before automatic fold assignment
LNKnet – E
QUIT
CONTROL EXPERIMENT:
START New Exper.
STOP Exper.
CONTINUE Current Exper.
PLOT ONLY
Only store shell script, do not run (No Plots or Reports to Screen)
ACTION:
Train
Test
Train and Test
N-Fold Cross Validation
Test File:
Train
Eval
Test
Cross Validation Parameters:
Autosplit
From file
Randomize Patterns Before Assigning to Folds
Present Patterns in Random Order
NOTE: Entries in text fields MUST be fo
Kill the process of the shell script most recently started
Write shell scripts on request but do not run them
Select the data file for testing
Select actions to perform in an experiment:
•Train only
•Test using the test file only
•Train then test using the test file
•Perform N-fold Cross validation on the training file
Seed for all random number generation
Randomize patterns each training epoch

Folds: 4
File Extens.: .cv
CV File Name: vowel.train.cv
Seed for All Random Numbers: 0

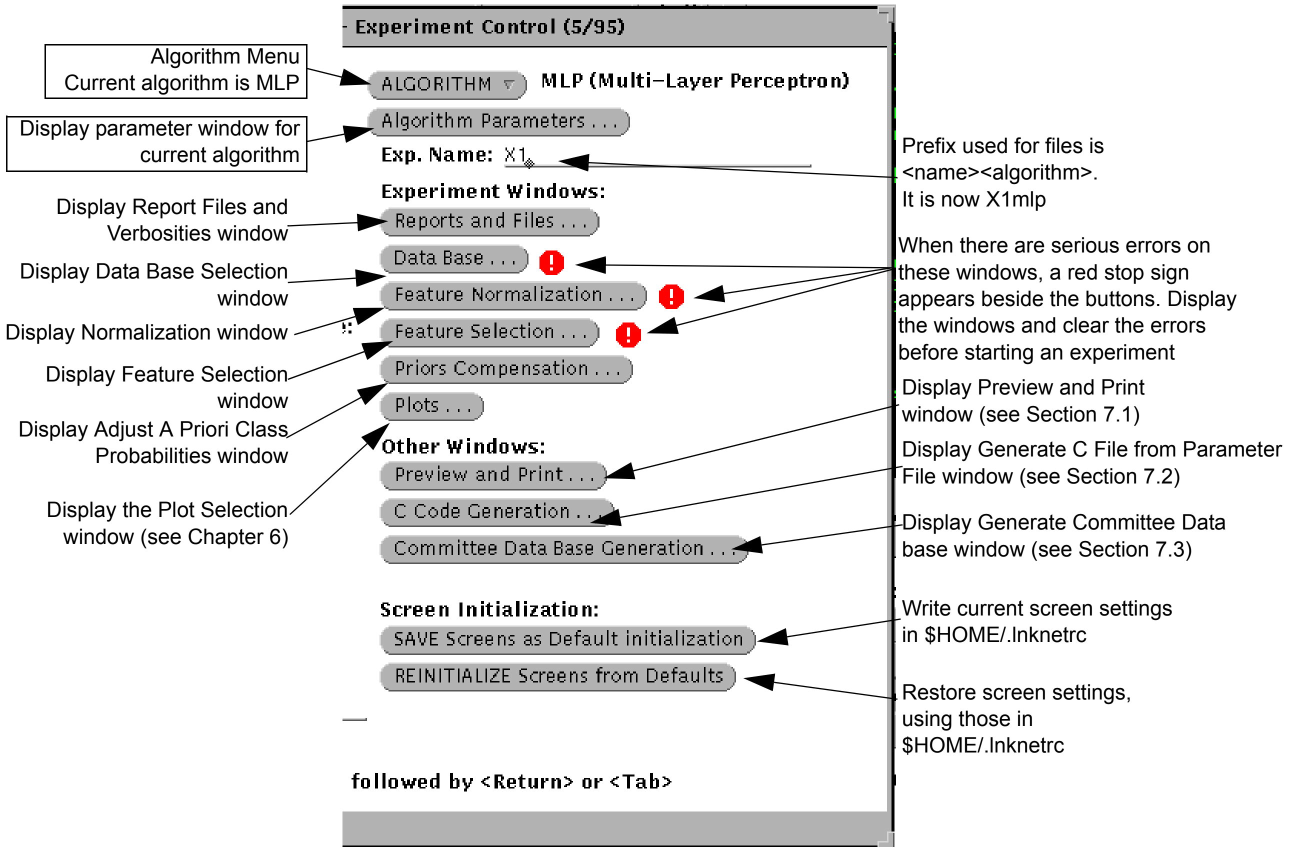
number generation **seed**. Changing this seed changes the values for random initial weights in some classifiers and the presentation order of randomized training patterns for all classifiers.

The first button on the right side of the main window has a menu which selects the classification or clustering algorithm to use in the current experiment. The current algorithm is displayed beside the menu button. Below the menu is a button which displays the parameter window for the current algorithm. This window sets parameters specific to the classification or clustering algorithm. The algorithm parameter windows are described in Chapter 3 and Chapter 4.

Below the algorithm parameters button is a text field for setting the **Experiment name prefix**. The experiment name is used for naming the files generated during an experiment. These include the shell script, screen file, log file, error files, plot files, and classifier parameter files. The prefix set here is added to the algorithm name to create the full experiment name. For the window in Figure 5.2 the full experiment name is X1mlp.

Next on the right side of the main window is a column of buttons which display other LNKnet popup windows. These windows are described in this chapter and in following chapters. The first six are typically accessed in an experiment in the order they appear on this window from top to bottom. The next three buttons display windows for performing further processing after an experiment has finished running. The last two buttons are for saving and restoring screen settings in a defaults file. This file, ~/.lnknetrc, is read when LNKnet is started. A new set of defaults can be created by selecting **Save Screens as Default Initialization**. The screens can be reinitialized to the settings in the current defaults file by selecting **Reinitialize screens from defaults**.

FIGURE 5.2 The LNKnet Main window (right side)


Experiment Control (5/95)
ALGORITHM
MLP (Multi-Layer Perceptron)
Algorithm Parameters . . .
Exp. Name: X1
Experiment Windows:
Reports and Files . . .
Data Base . . .
Feature Normalization . . .
Feature Selection . . .
Priors Compensation . . .
Plots . . .
Other Windows:
Preview and Print . . .
C Code Generation . . .
Committee Data Base Generation . . .
Screen Initialization:
SAVE Screens as Default initialization
REINITIALIZE Screens from Defaults
followed by <Return> or <Tab>
Algorithm Menu
Current algorithm is MLP
Display parameter window for current algorithm
Display Report Files and Verbosities window
Display Data Base Selection window
Display Normalization window
Display Feature Selection window
Display Adjust A Priori Class Probabilities window
Display the Plot Selection window (see Chapter 6)
Prefix used for files is <name><algorithm>. It is now X1mlp
When there are serious errors on these windows, a red stop sign appears beside the buttons. Display the windows and clear the errors before starting an experiment
Display Preview and Print window (see Section 7.1)
Display Generate C File from Parameter File window (see Section 7.2)
Display Generate Committee Data base window (see Section 7.3)
Write current screen settings in $HOME/.lnknetrc
Restore screen settings, using those in $HOME/.lnknetrc

5.2 Experiment Directory Files

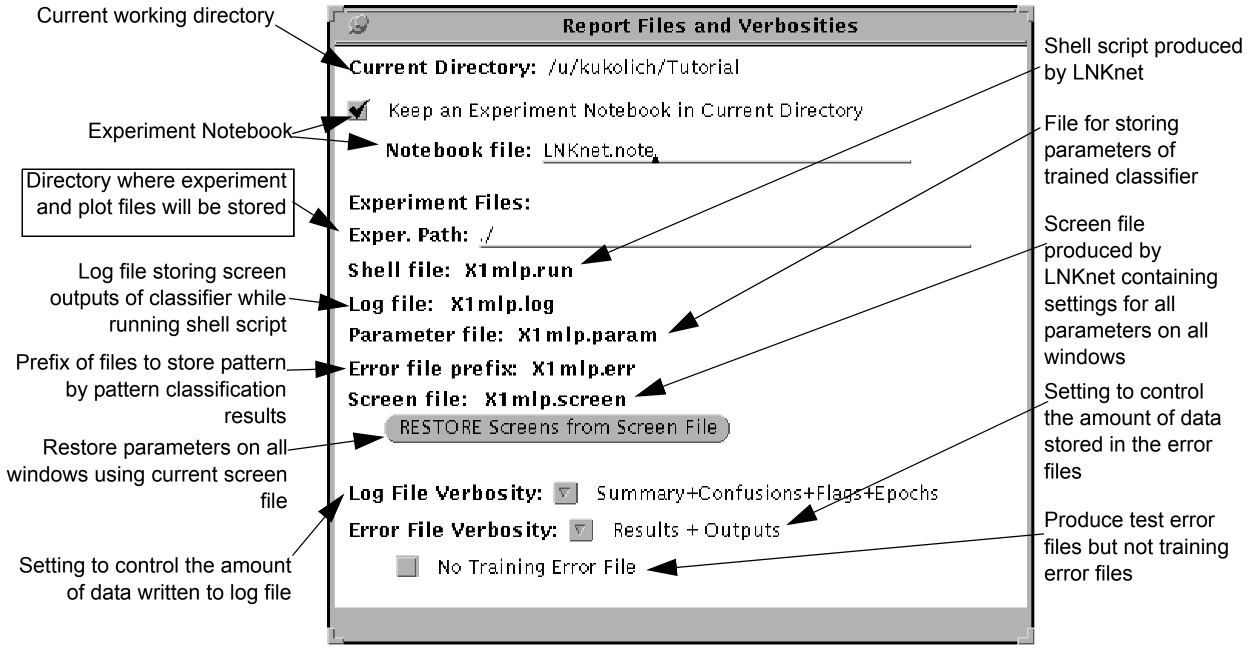
The Report Files and Verbosities window shown in Figure 5.3 sets the names of files created by the classifier during an experiment. The first field is the current working directory. This is the directory the user was in when LNKnet was started. It cannot be changed by the user. The next field is the **experiment path**. This is the path to the directory where all shell scripts, experiment files, and plot files will be written. This path can be an absolute path which starts from the root directory / or it can be a relative path which starts from the current working directory. In this example, the experiment path is a relative path, making the full path /u/kukolich/lnknet/Tutorial. An optional Experiment notebook is kept in the current working directory. When each experiment is started, a line is added to the notebook file with the experiment name, data base name, normalization parameters, feature selection parameters, and parameters for the classification algorithm. The experiment shell script writes training and testing results to the notebook as well as results from some plots. Below the experiment path are the names of files created by LNKnet or the classifier during an experiment. These names are automatically generated based on the experiment name and classifier and cannot be edited. The first file is the **shell script** created by LNKnet when START, CONTINUE or PLOT ONLY are selected on the main window. This shell script contains requested calls to the classifier for training or testing and calls to any requested plot programs.

When an experiment shell script is run, certain training and testing status information and results are stored in a **log file**. The type of information stored is controlled by the **log file verbosity flag**. The verbosity levels are described in Section 8.2.4. The log file can be viewed or printed from the **Preview and Print window** which is described in Section 7.1. When training is complete, a classifier **parameter file** is stored. This file contains all the information needed by a LNKnet program to recreate the trained classifier. Finally, when a classifier, in training or testing, finds the class of an input pattern, the results can be written to an **error file**. The amount of information in the error file is controlled by the **error file verbosity** flag. The error file verbosity levels are described in Section 8.2.6. Because the training error files can be very long, the user can select **No Training Error Files** to write test error files but not training files. The full name of the error file depends on the data file type used for input data (train, eval, test) and the action being performed (training, testing, or cross validation). These parameters are set on the main screen. The specifications of these names are found in Table 8.9 on page 125. Whenever this shell script is stored, LNKnet also creates a **screen file**. This screen file saves all the parameters set on all LNKnet windows. To restore a LNKnet experiment, set the experiment name and classifier on the main window, set the experiment path to the directory of the desired screen file, and select **Restore Exp. Screens**.

5.3 Data Base Selection

The data base selection window shown in Figure 5.4 selects the data files to be used for training and testing a classifier. The first field on the window is the **data path**, the directory where all the data base files, normalization files, and feature selection files for an experiment are stored. This can be an absolute path starting from the root directory or a relative path starting from the current working directory on the Reports window shown in Figure 5.3. All the data base description files in the data base directory are included in

FIGURE 5.3 Report Files and Verbosities Window

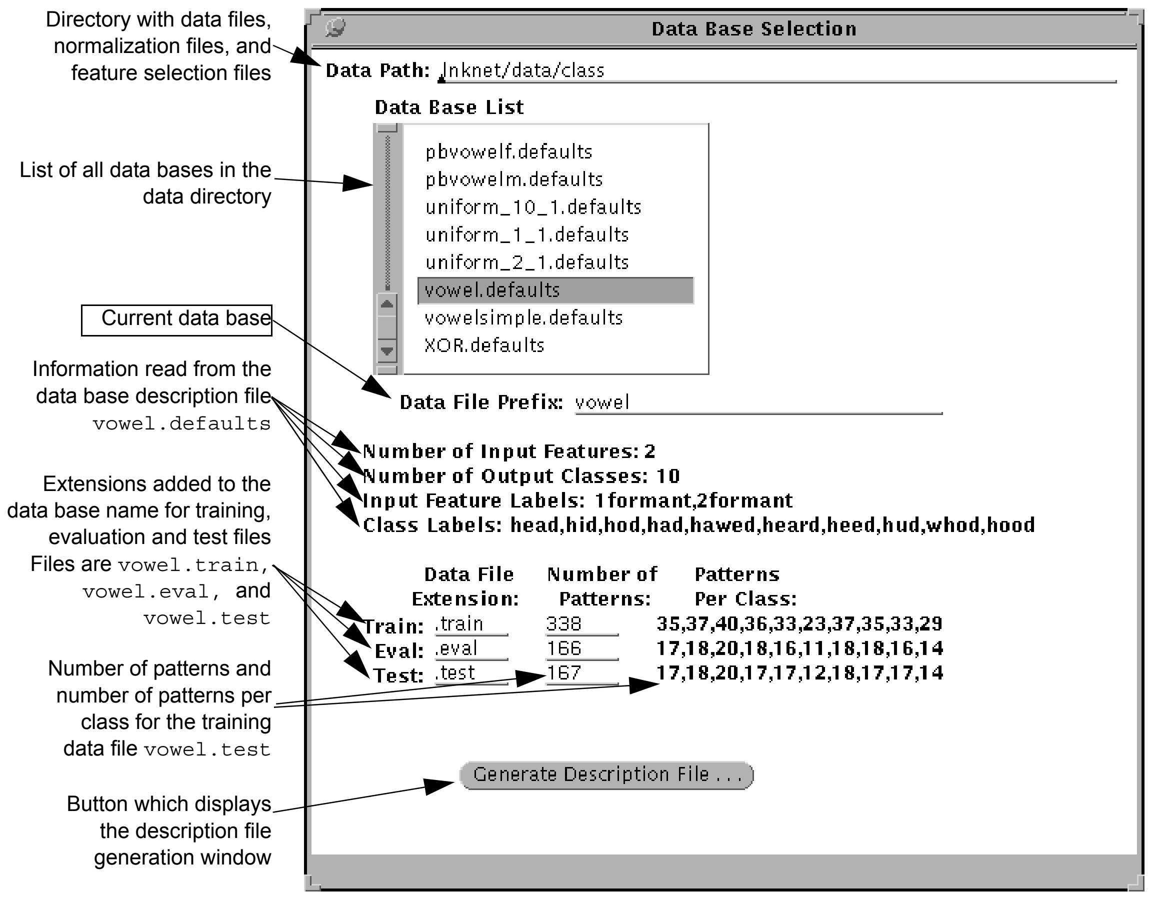

Current working directory
Experiment Notebook
Directory where experiment and plot files will be stored
Log file storing screen outputs of classifier while running shell script
Prefix of files to store pattern by pattern classification results
Restore parameters on all windows using current screen file
Setting to control the amount of data written to log file
Shell script produced by LNKnet
File for storing parameters of trained classifier
Screen file produced by LNKnet containing settings for all parameters on all windows
Setting to control the amount of data stored in the error files
Produce test error files but not training error files
Report Files and Verbosities
RESTORE Screens from Screen File

Current Directory: /u/kukolich/Tutorial
✓ Keep an Experiment Notebook in Current Directory
Notebook file: LNKnet.note
Experiment Files:
Exper. Path: ./
Shell file: X1mlp.run
Log file: X1mlp.log
Parameter file: X1mlp.param
Error file prefix: X1mlp.err
Screen file: X1mlp.screen
Log File Verbosity: Summary+Confusions+Flags+Epochs
Error File Verbosity: Results + Outputs
☐ No Training Error File

the **Data Base List** scroll list. These description files all have the suffix `.defaults` in their names. A new data base which does not yet have a description file will not be listed in the Data Base list. Use the description file generation window to create the missing description file. A data base can be selected from the scroll list or its name can be typed (without the .defaults suffix) in the **data file prefix** field below the scroll list. When a data base is selected, information about the data base is read from the description file. If LNKnet cannot find the description file, an error appears at the bottom of the screen and a stop sign will appear beside the Data Base... button on the main window. The description file can be created using the **Description File Generation** window shown in Figure 5.5. When a data base is selected, LNKnet also finds the data files included in the data base. The **file name extensions** for training, evaluation, and test files are specified at the bottom of the data base window. The actual file names are gotten by appending the extension to the data base name. If LNKnet finds these files it counts the total **number of patterns** and the number of patterns assigned to each class in the file. To use fewer patterns than are present in a file, change the number of patterns field. To reset the field, cause LNKnet to reread the file by reselecting the data base or by putting the cursor on the file name extension and hitting return.

The description file generation window shown in Figure 5.5 allows the user to create or modify a description file for a LNKnet data base. The user specifies the **number of input features**, **number of output classes**, and **labels** for the input features and classes. Selecting **Generate** writes the description file and adds it to the data base list on the data base selection window. The user can select **Cancel** to leave the description file generation window without creating a description file. It is important to get this right. An error in the data base description file can cause serious problems when an experiment is run.

FIGURE 5.4 Data Base Selection window


Directory with data files, normalization files, and feature selection files
List of all data bases in the data directory
Current data base
Information read from the data base description file vowel.defaults
Extensions added to the data base name for training, evaluation and test files
Files are vowel.train, vowel.eval, and vowel.test
Number of patterns and number of patterns per class for the training data file vowel.test
Button which displays the description file generation window
Generate Description File . . .

Data Base Selection

Data Path: lnknet/data/class

Data Base List

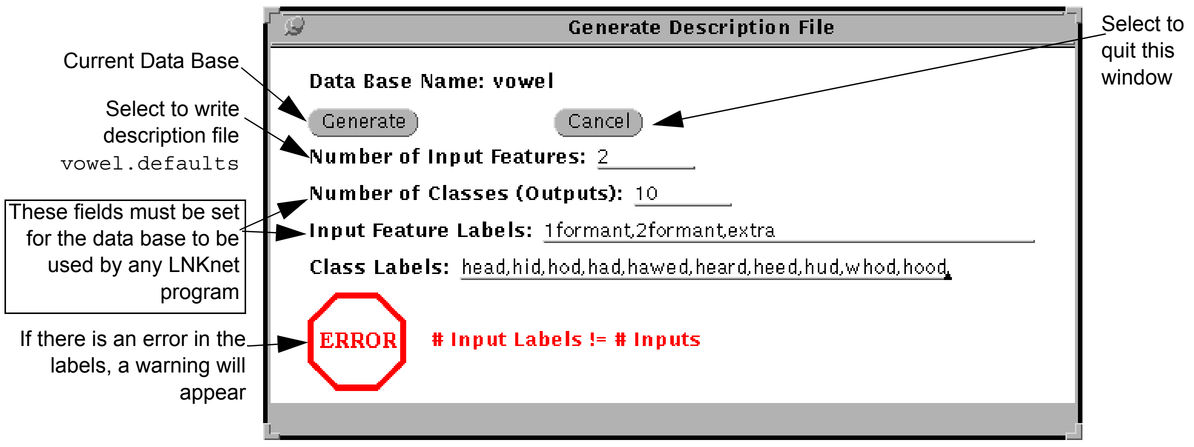
```
pbvowelf.defaults
pbvowelm.defaults
uniform_10_1.defaults
uniform_1_1.defaults
uniform_2_1.defaults
vowel.defaults
vowelsimple.defaults
XOR.defaults
```

Data File Prefix: vowel

Number of Input Features: 2
Number of Output Classes: 10
Input Feature Labels: 1formant,2formant
Class Labels: head,hid,hod,had,hawed,heard,heed,hud,whod,hood

	Data File Extension:	Number of Patterns:	Patterns Per Class:
Train:	.train	338	35,37,40,36,33,23,37,35,33,29
Eval:	.eval	166	17,18,20,18,16,11,18,18,16,14
Test:	.test	167	17,18,20,17,17,12,18,17,17,14

FIGURE 5.5 Description File Generation window


Current Data Base
Select to write description file vowel.defaults
These fields must be set for the data base to be used by any LNKnet program
If there is an error in the labels, a warning will appear
Select to quit this window
Generate
Cancel
ERROR

Generate Description File

Data Base Name: vowel

Number of Input Features: 2
Number of Classes (Outputs): 10
Input Feature Labels: 1formant,2formant,extra
Class Labels: head,hid,hod,had,hawed,heard,heed,hud,whod,hood

Input Labels != # Inputs

5.4 Normalization

When a data file is read by a classifier, it is possible to perform preprocessing for normalization. The preprocessing methods available in LNKnet either scale or rotate the input space. The normalization parameters for a data base are calculated using only training data.

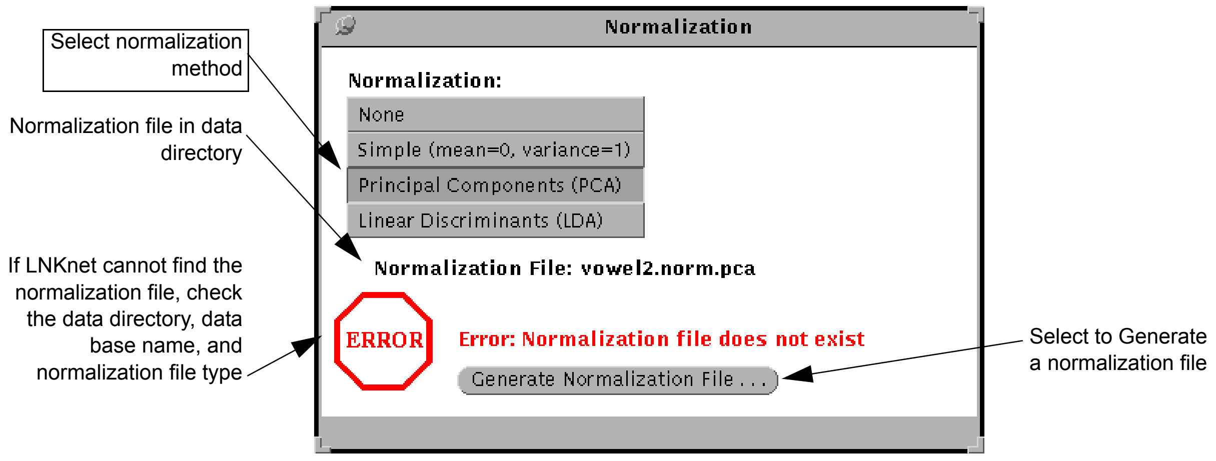
Simple normalization rescales each input feature independently to have a mean of 0 and a variance of 1. This compensates for the differences in the means and variances of the input dimensions. This should always be used for MLP and SVM classifiers.

Principal components analysis (PCA) rotates the input space to make the direction of greatest variance the first dimension. The remaining orthogonal dimensions correspond to directions of decreasing variance in the original input space. PCA can be used to reduce the number of input dimensions by first performing PCA and then selecting only the top N most important PCA features.

Linear discriminant analysis (LDA) assumes that classes and class means can be modeled using Gaussian distributions. It rotates the input space to make the first dimension the direction along which the classes can be most easily discriminated. The remaining dimensions are ordered by decreasing ability to be used to discriminate the classes. The number of features after LDA normalization is the minimum of D and M-1 where D is the original number of input features and M is the number of classes in the data base.

The normalization method used in a LNKnet experiment is selected on the normalization window shown in Figure 5.6. Selecting a **normalization method** sets the **normalization file name**. This file is stored in the data base directory which is set on the data base window. If this file does not exist an error will appear at the bottom of the window and beside the "Feature Normalization..." button on the main window. The normalization file can be created on the **Normalization File Generation** window shown in Figure 5.7.

FIGURE 5.6 Normalization Algorithm Selection


Select normalization method
Normalization file in data directory
If LNKnet cannot find the normalization file, check the data directory, data base name, and normalization file type
Select to Generate a normalization file
Normalization
Normalization:
None
Simple (mean=0, variance=1)
Principal Components (PCA)
Linear Discriminants (LDA)
ERROR
Generate Normalization File . . .

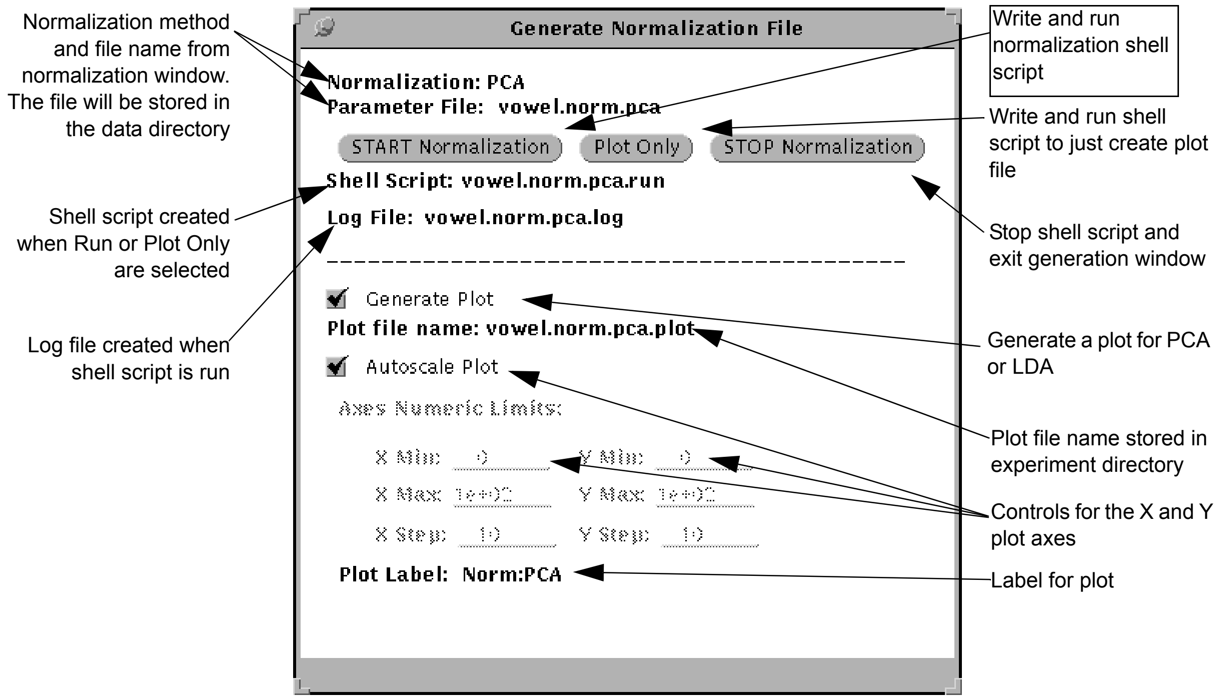
Normalization File: vowel2.norm.pca

Error: Normalization file does not exist

5.4.1 Generating Normalization Files

The normalization file generation window writes and runs shell scripts that calculate and plot normalization parameters. The normalization method and the parameter file to be written are listed at the top of the window. When the user selects **Run** on this window a **shell script** calls a normalization program which calculates normalization parameters based on the training data file and stores them to a parameter file in the data base directory. The data directory is specified on the data base window. If **Only store shell script, do not run** is selected on the main window, the shell script will be written but not run. When the shell script is run some status information is printed to a **log file** in the experiment directory and to the text window from which LNKnet was originally started. Because this is not an experiment, no information is printed to the experiment notebook file. Selecting **Cancel** stops the normalization shell script and removes the Normalization File Generation window. If the **Generate Plot** box is checked, a plot is generated and displayed after the normalization file has been created. This plot can also be generated without recalculating the normalization parameters by selecting the **Plot Only** button. The plot shows the relative importance of the features created when the input space is rotated using either the PCA or LDA normalization algorithms. **X and Y limits** of the plot can be chosen by the user or the plot program can choose them using the **autoscale** flag. The X dimension of this plot is the number of normalized input features. The Y value for each feature is the percentage of the total of the rotation matrix eigenvalues accounted for by that feature. An example of a normalization plot is found in Figure 2.40 on page 46.

FIGURE 5.7 Normalization File Generation window


Generate Normalization File
START Normalization
Plot Only
STOP Normalization
Generate Plot
Autoscale Plot
Normalization method and file name from normalization window. The file will be stored in the data directory
Shell script created when Run or Plot Only are selected
Log file created when shell script is run
Write and run normalization shell script
Write and run shell script to just create plot file
Stop shell script and exit generation window
Generate a plot for PCA or LDA
Plot file name stored in experiment directory
Controls for the X and Y plot axes
Label for plot

Normalization: PCA
Parameter File: vowel.norm.pca

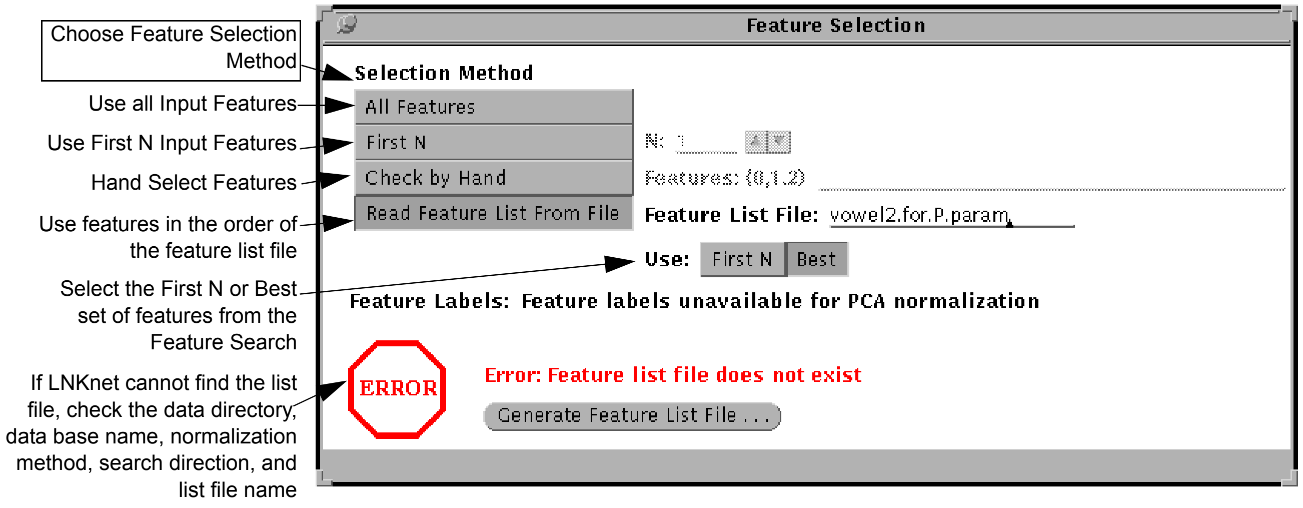
Shell Script: vowel.norm.pca.run
Log File: vowel.norm.pca.log

✓ Generate Plot
Plot file name: vowel.norm.pca.plot
✓ Autoscale Plot
Axes Numeric Limits:
X Min: 0 Y Min: 0
X Max: 1e+02 Y Max: 1e+02
X Step: 10 Y Step: 10
Plot Label: Norm:PCA

5.5 Feature Selection

Sometimes you do not want to use all of the input features available in a data base. LNKnet algorithms can select a subset of them. This subset can be the **first N features**, a **hand picked set**, or a set of features **read from a file**. When using PCA or LDA as the normalization algorithm, the first N features are usually chosen, which occurs after normalization has been applied to the input data. When hand picking a set of features, the original features are numbered from 0 to ninputs-1. The selected features are given in a comma delimited list with integers, commas, and no spaces. The labels which match the selected input features are listed in the middle of the window. If there is a problem with the input features or labels, an error or warning message appears. **Feature list files** are stored in the data directory which is set on the data base selection window shown in Figure 5.4 on page 82. LNKnet opens the specified file and reads a feature list from it. The user may choose the **number of features** to use from this list or the **best** set in the file can be used. If LNKnet cannot open the feature list file, an error message appears at the bottom of the window and a stop sign appears on the main window. Feature list files can be created using the **Feature List Generation** window shown in Figure 5.9.

FIGURE 5.8 Feature Selection Parameters


Choose Feature Selection Method
Use all Input Features
Use First N Input Features
Hand Select Features
Use features in the order of the feature list file
Select the First N or Best set of features from the Feature Search
If LNKnet cannot find the list file, check the data directory, data base name, normalization method, search direction, and list file name
Feature Selection
Selection Method
All Features
First N
Check by Hand
Read Feature List From File
N: 1
Features: (0,1,2)
Use: First N Best
ERROR
Generate Feature List File . . .

Feature List File: vowel2.for.P.param

Feature Labels: Feature labels unavailable for PCA normalization

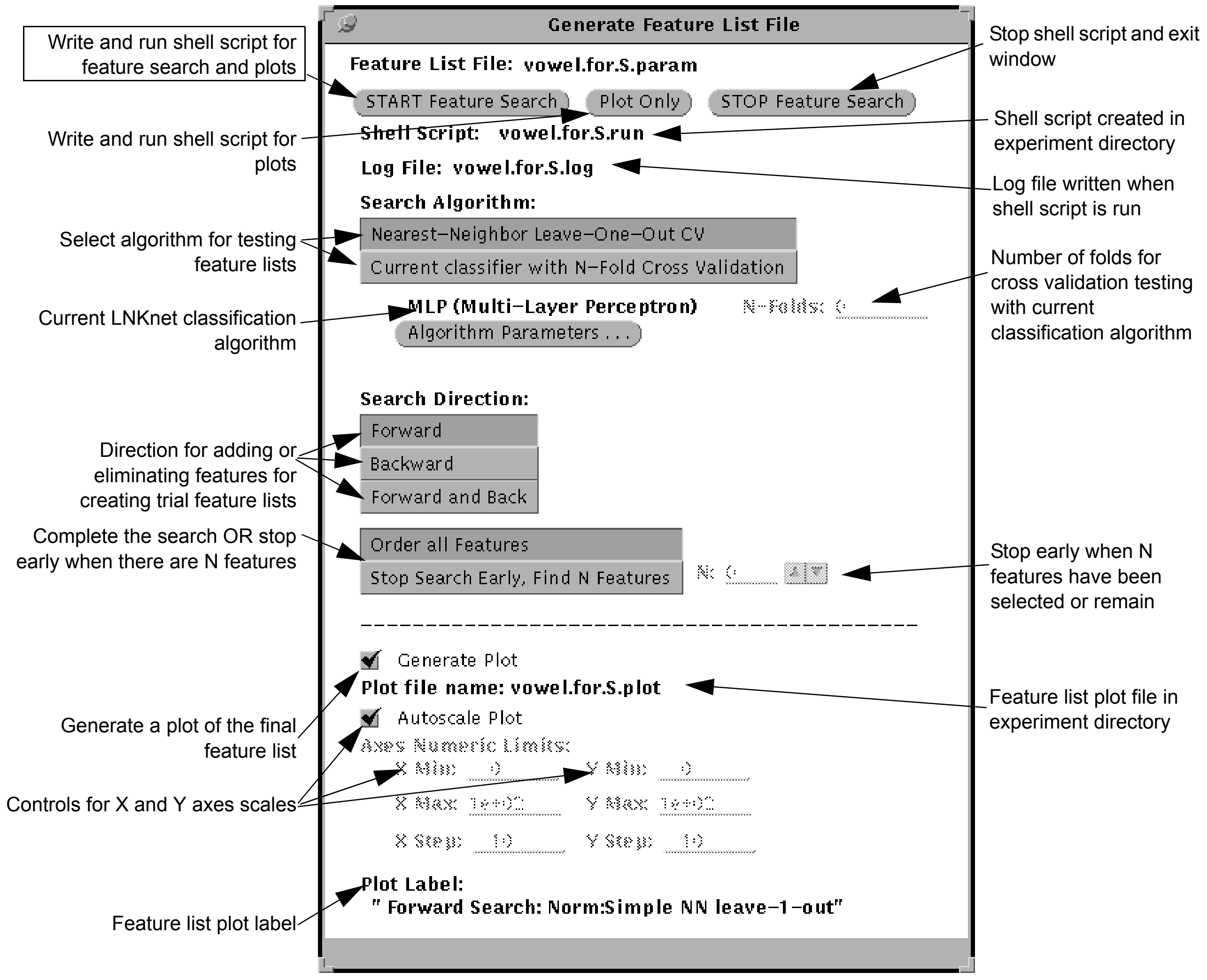
Error: Feature list file does not exist

The feature list file generation window writes and runs shell scripts that create and plot feature lists. When **Run** is selected on this window a **shell script** is written to the experiment directory and run. If **Only store shell script, do not run** is selected on the main window, the shell script is not run. When the shell script is run, status information from the feature search is printed to a **log file** and to the window LNKnet was originally started in. The shell script creates a feature list file. The features in this file are plotted if **Generate Plot** is checked in the lower half of the window. This plot can also be generated without repeating the feature search by selecting the **Plot Only** button. Selecting **Cancel** stops the feature selection shell script and removes the generation window. The shell script, log file, and plot file names are automatically generated based on the feature list file name which is set on the feature selection window.

To run a feature search, the program generates a series of feature lists and for each list performs a cross validation test on a classifier. The classification algorithm used in the tests can be a nearest neighbor algorithm with **leave-one-out cross validation** or any LNKnet classification algorithm with **N-fold cross validation**. The classifier used in the second case is the one selected on the main LNKnet window. If the classification algorithm uses a clustering algorithm for initialization, the clustering algorithm for feature searches is Kmeans, not the algorithm selected on the classifier's parameter window.

There are three **directions** for the selection of features for inclusion in the feature lists tested by the classifier. The search can go forward, backward, or forward and back. In a forward search, each feature is tried singly and the feature which gets the best classification rate is selected as the first feature. The remaining features are tested in combination with the first feature and the best of them is added as the second feature. Features are added this way until none are left to add.

FIGURE 5.9 Feature Search Parameters


Write and run shell script for feature search and plots
Write and run shell script for plots
Select algorithm for testing feature lists
Current LNKnet classification algorithm
Direction for adding or eliminating features for creating trial feature lists
Complete the search OR stop early when there are N features
Generate a plot of the final feature list
Controls for X and Y axes scales
Feature list plot label
Stop shell script and exit window
Shell script created in experiment directory
Log file written when shell script is run
Number of folds for cross validation testing with current classification algorithm
Stop early when N features have been selected or remain
Feature list plot file in experiment directory
START Feature Search
Plot Only
STOP Feature Search
Algorithm Parameters ...

Generate Feature List File

Feature List File: vowel.for.S.param

Shell Script: vowel.for.S.run

Log File: vowel.for.S.log

Search Algorithm:
- Nearest-Neighbor Leave-One-Out CV
- Current classifier with N-Fold Cross Validation

MLP (Multi-Layer Perceptron) N-Folds: 0

Search Direction:
- Forward
- Backward
- Forward and Back

- Order all Features
- Stop Search Early, Find N Features N: 0

✓ Generate Plot

Plot file name: vowel.for.S.plot

✓ Autoscale Plot

Axes Numeric Limits:

X Min: 0 Y Min: 0

X Max: 1e+02 Y Max: 1e+02

X Step: 10 Y Step: 10

Plot Label:
" Forward Search: Norm:Simple NN leave-1-out"

In a backward search, the program starts with all of the features selected and tries leaving each one out. The feature which the classifier did best without is selected as the last feature. The program goes on taking features away until none are left. The idea of a backward search is that there may be some set of features which do well when they are together but which do poorly individually. This set of features would not be found by a forward search.

A forward and backward search combines the two search methods above. The program starts searching forward with no features selected. When it has added two features, it searches for one to take away. It continues then, adding two and taking away one, until it has added all of the available features. This forward and backward search can find some interdependencies in the input features which are not found using the other two searches.

It is possible to **stop** a feature search early, when there are N features on the list. In the case of a forward search this is when N have been selected. For a backward search this is when there are N features left.

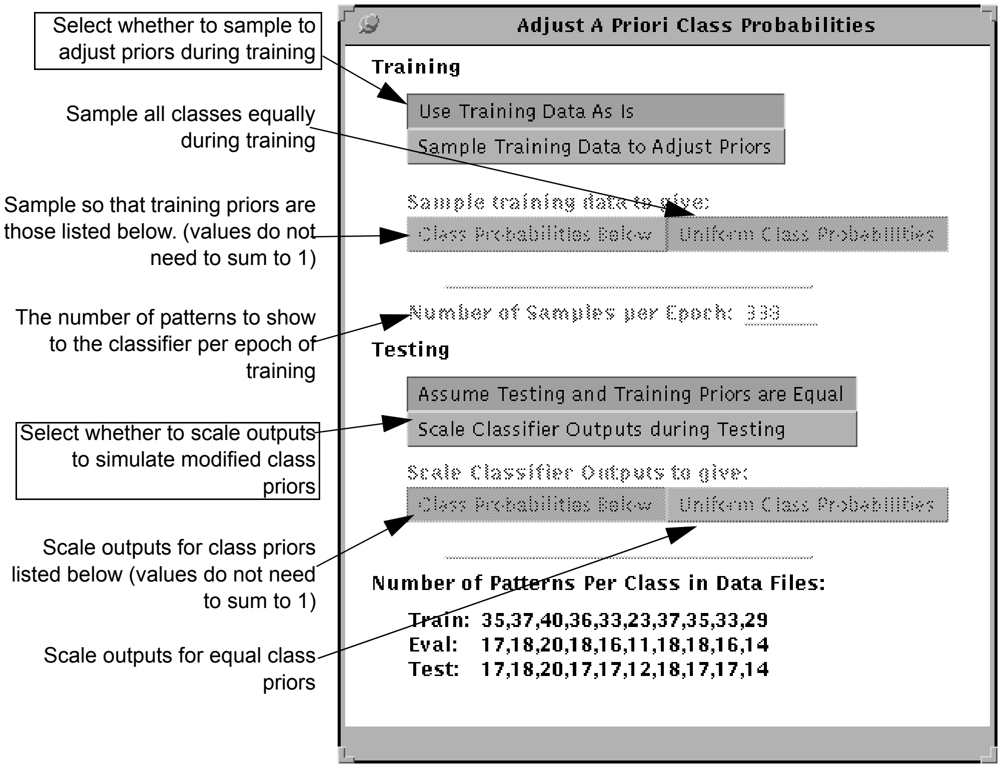
The feature selection **plot** shows the error rate for sets of features found during a feature search. The **X and Y limits** of the plot can be chosen by the user or the plot program can choose them using the **autoscale** flag. The X dimension is the feature added to the feature list to generate the classification error rate given in the Y direction. An example of a feature selection plot is in Figure 2.37 on page 43.

5.6 *A Priori* Probabilities

Normally the frequency of occurrence of different classes is equal in training, evaluation, and test data and no special actions are necessary to train and test classifiers. In some classification problems, however, the class prior probabilities in the training data do not match the probabilities in the test data. For instance, in a heart monitoring system there may be as many training examples of normal as there are of abnormal heart beats even though during testing there are ten times as many normal as abnormal patterns. This imbalance in class probabilities can be compensated for by **sampling during training** or by **scaling class outputs** during testing. Those LNKnet classifiers which have continuous outputs support both of these kinds of class probability adjustment. The nearest neighbor style classifiers only support priors adjustment during training.

There is a set of data bases which illustrates use of priors adjustment. They are uniform_1_1, uniform_2_1, and uniform_10_1. In all of these, there are two classes which have uniform Gaussian distributions. The centers of these classes are one standard deviation apart. In uniform_1_1, there are 500 patterns from each class. In uniform_2_1 there are 666 patterns from one class and 333 from the other, giving a ratio of 2 to 1 in their class probabilities. In uniform_10_1 there are 1000 patterns from one class and 100 from the other, giving a ratio of 10 to 1 in the class probabilities. Below is a table giving overall error rates on testing data for various training and test situations generated using these data bases and the classifier **gauss** with separate diagonal variances for each class.

FIGURE 5.10 Priors Adjustment Parameters


Select whether to sample to adjust priors during training
Sample all classes equally during training
Sample so that training priors are those listed below. (values do not need to sum to 1)
The number of patterns to show to the classifier per epoch of training
Select whether to scale outputs to simulate modified class priors
Scale outputs for class priors listed below (values do not need to sum to 1)
Scale outputs for equal class priors

Adjust A Priori Class Probabilities

Training

Use Training Data As Is
Sample Training Data to Adjust Priors

Sample training data to give:
Class Probabilities Below | Uniform Class Probabilities

Number of Samples per Epoch: 333

Testing

Assume Testing and Training Priors are Equal
Scale Classifier Outputs during Testing

Scale Classifier Outputs to give:
Class Probabilities Below | Uniform Class Probabilities

Number of Patterns Per Class in Data Files:

```
Train:  35,37,40,36,33,23,37,35,33,29
Eval:   17,18,20,18,16,11,18,18,16,14
Test:   17,18,20,17,17,12,18,17,17,14
```

TABLE 5.1 Percent Error Rates with two sets of data from the same Two Class Problem

Training		**Test Error Rate with given Ratio of Class A to Class B during Testing**	
Training File Ratio of Class A to Class B	**Priors Adjustment to Training data**	**1:1**	**10:1**
1:1	No Adjustment	16.7	15.05
	Sample during Training to give 10:1		6.5
	Scale Outputs to Simulate 10:1		6.55
10:1	No Adjustment	28.7	6.5
	Sample to give 1:1	16.9	
	Scale to Simulate 1:1	16.6	

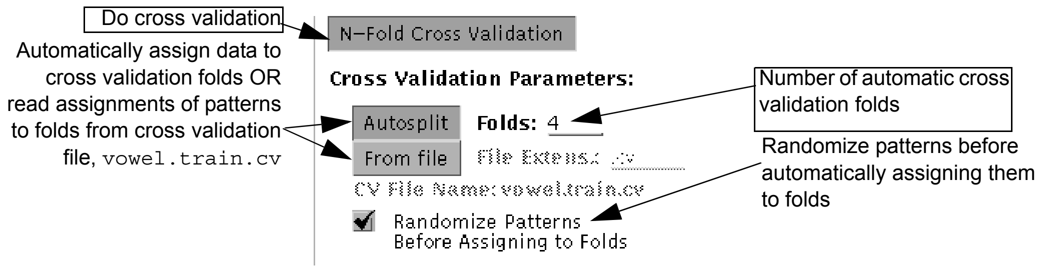
Table 5.1 shows that the testing error rates can be greatly reduced by priors adjustment when testing and training priors differ substantially. With two evenly sampled Gaussian classes, an error rate of 16.7% is expected with decision boundaries equidistant from the means of the classes as shown in the first row of Table 5.1 in the column labeled “1:1”. When there are considerably fewer patterns from one class, the overall error rate can be

improved to 6.5% by moving the boundary closer to the undersampled class's center. This greatly reduces the error rate for the more common class and increases the error rate for the undersampled class. When evenly sampled data is used for training and 10 to 1 unevenly sampled data is used for testing, the error rate is near 15%, as shown in the 10:1 column, unless some adjustment is made. Either method of priors adjustment can be used to bring the overall error rate down to 6.5% on the unevenly sampled data as shown in the second and third row of Table 5.1 under the column labeled "10:1". Conversely, when 10 to 1 unevenly sampled data is used in training and evenly sampled classes are used for testing the error rate is above 28%, as shown in the fourth row of Table 5.1 under the column labeled "1:1". Priors adjustment by sampling the training data uniformly or scaling the outputs brings the class error rates back to roughly 16.7% on evenly sampled data as shown in the bottom two rows of Table 5.1.

5.7 Cross Validation

Sometimes there is not enough data available to split it into three partitions, one for training, one for evaluation, and one for testing. In such a case, N-fold cross validation can be used to estimate the classification error rate on new data. The idea of cross validation is to split the data into several folds and test each fold against a classifier trained on the data in the other folds. Cross validation is primarily used where there are few patterns (< 1000) and with a small number of folds (between 4 and 10).

FIGURE 5.11 Cross Validation Parameters (on Main Window)


Do cross validation
Automatically assign data to cross validation folds OR read assignments of patterns to folds from cross validation file, vowel.train.cv
Number of automatic cross validation folds
Randomize patterns before automatically assigning them to folds
N-Fold Cross Validation
Autosplit
From file

Cross Validation Parameters:
Folds: 4
File Extens.: .cv
CV File Name: vowel.train.cv
[✓] Randomize Patterns Before Assigning to Folds

The most significant task in cross validation is the assignment of patterns to their training and testing folds. This can be performed automatically or by hand. The algorithm which does the **automatic fold** assignments attempts to preserve class prior probabilities while keeping the size of test folds constant. If **Randomize Patterns Before Assignment** is selected, the fold assignments depend on the random number seed. The user can test a classifier several times by rerunning an experiment with different seeds.

If the training data is collected from different places or at different times, it can be important that the data from different collection conditions is split up evenly during cross validation. In such a situation, the training data can be split into folds by hand. Because the specifications for these divisions are complicated, they are stored in a **cross validation file**. Patterns are divided into partitions called splits and then the splits are assigned to cross validation folds for training and for testing. The name of the cross validation file is set by appending the cross validation **file extension**, .cv, to the training data file name, which is set on the data base window shown in Figure 5.4.

In the example below, there is a speaker independent speech recognizer which is being trained on twenty-one patterns taken from four speakers. To complicate matters, speaker 1 and speaker 3 sound very similar, so speaker 3 should not give training data for speaker 1's test and vice versa. Also, there is some data for three of the speakers which I don't want to include in the tests. To make the testing folds easier to understand, I have added an empty split to the middle of the fourth speaker's data.

Figure 5.12 shows the speakers for each data pattern. Figure 5.13 is a list which is used to split the data up by speaker while identifying the patterns which will not be tested. Finally, Figures 5.14 and 5.15 are bit vectors for the train and test folds which identify the splits to use for each. Figure 5.16 shows the cross validation file 4SPEAK.train.cv which specifies the fold assignments for this cross validation experiment. There are backslashes at the end of the first three lines to indicate that there are more flags on the following line. The backslashes are immediately followed by a carriage return. When using the backslashes, you must be careful to remember to put spaces after the comma delimited lists. The backslash character and carriage return do not count as spaces.

FIGURE 5.12 Pattern numbers and speaker for each pattern for 4SPEAK.train

```
0   1   |2   |3   4   |5   |6   |7   8   |9   10  11  |12  |13  14  |15  16  17  |18  19  20
Sp1 Sp1|Sp1  |Sp1 Sp1|Sp2|Sp2|Sp2 Sp2|Sp3 Sp3 Sp3|Sp3|Sp3 Sp3|Sp4 Sp4 Sp4|Sp4 Sp4 Sp4
```

FIGURE 5.13 Splits: (first and last pattern in each split, -1:-1 is an empty split)

```
0:1,2:2,3:4,5:5,6:6,7:8,9:11,12:12,13:14,15:17,-1:-1,18:20
```

FIGURE 5.14 Testing folds: (do not test on middle split for each speaker)

```
101000000000,000101000000,000000101000,000000000101
```

FIGURE 5.15 Training folds: (do not train on Sp1 for Sp3 test)

```
000111000111,111000111111,000111000111,111111111000
```

FIGURE 5.16 Cross Validation File: (4SPEAK.train.cv)

```
cross_valid -nfolds 4 -nsplits 12 \
-cv_splits 0:1,2:2,3:4,5:5,6:6,6:8,9:11,12:12,13:14,15:17,-1:-1,18:20 \
-cv_train_mask 000111000111,111000111111,000111000111,111111111000 \
-cv_test_mask 101000000000,000101000000,000000101000,000000000101
```

CHAPTER 6 Plots

Some of the most visible and useful features of LNKnet are the many types of plots produced. All of the classifiers and clusterers can produce decision region plots that can be overlaid with a scatter plot of the data and an internals plot of classifier parameters. Those classifiers which have continuous outputs can produce a profile plot and a histogram plot of the data. Error files from these classifiers can be used to produce posterior probability plots, receiver operating characteristics (ROC) curve or detection plots, and rejection plots. Incrementally trained classifiers, those which go over the training data multiple times, can use the cost plot or percent error plot. Many classifiers have a structure plot which shows connections between classifier nodes. Figure 6.1 shows the LNKnet window used to select these plots. Once a plot file has been generated, it can be redisplayed or printed from the Preview and Print window described in Section 7.1. There are also plots for showing the results from a normalization run or from a feature search. These plots are generated on the Normalization File Generation window and the Feature List File Generation window respectively. The normalization plot and feature list plot are explained in Chapter 5.

6.1 Decision Region Plots

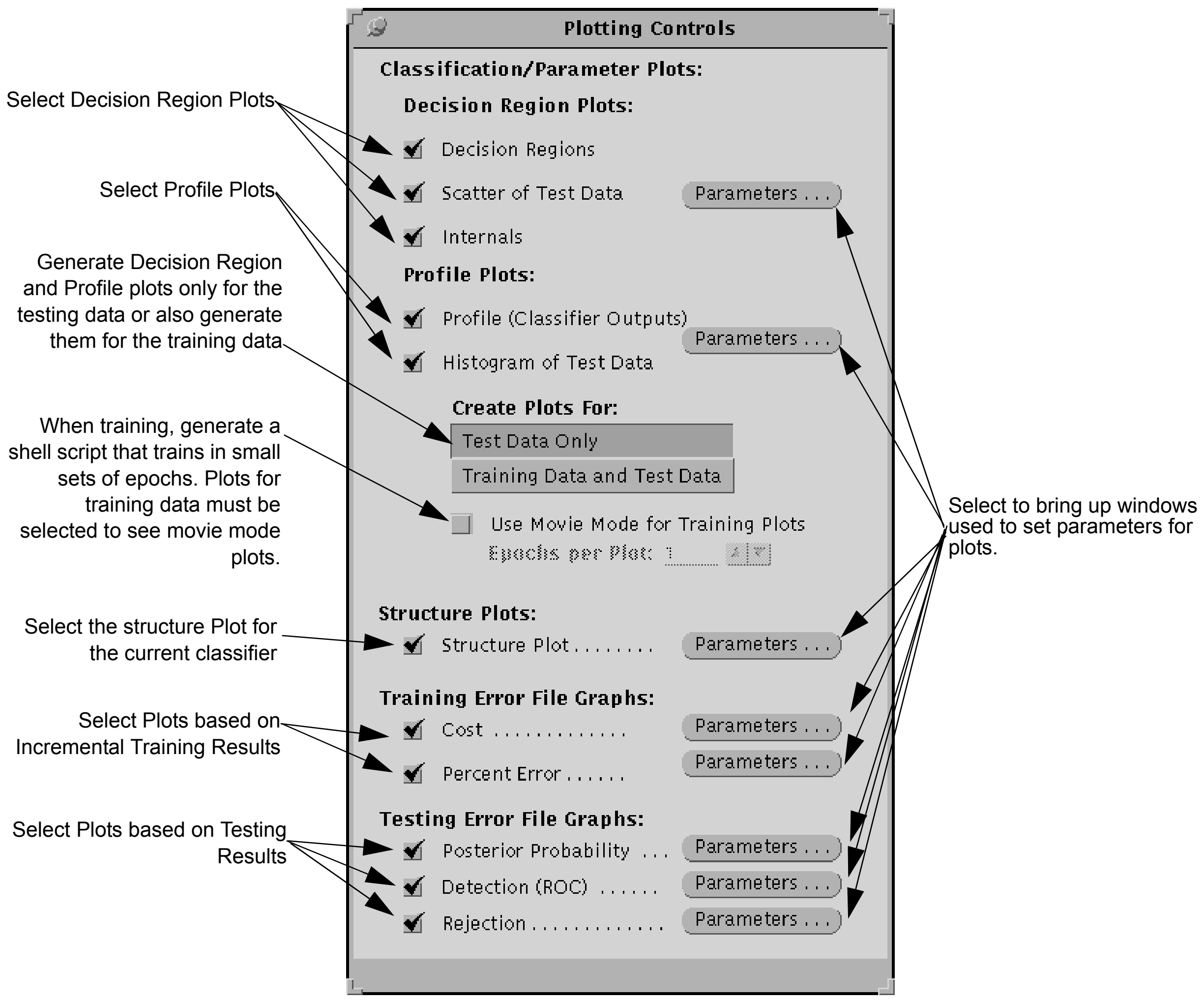
The decision region plot parameters window, shown in Figure 6.2, is displayed by selecting the top most **Parameters...** button on the plotting controls window. It sets parameters for three 2-dimensional plots. When these plots are displayed, all three plots are combined in one plot window. An example of these plots is in the tutorial in Figure 2.15.

The **decision region plot** shows what class would be returned for each point in the plotted area, given the current classifier model. If the current algorithm is a clusterer, borders are displayed which show the area assigned to each cluster. When **color** is used, decision regions for different classes are in different colors. When color is not used, only decision region boundaries are plotted.

Decision regions are created by sampling the plotted area uniformly and filling in each square cell in the plotting area with the color of the class at the cell’s center. A quick rough plot can be made using a coarse grid with 50 **intervals per dimension**. More refined decision region plots can be obtained after a longer time if more points (100 to 500) are used per dimension. This approach to forming decision regions was selected because it can produce accurate plots and can be used with any type of classifier.

A color coded **scatter plot** can be overlaid on a decision region plot. Each pattern is represented by a white bordered square. If color is not used, various plot symbols are used

FIGURE 6.1 Plot Selection Window


Plotting Controls
Select Decision Region Plots
Select Profile Plots
Generate Decision Region and Profile plots only for the testing data or also generate them for the training data
When training, generate a shell script that trains in small sets of epochs. Plots for training data must be selected to see movie mode plots.
Select the structure Plot for the current classifier
Select Plots based on Incremental Training Results
Select Plots based on Testing Results
Select to bring up windows used to set parameters for plots.
Parameters ...

Classification/Parameter Plots:

Decision Region Plots:
- ✓ Decision Regions
- ✓ Scatter of Test Data
- ✓ Internals

Profile Plots:
- ✓ Profile (Classifier Outputs)
- ✓ Histogram of Test Data

Create Plots For: [✓ Test Data Only] Training Data and Test Data

- ☐ Use Movie Mode for Training Plots
- Epochs per Plot: 1

Structure Plots:
- ✓ Structure Plot........

Training Error File Graphs:
- ✓ Cost ....
- ✓ Percent Error ......

Testing Error File Graphs:
- ✓ Posterior Probability ...
- ✓ Detection (ROC) ......
- ✓ Rejection ....

to identify the classes of the patterns. Increasing the **Level of Detail** to 2 changes the plot symbols to capital letters for each class. If color is used, and there are two input features, squares with the same color as the background region are classified correctly and those with differing colors are classified incorrectly. If there are more than two input features, it may be necessary to limit the number of patterns displayed in the scatter plot to get this same result. By NOT selecting **Show All Data** and setting the **distance limit**, it is possible to plot only those patterns that fall close to the decision region plane. When **highlight misclassified data points** is selected, these misclassified points are shown as grey and correct patterns are shown colored. In a black and white plot, misclassified points are shown normally and correct patterns are shown as tiny dots.

Finally, an **internals plot** can be overlaid on the scatter plot and decision region plot. The form of the internals plot depends on the algorithm. There are three basic types. Three classifiers use lines to show the internals. The multi-layer perceptron shows the planes defined by the first hidden layer. The binary tree classifier shows the node tests for each non-terminal node. The histogram classifier shows the edges of the histogram bins. The second type of internals plot uses ovals, circles, or rectangles to show the size and position of Gaussians, spheres, or hyper-rectangles used in classification or clustering. RBF, GAUSS, and HYPER, are examples of algorithms which have this type of internals plot. The global scale factor can be used to alter the size of these figures. Finally, the nearest neighbor algorithms which use only the positions of centers to determine the class show small squares for each stored center. KNN and LVQ are examples of this type of algorithm. When the **level of detail** is raised to 3, the internals plot elements are labeled by class or by node number.

There are two other features that are related to plotting with many-dimensional data bases. First, two **plotting dimensions** can be selected. The dimension numbers are counted from zero. The plot axes limits can be specified by the user or they can be set automatically based on the range of the scatter plot data. Second, the **values for the non-plotted dimensions** can be set using a comma delimited list. The list must have values for all dimensions, including X and Y. The X and Y settings will be ignored. For example, if there are five input features and dimensions 0 and 4 are plotted, then the list “0,1,-75,0.5,0” sets the second, third and fourth dimensions to 1, -75, and 0.5 when decision regions are plotted for dimensions 0 and 4. If no settings are provided, all of the other features are set to 0 when the decision region plot is generated. Combining these two features, selection of the plotted features and setting values for non-plotted features, it is possible to gain some understanding of the shapes of multi-dimensional decision regions.

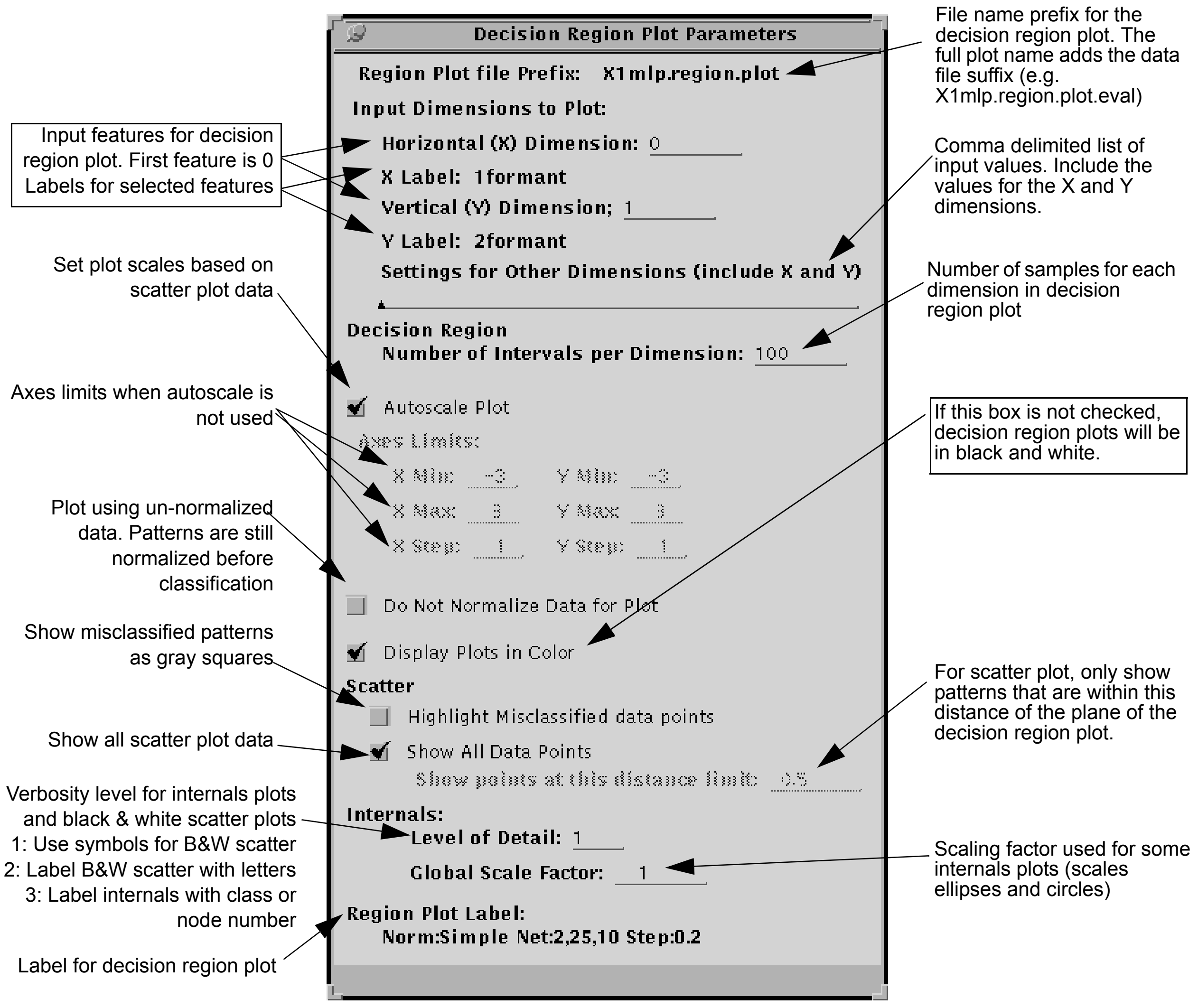
There is one final feature that relates to the plotting dimensions. The plots can be generated for data **before or after normalization**. In the case of simple normalization, this will change the values on the axes. When PCA or LDA normalization is being used, this means that the decision regions can be generated using the original input dimensions or using the rotated dimensions generated by the normalization. When the plots are being generated for un-normalized data, there will be no internals plot. The internals plots are derived from classifier parameters that were trained in the normalized data space and they cannot easily be translated back into the un-normalized space.

6.2 Profile Plots

The profile plot parameters window shown in Figure 6.3 is displayed by selecting the second **Parameters...** button on the plotting controls window. It sets parameters for two 1-dimensional plots. When these plots are displayed, both plots are combined in one plot window. An example of these plots is in the tutorial in Figure 2.16 on page 25. The profile plot is only available for classifiers with continuous outputs.

The **profile plot** shows the outputs for each classifier output node for each point along a selected dimension given the current classifier model. The **dimension plotted** is the first field on the plot window. Dimension numbers are counted from zero. The curve for each output node is color coded for the class of the node. The sum of the outputs for each point is plotted as a black line above the colored output lines. A bar below the zero line on the plot shows the class returned for each point sampled to create the profile plot.

FIGURE 6.2 Parameter window for the Decision Region Plot



Decision Region Plot Parameters

Region Plot file Prefix: X1mlp.region.plot

Input Dimensions to Plot:

Horizontal (X) Dimension: 0

X Label: 1formant

Vertical (Y) Dimension: 1

Y Label: 2formant

Settings for Other Dimensions (include X and Y)

Decision Region

Number of Intervals per Dimension: 100

✓ Autoscale Plot

Axes Limits:

X Min: -3 Y Min: -3

X Max: 3 Y Max: 3

X Step: 1 Y Step: 1

☐ Do Not Normalize Data for Plot

✓ Display Plots in Color

Scatter

☐ Highlight Misclassified data points

✓ Show All Data Points

Show points at this distance limit: 0.5

Internals:

Level of Detail: 1

Global Scale Factor: 1

Region Plot Label:

Norm:Simple Net:2,25,10 Step:0.2

File name prefix for the decision region plot. The full plot name adds the data file suffix (e.g. X1mlp.region.plot.eval)

Input features for decision region plot. First feature is 0

Labels for selected features

Comma delimited list of input values. Include the values for the X and Y dimensions.

Set plot scales based on scatter plot data

Number of samples for each dimension in decision region plot

Axes limits when autoscale is not used

If this box is not checked, decision region plots will be in black and white.

Plot using un-normalized data. Patterns are still normalized before classification

Show misclassified patterns as gray squares

For scatter plot, only show patterns that are within this distance of the plane of the decision region plot.

Show all scatter plot data

Verbosity level for internals plots and black & white scatter plots
1: Use symbols for B&W scatter
2: Label B&W scatter with letters
3: Label internals with class or node number

Scaling factor used for some internals plots (scales ellipses and circles)

Label for decision region plot

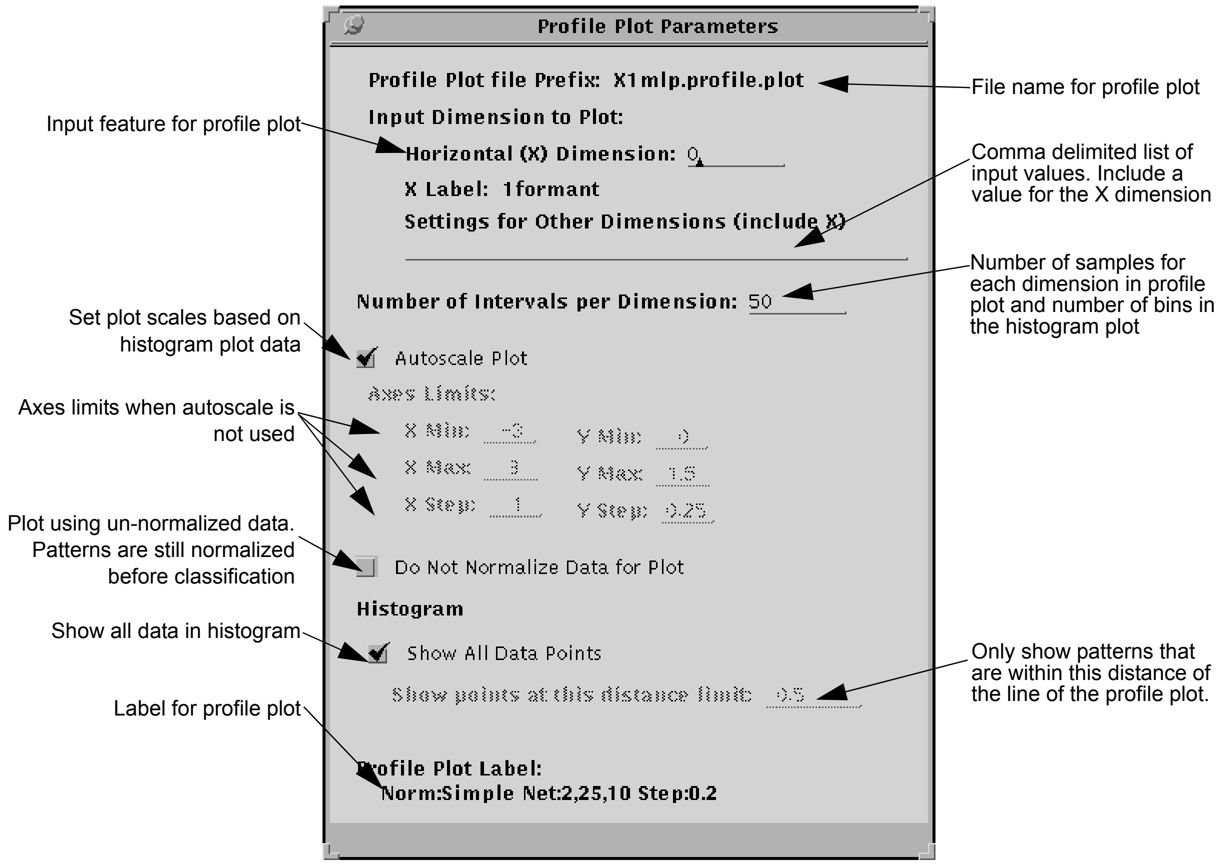
A color coded **histogram plot** is displayed in the lower half of the plot window. The line being sampled for the profile plot is divided into a number of bins. The number of bins is the **Number of Intervals per Dimension**. Each pattern is tested for its distance from the plotted line. If **Show All data Points** is selected, all patterns are included in the histogram. If Show all data is NOT set, only those patterns closer to the profile plot line than the **distance limit** are plotted. Each included pattern is assigned to a bin based on its X value. A colored square is drawn for it in that bin. If the profile plot has been selected, the patterns are also tested using the current classifier. If the pattern is classified correctly, its square is drawn above the histogram baseline. If the pattern is misclassified the square is drawn below the baseline.

When the autoscale flag is set, the horizontal or X axis limits are set according to the range of the input data in the dimension being plotted. The two vertical or Y axes are scaled by the range of the profile plot outputs and histogram bin heights. The user has the option of specifying the horizontal axis limits and the profile plot vertical axis limits.

As in the decision region plot, the **values for the non-plotted dimensions** can be set using a comma delimited list. The list must have values for all dimensions, including the plotted dimension X. The X setting will be ignored. For example, if there are five input features and dimension 0 is plotted, then the list “0,1,-75,0.5,0” sets the second, third, fourth, and fifth dimensions to 1, -75, 0.5, and 0 when a profile is plotted for dimension 0. If no settings are provided, all of the other features are set to 0 when the profile plot is generated.

Also as in the decision region plot, the profile and histogram plots can be generated for **data before or after normalization**. In the case of simple normalization, this will change the values on the X axis. When PCA or LDA normalization is being used, this means that the output profiles can be generated using the original input dimensions or using the rotated dimensions generated by the normalization.

FIGURE 6.3 Parameter window for the Profile Plot


File name for profile plot
Input feature for profile plot
Comma delimited list of input values. Include a value for the X dimension
Number of samples for each dimension in profile plot and number of bins in the histogram plot
Set plot scales based on histogram plot data
Axes limits when autoscale is not used
Plot using un-normalized data. Patterns are still normalized before classification
Show all data in histogram
Only show patterns that are within this distance of the line of the profile plot.
Label for profile plot

Profile Plot Parameters

Profile Plot file Prefix: X1mlp.profile.plot
Input Dimension to Plot:
Horizontal (X) Dimension: 0
X Label: 1formant
Settings for Other Dimensions (include X)

Number of Intervals per Dimension: 50

✓ Autoscale Plot
Axes Limits:
X Min: -3 Y Min: 0
X Max: 3 Y Max: 1.5
X Step: 1 Y Step: 0.25

☐ Do Not Normalize Data for Plot

Histogram
✓ Show All Data Points
Show points at this distance limit: 0.5

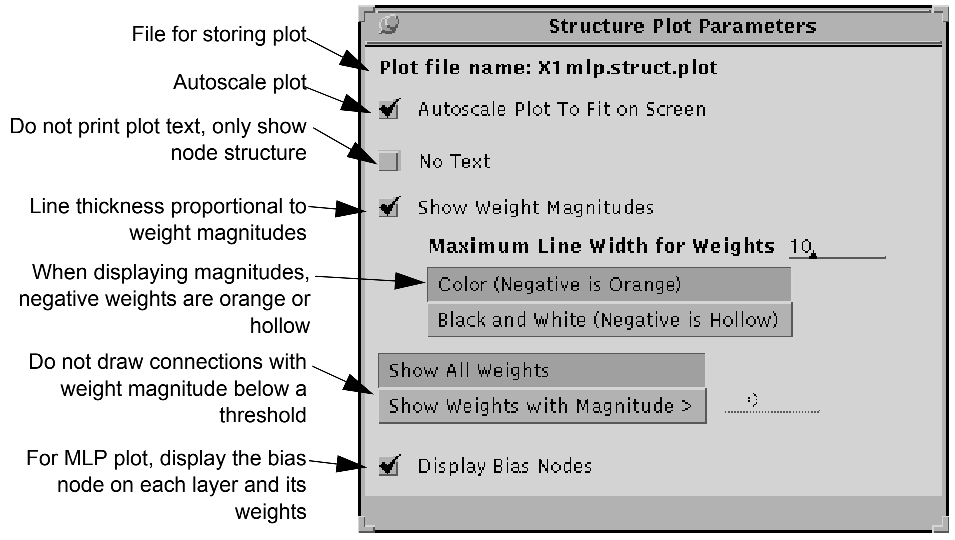
Profile Plot Label:
Norm:Simple Net:2,25,10 Step:0.2

6.3 Structure Plots

Structure plots show the number of nodes in a classifier and the connections between them. In combination with internals plots they can be very helpful in understanding the parameters of a trained classifier. The following LNKnet classifiers have structure plots: BINTREE, MLP, GAUSS, GMIX, RBF, and IRBF. For the other classifiers and clusterers, the internals plot generated with the decision region plot is more informative than a structure plot would be. The appearance of the each structure plot depends on the type of classifier being plotted. Because of these differences, some of the flags on the Structure Plot Parameter window shown in Figure 6.4 are not available for certain plots.

All the structure plots can be automatically scaled to fit in the plot window or they can be plotted using a default scale. Another feature available for all plots is that the node labels can be left off, displaying just the structure of the classifier.

FIGURE 6.4 Structure Plot Parameters


File for storing plot
Autoscale plot
Do not print plot text, only show node structure
Line thickness proportional to weight magnitudes
When displaying magnitudes, negative weights are orange or hollow
Do not draw connections with weight magnitude below a threshold
For MLP plot, display the bias node on each layer and its weights

Structure Plot Parameters

Plot file name: X1mlp.struct.plot

[✓] Autoscale Plot To Fit on Screen

[ ] No Text

[✓] Show Weight Magnitudes

Maximum Line Width for Weights 10

[✓ Color (Negative is Orange)] Black and White (Negative is Hollow)

[✓ Show All Weights] Show Weights with Magnitude >

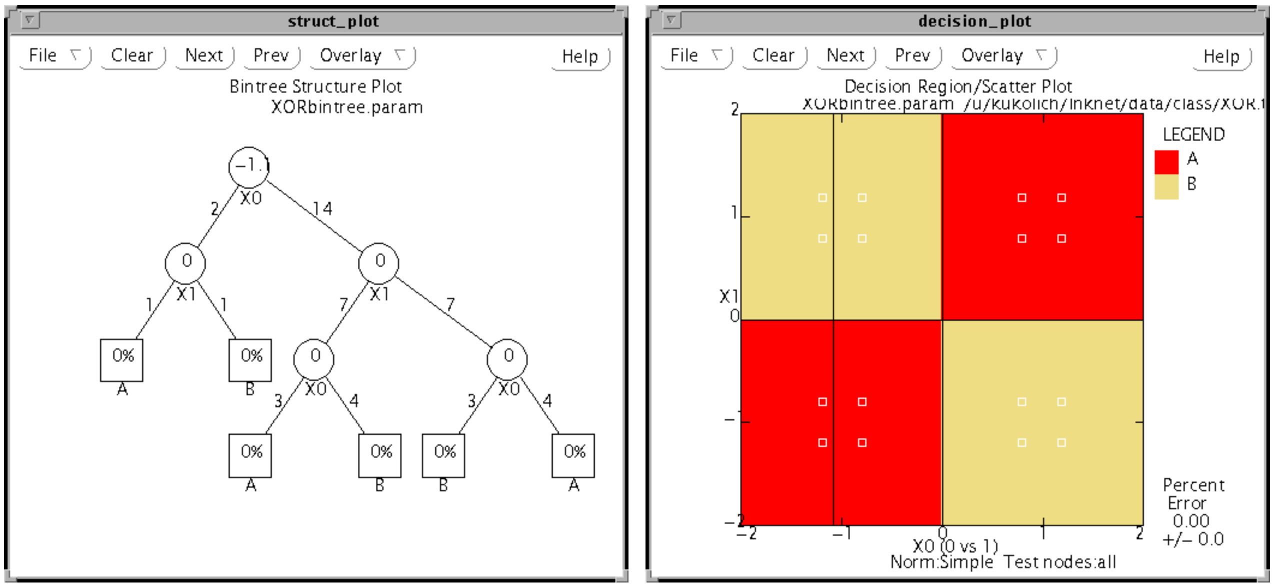
[✓] Display Bias Nodes

6.3.1 Binary Tree Structure Plot

A binary tree classifier has terminal and non-terminal nodes. On a BINTREE structure plot, the non-terminal nodes are represented by large circles. Each non-terminal node has a test of the form $x_i \le C$. The input dimension being tested, x_i, is printed below the circle for a node. The constant C is printed inside the circle. If the node tests use linear combinations of features, that is if the node has a test of the form $\sum_i x_i \beta_i \le C$, no test information is printed with the node. Each non-terminal node has two children. Those training patterns which pass the node test are assigned to the left child of the node. Those patterns that fail are assigned to the right child. The number of training patterns assigned to each child is printed above the line connecting the non-terminal node with that child. The terminal nodes are represented by large squares. Each terminal node is assigned a class. The class label is printed below the square. The percentage of patterns assigned to the terminal node that belong to other classes is printed inside the square.

Figure 6.5 shows a decision region plot and a structure plot for a binary tree classifier trained on the XOR problem.

FIGURE 6.5 Bintree Structure Plot and Internals Plot



6.3.2 Gaussian Structure Plot

For a Gaussian classifier, a set of input nodes is drawn as small black circles at the bottom of the plot. The label for each input feature is printed below each node. A set of output nodes is drawn as large white circles at the top of the plot with the class label for each output printed above each node. The input and output nodes are shown as lines between the nodes in each layer. The type of covariance matrices used in the classifier is printed below the plot. Figure 6.6 shows a structure plot and decision region plot for a Gaussian classifier trained on the XOR problem.

6.3.3 Support Vector Machine Structure and Internals Plots

The support vector machine classifier includes a structure plot that shows how binary classifiers are combined to form a decision for multi-class problems and an internals plot that shows the location of support vectors. Examples of structure plots for the 10-class vowel problem are shown in Figure 6.7. The left side shows the structure plot for the “each class versus other” mode of making multi-class decisions. It shows the ten binary classifiers this creates (middle nodes), each connected to input features and to the output representing the primary class for that binary classifier. The number in each classifier node representes the number of support vectors in that binary classifier. The right side of this figure shows the structure plot for the “all two-class combinations” mode of making multi-class decisions. In this plot there are 45 binary classifiers each connected to the input features and the two output classes the classifier is designed to discriminate. Numbers in the binary classifiers again represent the number of support vectors in each binary classifier.

FIGURE 6.6 Gauss Structure Plot and Internals Plot

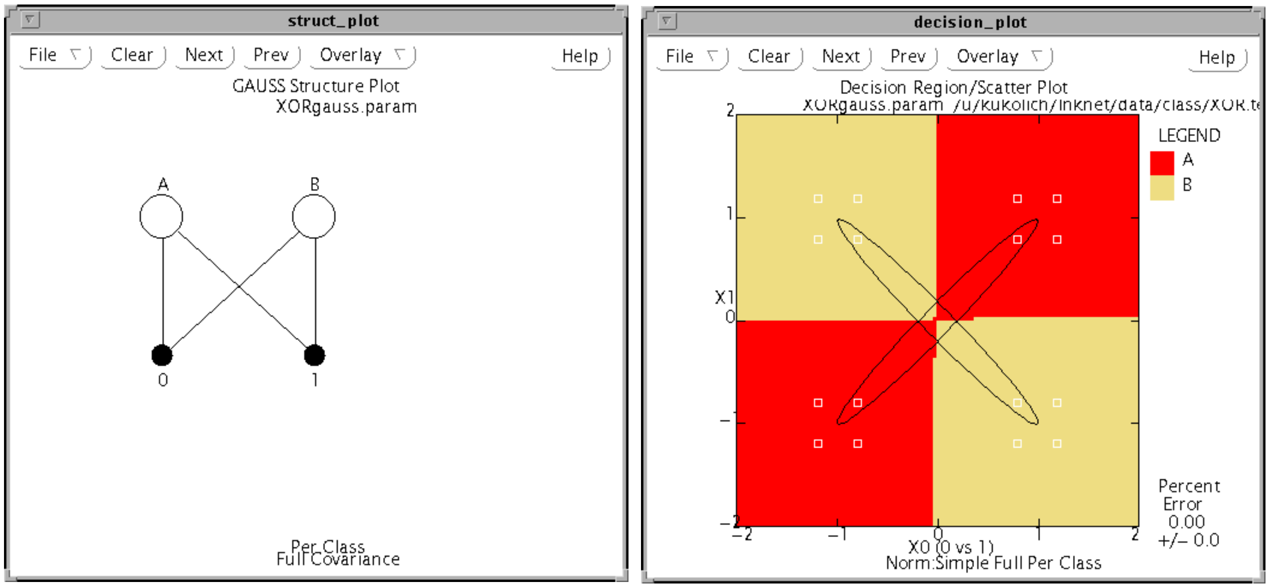
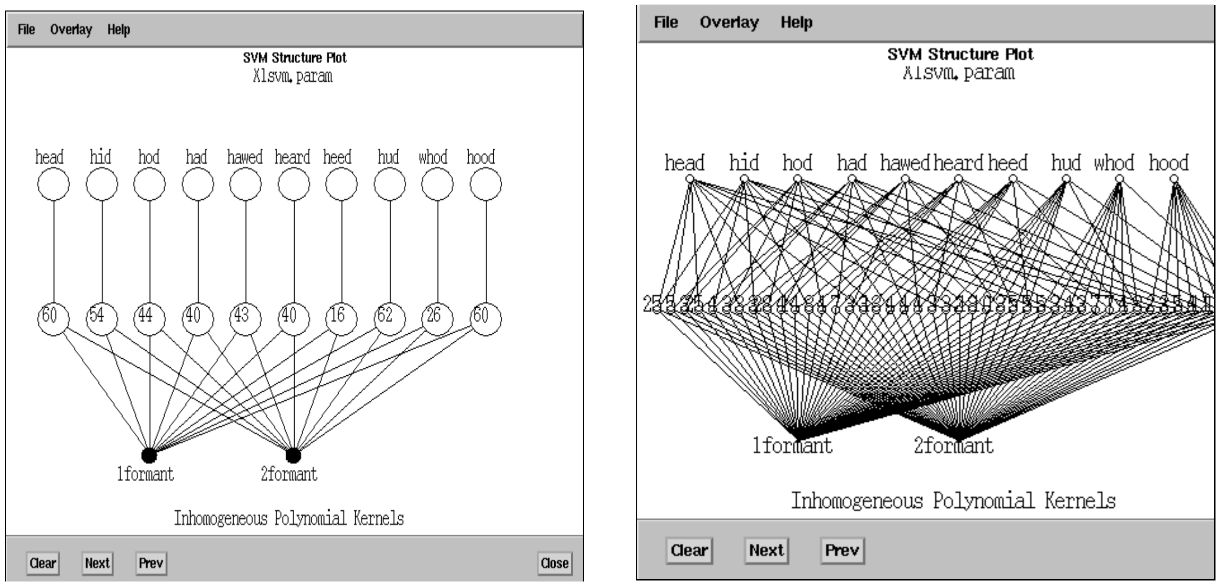
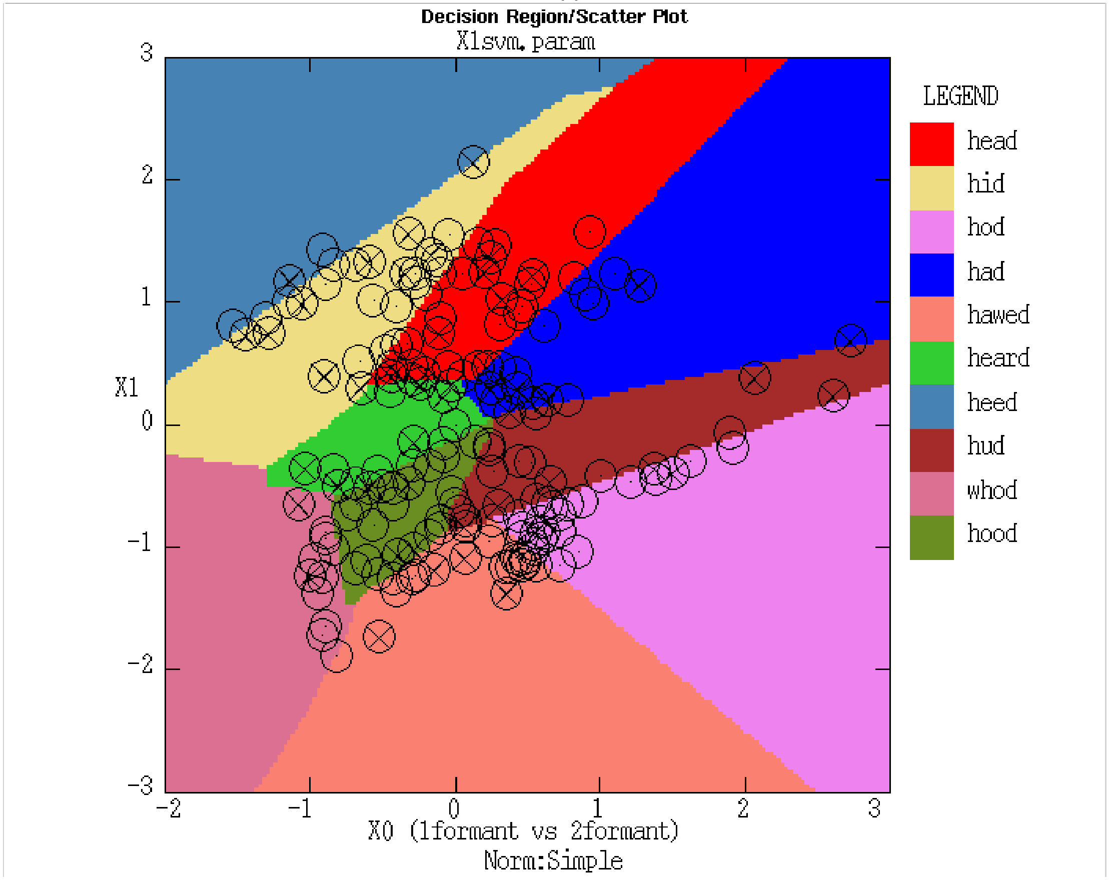


FIGURE 6.7 Support vector machine structure plots for the 10-class vowel problem using the “each class versus others” multi-class mode on the left and the “all two-class combinations” multi-class mode the right.



Support vector machine internals plots show the location of support vectors. An example for the vowel problem is shown in Figure 6.8. Support vectors that are at the Lagrange multiplier upper bound (cbound) are shown as circles and support vectors that are below this bound are shown as circles around an “x”. Internals plots with a linear kernal will show support vector locations only if the “Store linear support vectors” check box is filled in in the “SVM parameters” window shown in Figure 3.18.

FIGURE 6.8 Decision regions for a support vector machine classifier for the vowel problem where the internals shows the locations of support vectors.

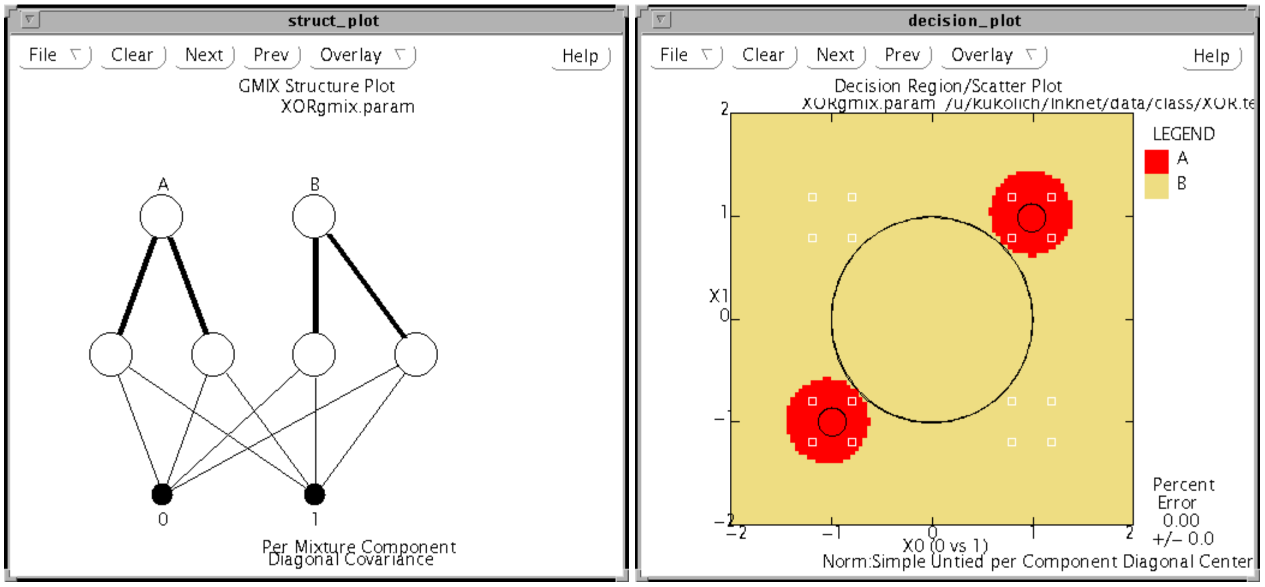


6.3.4 Gaussian Mixture and Radial Basis Function Structure Plots

The Gaussian mixture, radial basis function and incremental radial basis function classifiers use Gaussian functions to provide information about the positions of input patterns. Weighted sums of the Gaussian outputs are then used to make classification decisions. For all three classifiers, input nodes are represented as small black circles at the bottom of the plot. The input feature labels are printed below the input nodes. Lines connect each input to each Gaussian hidden node represented by a large white circle in the middle of the plot. The hidden nodes are connected by lines to the output nodes, represented by large white circles at the top of the plot. The class labels are printed above the output nodes. The connections from the hidden nodes to the output nodes are weighted. The magnitude of the weights can be shown by increasing the thickness of the lines for large weights. The maximum thickness of these lines can be set by the user. This can reveal the importance of particular hidden nodes. Negative weights are drawn as hollow tubes or are colored orange when weight magnitudes are shown. The plot can also only display those connections with weight magnitudes above a certain value. This can help clarify plots for classifiers with many hidden nodes.

For the Gaussian mixture classifier, the hidden nodes can be combined into one tied mixture shared by all the class output nodes or each class can have its own mixture of Gaussian nodes. In the first case all the hidden nodes will be connected to all the output nodes. In the second case the nodes assigned to each class mixture will be connected only to the output node for that class. Figure 6.9 shows a structure plot and decision region plot for a Gaussian mixture classifier trained on the XOR problem.

FIGURE 6.9 GMIX Structure Plot and Internals Plot

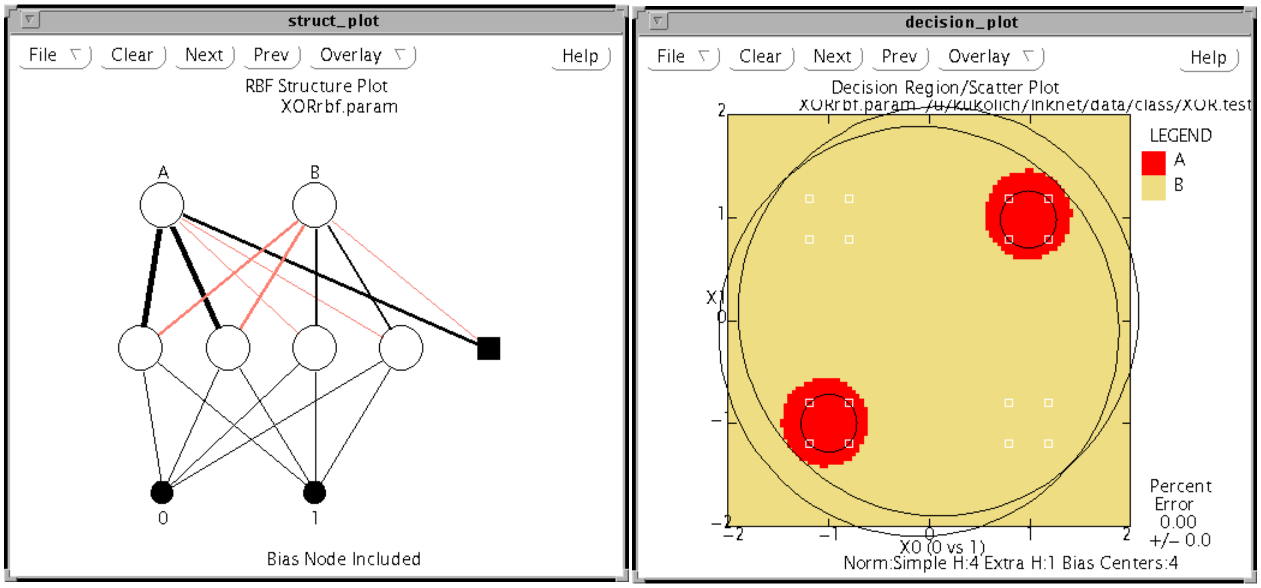


For the two radial basis function classifiers, all hidden nodes are connected to all output nodes. The RBF classifiers can also have a constant bias hidden node. If so, it is represented as a small square beside the other hidden nodes. Figure 6.10 shows a structure plot and decision region plot for a Radial Basis Function classifier trained on the XOR data base.

6.3.5 Multi-Layer Perceptron Structure Plot

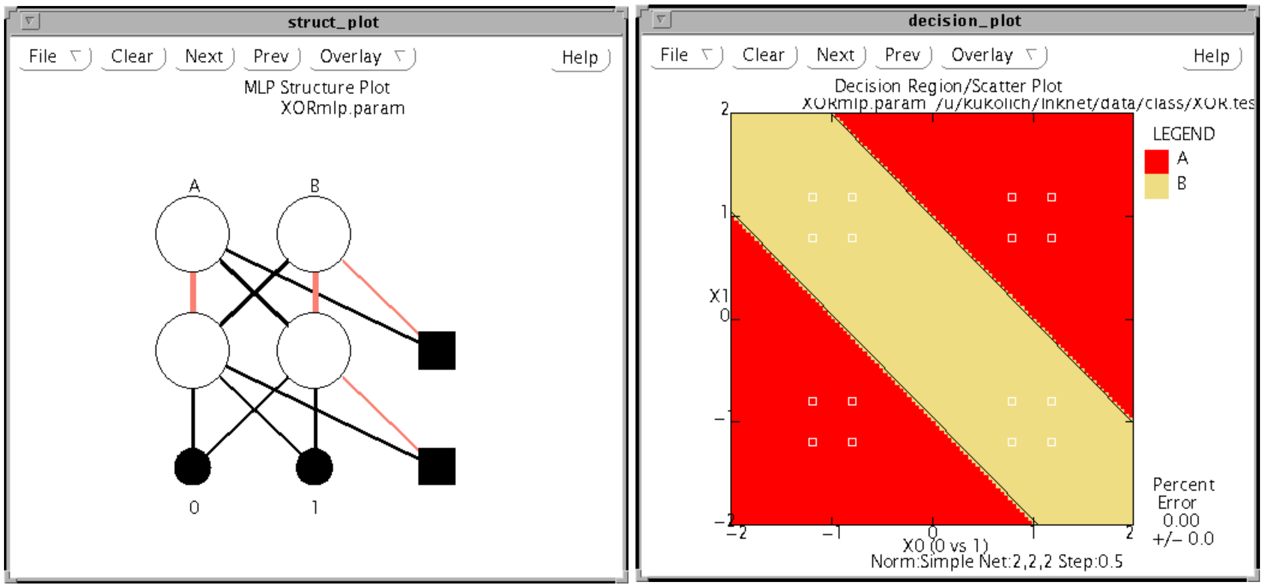
For the multi-layer perceptron, there can be up to 10 layers of nodes with an input layer at the bottom of the plot, as before, and an output layer at the top. All the connections between all the layers are weighted. The weight magnitudes can be shown and connections with very small weights can be made invisible. Negative weights can be shown as hollow lines or they can be colored orange. Each MLP layer has a constant bias node which can be displayed as a small black square beside the weights for that layer. Figure

FIGURE 6.10 RBF Structure Plot and Internals Plot



6.11 shows a structure plot and decision region plot for a Multi-Layer Perceptron classifier trained for 300 epochs on the XOR problem.

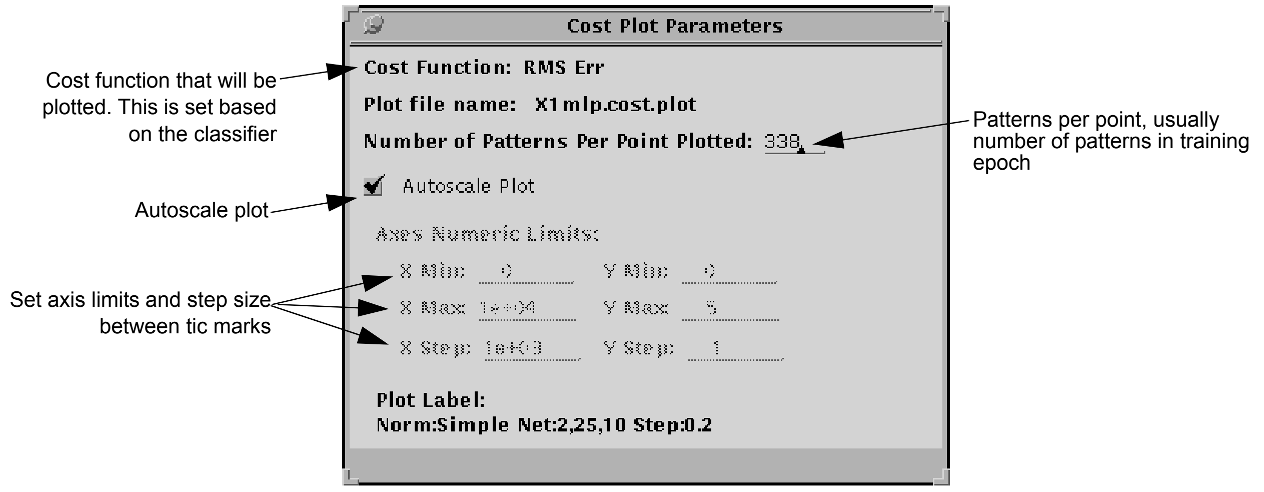
FIGURE 6.11 Multi-Layer Perceptron Structure Plot and Internals Plot



6.4 Cost Plot and Percent Error Plot

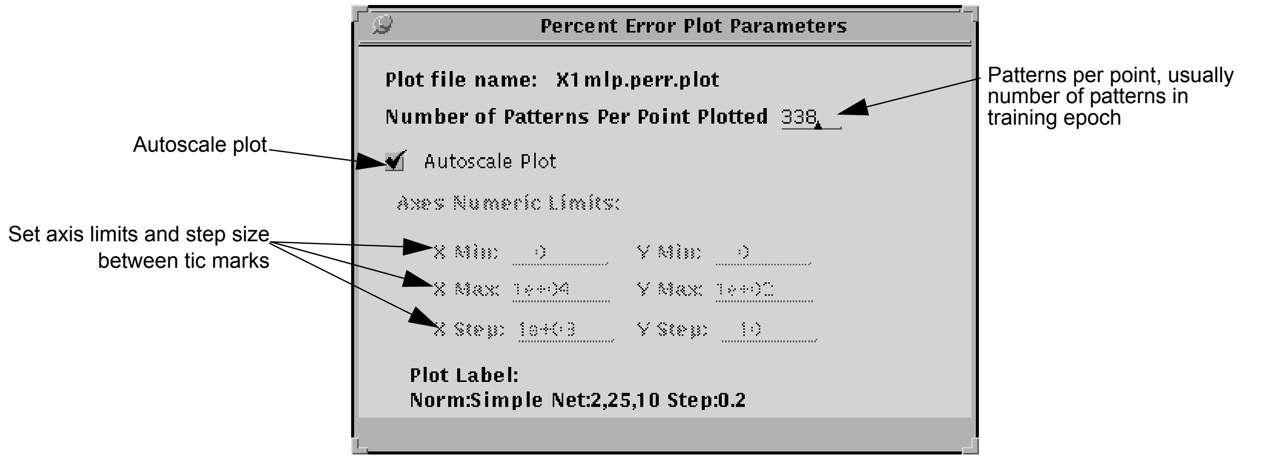
Those classifiers that train incrementally using multiple passes through the data generate training error or results files. These error files can be used to plot the cost and percent error over training. Examples of these plots are in Figure 2.18 and Figure 2.19 on page 27 in the tutorial. For a percent error plot, the classification error rates for each successive group of N patterns is calculated and plotted. A cost plot does the same with the cost information stored in the training error file. The default value for N is the number of patterns in each training epoch. These plots can be scaled automatically based on the range of the data and the number of patterns represented or the user can manually set the axes limits.

FIGURE 6.12 Cost Plot Parameters


Cost Plot Parameters
Cost function that will be plotted. This is set based on the classifier
Patterns per point, usually number of patterns in training epoch
Autoscale plot
Autoscale Plot
Axes Numeric Limits:
Set axis limits and step size between tic marks

Cost Function: RMS Err
Plot file name: X1mlp.cost.plot
Number of Patterns Per Point Plotted: 338
☑ Autoscale Plot
X Min: 0 Y Min: 0
X Max: 1e+04 Y Max: 5
X Step: 1e+03 Y Step: 1
Plot Label:
Norm:Simple Net:2,25,10 Step:0.2

FIGURE 6.13 Percent Error Plot Parameters


Percent Error Plot Parameters
Patterns per point, usually number of patterns in training epoch
Autoscale plot
Autoscale Plot
Axes Numeric Limits:
Set axis limits and step size between tic marks

Plot file name: X1mlp.perr.plot
Number of Patterns Per Point Plotted 338
☑ Autoscale Plot
X Min: 0 Y Min: 0
X Max: 1e+04 Y Max: 1e+02
X Step: 1e+03 Y Step: 10
Plot Label:
Norm:Simple Net:2,25,10 Step:0.2

6.5 Posterior Probability Plot

A posterior probability plot shows how closely continuous classifier outputs approximate the observed posterior probabilities for patterns in a given class. There are two formats for the plot. There is a scatter plot, which plots the observed posterior probabilities in each bin against the average output value for the patterns in the bin. A line is drawn along the diagonal (posterior=output) to indicate where perfect posterior probability outputs should lie. The second form of the plot displays a pair of values for each bin. The observed posterior probabilities are drawn as blue circles with lines indicating plus and minus two standard deviations. The average bin output value is indicated with an X. For both versions of the plot, if the observed probabilities are within two standard deviation of the average bin output, indicated by the line or the X's, then the classifier is adequately modeling the posterior class probabilities.

To generate the plot, test patterns are assigned to bins according to their output values for the given class. These bins can be uniformly placed from zero to 100 or the ends of the bins can be specified using a list of floating point numbers. When specifying the bin ends, remember that there is one more end than there are bins. The quality of the posterior probability fit is determined using a chi squared fit. To insure that there are enough patterns in each bin to make that number valid, a minimum number of patterns per bin (typically 5) is enforced. Bins with too few patterns are combined with neighboring bins until the minimums are met in all the remaining bins.

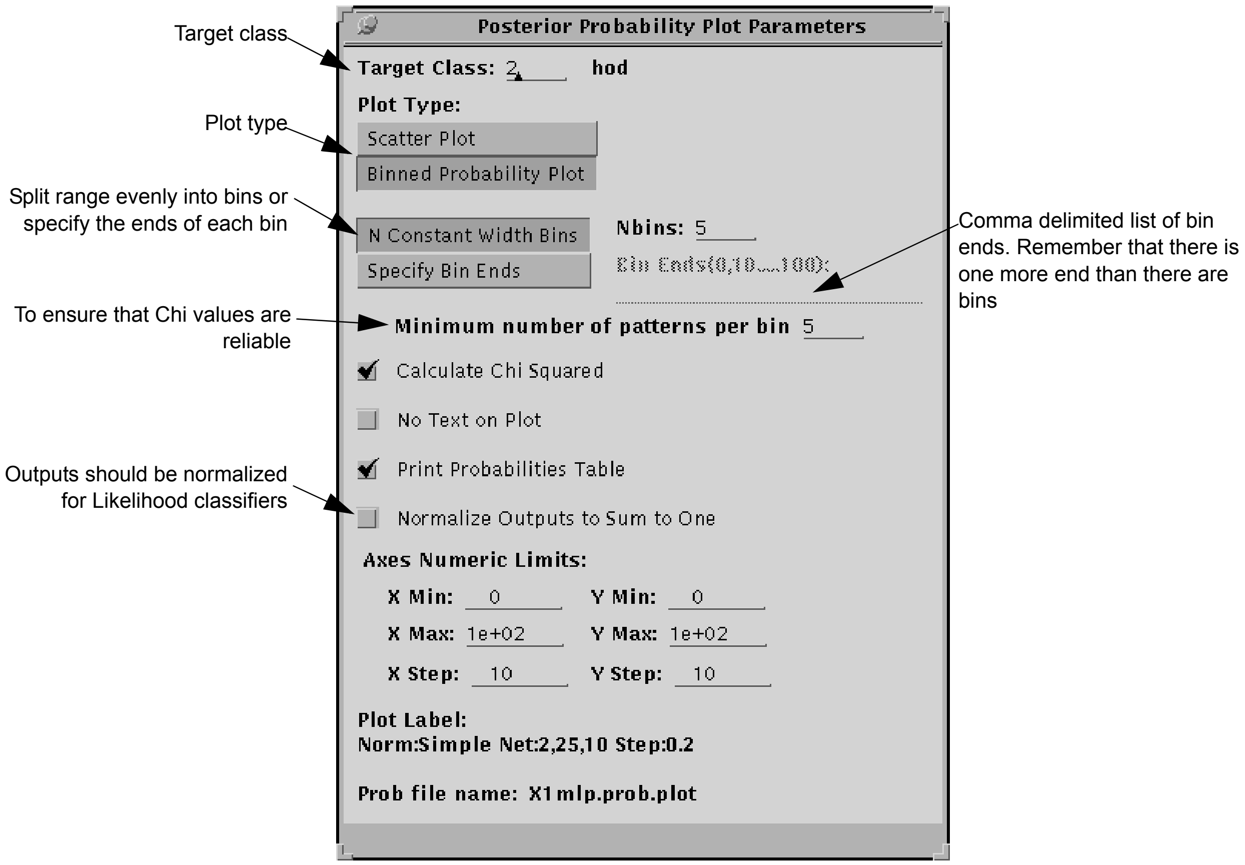
The plotted values can be printed to a table in the log file. A small table showing the chi square value for the bins and the quality or significance of the fit can be added. For this plot, higher significance values indicate better fits. Significance values of less than 0.05 are labeled poor. These chi and significance values are also printed to the experiment notebook file. The axes limits of the plot can be changed to examine smaller sections of the plot area. The default setting are for 0-100% probability on both axes.

The actual number of patterns and the number of patterns in the target class for each bin are printed above the upper two standard deviation indicator. This label text can be left off by selecting **No Text on Plot**. To use the posterior probability plot with likelihood classifiers, it is necessary to **normalize the outputs to sum to one**. Figure 6.14 shows the LNKnet parameter window for the posterior probability plot. Figure 2.20 shows an example of a posterior probability plot.

6.6 ROC (Detection) Plot

Receiver Operating Characteristic (ROC) or detection plots can be generated using the test error files from classifiers that produce continuous outputs. To generate the plot, each test pattern is sorted by its output value for the target class. A threshold is moved over the outputs with patterns below the threshold being rejected and patterns above the threshold being labeled as belonging to the target class. The curve which is plotted shows the percentage of the target patterns *versus* the percentage of non-target patterns that are accepted for each threshold value. When the threshold is at its maximum, the curve is at (0,0) because no patterns are accepted. When the threshold is at its minimum, the curve is at (100,100) because all the patterns are accepted. The area under the ROC curve is a measure of the quality of a classifier for detection problems. An area of 100% corresponds to a perfect ROC curve. In this case, there exists some threshold such that all the patterns in the target class produce an output above the threshold and all the non-target patterns produce an output below the threshold. An area of 50% corresponds to an

FIGURE 6.14 Posterior Probability Plot Parameters



Target class

Plot type

Split range evenly into bins or specify the ends of each bin

To ensure that Chi values are reliable

Outputs should be normalized for Likelihood classifiers

Comma delimited list of bin ends. Remember that there is one more end than there are bins

Posterior Probability Plot Parameters

Target Class: 2 **hod**

Plot Type:

Nbins: 5

Bin Ends(0,10,...,100):

Minimum number of patterns per bin 5

[✓] Calculate Chi Squared

[ ] No Text on Plot

[✓] Print Probabilities Table

[ ] Normalize Outputs to Sum to One

Axes Numeric Limits:

X Min: 0 **Y Min:** 0

X Max: 1e+02 **Y Max:** 1e+02

X Step: 10 **Y Step:** 10

Plot Label:
Norm:Simple Net:2,25,10 Step:0.2

Prob file name: X1mlp.prob.plot

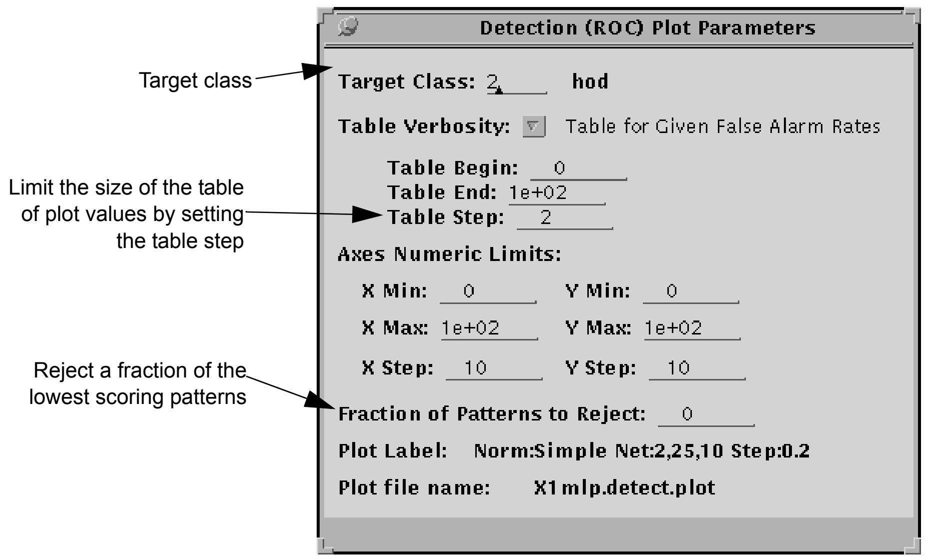
ROC curve for a random classifier, equally likely to return a given output value for any class. Because an ROC curve depends on only one output value, the ROC area does not necessarily indicate the quality of the classifier in classification, where all outputs are compared and the class of the maximum output is chosen.

A table of ROC plot data can be printed to the experiment log file. This table can include interpolated values or it can include the values of all the points in the plot. The ROC area is also printed to the log file and to the experiment notebook file. A fraction of the patterns can be rejected, eliminating from the plot those patterns with the lowest output values. The plot axes limits can be altered to more closely examine a certain section of the ROC curve. Figure 6.15 shows the LNKnet parameter window for the ROC plot. Figure 2.21 shows an example ROC plot generated during the tutorial.

6.7 Rejection Plot

A rejection plot shows the classification error rate on the remaining test patterns as patterns with low maximum output values are rejected. To generate the plot, all evaluation patterns are sorted by their highest output value across all classifier outputs. Patterns whose highest output is below a rejection threshold are rejected and not classified. If all

FIGURE 6.15 ROC (Detection) Plot Parameters


Target class
Limit the size of the table of plot values by setting the table step
Reject a fraction of the lowest scoring patterns

Detection (ROC) Plot Parameters

Target Class: 2 hod

Table Verbosity: Table for Given False Alarm Rates

Table Begin: 0
Table End: 1e+02
Table Step: 2

Axes Numeric Limits:

X Min: 0 Y Min: 0
X Max: 1e+02 Y Max: 1e+02
X Step: 10 Y Step: 10

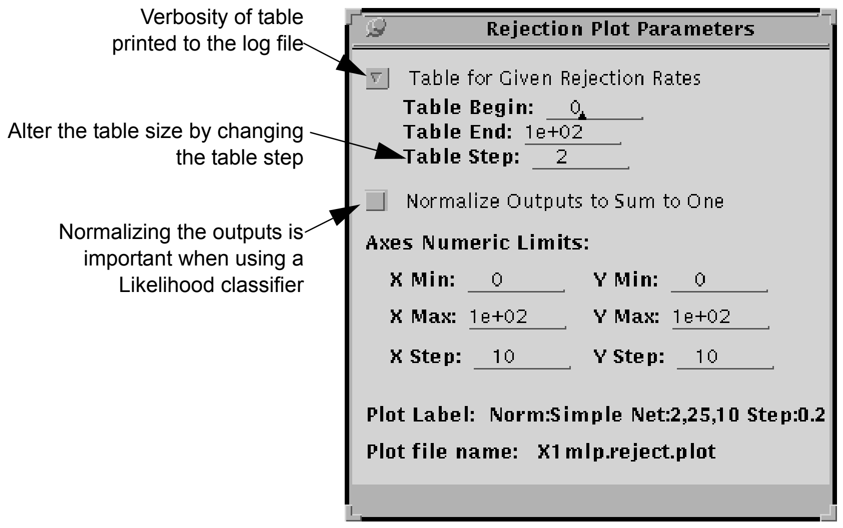
Fraction of Patterns to Reject: 0

Plot Label: Norm:Simple Net:2,25,10 Step:0.2

Plot file name: X1mlp.detect.plot

the patterns which cause errors have low maximum output values, the error rate on the remaining patterns can be reduced by setting the threshold to reject those low scoring patterns. As with the ROC plot, the rejection plot can print a table of plot values to the log file. The outputs for each pattern can be normalized to sum to one, which is important if a likelihood classifier is being used. The scale of the plot can be changed to focus on a particular section of the curve. Figure 6.16 shows the parameter window for a rejection plot. An example rejection plot can be found in Figure 2.22.

FIGURE 6.16 Rejection Plot Parameters


Verbosity of table printed to the log file
Alter the table size by changing the table step
Normalizing the outputs is important when using a Likelihood classifier

Rejection Plot Parameters

Table for Given Rejection Rates

Table Begin: 0
Table End: 1e+02
Table Step: 2

Normalize Outputs to Sum to One

Axes Numeric Limits:

X Min: 0 Y Min: 0
X Max: 1e+02 Y Max: 1e+02
X Step: 10 Y Step: 10

Plot Label: Norm:Simple Net:2,25,10 Step:0.2

Plot file name: X1mlp.reject.plot

6.8 Movie Mode

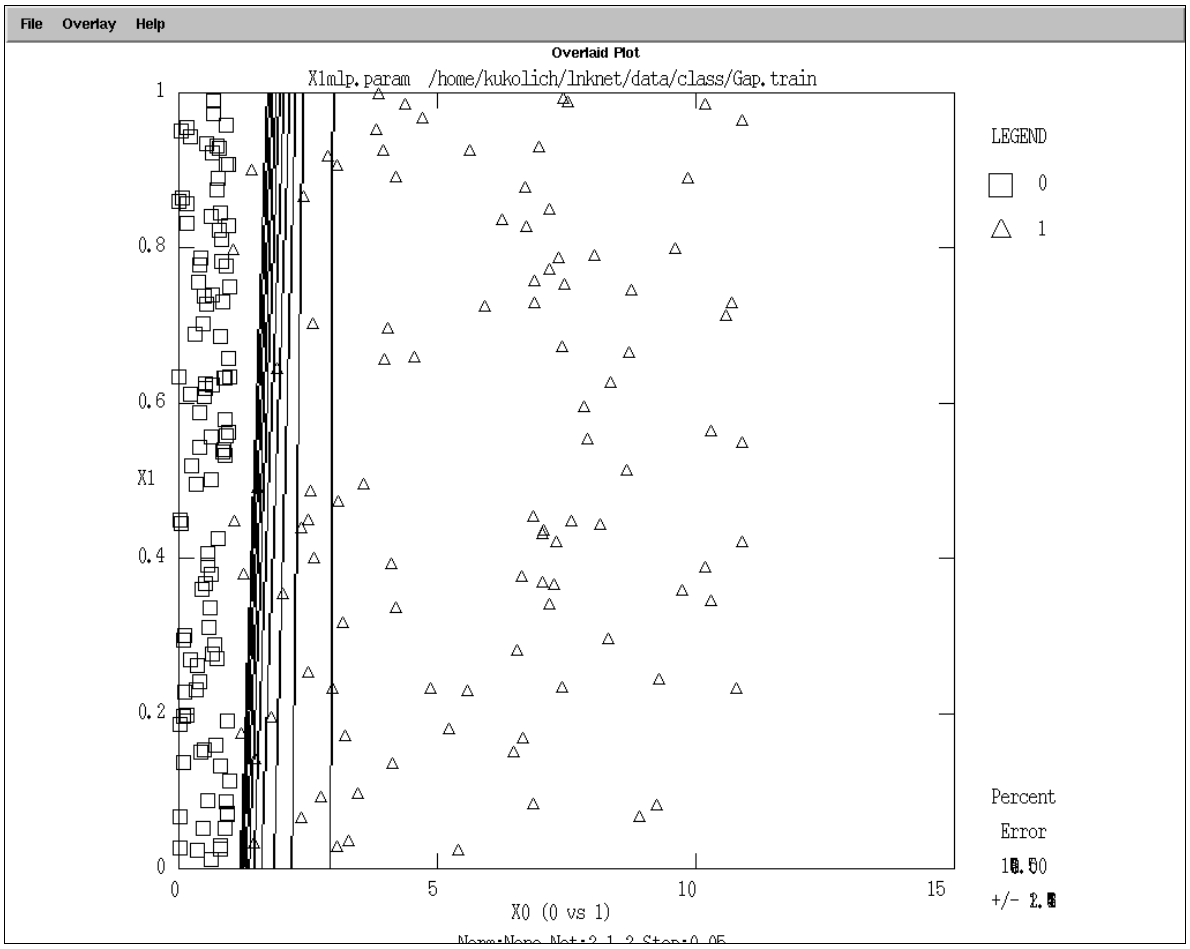
Looking at a series of plots after training as in a flip book movie illustrates changes that occur during training. Such series are often as informative as a slow motion movie to understand what happens during training. LNKnet has a feature which lets you do epoch training but which periodically stores classifier parameters and generates plots every N epochs of training. The plots which are generated are shown together in one olxplot window at the end of training. Olxplot allows you to page forward or backward by typing ‘f’ or ‘p’, as described in Section 6.10 on page 108, showing the plots as frames of a movie. Olxplot also lets you to create an overlay of the training plots and to save that overlay to a plot file.

To generate a movie mode plot select **Use Movie Mode for Training Plots** on the Plot Selection window shown in Figure 6.1 on page 92. Set the **Epochs per Plot** field. Also select **Create Plots for Training and Test Data**. When START or CONTINUE is selected on the Main window, a shell script will be written which trains a classifier in batches of epochs and stores decision region and profile training plots in numbered files for each batch. These plots are not displayed until all the training is complete. Because the movie mode plots are generated as training progresses, movie mode plots cannot be generated using the PLOT ONLY button on the Main window.

Below is an overlay of the training plots for an MLP classifier training on the Gap data base. There are two classes containing patterns uniformly sampled from rectangles of equal height. The rectangle for the first class is one tenth as wide as the one for the second class. The MLP had one hidden node and trained for a total of 10 epochs on 200 patterns per epoch. The training was done in batches of one epoch and thus plots were produced every epoch. The lines on the plot show the position of the decision region boundary defined by the hidden node. It starts at the right and gradually moves to the left to a position between the classes.

The movie mode feature can also be used to train for a large, unknown number of epochs. Training is done in batches even when there are no training plots selected. Select **Movie Mode** on the plot selection window but do not select **Create Plots for Training and Test Data**. Set the number of epochs per plot on the plot window and the maximum number of epochs to train on the algorithm parameter window. When you feel that training has gone on long enough, you can select STOP on the main window. This will terminate the shell script that was controlling the training. Provided you did not stop the training while a parameter file was being written, the stored parameters will reflect the training as of the end of the last batch. Because the training shell script will have stopped, tests of the trained classifier and plots must be run using a new shell script.

FIGURE 6.17 Internals Plots and Scatter Plot for MLP training on Gap.train,1 epoch per batch, 10 epochs total



6.9 Including Plots in Documents

Four approaches can be used to include LNKnet graphs in reports and to make hard-copies of plots. First, a bit map of a plot can be obtained by producing plots in a plot window and using standard tools that capture the screen image to produces a file from this image. It is possible to capture screen images on almost all computers, but this approach does not lead to the highest resolution plots. A second approach is to translate plot files to another format. Plot files can be translated to FrameMaker .mif files (Maker Interchange Format) using the filter provided named plot2mif. The mif versions of a .plot file can then be imported into a FrameMaker document. The plot shown in Figure

6.17 was imported using a MIF file. Alternatively, the plot2ps tool can be used to convert a .plot file to a PostScript file which can be imported into other document preparation programs or printed on a postscript printer. Postscript files can also be converted to many other graphics formats using standard plotting utilities. MIF and PostScript files can be created on the Preview and Print window described in Section 7.1.

A final approach to including plots in reports is to edit the shell script written by LNKnet, adding the -mac and other flags beginning with -mac to all plotting commands. This produces new files containing x,y coordinates of all points in plots suitable for importing into a spreadsheet on a Macintosh or IBM PC. These points can be used by programs such as Delta Graph or Excel to create carefully formatted and annotated plots.

6.10 Manipulating Plot Windows

LNKnet plots are created in separate windows using a program named olxplot. Olxplot uses several menu buttons to manipulate the plot. There are also keyboard short cuts which control the window, as shown in Table 6.1.

When a list of several plots is given to olxplot, as happens in movie mode, the user can use the next and previous commands to examine the plots in the list. Enabling overlay mode allows the user to examine multiple plots simultaneously. Use next or previous to display the first plot in the overlay, turn overlay mode on and select next or previous until all the desired plots are shown. The print and save commands can be used to generate a plot file or hardcopy of the displayed overlay plot.

TABLE 6.1 Olxplot Commands

Menu Item	Keyboard Command	Function
File->Print->(printer name)	h	Print current plot to default printer
File->Print->Printer...	P	Change default printer name
File->Save	s	Save current plot
File->Quit	q	Quit olxplot
Clear	c	Clear plot window
Next	n	Show the next plot or add it to an overlay plot
Prev	p	Show previous plot or add it to an overlay plot
Overlay->Enable	o	Toggle the overlay mode
Overlay->Disable		
Help	?	Display the Help window

CHAPTER 7 Other LNKnet Programs

This chapter describes additional features and tools that are available with the LNKnet package. Some of these tools (the Preview and Print window, C code generation from a parameter file, and committee data base creation) are available from the LNKnet graphical user interface. The others (batch file creation from LNKnet shell scripts, multi-layer perceptron initialization using binary tree parameters, data file creation with normalized patterns, and data exploration with xgobi) must be run from a shell or in a shell script.

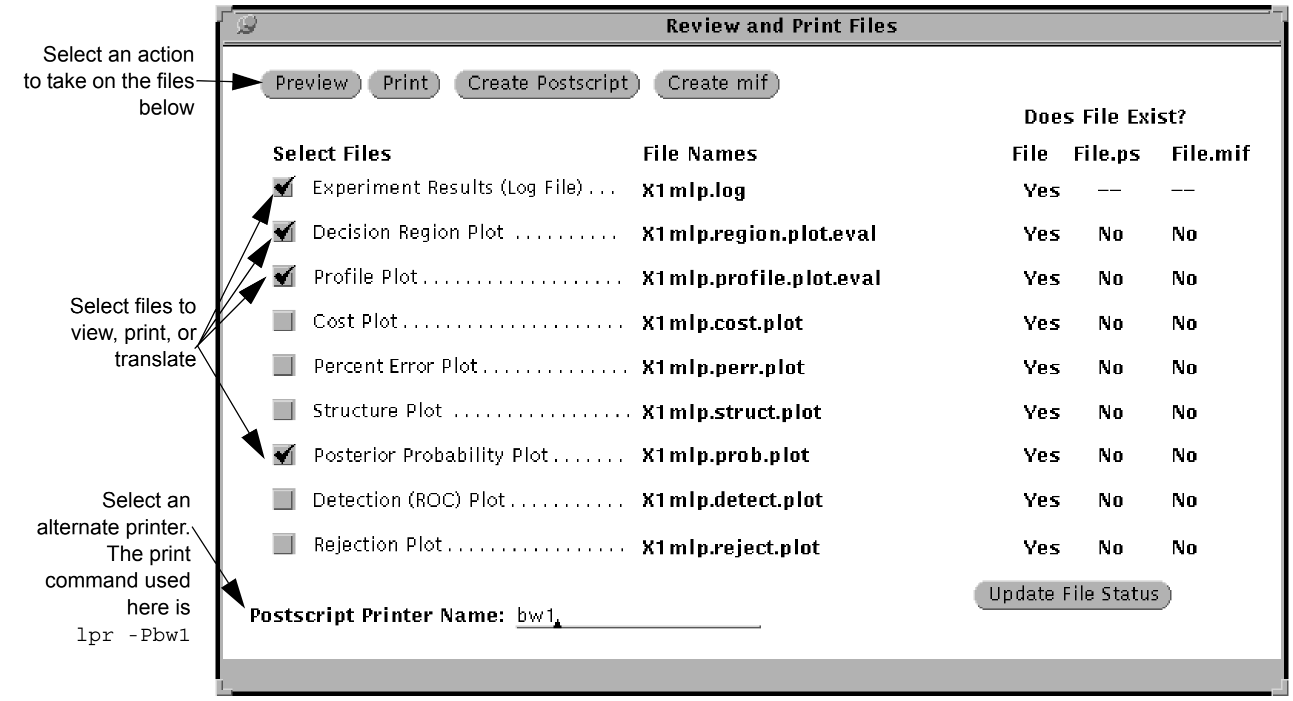
7.1 Preview and Print Window

Experiment log files and plot files can be viewed and printed from the Preview and Print File window shown in Figure 7.1. The available functions are at the top of the window. The user can display the current log file or experiment plot files, or print them to a PostScript printer. The user can also translate plot files to PostScript or Maker Interchange Format (MIF) using the plot2ps and plot2mif programs. The files to be acted on are selected using the check boxes on the left side of the window. The file names are listed beside the check boxes. To the right of the names is a table listing which log and plot files actually exist, as well as which plots have already been translated to PostScript or MIF format. Use the **Update file status** button to update this table after an experiment is run. An optional PostScript printer name is specified at the bottom of the window. When this field is filled, the command `lpr -P<name>` is used to print log or plot files.

7.2 Code Generation Using a LNKnet Parameter File

Many LNKnet users need to include a trained classifier as part of a larger system. Although the LNKnet classifier programs have subroutines that perform classification given a set of classifier parameters, a short stand-alone C subroutine would be much easier to integrate into most systems. LNKnet has a filter program for each classification algorithm that generates C subroutines for pattern classification. Each filter program takes as an argument an algorithm parameter file. The program prints a subroutine, classify(), to the UNIX standard output stream. This subroutine can be called from a C program to classify patterns. The outputs generated by classify() for an input pattern will be the same as those that the algorithm testing program would generate on the same pattern. The difference is that the testing program reads classifier parameters from a file which can be used to continue training. The parameters in the subroutine classify() cannot be changed.

FIGURE 7.1 Preview and Print Window


Select an action to take on the files below
Select files to view, print, or translate
Select an alternate printer. The print command used here is lpr -Pbw1
Review and Print Files
Preview
Print
Create Postscript
Create mif
Update File Status

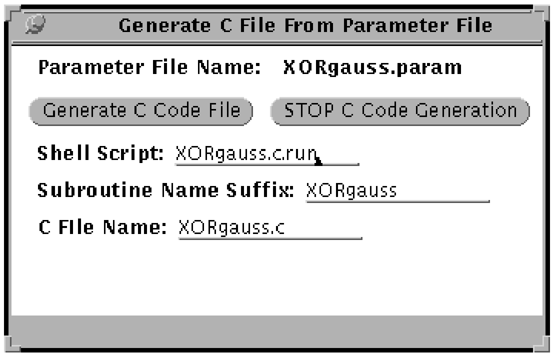
		Does File Exist?		
Select Files	File Names	File	File.ps	File.mif
✓ Experiment Results (Log File) . . .	X1mlp.log	Yes	--	--
✓ Decision Region Plot	X1mlp.region.plot.eval	Yes	No	No
✓ Profile Plot	X1mlp.profile.plot.eval	Yes	No	No
☐ Cost Plot	X1mlp.cost.plot	Yes	No	No
☐ Percent Error Plot	X1mlp.perr.plot	Yes	No	No
☐ Structure Plot	X1mlp.struct.plot	Yes	No	No
✓ Posterior Probability Plot	X1mlp.prob.plot	Yes	No	No
☐ Detection (ROC) Plot	X1mlp.detect.plot	Yes	No	No
☐ Rejection Plot	X1mlp.reject.plot	Yes	No	No

Postscript Printer Name: bw1

Classify() takes a raw input pattern and a pointer to an output vector. It normalizes the inputs and performs any feature selection that was used with LNKnet to train the classifier. For this, the classify routine uses the function normalize() which is included in the generated C file. The routine then calculates classifier outputs using the classifier parameters taken from the algorithm parameter file. These outputs are copied into the output vector and the index of the output with the largest output value is returned as the class of the input pattern.

To create a C subroutine file, first train a classifier as described in the tutorial. Bring up the C File generation window, shown in Figure 7.2, by selecting **C Code Generation...**. on the main LNKnet window. The subroutine name suffix field sets an extension for the classify routine name. For the window in Figure 7.2 the subroutine would be classify_XORgauss(). To have no subroutine suffix, make the field blank. Select **Generate C Code File** to write and run a shell script that creates the C subroutine file. An example parameter file for a Gaussian classifier trained on the XOR problem, the C subroutine classify_XORgauss() produced from it, and a short program that uses the subroutine to generate a decision region plot are included in Appendix C.6 on page 159.

FIGURE 7.2 C code window


Generate C File From Parameter File

Parameter File Name: XORgauss.param

Shell Script: XORgauss.c.run

Subroutine Name Suffix: XORgauss

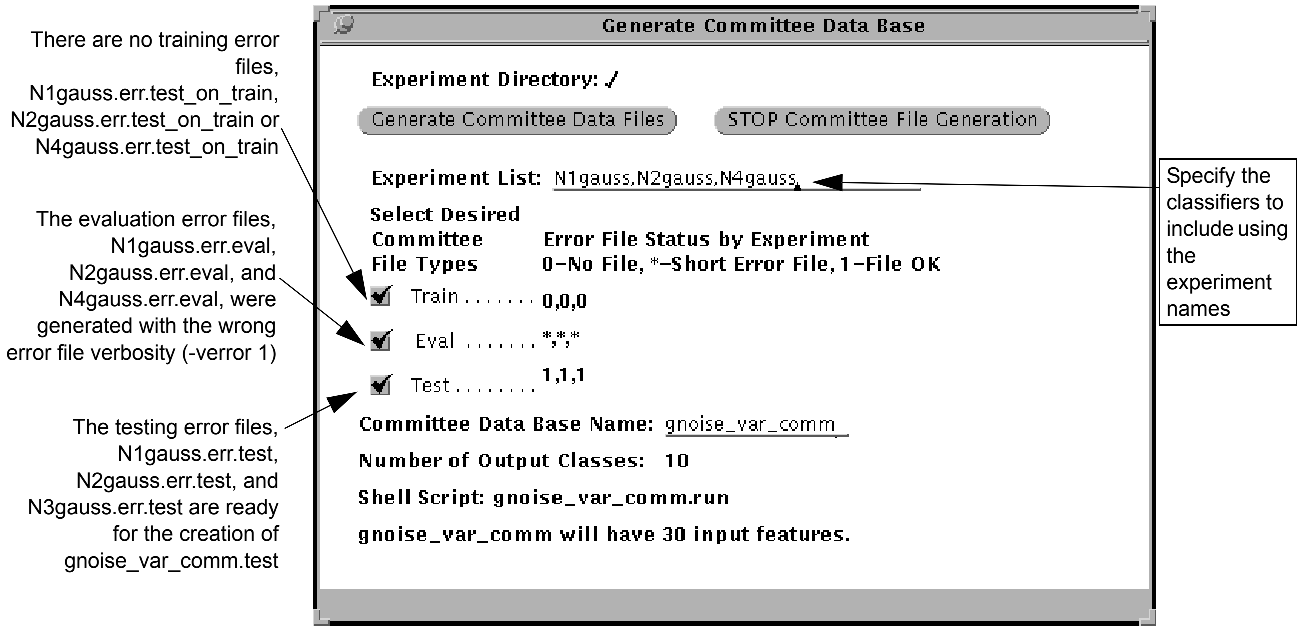
C File Name: XORgauss.c

7.3 Committee Data Base Generation

Because not all classifiers are the same, classification results can differ across classifiers. One way to improve overall classification error rates is to use several classifiers and combine the results. The LNKnet package has a program, committee, which takes a list of classification error files and combines them to create an input data file which can be used for this type of processing. The LNKnet graphical user interface has a window, shown in Figure 7.3, that helps the user use this program.

Because committee uses error files as its inputs, the first step in generating a committee data base is generating these error files. Train a set of classifiers and enter the experiment name for each classifier in the experiment list on the committee data base generation window. For each classifier, generate a testing error file for each data file (training, evaluation, and test). The **error file verbosity** on the Reports window must be set to **Results+Outputs**. Note that the training error file is generated by doing a test on the training data. It is not the error file generated while training the classifier. On the committee data base generation window there is a set of check boxes that specify which error files to generate committee data bases for. Beside each choice is a list which shows the current status of those error files. The lists should be all ones for any file to be generated. A zero (0) indicates a missing file. A star (*) indicates that the file was generated using the wrong error file verbosity. A one (1) indicates the file exists and was created with an error file verbosity that was high enough. If the lists seem to be incorrect, try clicking on them to bring them up to date. On the window in Figure 7.3, the test files are ready, the evaluation files were generated with the wrong verbosity and must be redone, and no training files have been created yet. Selecting **Generate Committee Data Files** writes and runs a shell script that generates a committee data file for each of the requested file types. If any files are missing or the wrong size, the shell script is not run and an error appears. A description file for the data base is also created. The number of classes in the data base is taken from the number of output classes field near the bottom of the committee data base generation window. The class labels are copied from the current data base selected on the data base selection window. The number of input features in the data base is the number of classes times the number of classifiers in the committee. The input labels are generated from the experiment names and output numbers. The input labels are more fully described in Section 8.4 on page 127. The data files and data base description file are stored in the experiment directory.

FIGURE 7.3 Committee Data Base Window


There are no training error files, N1gauss.err.test_on_train, N2gauss.err.test_on_train or N4gauss.err.test_on_train
The evaluation error files, N1gauss.err.eval, N2gauss.err.eval, and N4gauss.err.eval, were generated with the wrong error file verbosity (-verror 1)
The testing error files, N1gauss.err.test, N2gauss.err.test, and N3gauss.err.test are ready for the creation of gnoise_var_comm.test
Specify the classifiers to include using the experiment names
Generate Committee Data Base
Generate Committee Data Files
STOP Committee File Generation

Experiment Directory: ./

Experiment List: N1gauss,N2gauss,N4gauss

Select Desired Committee File Types — Error File Status by Experiment 0–No File, *–Short Error File, 1–File OK

✓ Train 0,0,0

✓ Eval .... *,*,*

✓ Test 1,1,1

Committee Data Base Name: gnoise_var_comm

Number of Output Classes: 10

Shell Script: gnoise_var_comm.run

gnoise_var_comm will have 30 input features.

7.4 Batch File Creation from LNKnet Shell Scripts

The LNKnet system has many classification and plotting programs. To provide flexibility to the users of these programs, they have many command line arguments which must be set each time the programs are called. The LNKnet graphical user interface was written to simplify the creation of shell scripts which call these classification and plotting programs. These shell scripts can be run by LNKnet, as was done in the tutorial. The shell scripts can also be run from a shell window or be called from another script. This allows the user to include LNKnet classifiers in larger experiments.

To run a LNKnet experiment from the shell, first set up the experiment in the LNKnet graphical user interface. On the main window, select **Only store shell script, do not run**. Then select **START**, as you would normally. LNKnet will write a shell script for the experiment. This script can then be edited and started from a shell window or called from another script, just like any other C-shell script. Note that this script is different from normal LNKnet shell scripts because outputs are stored in the log file but are not printed to the shell window and plot files are created but the plots are not displayed on your workstation screen.

An example of using a script in a batch mode would be an experiment which tests different weight step sizes for a Multi-Layer Perceptron. This can be done interactively from the LNKnet interface, but it might be faster to make a single script. The user could then edit the script to make the step size parameter a variable and put the classifier training and testing commands in a loop. The new script would cycle through a list of step size values, training and testing a classifier for each value.

7.5 File Generation with Normalized Data

In many applications, normalization and feature selection are performed as an external preprocessing step. This has the advantage of allowing many functions to be applied to the data before it is written to a file and given to the analysis software. A disadvantage of this method is that each normalization and feature selection method creates a new copy of the data base which can use prohibitive amounts of memory. In LNKnet we have chosen to precalculate and store certain normalization parameters which are then applied on the fly as training or test data is read into a classifier. This slightly increases calculation times but decreases the amount of file storage required to run an experiment. Unfortunately this restricts the preprocessing that can be performed as part of that experiment.

LNKnet has a program, norm_apply, which takes a normalization parameter file, feature selection specification, and a data file. The program applies the normalization and feature selection to each data pattern and then writes the modified pattern to a new data file. The UNIX manual page for norm_apply describes all the flags used by the program.

7.6 Multi-Layer Perceptron Initialization from Binary Tree Parameters

The multi-layer perceptron is a flexible algorithm suitable for solving classification problems, detection problems, and input-output mapping problems. In testing, the MLP classifier can produce outputs for an input pattern quickly and those outputs can be useful in determining a confidence measure for the network's response. Unfortunately, the MLP classifier trains slowly. One way to avoid this problem is to initialize the MLP using parameters from another algorithm. The program bintree2mlp initializes the first layer weights of a Multi-Layer Perceptron using the non-terminal node tests of a Binary Tree classifier. On some problems, this method greatly reduces the training time for the MLP classifier.

Take as an example the LNKnet disjoint data base. In this two class data base, the data in class 1 is found in two squares surrounded by the data in class 0. While most classifiers can correctly classify most of the data, the class 0 patterns which lie between the two class 1 squares are often misclassified. A binary tree classifier with 6 non-terminal nodes can achieve an error rate of 1.2% on the disjoint testing data. The structure and decision region plots for this classifier are shown in Figure 7.4. A multi-layer perceptron classifier with 6 hidden nodes requires 1000 epochs of training to achieve a similar error rate of 1.6%. This training time can be cut to 10 epochs by initializing the multi-layer perceptron using the binary tree parameters. The structure plot and decision region plot for this classifier are shown in Figure 7.5.

FIGURE 7.4 Binary Tree Structure plot and Internals Plot

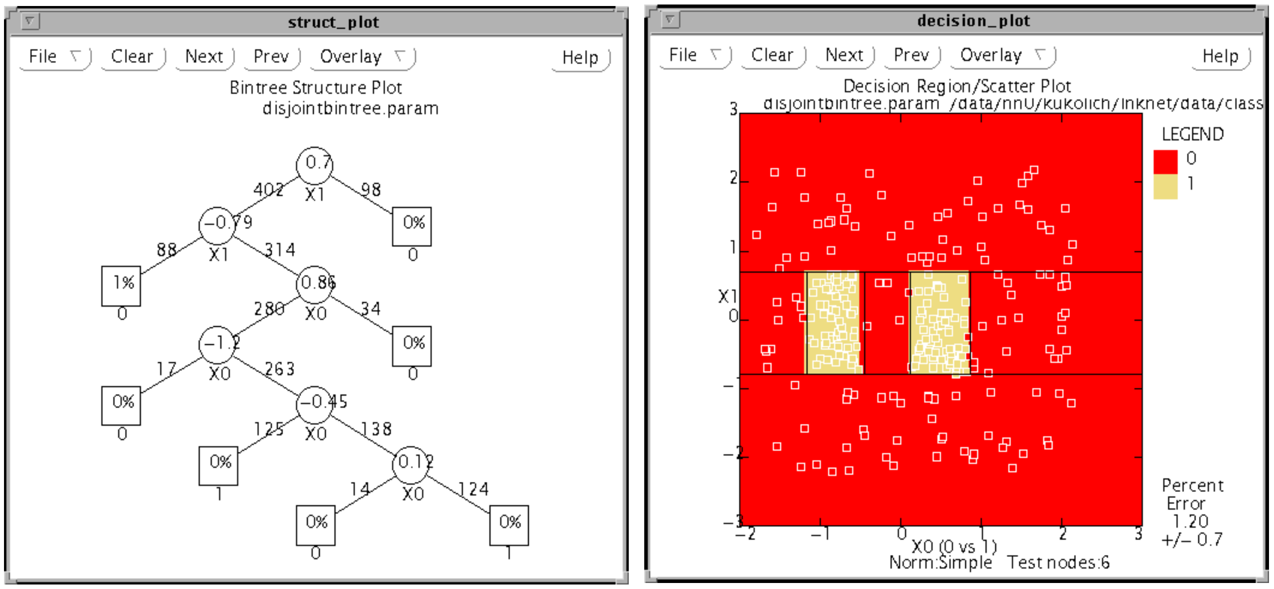
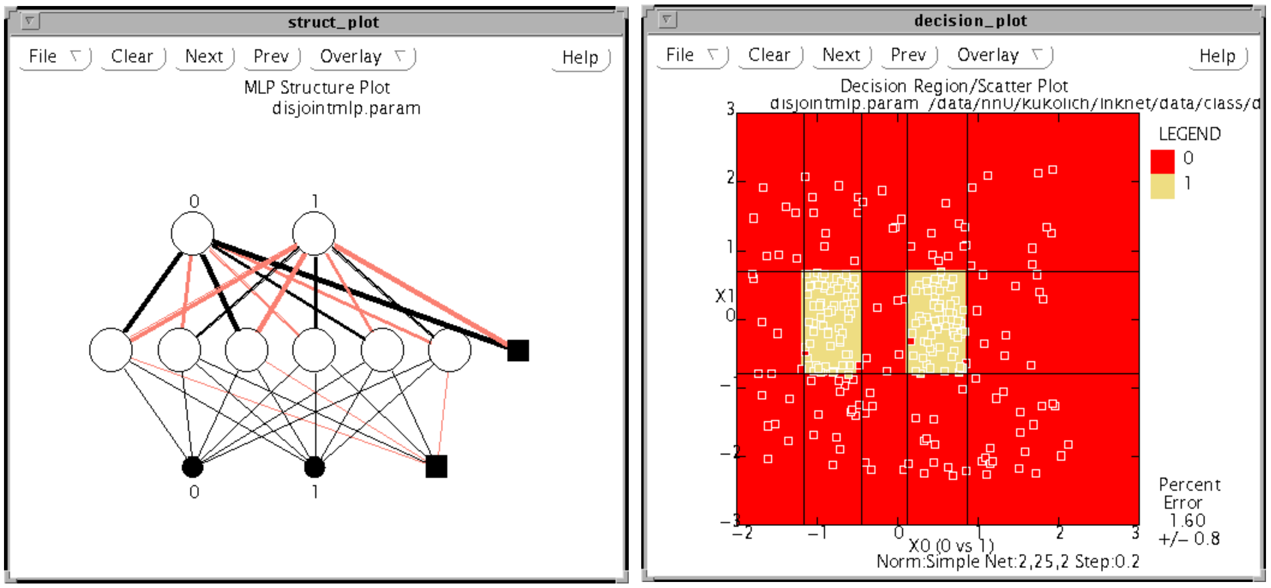


FIGURE 7.5 Initialized MLP Structure Plot and Internals Plot


struct_plot
File
Clear
Next
Prev
Overlay
Help
MLP Structure Plot
disjointmlp.param
0
1
0
1
decision_plot
Decision Region/Scatter Plot
disjointmlp.param /data/nnU/kukolich/lnknet/data/class/d
LEGEND
0
1
X1
X0 (0 vs 1)
Norm:Simple Net:2,25,2 Step:0.2
Percent Error 1.60 +/- 0.8

To perform this experiment, first select the LNKnet disjoint data base on the data base selection window. The LNKnet data base directory is $LNKHOME/data/class. Next, change the **experiment name prefix** to **disjoint** and select the binary tree classifier. On the BINTREE parameter window, select **Maximum Number of Nodes during Testing** and set the number to **6**. Train and test the binary tree classifier.

Next select the multi-layer perceptron classifier. On the MLP parameter window set the number of epochs to **1000** and the node structure to **2,6,2**. Train and test the multi-layer perceptron classifier by selecting **START**. It may be interesting to display training plots using movie mode. A plot every 50 or 100 epochs should be sufficient.

Now, initialize a multi-layer perceptron classifier using the binary tree classifier. First, run the following command in your shell window:

```
bintree2mlp -bin_fparam disjointbintree.param \
-prune_tree -max_nodes 6 \
-mlp_fparam disjointmlp.param -nodes 2,6,2
```

The parameter file disjointmlp.param now holds a multi-layer perceptron with first layer weights that match the binary tree decision node lines. The other weights in the network are set to random values and need to be trained. On the MLP parameter window, set the number of epochs to **10**. Display the MLP weight parameter window and select **Use Step size list for weights in each layer**. Set the **Step size list** to **0,.1**. This freezes the first layer weights to the initialized values while allowing the other weights to be trained. Display the MLP node parameter window and select **Specify sigmoid steepness for each layer**. Set the **Sigmoid Steepness List** to 50,1. This makes the sigmoid functions for the hidden layer act like the non-terminal node tests used in the binary tree classifier. Now train the initialized multi-layer perceptron by selecting **CONTINUE** on the main window. The pre-initialized MLP classifier achieves the same error rate as the randomly initialized MLP classifier using 1/100 the epochs of training.

7.7 Data Exploration with Xgobi

Xgobi is a public domain plotting package which permits exploration of multi-dimensional data bases. It displays one, two, and three dimensional scatter plots of ASCII data files of a format similar to that used by LNKnet. The source code and documentation for xgobi can be found in the LNKnet software release under $LNKHOME/src/xgobi. The software can also be obtained using an anonymous ftp to lib.stat.cmu.edu. Compressed tar files of the current xgobi release are found in general/XGobi at that site.

LNK2gobi creates several Xgobi label files to facilitate exploration of LNKnet classification data files. To use LNK2gobi, first create a LNKnet data file and data base description file, as described in Section 8.1. The data base description files can be created using the LNKnet graphical user interface, as described in Section 5.3. Then run LNK2gobi to generate xgobi label files. Finally, run xgobi on the data file. For example, use these commands to run xgobi on the normalized vowel data used in the tutorial:

```
> cd ~/Tutorial

> LNK2gobi -fdata vowel.train -fdesc vowel.defaults
> xgobi vowel.train
```

The data file and the data base description file are unaltered by LNK2gobi. The following files are created:

TABLE 7.1 Files created by LNK2gobi

vowel.train.col	Labels for each input feature
vowel.train.colors	Colors for each pattern based on the class
vowel.train.row	Class labels for each pattern
vowel.train.glyphs	Plotting shapes for each pattern based on the class

CHAPTER 8

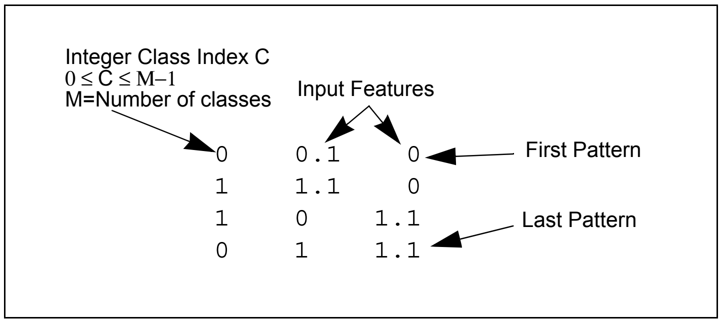
Input and Output File Formats

The LNKnet system uses many files to store classification data, normalization and feature selection parameters, experiment commands and results, classification algorithm parameters, plots, generated C subroutines and committee data bases. This chapter describes the default names and the formats of the files created and used by LNKnet programs.

8.1 Input Data File Formats

Before running an experiment it is necessary to create a data base of classification patterns. A LNKnet data base has a data base description file and one or more data files. A data base usually also has normalization files and feature selection files. When cross validation experiments are run there can also be a cross validation file, although this is usually not necessary. See Section 5.7 for a discussion of cross validation files.

FIGURE 8.1 File format for train, eval, and test data files



8.1.1 Train/Eval/Test Data Files

A LNKnet data base usually has three data files. There is a training file, an evaluation file, and a testing file. The training file is for training the classifier. The evaluation file is usually used for evaluating the classifier each time it is trained to select the classifier size and tune classifier regularization parameters such as K for a K nearest neighbor classifier and network structure for a multi-layer perceptron classifier. The test file is saved for generating the final generalization error rate that should be reported using data never used during training.

The format of these three files is identical and is shown in Figure 8.1. The files are ASCII. There is one pattern per line. On each line, the first number is an integer for the class. The class numbers go from zero to the number of classes minus one. The remaining numbers are floating point values of input features of the pattern. The numbers on each line must be separated by at least one space or tab. Every line, including the last, should have a carriage return at its end.

A data base is selected on the LNKnet data base window which is described in Section 5.3. The names of the data files are generated by adding extensions to the data base name. The default extensions and the resulting file names for the pbvowel data base are given in Table 8.1. Figure 8.1 shows five patterns taken from the pbvowel

TABLE 8.1 Data Files for pbvowel data base

File Type	Suffix	File Name
Training	.train	pbvowel.train
Evaluation	.eval	pbvowel.eval
Testing	.test	pbvowel.test

training data file.

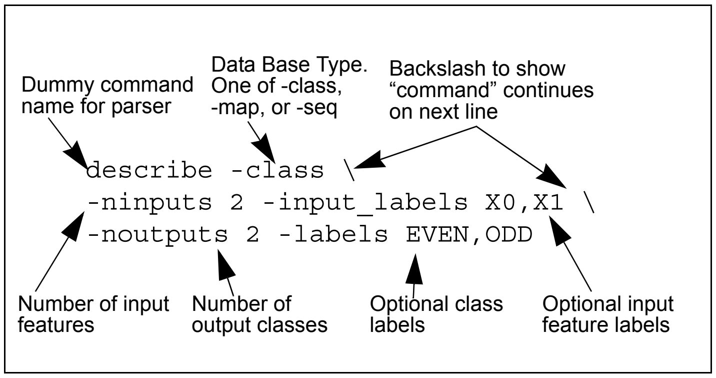
FIGURE 8.2 Five Patterns taken from pbvowel.train

```
0 228. 460. 3300. 3950. 3
1 205. 600. 2550. 4000. 3
3 220. 820. 2180. 2850. 2
6 228. 460. 900. 2830. 2
0 150. 300. 2240. 3200. 1
```

8.1.2 Description Files

When a data base is selected in LNKnet, the program displays the number of inputs, the number of classes, the labels for the input features, and the labels for the classes. This information is obtained from the description file for each data base. These files have the suffix ".defaults". The recommended approach to generating description files is to use the **Generate Description File** popup window which is described in Section 5.3.

FIGURE 8.3 File format for a description file


Dummy command name for parser
Data Base Type. One of -class, -map, or -seq
Backslash to show "command" continues on next line
Number of input features
Number of output classes
Optional class labels
Optional input feature labels

```
describe -class \
-ninputs 2 -input_labels X0,X1 \
-noutputs 2 -labels EVEN,ODD
```

A description file has the same general format as the .lnknetrc file. There is a dummy command name followed by a list of flags and their values. The flags in the description file must match those in the example below. The flag **-ninputs** is followed by the number of input features. The flag **-noutputs** is followed by the number of classes. The flag **-labels** is followed by a comma delimited list containing the names of all classes beginning with class zero. The flag **-input_labels** is followed by a similar comma delimited list of labels for the input features starting from feature zero. The delimiter for the label lists can be comma(,), colon(:), or dash(-). A label list ends at the first space encountered. None of these characters can be used in class labels or input feature labels. Table 8.2 gives examples of acceptable and unacceptable labels. A data base description

TABLE 8.2 Acceptable and Unacceptable labels for string lists

Acceptable	Unacceptable	Reason for Unacceptability
10/jan/94	10:jan:94	colons (:)
1cepstra	1 cepstra	space
delta_cepstra	delta-cepstra	dash (-)
July_4_95	July 4, 1995	spaces and commas

file can also have a flag for the type of data base. The data base type flags are **-class** for static pattern classification data bases, **-map** for input/output mapping data bases, and **-seq** for sequence classification data bases. Only static pattern classification data bases can be used with the programs described in this User's Guide. Description files are read using the command line argument parsing routines used by all LNKnet programs. Like a UNIX command, the description file flag list must either be one line long or every line but the last must end with a backslash (\) immediately followed by a carriage return. When using backslashes, it is important to remember to put spaces at the end of each comma delimited list. The backslash and carriage return are not interpreted as spaces

For LNKnet to find the description of a particular data base, the name must be `<data_base>.defaults.` For example, the description file for `pbvowel` is `pbvowel.defaults.`

FIGURE 8.4 The description file for pbvowel, pbvowel.defaults

```
describe -class -ninputs 5 -noutputs 10 \
-labels heed,hid,head,had,hud,hod,hawed,hood,whod,heard \
-input_labels pitch,1formant,2formant,3formant,MFC
```

8.1.3 Normalization Files

Normalization files contain normalization parameters calculated based on training data. These parameters are applied to training and test patterns during classification experiments and for plot generation. A copy of the normalization parameter file is stored in each classification and clustering parameter file to insure that the same normalization is used throughout an experiment. The parameters stored in the file include the number of inputs and outputs for the data base, the number of patterns in the training file, the type of normalization, and those parameters necessary to perform the normalization. Normalization parameter files can be created from the normalization file generation window as described in Section 5.4. Normalization file names start with the data base name followed by .norm. There is an extension for the type of normalization, as shown in Table 8.3.

TABLE 8.3 Normalization File Names for vowel data base

Normalization Type	File name extension	file name (pbvowel data base)
simple	.simple	pbvowel.norm.simple
principal components analysis	.pca	pbvowel.norm.pca
linear discriminant analysis	.lda	pbvowel.norm.lda

8.1.4 Feature List Files

A feature list file contains an ordered list of input feature numbers. The order is based on a feature search of the type described in Section 5.5. The file contains the number of input features, the number of features in the best feature list found during the search, the list of features, and the classification error rates of the feature sets on the list. Feature list file names start with the data base name followed by an extension for the search direction, a character for the normalization file type, and .param. Table 8.4 shows some feature list file names for the pbvowel data base.

TABLE 8.4 Feature List file names for pbvowel data base

Search Direction	Direction Extension	Normalization Type	Normalization Character	File Name
forward	.for	none	.N	pbvowel.for.N.param
forward	.for	simple	.S	pbvowel.for.S.param

TABLE 8.4 Feature List file names for pbvowel data base

Search Direction	Direction Extension	Normalization Type	Normalization Character	File Name
backward	.back	PCA	.P	pbvowel.back.P.param
forward and back	for_bk	LDA	.D	pbvowel.for_bk.D.param

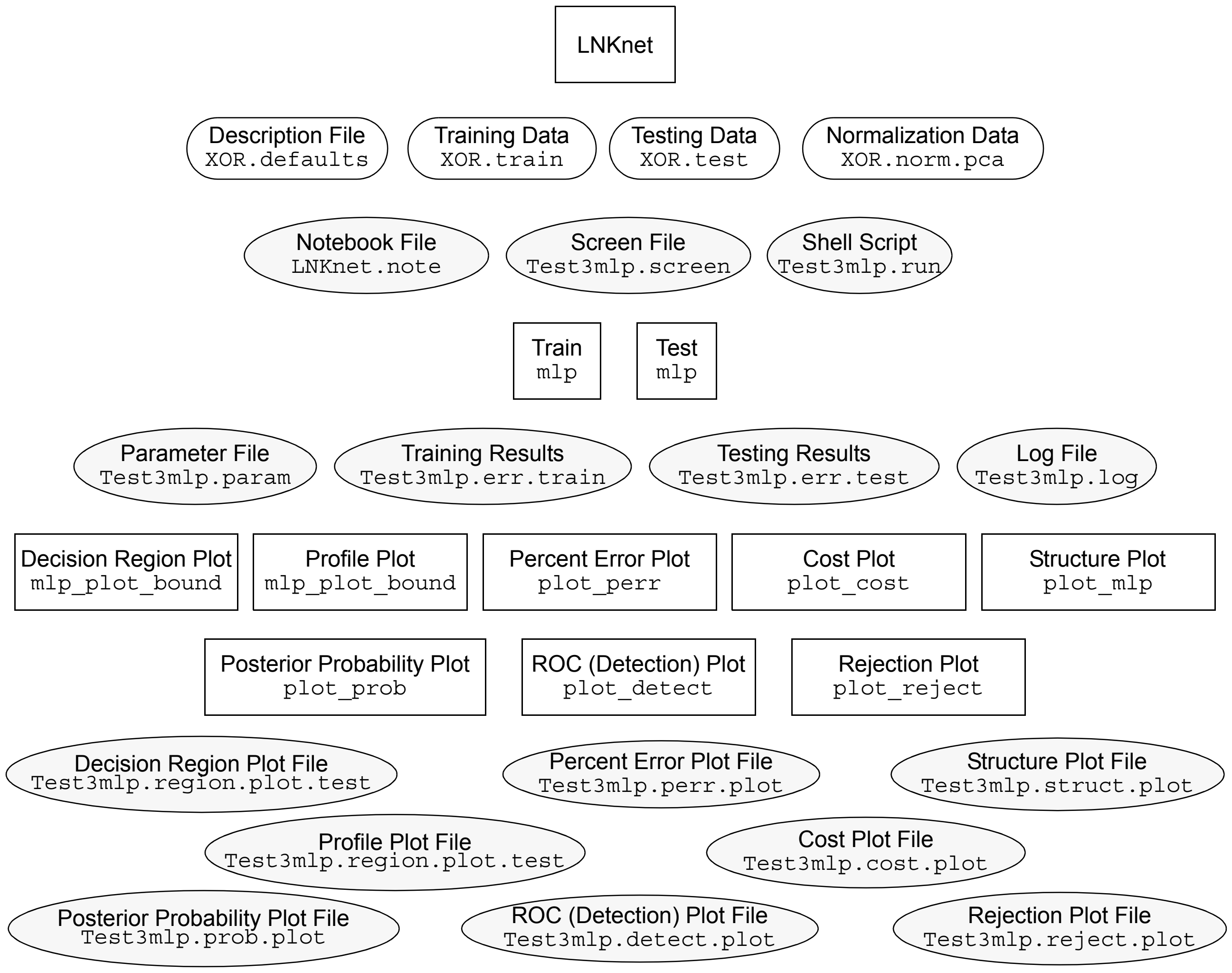
8.2 Files Generated by LNKnet

LNKnet programs generate several types of files during an experiment. There are shell scripts, screen files, a notebook file, log files, parameter files, error files, and plot files. Shell scripts and screen files are produced by LNKnet itself. A notebook file is created by LNKnet and added to by the shell scripts. Log files, parameter files and error files are generated by the classifiers. Plot files are generated by the plot programs associated with the classifiers.

The files generated during a classification experiment are shown in Figure 8.5 using shaded ellipses. Data base files in this figure are shown using unshaded rounded edged rectangles, and LNKnet programs are shown as rectangles. LNKnet creates a screen file which stores the settings for all LNKnet windows at the time of the experiment for possible restoration and continuation of the experiment. LNKnet writes a shell script which includes calls to all of the programs needed in an experiment. Finally, LNKnet appends an entry to the notebook file briefly describing the experiment. In a normal experiment, a classifier model is created, trained, and stored. Pattern by pattern classification results may be stored in an error file. A summary of the training is printed to the terminal window in which the LNKnet program was started. The summary is also printed to a log file and a one line training entry may be added to the notebook file. After training, the stored classifier is tested on a new data file. The pattern by pattern results are stored in an error file and a summary is written to the terminal window and appended to the log file. A one line test entry is added to the notebook file. Finally, decision region and profile plots are generated based on the classifier parameter file and the test data file. Structure plots are generated based on the classifier parameter file. Percent error and cost plots can be generated based on the training error file. Posterior probability, ROC, and rejection plots can be generated based on the test error file. The posterior probability and ROC plots each add one line to the notebook file.

The file names are generated by LNKnet using a few simple rules. The file names for an experiment all start with the same experiment name. The name is the experiment name prefix followed by the classifier name. If the experiment name prefix is Test3, and the classifier is a Multi-Layer Perceptron classifier, the experiment name is Test3mlp. Each file type has its own default extension as described below and shown in Figure 8.5. The notebook file is not part of a particular experiment. It is usually called LNKnet.note but the name can be changed by the user. It is stored in the directory in which LNKnet was started.

FIGURE 8.5 Files used and created in a LNKnet experiment


LNKnet
Description File XOR.defaults
Training Data XOR.train
Testing Data XOR.test
Normalization Data XOR.norm.pca
Notebook File LNKnet.note
Screen File Test3mlp.screen
Shell Script Test3mlp.run
Train mlp
Test mlp
Parameter File Test3mlp.param
Training Results Test3mlp.err.train
Testing Results Test3mlp.err.test
Log File Test3mlp.log
Decision Region Plot mlp_plot_bound
Profile Plot mlp_plot_bound
Percent Error Plot plot_perr
Cost Plot plot_cost
Structure Plot plot_mlp
Posterior Probability Plot plot_prob
ROC (Detection) Plot plot_detect
Rejection Plot plot_reject
Decision Region Plot File Test3mlp.region.plot.test
Percent Error Plot File Test3mlp.perr.plot
Structure Plot File Test3mlp.struct.plot
Profile Plot File Test3mlp.region.plot.test
Cost Plot File Test3mlp.cost.plot
Posterior Probability Plot File Test3mlp.prob.plot
ROC (Detection) Plot File Test3mlp.detect.plot
Rejection Plot File Test3mlp.reject.plot

8.2.1 Shell scripts

When START, CONTINUE or PLOT ONLY is selected on the main LNKnet screen, a shell script is written and an entry is added to the notebook file. The shell script contains calls to all of the programs requested by the user. In general, the classifier will be trained, training plots will be generated, the classifier will be tested and testing plots will be generated. If movie mode was selected on the plot window, there will be a loop for the training. In the loop, the classifier is trained for a few epochs, then any training plots are generated, but not displayed. When the total number of requested epochs of training have been completed, all of the training plots are displayed together in one olx-plot window. Testing the classifier and generating testing plots proceed as before.

The file name extension for shell scripts is .run. Thus the full file name for the example experiment is Test3mlp.run. An example shell script is found in Section C.1.1.

8.2.2 Screen files

Whenever LNKnet writes a shell script, it also saves the settings of all screens in a screen file. The entries of a screen file look like calls to the LNKnet classifiers and plotting programs. There are settings for all LNKnet classification, clustering, plotting, and general parameters.

The file name extension for screen files is .screen. Thus the full screen file name for the example experiment is Test3mlp.screen.

8.2.3 Notebook File

Each experiment adds one or more lines to the LNKnet notebook file. Each line starts with the name of the shell script file. The first line describes the experiment. If this is the first time the experiment has been run, the data base, normalization, feature selection, and priors adjustment are described. Parameters to the classification algorithm are also included. If the current experiment is being rerun, only the changes to the experiment parameters are recorded. In this case, after the shell script name is a list of the flags that have been added or changed. Flags that have been removed are also listed in square brackets. To create this file, LNKnet compares the current experiment screen file with a copy stored when the previous experiment was written. This backup copy is stored in LNKnet.note.screen. If this is a classification algorithm which uses several passes through the data to train, the next entry in the experiment notebook gives the number of training epochs, the classification error rate and cost for the last epoch of training, and the number of seconds the training took. The next line gives the data file used in testing the classifier, the average classification error rate and cost on that data, and the number of required seconds to test the data. This information is also printed out for cross validation experiments, along with the number of automatic cross validation folds. If a posterior probability plot was requested, there is a line with the target class, the chi value, degrees of freedom, and significance of the fit of the binned output values to the actual posterior class probabilities in the bins. For an ROC plot, a line is included with the target class and the area under the ROC curve. The notebook file generated during the LNKnet tutorial is found in Section C.5.

8.2.4 Log Files

When a classifier or clusterer is run, it prints certain information to the screen and also stores that information in a log file. The contents of that log file depends on the **Report Verbosity,** as shown in Table 8.5. The log file verbosity is set on the LNKnet Report Files and Verbosities window. In a shell script, this parameter is -verbosity.

TABLE 8.5 Log file verbosity levels and Contents

Log file contents	Verbosity Level	Notes
=========<classifier> Begin	All Levels	
Settings for all command line variables	Verbosity 3 or over	
Average error or cost for each epoch of training	Verbosity 3 or over	Incrementally Trained classifiers only
Confusion Matrix	Verbosity 2 or over	

TABLE 8.5 Log file verbosity levels and Contents

Log file contents	Verbosity Level	Notes
Error Summary	Verbosity 1 or over	
Overall Error Rate	All Levels	
Summary for entry into Notebook	All Levels	
========<classifier> End	All Levels	

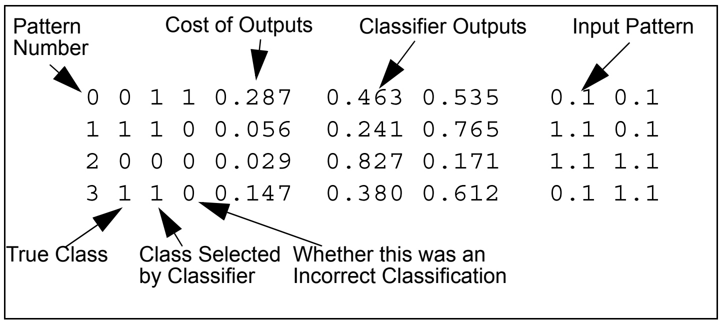
The file name extension for log files is .log. Thus the full log file name for the example experiment is Test3mlp.log. Section C.1.2 gives an example log file from the LNKnet tutorial.

8.2.5 Algorithm Parameter Files

After a classifier has been trained, it is saved in a parameter file. The first items in the parameter file are all program flags and their settings and the date and time that training was started. Following this is information on any normalization performed on the training data before it was presented to the classifier. Any data presented to this classifier for testing will use these same normalization parameters. Finally, the classifier parameters are stored. If this parameter file was generated during N-fold cross validation, it will have several sets of classifier parameters.

The file name extension for parameter files is .param. Thus the full parameter file name for the example experiment is Test3mlp.param. A parameter file for the Gaussian classifier is found in Section C.6.1.

FIGURE 8.6 Format of an error file


```
0 0 1 1 0.287  0.463 0.535   0.1 0.1
1 1 1 0 0.056  0.241 0.765   1.1 0.1
2 0 0 0 0.029  0.827 0.171   1.1 1.1
3 1 1 0 0.147  0.380 0.612   0.1 1.1
```
Pattern Number
Cost of Outputs
Classifier Outputs
Input Pattern
True Class
Class Selected by Classifier
Whether this was an Incorrect Classification

8.2.6 Error Files

When a classifier is tested, the classification results for each pattern can be stored in an error file. An annotated example of an error file is shown in Figure 8.6. Whether to create an error file and how much information to store in it is controlled by the Error File Verbosity on the LNKnet Report Files and Verbosities window. In a shell script, this parameter is the -verror flag. If Error File Verbosity is set to None (-verror 0), no error file is written. If Error File Verbosity is set to Classification Results (-verror 1), for each pattern which is tested, the entries shown in Table 8.6 are written to the error file. Each

TABLE 8.6 Fields in a Results Error File (-verror 1)

Pattern Number	Correct Class	Classifier's Class	Classification Error	Cost

tested pattern generates a line in this file. If the Error File Verbosity is Results+Outputs (-verror 2), after the results entries, the classifier outputs are written to the file, as shown

TABLE 8.7 Fields in a Results+Outputs Error File (-verror 2)

Five Results Fields	Classifier Outputs for this pattern (nclasses fields)

in Table 8.7. Finally, if the Error File Verbosity is Results+Outputs+Inputs (-verror 3), the normalized input pattern is written to the file after the outputs as shown in Table 8.8.

TABLE 8.8 Fields in a Results+Outputs+Inputs Error File (-verror 3)

Five Results Fields	Nclasses Outputs Fields	Normalized Inputs (ninputs fields)

The file name extension for error files is .err. The data base file extension is also used to tell which data base file these are the results for. Thus, if the example multi-layer perceptron classifier stores pattern by pattern classification results during training, they go into Test3mlp.err.train. Table 8.9 shows the default extensions for data file types and the resulting default error file names. The .test_on_train extension cannot be changed from the LNKnet graphical user interface.

TABLE 8.9 Error File Names for Experiment Test3mlp

File Type	File Type Extension	Error File Name
Train on train	.train	Test3mlp.err.train
Test on train	.test_on_train	Test3mlp.err.test_on_train
Test on eval	.eval	Test3mlp.err.eval
Test on test	.test	Test3mlp.err.test
Cross valid on train	.cv	Test3mlp.err.cv

8.2.7 Plot Files

When a plot is generated, it is stored in a file which is formatted as described on the plot(5) UNIX manual page. The format has been extended to allow the use of colors in plots. The file name extension depends on the plot type. Because scatter and histogram plots can be generated for any data base file, the decision region and profile plots names also include a data base file extension. These file name extensions are the same as those used for error files and are given in Table 8.9. The file names for normalization and feature selection plots depend on the name of the parameter file on which the plot is based. For normalization plots, add .plot to the parameter file name. For feature selection plots, replace the extension .param with .plot. Table 8.10 shows the extensions and plot names for the example classification experiment. The decision region and profile plot names are for plots using the evaluation data file.

TABLE 8.10 Plot file names for Experiment Test3mlp and Evaluation data file

Plot Type	File name Extension	Plot File name
Decision region	.region.plot<file type>	Test3mlp.region.plot.eval
Profile	.profile.plot<file type>	Test3mlp.profile.plot.eval
Structure	.struct.plot	Test3mlp.struct.plot
Cost	.cost.plot	Test3mlp.cost.plot
Percent error	.perr.plot	Test3mlp.perr.plot
Posterior probability	.prob.plot	Test3mlp.prob.plot
ROC (detection)	.detect.plot	Test3mlp.detect.plot
Rejection	.reject.plot	Test3mlp.reject.plot

These files can be displayed using olxplot under OpenWindows or xplot under MIT X. One way to print plot files is to first convert them to PostScript using plot2ps. Plots can be added to FrameMaker documents if they are first translated into Maker Interchange Format using plot2mif. Both of these programs are available on the Print window described in Section 7.1.

8.3 C Code Files

Each LNKnet classifier has a filter program which generates a C classification subroutine based on a classifier parameter file. The actual contents of such a file depend on the classification algorithm being implemented. In general, the file contains the original parameter file name, a description of the normalization and feature selection parameters, a description of those classifier parameters that affect the classification subroutine, necessary #include statements, declarations of the classify() and normalize() subroutines created in the file, and finally the classify() and normalize() routines themselves. The routines declare most classifier parameters in the form of structures or arrays. The routine classify() takes two float arrays as input, the raw input pattern and an array for the resulting classifier outputs. The routine returns an integer representing the class of the input pattern. The classification routine calls the normalize() routine for normalization and feature selection. The classification routine itself then uses the classifier parameter structures or arrays for its calculation of the classifier outputs given the now normalized inputs. Finally, the normalization routine takes a float array for the input vector. The input vector is normalized and copied back into the array. The integer number of inputs after normalization and feature selection is then returned. C code files and how to generate them from LNKnet are described in Section 7.2. The file name extension for C code files is .c. Thus the full C code file name for the example experiment is Test3mlp.c. An example of a C code file for a Gaussian classifier is found in Section C.6.3.

8.4 Committee Data Base Files

Committee data bases are generated from the class and outputs fields of testing error files, as described in Section 7.3. The format of a committee data base file is the same as for any other classification data file. The first field of each line holds the integer class of a data pattern. The next $N \times M$ fields on the line are the M floating point class outputs for each of the N classifiers in the committee for the original input pattern. Each line ends with a newline character. The input fields for the committee data base are illustrated in the description file shown in Figure 8.7. This is the description file for the committee data base being generated in Figure 7.3 on page 112. The class labels for the committee data base are the same as the class labels of the original classification data base. The input labels are generated from the experiment labels for the committee members and the output numbers. The total number of input features here is 30 because there are 10 classes and 3 committee members. The default name for a committee data base starts with the original data base name followed by _comm.

FIGURE 8.7 Description file for gnoise_var committee data base (gnoise_var_comm.defaults)

```
describe -labels 0,1,2,3,4,5,6,7,8,9  \
-input_labels N1gauss0,N1gauss1,N1gauss2,N1gauss3,N1gauss4,N1gaus
s5,N1gauss6,N1gauss7,N1gauss8,N1gauss9,N2gauss0,N2gauss1,N2gauss2
,N2gauss3,N2gauss4,N2gauss5,N2gauss6,N2gauss7,N2gauss8,N2gauss9,N
4gauss0,N4gauss1,N4gauss2,N4gauss3,N4gauss4,N4gauss5,N4gauss6,N4g
auss7,N4gauss8,N4gauss9 \
-class  -ninputs 30  -noutputs 10
```

APPENDIX A Common Questions and Problems

1. UNIX and Shell Scripts

1.1. Problem: The command “lnknet” cannot be found when I type it at the command line.
Solution: Add the LNKnet bin directory to your environment variable PATH. (Contact your system administrator for help)

1.2. Problem: The LNKnet manual pages cannot be found.
Solution: Add the LNKnet man directory to your environment variable MANPATH. (Contact your system administrator for help).

1.3. Problem: A LNKnet experiment starts but doesn’t finish.
Solution: There was an error in the one of the programs called by the shell script. The shell script stops whenever a program returns a non-zero value. Look in the log file to find the error message. Check the flags of the program that generated the error to determine how to fix the error.

1.4. Problem: How do I create shell scripts to run background batch jobs?
Solution: Create a shell script with the suffix .run with LNKnet, edit the shell script if necessary, and then run this shell script in the background as you would a user-generated shell script. You may want to eliminate some plotting by adding the **-no_graphics** flag to plotting programs and replace the string “|& **nn_tee** -h -a” with “>>” to send output to the log file without sending it to the screen. These modifications are automatically made if you check the **Only Store Shell Scripts, do not Run** box on the main LNKnet window.

2. Files and User I/O

2.1. Problem: LNKnet won’t start: Error message is “bad flag”.
Solution: The current copy of LNKnet uses a different set of flags than those in your current .lnknetrc file. Delete your .lnknetrc file or change its name to start LNKnet without the default settings.

2.2. Problem: There are no files in the data base scroll list on the data base window or the desired data base is not on the list.
Solution: There are several possibilities here:

1. The path to the data base directory is wrong. Change the data base directory path.
2. There are no data base description files for the data bases in this directory. Enter the data base names in the data file prefix field and generate description files for the data bases on the Description File Generation window.
3. The description files use the wrong suffix. Data base description files must be named <database>.defaults. Only file names which include the string ".defaults" are displayed in the data base scrolling list.

2.3. Problem: On the data base window, under patterns per class it says "Not classification data"
Solution: Check that the data base selected is a static pattern classification data base, not an input/output mapping or sequence classification data base.

2.4. Problem: When I select a data base I get a warning on my shell window, "Noutputs from defaults file doesn't match data file"
Solution: There are two possible problems here:

1. The training, testing, or evaluation file being read is not a LNKnet classification data file.
2. There is a pattern in the data file with a class label which is out of range. The class numbers at the start of every LNKnet pattern are numbered from 0 to Noutputs-1.

2.5. Problem: There are red stop signs beside some of the buttons on the main window.
Solution: There are important errors on these windows. Unless these errors are cleared, an experiment started now will not run correctly. Select the buttons and clear the errors before starting the experiment.

2.6. Problem: File names and parameters entered in LNKnet windows are not updated during an experiment.
Solution: A carriage return or a tab must be entered after typing anything in a LNKnet window before the new entry is read in. Play it safe and hit carriage return after typing anything.

2.7. Problem: The number of patterns in a data base file is not set when the data base is selected.
Solution: If the data base directory is correct, check the data file extension.

2.8. Problem: There is an error on the normalization window, "Normalization file does not exist."
Solution: If the normalization file DOES exist, check the data base directory, data base selection, and the normalization selection. Otherwise, create the normalization file on the normalization file generation window.

2.9. Problem: There is an error on the feature selection window, bad feature list or not enough labels.
Solution: There are two possibilities here:

1. Check the feature selection parameters.

2. Check that the number of input features on the data base window is the same as the number of inputs. Change the input feature list on the description file generation window.

2.10. Problem: There is an “File does not exist” error on one of the windows after the missing file has been created.
Solution: Click the mouse on the error Stop sign to erase the message.

2.11. Problem: Options are grayed out on a parameter window and cannot be selected.
Solution: Options that are inconsistent with previous selections are grayed out and cannot be selected. For example, the perceptron convergence procedure cost function of the multi-layer perceptron classifier is only available if there are no hidden layers. When it is selected, the field for specifying network topology is grayed out. Select a parameter value that is consistent with the desired option.

2.12. Problem: A comma delimited list is not read correctly.
Solution: There are two possibilities here:

1. Do not leave spaces between the commas. If this is a list of strings, there are three delimiter characters that can be used: comma (,), colon (:), and dash (-). Check that these characters are not included inside any of your desired list strings. For example first_formant is a valid label because an underscore is used to indicate the space between the words, but first-formant is not valid because a dash is used within the label.
2. Always put a space after a comma delimited list. Because dash is used to mark flags and is also a list delimiter, forgetting the space after a list can cause LNKnet to add the name of the next flag to the list. That flag will not be set, since it has already been parsed as being a part of a list. This is most often a problem when flags are put on multiple lines in a file with backslashes (\) at the ends of lines. When the file is read, the first character of the second line is placed directly after the last character of the first line. No extra spaces are inserted.

2.13. Problem: Per-epoch error information is not printed out when training an MLP classifier.
Solution: Set the log file verbosity flag on the Report Files and Verbosities window to print out Summary+Confusion+Flags+Epochs.

3. Misc.

3.1. Problem: It is annoying when I keep the same experiment name to keep having to move my mouse after starting another experiment and clicking on the button that says it’s ok to overwrite the old experiment
Solution: Experts move the small window that queries you about overwriting to be located over the button used to start a new experiment. You can then dismiss the second small verification window with a second mouse click in the same location.

3.2. Problem: It is difficult to select features using a comma separted list of feature numbers because I keep forgetting which numbers correspond to which feature names.
Solution: Most of us get around this by keeping a listing of the feature names along with their numbers. This list is provided at the bottom of the feature selection window if you select all features. Once you guess at the feature numbers and hit carriage return, the feature

names are displayed at the bottom of the feature-selection window. Also remember that feature numbers start at zero and not at one.

3.3. Problem: When I have many input features, the feature selection window extends way off the screen and I can't see the whole window at once.
Solution: Either use smaller feature names, or drag the window to left or right using the mouse. You can also resize this window, after performing feature selection.

4. Known Limitations

4.1. Problem: I think I found a problem or want a new feature.
Solution: Send questions, requests, and bug reports to Linda Kukolich (kukolich@sst.ll.mit.edu) or Richard Lippmann (rpl@sst.ll.mit.edu).

4.2. Problem: LNKnet windows come up all black or with black writing on black buttons.
Solution: LNKnet was developed on a color Sparc station. It has not been debugged on black and white terminals and may not work on them. The problem may be solved in newer versions of OpenWindows.

4.3. Problem: When specifying a large number of cross validation folds in a file, the following error occurs: "Number of labels has exceeded 255. The list has been truncated."
Solution: The maximum number of entries in the -cv_splits, -cv_train_mask, and -cv_test_mask arguments of a cross validation file is 255. Because a split is defined by pairs of entries, the maximum number of splits is 127. The maximum number of cross validation folds is 255. This may be changed in a future version of LNKnet.

5. MLP Training

5.1. Problem: MLP training is slow
Solution: There are several things that can be tried to speed up MLP training:

1. Make sure that random presentation order is selected on the main window.
2. Make sure that the weights are being updated after every trial.
3. Increase the step size.

5.2. Problem: MLP cost is not decreasing.
Solution: Decrease step size.

6. Plots

6.1. Problem: Decision region plots are not in color.
Solution: Check that color plots are selected on the Decision Region Plot window.

6.2. Problem: Decision region plots are too jagged and blocky.
Solution: Increase the number of points per dimension on the Decision Region Plot window.

6.3. Problem: Scatter plot or Histogram plot does not show patterns.
Solution: Turn on autoscale, adjust the setting for Xmin, Xmax, Ymin, and Ymax, turn on show all patterns, or increase the distance limit for pattern display.

6.4. Problem: I have problems looking at plots while running MIT X.
Solution: Change the name of the binary file xplot to olxplot, after saving olxplot. Xplot is designed to run under MIT X. Olxplot is designed to run under SUN OpenLook.

6.5. Problem: The screen rapidly gets cluttered with plots.
Solution: Turn plots off and eliminate existing plots rapidly by typing ‘q’ when the mouse is over a plot window.

6.6. Problem: How do I make hard copies of plots?
Solution: See Section 7.1 on page 109 or Section 6.9 on page 107.

6.7. Problem: Plots do not run and generate the error:

```
ld.so: Undefined symbol: _XtQString
```

Solution: The program which displays plots, olxplot, was written using the OpenLook Intrinsics library, olit. Olit uses X11R4, as does the rest of the OpenWindows environment. If your environment variable $LD_LIBRARY_PATH includes the X11R5 libraries, olxplot will not run because of incompatibilities between X11R4 and X11R5. Remove the X11R5 libraries from the LD_LIBRARY_PATH environment variable in your terminal window shell before starting LNKnet. The following commands can be used to display and correct the LD_LIBRARY_PATH variable:

```
> echo $LD_LIBRARY_PATH

/usr/local/X11R5/lib:/src/openwin3.o/lib:/usr/local/lib:/usr/
local/lib/X11:/usr/lib

> setenv LD_LIBRARY_PATH /src/openwin3.0/lib:/usr/local/lib:/
usr/local/lib/X11:/usr/lib
```

6.8. Problem: Plots generate a warning:

```
Warning: XtRemoveInput: Input handler not found
```

Solution: This is a known bug in olxplot. It has no effect on the plots and can be ignored.

APPENDIX B Installing LNKnet

B.1 What you need

- 120 Mbytes disc space (90 Mbytes without the source)
- Sun Solaris 2.5, 2.6 or later with OpenWindows
- RedHat or other versions of Linux
- Microsoft windows with a current version of the Cygwin environment

B.2 Read Tar Tape or Download from Web Site

Make a directory to install LNKnet and change to it. This is usually in your home directory

```
> mkdir lnknet
> cd lnknet
```

Load the tape in the appropriate device and read it using

```
> tar xvf /dev/rst0
```

or download, unzip and untar the gzipped LNKnet tar file from the LNKnet web site that is currently hhttp://www.ll.mit.edu/IST/lnknet/index.html. Under RedHat linux the command to perform these actions would be

```
> tar zxvf lnknet.linux.tgz
```

Alist of all the files will be displayed as they are read.

Files in the lnknet directory:

```
Makefile          changes/        lib/
RCS/              data/           loading_instructions
README            demo/           src/
bin/              man/
```

At an absolute minimum, you need the bin/ and man/ directories. In order to do the tutorial, you will also need the data/ directory. If space is tight, the other directories may be deleted. This would save 34 Mbytes of space.

Add lnknet/man to the MANPATH environment variable.

Add lnknet/bin to the PATH environment variable. See the README and INSTALL files for other information.

B.3 LNKnet updates

If this is an update, the old copy of LNKnet can be overwritten with this one. The files in the directory “changes” document new features, bug fixes, and changes that will be necessary to get old shell scripts to run in the new version. In addition, delete any .lnknetrc files in users’ home directories.

B.4 Recompiling LNKnet

LNKnet users have ported LNKnet to other platforms or have modified LNKnet programs by enhancing them or by entering small bug fixes themselves. This requires that the affected programs be recompiled. To recompile LNKnet programs you must first define an environment variable, LNKHOME, as the path to the directory in which LNKnet was installed. For example, on my system I define LNKnet home using this command:

```
>setenv LNKHOME /home/kukolich/lnknet
```

LNKHOME is used by the Makefiles to determine the paths to the LNKnet source, include, library, and binary directories. To recompile the graphical user interface it is also necessary that you have the OpenWindows XVIEW 3.0 library. The plotting programs require the OpenLook Intrinsics library, olit.

The LNKnet binaries can be recompiled in part or as a whole by changing the directory in which the make command is issued. Table B.1 shows the directories and the type of binaries created.

TABLE B.1: LNKnet directories, and results

directory	**result of make**
$LNKHOME	all binaries
$LNKHOME/src	all binaries
$LNKHOME/src/lib	library nnlib.a†
$LNKHOME/src/algorithm	all classifiers and clusterers
$LNKHOME/src/algorithm/mlp	multi-layer perceptron program, mlp†
$LNKHOME/src/plot	plot programs†
$LNKHOME/src/gclass	LNKnet GUI†

†The command **make** creates the binary. The command **make copy** creates the binary and copies it into the bin directory, $LNKHOME/bin.

If there are any problems with compilation, call or e-mail Linda Kukolich (KUKOLICH@LL.MIT.EDU) for help.

APPENDIX C Tutorial Scripts and Outputs

The experiments shown here were performed as part of the LNKnet tutorial in Chapter 2 and as examples in Chapter 6.

C.1 MLP

This experiment was run in two parts. During the first part, the MLP classifier was trained for 20 epochs on 338 samples from the vowel data base. When evaluated, the resulting classifier obtained an error rate of 30%. The training was then continued for another 20 epochs which brought the evaluation error rate down to 20%. The shell script shows the calls for the first half of the training. It differs from the script for the second half only in that the calls to MLP use the -create flag. The log file shown below has the results from both halves of the experiment. Because this classifier looks at the same training patterns multiple times, there are classification results for the training as well as for the evaluation portions of the experiment.

C.1.1 MLP Shell Script

```
#!/bin/csh -ef
# ./X1mlp.run
set loc=`pwd`
```

Train MLP model (first 20 epochs) The second 20 epochs are the same except that the create flag is not set

```
#train
(time mlp\
 -train  -create  -pathexp $loc  -ferror X1mlp.err.train  -fparam X1mlp.param\
 -pathdata /u/kukolich/Tutorial  -finput vowel.train\
 -fdescribe vowel.defaults  -npatterns 338  -ninputs 2  -normalize\
 -fnorm vowel.norm.simple  -cross_valid 0  -fcross_valid vowel.train.cv\
 -random_cv  -random  -seed 0  -priors_npatterns 338  -debug 0  -verbose 3\
 -verror 2 \
-nodes 2,25,10  -alpha 0.6  -etta 0.2  -etta_change_type 0  -epsilon 0.1\
 -kappa 0.01  -etta_nepochs 0  -decay 0  -tolerance 0.01  -hfunction 0\
 -ofunction 0  -sigmoid_param 1  -cost_func 0  -cost_param 1  -epochs 20\
 -batch 1,1,0  -init_mag 0.1 \
```

Put training result in the experiment notebook

```
)|& nn_tee -h X1mlp.log
echo -n "X1mlp.run        " >> /u/kukolich/Tutorial/LNKnet.note
grep "LAST TRAIN EPOCH" X1mlp.log | tail -1 >> /u/kukolich/Tutorial/LNKnet.note
```

Evaluate MLP model

```
#test
(time mlp\
 -create  -pathexp $loc  -ferror X1mlp.err.eval  -fparam X1mlp.param\
 -pathdata /u/kukolich/Tutorial  -finput vowel.eval\
 -fdescribe vowel.defaults  -npatterns 166  -ninputs 2  -normalize\
 -fnorm vowel.norm.simple  -cross_valid 0  -fcross_valid vowel.train.cv\
 -random_cv  -random  -seed 0  -priors_npatterns 338  -debug 0  -verbose 3\
 -verror 2 \
-nodes 2,25,10  -alpha 0.6  -etta 0.2  -etta_change_type 0  -epsilon 0.1\
 -kappa 0.01  -etta_nepochs 0  -decay 0  -tolerance 0.01  -hfunction 0\
 -ofunction 0  -sigmoid_param 1  -cost_func 0  -cost_param 1  -epochs 20\
 -batch 1,1,0  -init_mag 0.1 \
)|& nn_tee -h -a X1mlp.log
```

Put testing result in the experiment notebook

```
echo -n "X1mlp.run        " >> /u/kukolich/Tutorial/LNKnet.note
grep "TEST" X1mlp.log | tail -1 >> /u/kukolich/Tutorial/LNKnet.note
```

Decision region plots

```
#decision region plot
mlp_plot_bound\
 -autoscale  -pathexp $loc  -fparam X1mlp.param\
 -fregion X1mlp.region.plot.eval  -mac_wireframe_output -1\
 -fmacdots boundary_dots.mac  -fmacscatter scatter.mac\
 -fmaclines profile_lines.mac  -fmachisto profile_histo.mac\
 -fmacwireframe boundary_wire.mac  -xmin -3  -xmax 3  -ymin -3  -ymax 3\
 -xstep 1  -ystep 1  -pymin 0  -pymax 1.5  -pystep 0.25  -ninputs 2\
 -noutputs 10  -tregion "Norm:Simple Net:2,25,10 Step:0.2"  -first_dim 0\
 -second_dim 1  -npatterns 166  -npoints 100  -region  -scatter  -internals\
 -internals_scale 1  -internals_level 1  -color  -all  -distance 0.5\
 -fdata /u/kukolich/Tutorial/vowel.eval\
 -fdesc /u/kukolich/Tutorial/vowel.defaults
```

Profile plots

```
#profile plot
mlp_plot_bound\
 -autoscale  -pathexp $loc  -fparam X1mlp.param  -mac_wireframe_output -1\
 -fmacdots boundary_dots.mac  -fmacscatter scatter.mac\
 -fmaclines profile_lines.mac  -fmachisto profile_histo.mac\
 -fmacwireframe boundary_wire.mac  -fprofile X1mlp.profile.plot.eval  -xmin -3\
 -xmax 3  -ymin -3  -ymax 3  -xstep 1  -ystep 1  -pymin 0  -pymax 1.5\
 -pystep 0.25  -ninputs 2  -noutputs 10\
 -tprofile "Norm:Simple Net:2,25,10 Step:0.2"  -first_dim 0  -second_dim 1\
 -npatterns 166  -npoints 50  -internals_scale 1  -internals_level 1  -profile\
 -histogram  -color  -all  -distance 0.5\
 -fdata /u/kukolich/Tutorial/vowel.eval\
 -fdesc /u/kukolich/Tutorial/vowel.defaults
```

Cost plot

```
#cost plot
plot_cost -pathexp $loc  -ferror X1mlp.err.train  -fplot X1mlp.cost.plot  -
autoscale\
 -xmin 0  -xmax 10000  -ymin 0  -ymax 5  -xstep 1000  -ystep 1  -trials 338\
 -title "Norm:Simple Net:2,25,10 Step:0.2"  -cost_func 0  -fmaclines cost.mac\
 -class
```

Percent error plot

```
#percent error plot
plot_perr -pathexp $loc  -ferror X1mlp.err.train  -fplot X1mlp.perr.plot  -
autoscale\
 -xmin 0  -xmax 10000  -ymin 0  -ymax 100  -xstep 1000  -ystep 10  -trials 338\
 -title "Norm:Simple Net:2,25,10 Step:0.2"  -fmaclines perr.mac  -class
```

Structure plot

```
#structure plot
plot_mlp -fparam X1mlp.param  -fplot X1mlp.struct.plot\
 -fdescribe /u/kukolich/Tutorial/vowel.defaults  -autoscale  \
 -threshold 0.000000  -show_weight_magnitude -max_line_width 10  -show_bias
```

Posterior probability plot

```
#prob plot
plot_prob -bin_plot  -target 2  -nbins 5  -min_bin_count 5  -chi_square  -pathexp
$loc\
 -ferror X1mlp.err.eval  -fplot X1mlp.prob.plot  -no_graphics  -noutputs 10\
 -npatterns 166  -xmin 0  -xmax 100  -ymin 0  -ymax 100  -xstep 10  -ystep 10\
 -verbose 1  -title "Norm:Simple Net:2,25,10 Step:0.2"  -fmaclines prob.mac \
 |& nn_tee -h -a X1mlp.log
```

Put chi square results in the experiment notebook

```
olxplot -geometry 500x480 -title prob_plot X1mlp.prob.plot&
echo -n "X1mlp.run        " >> /u/kukolich/Tutorial/LNKnet.note
grep "CHI" X1mlp.log | tail -1 >> /u/kukolich/Tutorial/LNKnet.note
```

ROC (Detection) plot

```
#detect plot
plot_detect -target 2  -pathexp $loc  -ferror X1mlp.err.eval  -fplot
X1mlp.detect.plot\
 -noutputs 10  -reject 0  -xmin 0  -xmax 100  -ymin 0  -ymax 100  -xstep 10\
 -ystep 10  -title "Norm:Simple Net:2,25,10 Step:0.2"  -table_begin 0\
 -table_end 100  -table_step 2  -verbose 2  -fmaclines detect.mac\
 -no_graphics \
 |& nn_tee -h -a X1mlp.log
olxplot -geometry 500x480 -title detect_plot X1mlp.detect.plot&
```

Put ROC area in the experiment notebook

```
echo -n "X1mlp.run        " >> /u/kukolich/Tutorial/LNKnet.note
grep "ROC AREA" X1mlp.log | tail -1 >> /u/kukolich/Tutorial/LNKnet.note
```

Rejection plot

```
#reject plot
plot_reject -pathexp $loc  -ferror X1mlp.err.eval  -fplot X1mlp.reject.plot  -nout-
puts 10\
 -npatterns 0  -xmin 0  -xmax 100  -ymin 0  -ymax 100  -xstep 10  -ystep 10\
 -title "Norm:Simple Net:2,25,10 Step:0.2"  -table_begin 0  -table_end 100\
 -table_step 10 -verbose 1  -fmaclines reject.mac  -no_graphics \
 |& nn_tee -h -a X1mlp.log
olxplot -geometry 500x480 -title reject_plot X1mlp.reject.plot&
echo "current directory:" >> X1mlp.log
echo $loc >> X1mlp.log
```

C.1.2 MLP Log File (Initial training Plus Continuation)

Initial training of MLP

```
================================================================ mlp BEGIN
mlp
-train  -create  -pathexp /u/kukolich/Tutorial\
 -ferror X1mlp.err.train  -fparam X1mlp.param\
 -pathdata /u/kukolich/Tutorial  -finput vowel.train\
 -fdescribe vowel.defaults  -npatterns 338  -ninputs 2  -normalize\
 -fnorm vowel.norm.simple  -cross_valid 0  -fcross_valid vowel.train.cv\
 -random_cv  -random  -seed 0  -priors_npatterns 338  -debug 0  -verbose 3\
 -verror 2  -nodes 2,25,10  -alpha 0.6  -etta 0.2  -etta_list 0.2,0.2\
 -etta_change_type 0  -epsilon 0.1  -kappa 0.01  -etta_nepochs 0  -decay 0\
 -tolerance 0.01  -hfunction 0  -ofunction 0  -sigmoid_param 1\
 -sig_param_list 1,1  -cost_func 0  -cost_param 1  -epochs 20  -batch 1,1,0\
 -init_mag 0.1
Wed Apr  5 11:10:24 1995
```

Percent error and average cost per epoch

```
Reading /u/kukolich/Tutorial/vowel.train

 EPOCH  %error    RMS Err(338 patterns/epoch)
   1      92.0    0.3051
   2      81.4    0.2977
   3      63.6    0.2781
   4      58.0    0.2628
   5      51.2    0.2565
   6      46.7    0.2508
   7      47.0    0.2479
   8      42.9    0.2436
   9      43.5    0.2396
  10      43.2    0.2388
  11      43.5    0.2359
  12      39.3    0.2335
  13      39.3    0.2306
  14      42.0    0.2305
  15      37.3    0.2248
  16      36.4     0.225
  17      34.6    0.2231
  18      36.1    0.2216
  19      32.8    0.2187
  20      33.7    0.2175
 LAST TRAIN EPOCH:  20 33.73 % Err  0.217 RMS Err  18.01 secs

Finished -- model saved in "/u/kukolich/Tutorial/X1mlp.param"


Classification Confusion Matrix - X1mlp.err.train
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Confusion matrix for first 20 epochs

```
Desired                          Computed Class
 Class     0     1     2     3     4     5     6     7     8     9   Total
 -----   ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
   0      308   157    12   151     6     3    49    10     2     2    700
   1      148   236    15    24     6     4   288     9     8     2    740
   2       11     6   588    19    99     1     5    60     8     3    800
   3      119    14    37   502     5     2    15    22     3     1    720
   4        5     8   152     4   397     2     4    34    41    13    660
   5       51    93     8    19    12   101    17    19    72    68    460
   6       33    56     5     7     6         614     9    10          740
   7       21     5   288   102    80     2    10   162     7    23    700
   8        2     7    15     8    28     6    11     3   552    28    660
   9       14    11    24    11   104    50     5    66   188   107    580
 -----   ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
 Total    712   593  1144   847   743   171  1018   394   891   247   6760

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

```
Error Report - X1mlp.err.train
--------------------------------------------------------------------------------
```

Error summary for first 20 epochs

```
--------------------------------------------------------------------------------
Class      Patterns   # Errors     % Errors StdDev RMS Err Label
  0          700        392        56.00   ( 1.9)     0.264  head
  1          740        504        68.11   ( 1.7)     0.272  hid
  2          800        212        26.50   ( 1.6)     0.214  hod
  3          720        218        30.28   ( 1.7)     0.217  had
  4          660        263        39.85   ( 1.9)     0.237  hawed
  5          460        359        78.04   ( 1.9)     0.292  heard
  6          740        126        17.03   ( 1.4)     0.190  heed
  7          700        538        76.86   ( 1.6)     0.289  hud
  8          660        108        16.36   ( 1.4)     0.180  whod
  9          580        473        81.55   ( 1.6)     0.293  hood
          --------     --------    ------- -------------

Overall     6760       3193        47.23   ( 0.6)    0.245

================================================================ mlp   END
17.9u 0.2s 0:23 78% 0+440k 5+111io 50pf+0w
```

Evaluate model based on initial training

```
================================================================ mlp BEGIN
mlp
-create  -pathexp /u/kukolich/Tutorial  -ferror X1mlp.err.eval\
 -fparam X1mlp.param  -pathdata /u/kukolich/Tutorial\
 -finput vowel.eval  -fdescribe vowel.defaults  -npatterns 166  -ninputs 2\
 -normalize  -fnorm vowel.norm.simple  -cross_valid 0\
 -fcross_valid vowel.train.cv  -random_cv  -random  -seed 0\
 -priors_npatterns 338  -debug 0  -verbose 3  -verror 2  -nodes 2,25,10\
 -alpha 0.6  -etta 0.2  -etta_list 0.2,0.2  -etta_change_type 0  -epsilon 0.1\
 -kappa 0.01  -etta_nepochs 0  -decay 0  -tolerance 0.01  -hfunction 0\
 -ofunction 0  -sigmoid_param 1  -sig_param_list 1,1  -cost_func 0\
 -cost_param 1  -epochs 20  -batch 1,1,0  -init_mag 0.1
Wed Apr  5 11:10:45 1995
```

```
Reading /u/kukolich/Tutorial/vowel.eval
```

Evaluation confusion matrix

```
Classification Confusion Matrix - X1mlp.err.eval
----------------------------------------------------------------------------
----------------------------------------------------------------------------

Desired                          Computed Class
 Class     0     1     2     3     4     5     6     7     8     9   Total
 -----   ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
   0      14                 3                                         17
   1       5     1                             12                      18
   2                  19     1                                         20
   3       5                11           1           1                 18
   4                   2           14                                  16
   5       1                             9                 1           11
   6                                           18                      18
   7                   6     3     2                 7                 18
   8                                                      15     1     16
   9                                     6           1     3     4     14
 -----   ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
 Total    25     1    27    18    16    16    30     9    19     5    166

----------------------------------------------------------------------------
----------------------------------------------------------------------------
```

Evaluation error summary

```
Error Report - X1mlp.err.eval
----------------------------------------------------------------------------
----------------------------------------------------------------------------
Class     Patterns   # Errors     % Errors StdDev RMS Err   Label
  0          17          3        17.65   ( 9.2)     0.173  head
  1          18         17        94.44   ( 5.4)     0.304  hid
  2          20          1         5.00   ( 4.9)     0.127  hod
  3          18          7        38.89   (11.5)     0.197  had
  4          16          2        12.50   ( 8.3)     0.182  hawed
  5          11          2        18.18   (11.6)     0.249  heard
  6          18          0         0.00   ( 0.0)     0.051  heed
  7          18         11        61.11   (11.5)     0.268  hud
  8          16          1         6.25   ( 6.1)     0.121  whod
  9          14         10        71.43   (12.1)     0.289  hood
          --------     --------    ------- -------------

Overall      166         54        32.53   ( 3.6)    0.207

 TEST:      vowel.eval 32.53 % Err  0.207 RMS Err   0.54 secs
 ============================================================ mlp   END
0.5u 0.0s 0:00 91% 0+376k 1+3io 1pf+0w
```

Posterior probability plot table

```
 ---------------------------------------------------------------------------

 target class =  2
   total number of patterns =  166
 ---------------------------------------------------------------------------
     BIN #     # PATTERNS  PREDICTED %     ACTUAL %      RANGE
 -----------  -----------  -----------  -----------  -----------
        0         131        1.84         0.76    -0.76 -   2.28
        1          11       28.08         9.09    -8.24 -  26.43
        2           6       52.66        33.33    -5.16 -  71.82
        3           7       69.71        71.43    37.28 - 105.58
        4          11       87.98       100.00   100.00 - 100.00

Chi Square Fit:
```

```
Chi                =   2.446746
Degrees of Freedom =   2
Significance       =   0.294236
 TARGET 2 CHI 2.446746 DOF 2 SIGNIFICANCE 0.294236
Created file /u/kukolich/Tutorial/X1mlp.prob.plot
ROC-x      ROC-y
```

ROC (detection) plot table

```
----------------------------------------------------------------
target class = 2          reject = 0.00%
number of patterns from target class = 20
----------------------------------------------------------------
# PATTERNS  # CORRECT     % CORRECT  # FALSE  % FALSE_ALARM  THRESHOLD
 ----------  ----------  ----------  ----------  ----------  ----------
    0           0         0.000           0       0.000       1.000
   20          16        82.500           4       2.000       0.564
   25          18        92.500           7       4.000       0.350
   35          19        97.500          16       6.000       0.199
  166          20       100.000         146      12.000       0.000
 TARGET 2 ROC AREA = 98.732872
Created file X1mlp.detect.plot
```

Rejection plot table

```
 ------------------------------------------------------------------------------
 # PATTERNS # REJECTIONS  % REJECTED  # ERRORS     % ERROR    THRESHOLD
 ----------  ----------  ----------  ----------  ----------  ----------
       166           0    0.000000          54   32.530121       0.000
       149          16   10.000000          49   32.798210       0.264
       132          33   20.000000          43   32.380184       0.347
       116          49   30.000000          29   24.957266       0.416
        99          66   40.000000          19   19.076767       0.513
        83          83   50.000000          11   13.253012       0.604
        66          99   60.000000           6    9.633648       0.684
        49         116   70.000000           1    2.008163       0.757
        33         132   80.000000           1    3.012477       0.862
        16         149   90.000000           0    0.000000       0.926
Created file /u/kukolich/Tutorial/X1mlp.reject.plot
current directory:
/u/kukolich/Tutorial
```

Continue training model stored in X1mlp.param for 20 more epochs

```
================================================================ mlp BEGIN
mlp
-train  -pathexp /u/kukolich/Tutorial  -ferror X1mlp.err.train\
 -fparam X1mlp.param  -pathdata /u/kukolich/Tutorial\
 -finput vowel.train  -fdescribe vowel.defaults  -npatterns 338  -ninputs 2\
 -normalize  -fnorm vowel.norm.simple  -cross_valid 0\
 -fcross_valid vowel.train.cv  -random_cv  -random  -seed 0\
 -priors_npatterns 338  -debug 0  -verbose 3  -verror 2  -nodes 2,25,10\
 -alpha 0.6  -etta 0.2  -etta_list 0.2,0.2  -etta_change_type 0  -epsilon 0.1\
 -kappa 0.01  -etta_nepochs 0  -decay 0  -tolerance 0.01  -hfunction 0\
 -ofunction 0  -sigmoid_param 1  -sig_param_list 1,1  -cost_func 0\
 -cost_param 1  -epochs 20  -batch 1,1,0  -init_mag 0.1
Wed Apr  5 15:35:39 1995

Reading /u/kukolich/Tutorial/vowel.train
```

Percent error and cost for continued training

```
 EPOCH  %error    RMS Err(338 patterns/epoch)
   1      34.6    0.2161
   2      35.2     0.214
   3      31.4    0.2129
   4      32.0    0.2105
   5      30.5    0.2081
   6      29.0    0.2074
   7      29.6    0.2047
   8      27.8    0.2044
   9      28.4    0.2022
  10      29.9    0.2044
  11      27.2    0.2013
  12      28.7    0.2009
  13      27.2    0.1973
  14      25.4    0.1964
  15      26.3    0.1959
  16      28.7    0.1973
  17      24.6    0.1935
  18      24.3    0.1932
  19      25.1    0.1933
  20      25.4    0.1935
 LAST TRAIN EPOCH:  20 25.44 % Err  0.194 RMS Err  17.80 secs

Finished -- model saved in "/u/kukolich/Tutorial/X1mlp.param"
```

Confusion matrix for new epochs

```
Classification Confusion Matrix - X1mlp.err.train
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

Desired                          Computed Class
 Class     0     1     2     3     4     5     6     7     8     9   Total
 -----   ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
   0      416   156         113         15                             700
   1      119   478                     17   126                       740
   2                  660          85               55                 800
   3      123               571                     26                 720
   4                  128         471               12    20    29     660
   5       43    39          20         268               24    66     460
   6       12    54                           674                      740
   7                  121    54    51              437          37     700
   8                               22    18               568    52    660
   9                               56    85         82    71   286     580
 -----   ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
 Total    713   727   909   758   685   403   800   612   683   470   6760

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

Error summary for new epochs

```
Error Report - X1mlp.err.train
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Class     Patterns   # Errors     % Errors StdDev RMS Err Label
  0         700        284        40.57   ( 1.9)     0.236  head
  1         740        262        35.41   ( 1.8)     0.220  hid
  2         800        140        17.50   ( 1.3)     0.177  hod
  3         720        149        20.69   ( 1.5)     0.171  had
  4         660        189        28.64   ( 1.8)     0.203  hawed
  5         460        192        41.74   ( 2.3)     0.236  heard
  6         740         66         8.92   ( 1.0)     0.142  heed
  7         700        263        37.57   ( 1.8)     0.229  hud
  8         660         92        13.94   ( 1.3)     0.145  whod
  9         580        294        50.69   ( 2.1)     0.256  hood
          --------     --------    ------- -------------

Overall     6760       1931        28.57   ( 0.5)    0.202

=============================================================== mlp   END
17.7u 0.1s 0:23 77% 0+440k 4+112io 49pf+0w
```

Re-evaluate model after new training

```
=============================================================== mlp BEGIN
mlp
-create  -pathexp /u/kukolich/Tutorial  -ferror X1mlp.err.eval\
 -fparam X1mlp.param  -pathdata /u/kukolich/Tutorial\
 -finput vowel.eval  -fdescribe vowel.defaults  -npatterns 166  -ninputs 2\
 -normalize  -fnorm vowel.norm.simple  -cross_valid 0\
 -fcross_valid vowel.train.cv  -random_cv  -random  -seed 0\
 -priors_npatterns 338  -debug 0  -verbose 3  -verror 2  -nodes 2,25,10\
 -alpha 0.6  -etta 0.2  -etta_list 0.2,0.2  -etta_change_type 0  -epsilon 0.1\
 -kappa 0.01  -etta_nepochs 0  -decay 0  -tolerance 0.01  -hfunction 0\
 -ofunction 0  -sigmoid_param 1  -sig_param_list 1,1  -cost_func 0\
 -cost_param 1  -epochs 20  -batch 1,1,0  -init_mag 0.1
Wed Apr  5 15:36:02 1995

Reading /u/kukolich/Tutorial/vowel.eval
```

Confusion matrix for re-evaluation

```
Classification Confusion Matrix - X1mlp.err.eval
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

Desired                          Computed Class
 Class     0     1     2     3     4     5     6     7     8     9   Total
 -----   ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
   0      13                 3           1                             17
   1            12                             6                       18
   2                  18                             2                 20
   3                        15           1           2                 18
   4                   2          13                             1     16
   5       1                             6           1           3     11
   6                                          18                       18
   7                               3                15                 18
   8                                                      13     3     16
   9                                     2           1     1    10     14
 -----   ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
 Total    14    12    20    18    16    10    24    21    14    17    166

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

Error summary for re-evaluation

```
Error Report - X1mlp.err.eval
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Class     Patterns   # Errors     % Errors StdDev RMS Err Label
  0          17          4        23.53   (10.3)     0.213  head
  1          18          6        33.33   (11.1)     0.227  hid
  2          20          2        10.00   ( 6.7)     0.152  hod
  3          18          3        16.67   ( 8.8)     0.172  had
  4          16          3        18.75   ( 9.8)     0.145  hawed
  5          11          5        45.45   (15.0)     0.231  heard
  6          18          0         0.00   ( 0.0)     0.038  heed
  7          18          3        16.67   ( 8.8)     0.187  hud
  8          16          3        18.75   ( 9.8)     0.171  whod
  9          14          4        28.57   (12.1)     0.224  hood
          --------     --------    ------- -------------

Overall      166         33        19.88   ( 3.1)    0.181

 TEST:      vowel.eval 19.88 % Err  0.181 RMS Err   0.48 secs
================================================================ mlp   END
0.4u 0.0s 0:00 92% 0+376k 0+3io 0pf+0w
```

Posterior probability plot table for new model

```
 ------------------------------------------------------------------------------

 target class =  2
   total number of patterns =  166
 ------------------------------------------------------------------------------
     BIN #     # PATTERNS  PREDICTED %     ACTUAL %      RANGE
 -----------  -----------  -----------  -----------  -----------
```

```
         0         142        1.36        1.41    -0.57 -   3.39
         1           5       30.01       20.00   -15.78 -  55.78
         2           9       63.01       77.78    50.06 - 105.49
         3          10       88.56      100.00   100.00 - 100.00

Chi Square Fit:
Chi               =   0.211159
Degrees of Freedom =   2
Significance      =   0.899803
 TARGET 2 CHI 0.211159 DOF 2 SIGNIFICANCE 0.899803
Created file /u/kukolich/Tutorial/X1mlp.prob.plot
ROC-x      ROC-y
```

ROC (detection) plot table for new model

```
-------------------------------------------------------------
target class = 2          reject = 0.00%
number of patterns from target class = 20
-------------------------------------------------------------
# PATTERNS  # CORRECT     % CORRECT  # FALSE  % FALSE_ALARM  THRESHOLD
 ----------  ----------  ----------  ----------  ----------  ----------
    0           0         0.000           0       0.000       1.000
   25          18        92.500           7       2.000       0.191
   33          19        97.500          14       6.000       0.081
  166          20       100.000         146      10.000       0.000
 TARGET 2 ROC AREA = 99.006859
Created file X1mlp.detect.plot
```

Rejection plot table for new model

```
 -----------------------------------------------------------------------------
 # PATTERNS # REJECTIONS  % REJECTED  # ERRORS     % ERROR    THRESHOLD
 ----------  ----------  ----------  ----------  ----------  ----------
       166           0    0.000000          33   19.879519       0.000
       149          16   10.000000          25   17.000448       0.426
       132          33   20.000000          21   16.112619       0.520
       116          49   30.000000          17   14.630121       0.583
        99          66   40.000000          15   15.660607       0.642
        83          83   50.000000           9   10.843373       0.693
        66          99   60.000000           5    7.530529       0.746
        49         116   70.000000           3    6.024490       0.813
        33         132   80.000000           0    0.000000       0.903
        16         149   90.000000           0    0.000000       0.949
Created file /u/kukolich/Tutorial/X1mlp.reject.plot
current directory:
/u/kukolich/Tutorial
```

C.2 KNN

This experiment was on the same training and evaluation data as the MLP experiment above. The KNN classifier obtained an 18% error rate on the evaluation data. Because this is a single pass classifier, there are no classification results from the training. Because this classifier does not produce continuous outputs, there are no tables from the posterior probability plot, ROC plot, or rejection plot.

C.2.1 KNN Shell Script

Train KNN

```
#!/bin/csh -ef
# ./X1knn.run
set loc=`pwd`

#train
(time knn\
 -train  -create  -pathexp $loc  -ferror X1knn.err.train  -fparam X1knn.param\
 -pathdata /u/kukolich/Tutorial  -finput vowel.train\
 -fdescribe vowel.defaults  -npatterns 338  -ninputs 2  -normalize\
 -fnorm vowel.norm.simple  -cross_valid 0  -fcross_valid vowel.train.cv\
 -random_cv  -random  -seed 0  -priors_npatterns 338  -debug 0  -verbose 3\
 -verror 2 \
-k 3 \
)|& nn_tee -h X1knn.log
```

Evaluate KNN model

```
#test
(time knn\
 -create  -pathexp $loc  -ferror X1knn.err.eval  -fparam X1knn.param\
 -pathdata /u/kukolich/Tutorial  -finput vowel.eval\
 -fdescribe vowel.defaults  -npatterns 166  -ninputs 2  -normalize\
 -fnorm vowel.norm.simple  -cross_valid 0  -fcross_valid vowel.train.cv\
 -random_cv  -random  -seed 0  -priors_npatterns 338  -debug 0  -verbose 3\
 -verror 2 \
-k 3 \
)|& nn_tee -h -a X1knn.log
echo -n "X1knn.run        " >> /u/kukolich/Tutorial/LNKnet.note
grep "TEST" X1knn.log | tail -1 >> /u/kukolich/Tutorial/LNKnet.note
```

Decision region plots

```
#decision region plot
knn_plot_bound\
 -autoscale  -pathexp $loc  -fparam X1knn.param\
 -fregion X1knn.region.plot.eval  -mac_wireframe_output -1\
 -fmacdots boundary_dots.mac  -fmacscatter scatter.mac\
 -fmaclines profile_lines.mac  -fmachisto profile_histo.mac\
 -fmacwireframe boundary_wire.mac  -xmin -3  -xmax 3  -ymin -3  -ymax 3\
 -xstep 1  -ystep 1  -pymin 0  -pymax 1.5  -pystep 0.25  -ninputs 2\
 -noutputs 10  -tregion "Norm:Simple K:3"  -first_dim 0  -second_dim 1\
 -npatterns 166  -npoints 100  -region  -scatter  -internals\
 -internals_scale 1  -internals_level 1  -color  -all  -distance 0.5\
 -fdata /u/kukolich/Tutorial/vowel.eval\
 -fdesc /u/kukolich/Tutorial/vowel.defaults \
 -k 3
echo "current directory:" >> X1knn.log
echo $loc >> X1knn.log
```

C.2.2 KNN Log File

Train KNN model

```
================================================================ knn BEGIN
knn
-train  -create  -pathexp /u/kukolich/Tutorial\
 -ferror X1knn.err.train  -fparam X1knn.param\
 -pathdata /u/kukolich/Tutorial  -finput vowel.train\
 -fdescribe vowel.defaults  -npatterns 338  -ninputs 2  -normalize\
 -fnorm vowel.norm.simple  -cross_valid 0  -fcross_valid vowel.train.cv\
 -random_cv  -random  -seed 0  -priors_npatterns 338  -debug 0  -verbose 3\
 -verror 2  -k 3
Wed Apr  5 14:15:56 1995

Reading /u/kukolich/Tutorial/vowel.train

Finished -- knn model saved in "/u/kukolich/Tutorial/X1knn.param"
================================================================ knn   END
0.2u 0.0s 0:07 4% 0+352k 5+2io 48pf+0w
```

Evaluate KNN model

```
================================================================ knn BEGIN
knn
-create  -pathexp /u/kukolich/Tutorial  -ferror X1knn.err.eval\
 -fparam X1knn.param  -pathdata /u/kukolich/Tutorial\
 -finput vowel.eval  -fdescribe vowel.defaults  -npatterns 166  -ninputs 2\
 -normalize  -fnorm vowel.norm.simple  -cross_valid 0\
 -fcross_valid vowel.train.cv  -random_cv  -random  -seed 0\
 -priors_npatterns 338  -debug 0  -verbose 3  -verror 2  -k 3
```

Confusion matrix

Error summary

```
Wed Apr  5 14:15:57 1995

Reading /u/kukolich/Tutorial/vowel.eval


Classification Confusion Matrix - X1knn.err.eval
------------------------------------------------------------------------------
------------------------------------------------------------------------------

Desired                        Computed Class
 Class    0     1     2     3     4     5     6     7     8     9   Total
 -----  ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
   0     15     1           1                                        17
   1           15                             3                      18
   2                 17           1                 2                20
   3      1               15            1           1                18
   4                  1          14                             1    16
   5      1                             8                       2    11
   6                                         18                      18
   7                        2                      15           1    18
   8                                                     12     4    16
   9                                    4           1     2     7    14
 -----  ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
 Total   17    16    18    18    15    13    21    19    14    15    166

------------------------------------------------------------------------------
------------------------------------------------------------------------------



Error Report - X1knn.err.eval
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Class     Patterns   # Errors     % Errors StdDev RMS Err    Label
  0          17          2        11.76   ( 7.8)     0.149  head
  1          18          3        16.67   ( 8.8)     0.136  hid
  2          20          3        15.00   ( 8.0)     0.145  hod
  3          18          3        16.67   ( 8.8)     0.169  had
  4          16          2        12.50   ( 8.3)     0.149  hawed
  5          11          3        27.27   (13.4)     0.234  heard
  6          18          0         0.00   ( 0.0)     0.000  heed
  7          18          3        16.67   ( 8.8)     0.153  hud
  8          16          4        25.00   (10.8)     0.154  whod
  9          14          7        50.00   (13.4)     0.270  hood
          --------     --------    ------- -------------

Overall      166         30        18.07   ( 3.0)    0.163

 TEST:      vowel.eval 18.07 % Err  0.163 RMS Err   0.42 secs
============================================================ knn   END
0.3u 0.0s 0:01 35% 0+392k 1+3io 1pf+0w
current directory:
/u/kukolich/Tutorial
```

C.3 GAUSS (Hand Picking Features)

This experiment was performed during the feature selection section of the tutorial. In this particular experiment a set of three features was typed in for use. The features are selected from a data base with eight features. The set of features was chosen by a forward feature search.

C.3.1 Gauss Shell Script

```
#!/bin/csh -ef
# ./last3gauss.run
set loc=`pwd`
```

Train Gaussian classifier

Hand-picked feature list is 7,5,6

```
#train
(time gauss\
 -train  -create  -pathexp $loc  -ferror last3gauss.err.train\
 -fparam last3gauss.param  -pathdata /u/kukolich/Tutorial\
 -finput gnoise_var.train  -fdescribe gnoise_var.defaults  -npatterns 200\
 -ninputs 3  -features 7,5,6  -normalize  -fnorm gnoise_var.norm.simple\
 -cross_valid 0  -fcross_valid gnoise_var.train.cv  -random_cv  -random\
 -seed 0  -priors_npatterns 200  -debug 0  -verbose 3  -verror 2 \
-minvar 1e-05  -max_ratio 1e+06 \
)|& nn_tee -h last3gauss.log
```

Evaluate classifier

```
#test
(time gauss\
 -create  -pathexp $loc  -ferror last3gauss.err.eval  -fparam last3gauss.param\
 -pathdata /u/kukolich/Tutorial  -finput gnoise_var.eval\
 -fdescribe gnoise_var.defaults  -npatterns 100  -ninputs 3  -features 7,5,6\
 -normalize  -fnorm gnoise_var.norm.simple  -cross_valid 0\
 -fcross_valid gnoise_var.train.cv  -random_cv  -random  -seed 0\
 -priors_npatterns 200  -debug 0  -verbose 3  -verror 2 \
-minvar 1e-05  -max_ratio 1e+06 \
)|& nn_tee -h -a last3gauss.log
echo -n "last3gauss.run   " >> /u/kukolich/Tutorial/LNKnet.note
grep "TEST" last3gauss.log | tail -1 >> /u/kukolich/Tutorial/LNKnet.note
```

Decision region plot

```
#decision region plot
gauss_plot_bound\
 -autoscale  -pathexp $loc  -fparam last3gauss.param\
 -fregion last3gauss.region.plot.eval  -mac_wireframe_output -1\
 -fmacdots boundary_dots.mac  -fmacscatter scatter.mac\
 -fmaclines profile_lines.mac  -fmachisto profile_histo.mac\
 -fmacwireframe boundary_wire.mac  -xmin -3  -xmax 3  -ymin -3  -ymax 3\
 -xstep 1  -ystep 1  -pymin 0  -pymax 1.5  -pystep 0.25  -ninputs 8\
 -noutputs 10  -tregion "Norm:Simple Diagonal Grand"  -first_dim 0\
 -second_dim 1  -npatterns 100  -npoints 100  -region  -scatter  -internals\
 -internals_scale 1  -internals_level 1  -color  -all  -distance 0.5\
 -fdata /u/kukolich/Tutorial/gnoise_var.eval\
 -fdesc /u/kukolich/Tutorial/gnoise_var.defaults
echo "current directory:" >> last3gauss.log
echo $loc >> last3gauss.log
```

C.3.2 Gauss Log File

Train Gaussian classifier

```
============================================================= gauss BEGIN
gauss
-train  -create  -pathexp /u/kukolich/Tutorial\
 -ferror last3gauss.err.train  -fparam last3gauss.param\
 -pathdata /u/kukolich/Tutorial  -finput gnoise_var.train\
 -fdescribe gnoise_var.defaults  -npatterns 200  -ninputs 3  -features 7,5,6\
 -normalize  -fnorm gnoise_var.norm.simple  -cross_valid 0\
 -fcross_valid gnoise_var.train.cv  -random_cv  -random  -seed 0\
 -priors_npatterns 200  -debug 0  -verbose 3  -verror 2  -minvar 1e-05\
 -max_ratio 1e+06
Wed Apr  5 17:04:10 1995

Reading /u/kukolich/Tutorial/gnoise_var.train

Finished -- gauss model saved in "/u/kukolich/Tutorial/last3gauss.param"
============================================================= gauss   END
0.3u 0.0s 0:07 4% 0+368k 6+1io 51pf+0w
```

Evaluate classifier

```
============================================================== gauss BEGIN
gauss
-create  -pathexp /u/kukolich/Tutorial\
 -ferror last3gauss.err.eval  -fparam last3gauss.param\
 -pathdata /u/kukolich/Tutorial  -finput gnoise_var.eval\
 -fdescribe gnoise_var.defaults  -npatterns 100  -ninputs 3  -features 7,5,6\
 -normalize  -fnorm gnoise_var.norm.simple  -cross_valid 0\
 -fcross_valid gnoise_var.train.cv  -random_cv  -random  -seed 0\
 -priors_npatterns 200  -debug 0  -verbose 3  -verror 2  -minvar 1e-05\
 -max_ratio 1e+06
Wed Apr  5 17:04:15 1995

Reading /u/kukolich/Tutorial/gnoise_var.eval


Classification Confusion Matrix - last3gauss.err.eval
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

Confusion matrix

```
Desired                          Computed Class
 Class     0     1     2     3     4     5     6     7     8     9   Total
 -----   ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
   0      10                                                           10
   1       1     9                                                     10
   2                   9     1                                         10
   3                        10                                         10
   4                              10                                   10
   5                                    10                             10
   6                                          10                       10
   7                                                10                 10
   8                                                       9     1     10
   9                                                            10     10
 -----   ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  -----
 Total    11     9     9    11    10    10    10    10     9    11    100

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------



Error Report - last3gauss.err.eval
-------------------------------------------------------------------------------
```

Error summary

```
-------------------------------------------------------------------------------
Class     Patterns   # Errors     % Errors StdDev Avg LogL   Label
  0          10         0          0.00   ( 0.0)    -0.466  0
  1          10         1         10.00   ( 9.5)    -0.610  1
  2          10         1         10.00   ( 9.5)    -0.542  2
  3          10         0          0.00   ( 0.0)    -0.259  3
  4          10         0          0.00   ( 0.0)    -0.548  4
  5          10         0          0.00   ( 0.0)    -0.363  5
  6          10         0          0.00   ( 0.0)    -0.074  6
  7          10         0          0.00   ( 0.0)    -0.449  7
  8          10         1         10.00   ( 9.5)    -0.663  8
  9          10         0          0.00   ( 0.0)    -0.810  9
          --------     --------    ------- ------------

Overall      100          3          3.00   ( 1.7)   -0.478

 TEST: gnoise_var.eval 3.00 % Err -0.478 Avg LogL   0.31 secs
============================================================== gauss   END
0.2u 0.0s 0:02 12% 0+404k 1+2io 1pf+0w
current directory:
/u/kukolich/Tutorial
```

C.4 RBF with KMEANS (Cross Validation)

In this experiment, a series of radial basis function classifiers were trained on data taken from different kinds of iris flowers. The basis functions used in each classifier were generated in a separate call to LNKnet's K-means algorithm. Because there is not much data for the iris problem, the experiment used cross validation to get a good estimate of the classification error rate. Cross validation is explained in Section 5.7 on page 89.

C.4.1 RBF Cross Validation Shell Script

```
#!/bin/csh -ef
# ./cvrbf.run
set loc=`pwd`
```

Cluster using Kmeans

```
#cross validation
(time kmeans\
 -create  -pathexp $loc  -ferror cvrbf.err.cv  -fparam cvkmeans.param\
 -pathdata /u/kukolich/Tutorial  -finput iris.train\
 -fdescribe iris.defaults  -npatterns 150  -ninputs 4  -normalize\
 -fnorm iris.norm.simple  -cross_valid 5  -fcross_valid iris.train.cv\
 -random_cv  -random  -seed 0  -priors_npatterns 150  -debug 0  -verbose 3\
 -verror 2 \
-cluster_by_class  -ncenters 2  -split_percentage 1  -add_random_offset\
 -max_iteration 10  -stop_percentage 1  -reduce_step 1 \
```

Do RBF 5 fold cross validation

```
)|& nn_tee -h cvrbf.log
(time rbf\
 -create  -pathexp $loc  -ferror cvrbf.err.cv  -fparam cvrbf.param\
 -pathdata /u/kukolich/Tutorial  -finput iris.train\
 -fdescribe iris.defaults  -npatterns 150  -ninputs 4  -normalize\
 -fnorm iris.norm.simple  -cross_valid 5  -fcross_valid iris.train.cv\
 -random_cv  -random  -seed 0  -priors_npatterns 150  -debug 0  -verbose 3\
 -verror 2 \
-fclparam cvkmeans.param  -hspread 1  -exhspread 1  -max_ratio 1e+06\
 -minvar 1e-06  -fast_nhidden 0 \
)|& nn_tee -h -a cvrbf.log
```

Put cross validation results in the notebook file

```
echo -n "cvrbf.run        " >> /u/kukolich/Tutorial/LNKnet.note
grep "CV" cvrbf.log | tail -1 >> /u/kukolich/Tutorial/LNKnet.note
echo "current directory:" >> cvrbf.log
echo $loc >> cvrbf.log
```

C.4.2 RBF Cross Validation Log File

Kmeans clustering

```
================================================================ kmeans BEGIN
kmeans
-create  -pathexp /u/kukolich/Tutorial  -ferror cvrbf.err.cv\
 -fparam cvkmeans.param  -pathdata /u/kukolich/Tutorial\
 -finput iris.train  -fdescribe iris.defaults  -npatterns 150  -ninputs 4\
 -normalize  -fnorm iris.norm.simple  -cross_valid 5\
 -fcross_valid iris.train.cv  -random_cv  -random  -seed 0\
 -priors_npatterns 150  -debug 0  -verbose 3  -verror 2  -cluster_by_class\
 -ncenters 2  -ncenters_list 2,2,2  -split_percentage 1  -add_random_offset\
 -max_iteration 10  -stop_percentage 1  -reduce_step 1
Thu Apr 13 14:04:11 1995
```

Finding clusters for fold 0

```
Reading /u/kukolich/Tutorial/iris.train
>>>>>> FOLD 0 <<<<<<
```

Finding clusters for class 0 fold 0

```
training centers for class 0
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
    1      0.000          1
    2      1.068          2
    3      0.469          2
```

```
   4      0.469          2
```

Finding clusters for class 1 fold 0

```
training centers for class 1
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
   1      0.000          1
   2      1.006          2
   3      0.520          2
   4      0.511          2
   5      0.511          2
```

Finding clusters for class 2 fold 0

```
training centers for class 2
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
   1      0.000          1
   2      1.441          2
   3      0.789          2
   4      0.785          2
```

Finding clusters for fold 1

```
>>>>>> FOLD 1 <<<<<<
```

Finding clusters for class 0 fold 1

```
training centers for class 0
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
   1      0.000          1
   2      1.031          2
   3      0.499          2
   4      0.499          2
```

Finding clusters for class 1 fold 1

```
training centers for class 1
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
   1      0.000          1
   2      0.971          2
   3      0.470          2
   4      0.466          2
```

Finding clusters for class 2 fold 1

```
training centers for class 2
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
   1      0.000          1
   2      1.195          2
   3      0.696          2
   4      0.692          2
```

Finding clusters for fold 2

```
>>>>>> FOLD 2 <<<<<<
training centers for class 0
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
   1      0.000          1
   2      0.894          2
   3      0.432          2
   4      0.432          2
training centers for class 1
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
   1      0.000          1
   2      1.090          2
   3      0.546          2
   4      0.545          2
training centers for class 2
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
   1      0.000          1
   2      1.431          2
   3      0.795          2
   4      0.795          2
```

Finding clusters for fold 3

```
>>>>>> FOLD 3 <<<<<<
training centers for class 0
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
   1      0.000          1
   2      0.891          2
   3      0.390          2
   4      0.384          2
   5      0.363          2
   6      0.356          2
   7      0.356          2
training centers for class 1
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
   1      0.000          1
   2      1.053          2
   3      0.510          2
   4      0.510          2
training centers for class 2
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
   1      0.000          1
   2      1.250          2
   3      0.725          2
   4      0.724          2
```

Finding clusters for fold 4

```
>>>>>> FOLD 4 <<<<<<
training centers for class 0
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
   1      0.000          1
   2      0.778          2
   3      0.358          2
   4      0.358          2
training centers for class 1
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
   1      0.000          1
   2      0.942          2
   3      0.510          2
   4      0.510          2
training centers for class 2
 EPOCH ave.sq.err ncenters (40 patterns/epoch)
   1      0.000          1
   2      1.291          2
   3      0.730          2
   4      0.725          2

Finished -- model saved in “/u/kukolich/Tutorial/cvkmeans.param”
================================================================ kmeans   END
0.2u 0.0s 0:02 11% 0+368k 3+1io 47pf+0w
```

Running RBF cross validation experiment

```
================================================================ rbf BEGIN
rbf
-create  -pathexp /u/kukolich/Tutorial  -ferror cvrbf.err.cv\
 -fparam cvrbf.param  -pathdata /u/kukolich/Tutorial\
 -finput iris.train  -fdescribe iris.defaults  -npatterns 150  -ninputs 4\
 -normalize  -fnorm iris.norm.simple  -cross_valid 5\
 -fcross_valid iris.train.cv  -random_cv  -random  -seed 0\
 -priors_npatterns 150  -debug 0  -verbose 3  -verror 2\
 -fclparam cvkmeans.param  -hspread 1  -exhspread 1  -max_ratio 1e+06\
 -minvar 1e-06  -fast_nhidden 0
Thu Apr 13 14:04:15 1995

Reading cluster parameters from “/u/kukolich/Tutorial/cvkmeans.param”

Building rbf model

Reading /u/kukolich/Tutorial/iris.train
>>>>>> FOLD 0 <<<<<<

Inverting rbf matrix
```

Confusion matrix for fold 0

```
Classification Confusion Matrix - cvrbf.err.cv
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

Desired                          Computed Class
 Class     0     1     2   Total
 -----   ----  ----  ----  -----
   0      10                 10
   1            10           10
   2                  10     10
 -----   ----  ----  ----  -----
 Total    10    10    10     30
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

Error summary for fold 0

```
Error Report - cvrbf.err.cv
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Class     Patterns   # Errors     % Errors StdDev RMS Err
   Label
  0          10          0         0.00   ( 0.0)     0.000  Setosa
  1          10          0         0.00   ( 0.0)     0.010  Versicolour
  2          10          0         0.00   ( 0.0)     0.098  Virginica
          --------     --------    ------- -------------

Overall       30          0         0.00   ( 0.0)    0.057

>>>>>> FOLD 1 <<<<<<

Inverting rbf matrix
```

Results for fold 1

```
Classification Confusion Matrix - cvrbf.err.cv
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

Desired                          Computed Class
 Class     0     1     2   Total
 -----   ----  ----  ----  -----
   0      10               10
   1            9     1     10
   2                 10     10
 -----   ----  ----  ----  -----
 Total    10     9    11     30

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

```
Error Report - cvrbf.err.cv
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Class     Patterns   # Errors     % Errors StdDev RMS Err
   Label
  0          10          0         0.00   ( 0.0)    39.669  Setosa
  1          10          1        10.00   ( 9.5)    39.669  Versicolour
  2          10          0         0.00   ( 0.0)    39.669  Virginica
          --------     --------    ------- -------------

Overall       30          1         3.33   ( 3.3)   39.669

>>>>>> FOLD 2 <<<<<<

Inverting rbf matrix
```

Results for fold 2

```
Classification Confusion Matrix - cvrbf.err.cv
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

Desired                          Computed Class
 Class     0     1     2   Total
 -----   ----  ----  ----  -----
   0      10                10
   1           10           10
   2                 10     10
 -----   ----  ----  ----  -----
 Total    10    10    10     30

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------



Error Report - cvrbf.err.cv
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Class     Patterns   # Errors     % Errors StdDev RMS Err
   Label
  0          10          0         0.00   ( 0.0)    39.669  Setosa
  1          10          0         0.00   ( 0.0)    39.669  Versicolour
  2          10          0         0.00   ( 0.0)    39.669  Virginica
          --------     --------    ------- ------------

Overall       30          0         0.00   ( 0.0)   39.669

>>>>>> FOLD 3 <<<<<<

Inverting rbf matrix
```

Results for fold 3

```
Classification Confusion Matrix - cvrbf.err.cv
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

Desired                          Computed Class
 Class     0     1     2   Total
 -----   ----  ----  ----  -----
   0      10                10
   1            8     2     10
   2            2     8     10
 -----   ----  ----  ----  -----
 Total    10    10    10     30

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------



Error Report - cvrbf.err.cv
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Class     Patterns   # Errors     % Errors StdDev RMS Err
   Label
  0          10          0         0.00   ( 0.0)     0.577  Setosa
  1          10          2        20.00   (12.6)     0.577  Versicolour
  2          10          2        20.00   (12.6)     0.577  Virginica
          --------     --------    ------- ------------

Overall       30          4        13.33   ( 6.2)    0.577

>>>>>> FOLD 4 <<<<<<

Inverting rbf matrix
```

Results for fold 4

```
Classification Confusion Matrix - cvrbf.err.cv
------------------------------------------------------------------------------
------------------------------------------------------------------------------

Desired                         Computed Class
 Class     0     1     2   Total
 -----   ----  ----  ----  -----
   0      10               10
   1            9     1    10
   2                 10    10
 -----   ----  ----  ----  -----
 Total    10    9    11    30

------------------------------------------------------------------------------
------------------------------------------------------------------------------
```

```
Error Report - cvrbf.err.cv
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Class     Patterns   # Errors     % Errors StdDev RMS Err
   Label
  0          10          0         0.00   ( 0.0)     0.577  Setosa
  1          10          1        10.00   ( 9.5)     0.577  Versicolour
  2          10          0         0.00   ( 0.0)     0.577  Virginica
          --------     --------    ------- -------------

Overall       30          1         3.33   ( 3.3)    0.577
```

```
Finished -- rbf model saved in “/u/kukolich/Tutorial/cvrbf.param”
>>>>>> OVERALL <<<<<<
```

Overall confusion matrix

```
Classification Confusion Matrix - cvrbf.err.cv
------------------------------------------------------------------------------
------------------------------------------------------------------------------

Desired                         Computed Class
 Class     0     1     2   Total
 -----   ----  ----  ----  -----
   0      50               50
   1           46     4    50
   2            2    48    50
 -----   ----  ----  ----  -----
 Total    50    48    52   150

------------------------------------------------------------------------------
------------------------------------------------------------------------------
```

Overall error summary

```
Error Report - cvrbf.err.cv
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Class     Patterns   # Errors     % Errors StdDev RMS Err
   Label
  0          50          0         0.00   ( 0.0)    25.092  Setosa
  1          50          4         8.00   ( 3.8)    25.092  Versicolour
  2          50          2         4.00   ( 2.8)    25.092  Virginica
          --------     --------    ------- -------------

Overall      150          6         4.00   ( 1.6)   25.092
```

```
 CV 5:      iris.train 4.00 % Err   25.1 RMS Err   0.41 secs
================================================================ rbf   END
0.3u 0.0s 0:04 8% 0+396k 2+3io 48pf+0w
current directory:
/u/kukolich/Tutorial
```

C.5 Experiment Notebook File

The experiment notebook holds a short description of each experiment run during a LNKnet session. A notebook entry includes a short list of experiment parameters, training results, testing results, and information from the posterior probability plot and the ROC (detection) plot. Each line starts with the experiment shell script name, which allows the user to find lines pertinent to a particular experiment using the UNIX grep and awk programs.

First 20 epochs of MLP training

```
X1mlp.run vowel simple   mlp -nodes 2,25,10  -alpha 0.6  -etta 0.2  -etta_change_type
0  -epsilon 0.1  -kappa 0.01  -etta_nepochs 0  -decay 0  -tolerance 0.01  -hfunction
0  -ofunction 0  -sigmoid_param 1  -cost_func 0  -cost_param 1  -epochs 20  -batch
1,1,0  -init_mag 0.1
X1mlp.run         LAST TRAIN EPOCH:  20 33.73 % Err  0.217 RMS Err  18.01 secs
X1mlp.run         TEST:      vowel.eval 32.53 % Err  0.207 RMS Err   0.54 secs
X1mlp.run         TARGET 2 CHI 2.446746 DOF 2 SIGNIFICANCE 0.294236
X1mlp.run         TARGET 2 ROC AREA = 98.732872
```

KNN experiment

```
X1knn.run vowel simple   knn -k 3
X1knn.run         TEST:      vowel.eval 18.07 % Err  0.163 RMS Err   0.42 secs
```

Second 20 epochs of MLP training

```
X1mlp.run vowel simple  -continue  mlp -nodes 2,25,10  -alpha 0.6  -etta 0.2  -
etta_change_type 0  -epsilon 0.1  -kappa 0.01  -etta_nepochs 0  -decay 0  -tolerance
0.01  -hfunction 0  -ofunction 0  -sigmoid_param 1  -cost_func 0  -cost_param 1  -
epochs 20  -batch 1,1,0  -init_mag 0.1
X1mlp.run         LAST TRAIN EPOCH:  20 25.44 % Err  0.194 RMS Err  17.80 secs
X1mlp.run         TEST:      vowel.eval 19.88 % Err  0.181 RMS Err   0.48 secs
X1mlp.run         TARGET 2 CHI 0.211159 DOF 2 SIGNIFICANCE 0.899803
X1mlp.run         TARGET 2 ROC AREA = 99.006859
```

Feature selection and normalization experiments

```
allgauss.run gnoise_var simple   gauss -minvar 1e-05  -max_ratio 1e+06
allgauss.run      TEST: gnoise_var.eval 1.00 % Err  -1.87 Avg LogL   0.38 secs

N1gauss.run gnoise_var simple  -ninputs 1  gauss -minvar 1e-05  -max_ratio 1e+06
N1gauss.run       TEST: gnoise_var.eval 78.00 % Err  -1.41 Avg LogL   0.36 secs

N2gauss.run gnoise_var simple  -ninputs 2  gauss -minvar 1e-05  -max_ratio 1e+06
N2gauss.run       TEST: gnoise_var.eval 62.00 % Err  -1.71 Avg LogL   0.36 secs

N4gauss.run gnoise_var simple  -ninputs 4  gauss -minvar 1e-05  -max_ratio 1e+06
N4gauss.run       TEST: gnoise_var.eval 48.00 % Err  -2.22 Avg LogL   0.35 secs

last3gauss.run gnoise_var simple -features 7,5,6 -ninputs 3  gauss -minvar 1e-05  -
max_ratio 1e+06
last3gauss.run    TEST: gnoise_var.eval 3.00 % Err -0.478 Avg LogL   0.31 secs

pcagauss.run gnoise_var pca  -ninputs 2  gauss -minvar 1e-05  -max_ratio 1e+06
pcagauss.run      TEST: gnoise_var.eval 25.00 % Err  -2.67 Avg LogL   0.36 secs

ldagauss.run gnoise_var lda  -ninputs 2  gauss -minvar 1e-05  -max_ratio 1e+06
ldagauss.run      TEST: gnoise_var.eval 0.00 % Err  -1.71 Avg LogL   0.33 secs
```

Cross validation experiment

```
cvrbf.run iris simple  -cross_valid 5  kmeans -cluster_by_class  -ncenters 2  -
split_percentage 1  -add_random_offset  -max_iteration 10  -stop_percentage 1  -
reduce_step 1 rbf -fclparam cvkmeans.param  -hspread 1  -exhspread 1  -max_ratio
1e+06  -minvar 1e-06  -fast_nhidden 0
cvrbf.run         CV 5:      iris.train 4.00 % Err   25.1 RMS Err   0.41 secs
```

Repeat of cvrbf with more kmeans clusters and fast training

```
cvrbf.run {kmeans -ncenters 4 } {rbf -fast_train -fast_nhidden 2 }
cvrbf.run         CV 5:      iris.train 4.67 % Err   17.7 RMS Err   0.51 secs
```

Second repeat with fast training off

```
cvrbf.run {rbf [-fast_train ] }
cvrbf.run         CV 5:      iris.train 4.67 % Err   25.1 RMS Err   0.51 secs
```

C.6 C Code Generation From a Parameter File

In this example a Gaussian classifier was trained on the XOR data base shown on page 188. The Gaussian classifier has a full covariance matrix for each class, as described in Section 6.3.2 on page 97. The input data was normalized using simple normalization. The decision regions produced by this Gaussian classifier are shown in Figure 7.5 on page 114. A C subroutine file was generated from the parameter file as described in Section 7.2. The shell script that produced the routine is shown in Figure C.6.2. The subroutine file is shown in Figure C.6.3. Finally, a small program was written that uses the generated classification subroutine to draw the decision region plot in Section C.6.4.

C.6.1 Gauss Parameter File

This parameter file was created by the program gauss which was called in the shell script XORgauss.run.

FIGURE C.1
Settings for all GAUSS command line arguments

Gauss Parameter file (XORgauss.param)

```
gauss
-train  -create  -pathexp /u/kukolich/Tutorial\
 -ferror XORgauss.err.train  -fparam XORgauss.param\
 -pathdata /u/kukolich/lnknet/data/class  -finput XOR.train\
 -fdescribe XOR.defaults  -npatterns 16  -ninputs 2  -normalize\
 -fnorm XOR.norm.simple  -cross_valid 0  -fcross_valid XOR.train.cv\
 -random_cv  -random  -seed 0  -priors_npatterns 16  -debug 0  -verbose 3\
 -verror 2  -full  -per_class  -minvar 1e-05  -max_ratio 1e+06
Fri Apr  7 09:49:19 1995
```

Normalization parameters from XOR.norm.simple

```
normalization data
<BEGIN_PARAM>
normalization  1
ninputs        2
nclasses       2
total_trials   16
max_features   2
<END_PARAM>
<BEGIN_PARAM>
{BEGIN_VECTOR
means 2
0    0.5
1    0.5
END_VECTOR}
{BEGIN_VECTOR
sdevs 2
0    0.504975
1    0.504975
END_VECTOR}
<END_PARAM>
```

Gauss parameters

```
gauss model
<BEGIN_PARAM>
```

Gauss constants

```
ninputs        2
noutputs       2
total_trials   16
full_covar     1
per_class      1
<END_PARAM>
<BEGIN_PARAM>
{BEGIN_VECTOR
```

No feature selection

```
features 1
0    0
END_VECTOR}
{BEGIN_VECTOR
```

Npatterns seen per class over all training

```
class_trials 2
0    8
1    8
END_VECTOR}
<END_PARAM>
<BEGIN_PARAM>
{BEGIN_VECTOR
```

Means for first class

```
mean 2
0    -1.49012e-08
1    -7.45058e-09
END_VECTOR}
<END_PARAM>
<BEGIN_PARAM>
{BEGIN_VECTOR
```

Means for second class

```
mean 2
0    0
1    -7.45058e-09
END_VECTOR}
<END_PARAM>
```

Determinant and inverse covariance matrix for first class

```
<BEGIN_PARAM>
log_det        -1.41082
{BEGIN_MATRIX
inv_covar 2 2
0 0    25.7526
0 1    -25.2477
1 0    -25.2477
1 1    25.7526
END_MATRIX}
<END_PARAM>
```

Determinant and inverse covariance matrix for second class

```
<BEGIN_PARAM>
log_det        -1.41082
{BEGIN_MATRIX
inv_covar 2 2
0 0    25.7524
0 1    25.2475
1 0    25.2475
1 1    25.7524
END_MATRIX}
<END_PARAM>
```

C.6.2 C code generation shell script

This shell script, XORgauss.c.run, was created from LNKnet's C Code generation window. The program gauss2c will produce two subroutines, classify_XORgauss() and normalize_XORgauss(), shown in Section C.6.3.

FIGURE C.2

GAUSS2C script (XORgauss.c.run)

```
#!/bin/csh -ef
# ./XORgauss.c.run
gauss2c -model_file XORgauss.param  -suffix XORgauss  >! XORgauss.c
```

C.6.3 Sample gauss2c Output

The filter gauss2c to produces two C subroutines, by default named classify() and normalize(). The routine classify() takes a vector of raw inputs and a pointer to a vector of outputs. It calls normalize() and then calculates the outputs of the Gaussian classifier for the normalized inputs. The outputs are copied into the output vector sent by the calling routine. The number of the highest output is returned as the class of the input vector. The routine normalize() takes a raw input pattern and performs normalization and feature selection. The normalized inputs are stored in the vector that held the original pattern. Normalize() returns the number of input features used by the classifier. gauss2c has a flag, -suffix, which adds a suffix to the subroutine names produced. This allows easy inclusion of multiple classification routines in one user program. For example, in Figure C.2 the suffix flag was set to XORgauss. The classify and normalize subroutine names are thus classify_XORgauss() and normalize_XORgauss().

For the parameter file in Figure C.1, simple normalization is used. There are two input features and two classes. Each class has its own covariance matrix and that is a full covariance matrix.

An example of the use of these routines in a program has been added after the routine normalize_XORgauss(), in Figure C.4. This example generates a list of patterns and their output classes for a decision region plot.

FIGURE C.3
Comments describing Gaussians and normalization

Output of gauss2c (XORgauss.c)

```
/* XORgauss.param */
/* Per Class, Full */
/* norm: SIMPLE */
/* 2 (all) features */


#include <math.h>


 /* macro definitions */
#ifndef SQR(x)
#define SQR(x) ((x) * (x))
#endif

#ifndef M_LOG10E
#define M_LOG10E 0.43429448190325182765
#endif

#ifndef M_LN10
#define M_LN10 2.30258509299404568402
#endif

#ifndef LOG_2PI
#define LOG_2PI 0.798179868358
#endif
```

Macros for functions and constants

```
 /* function declarations */
extern int classify_XORgauss(/* float *,float * */);
extern int normalize_XORgauss(/* float * */);


#define NRAW_XORgauss 2
#define NINPUTS_XORgauss 2
#define NCLASSES_XORgauss 2
```

classify_XORgauss

```
int classify_XORgauss (inputs, outputs)
   float *inputs,*outputs;
{
  int n, best;
  int j,k;
  float y[NCLASSES_XORgauss],p;
```

Means for Gaussians

```
 static float means[NCLASSES_XORgauss][NINPUTS_XORgauss] = {
        { -1.490120e-08,  -7.450580e-09, },
        {  0.000000e+00,  -7.450580e-09, },
};
```

Inverse covariance matrices for each class

```
 static float inv_var[NCLASSES_XORgauss][NINPUTS_XORgauss][NINPUTS_XORgauss] = {
        /* class 0 */{
          {  2.575260e+01,  -2.524770e+01, },
          { -2.524770e+01,   2.575260e+01, },
},
        /* class 1 */{
          {  2.575240e+01,   2.524750e+01, },
          {  2.524750e+01,   2.575240e+01, },
},
};
```

Log determinants of covariance matrices

```
 static float log_determinant[NCLASSES_XORgauss] = {
 -1.410820e+00,  -1.410820e+00, };
```

Log *a priori* class probabilities

```
static float log_class_priors[NCLASSES_XORgauss] = {
         -3.010300e-01,  -3.010300e-01, };
  float x[NRAW_XORgauss];
  int i;
```

Copy inputs into input array, x, then do normalization and feature selection

```
 /* load inputs */
  for(i = 0; i < NRAW_XORgauss; i++)
    x[i] = inputs[i];

  /* normalize loaded data */
  normalize_XORgauss(x);
```

Calculate outputs for each class

```
/* calculate outputs for each class */
  for(n = 0; n < NCLASSES_XORgauss; n++){
    y[n] = 0;
    for(j = 0; j < NINPUTS_XORgauss; j++){
      p = 0.;
      for(k = 0; k < NINPUTS_XORgauss; k++){
        p += (x[k]-means[n][k])*inv_var[n][j][k];
      }
      y[n] += (x[j] - means[n][j])*p;
    }
    y[n] *= 0.5 * M_LOG10E;
    y[n] += log_determinant[n] * 0.5 + NINPUTS_XORgauss*0.5*LOG_2PI;
    y[n] = -y[n];
    y[n] += log_class_priors[n];
  }
```

Copy outputs into user supplied array, making them linear

```
 /* copy outputs and make linear */
  for(n = 0; n < NCLASSES_XORgauss; n++)
    outputs[n] = exp(M_LN10 * (double)y[n]);
```

Find highest output and return its class

```
 /* find highest output */
  for(best = n = 0; n < NCLASSES_XORgauss; n++)
```

```
    if(y[best] < y[n]) best = n;
  return(best);
}
```

normalize_XORgauss

```
int normalize_XORgauss(inputs)
   float *inputs;
{
  int n;
  float new_inputs[NRAW_XORgauss];
```

Means and variances for simple normalization

```
static float means[NINPUTS_XORgauss] = {
          5.000000e-01,   5.000000e-01, };

static float sdevs[NINPUTS_XORgauss] = {
          5.049750e-01,   5.049750e-01, };
```

Normalize inputs and copy result to new_inputs

```
 for(n = 0; n < NINPUTS_XORgauss; n++){
    new_inputs[n] = (inputs[n]- means[n]) / sdevs[n];
  }
```

Do features selection on new_inputs, copying selected features back into inputs. Return the number of normalized selected features

```
 for(n = 0; n < NINPUTS_XORgauss; n++){
    inputs[n] = new_inputs[n];
  }
  return((int) NINPUTS_XORgauss);
}
#undef NRAW_XORgauss
#undef NINPUTS_XORgauss
#undef NCLASSES_XORgauss
```

FIGURE C.4 Main driver program that uses LNKnet classifier code.

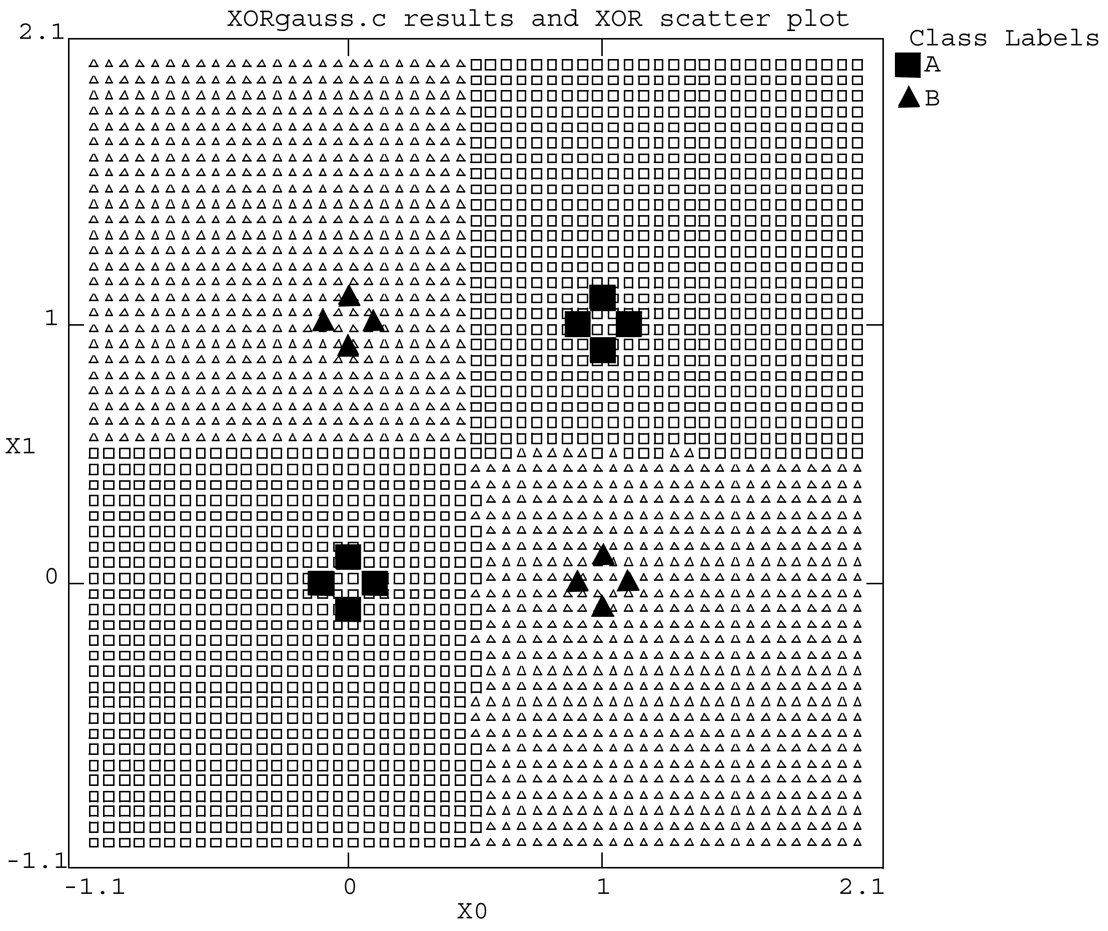
Main routine for generating decision region patterns and classes.

```
/* program for generating decision regions */
main()
{
  float x,y, xstep, ystep;
  float inputs[2], outputs[2];
  int class;
  float xlow = -1, xhigh = 2, ylow = -1, yhigh = 2;

  /* sample a 50 by 50 grid of patterns between (-1,-1) and (2,2) */
  xstep = (xhigh - xlow)/50.;
  ystep = (yhigh - ylow)/50.;
  for(x = xlow; x < xhigh; x += xstep){
    inputs[0] = x;
    for(y = ylow; y < yhigh; y += ystep){
      inputs[1] = y;
      /* find the class for the current pattern */
      class = classify_XORgauss(inputs, outputs);
      /* print out the results */
      printf("%f %f %d\n", x, y, class);
    }
  }
}
```

C.6.4 XOR Decision Region and Scatter Plot

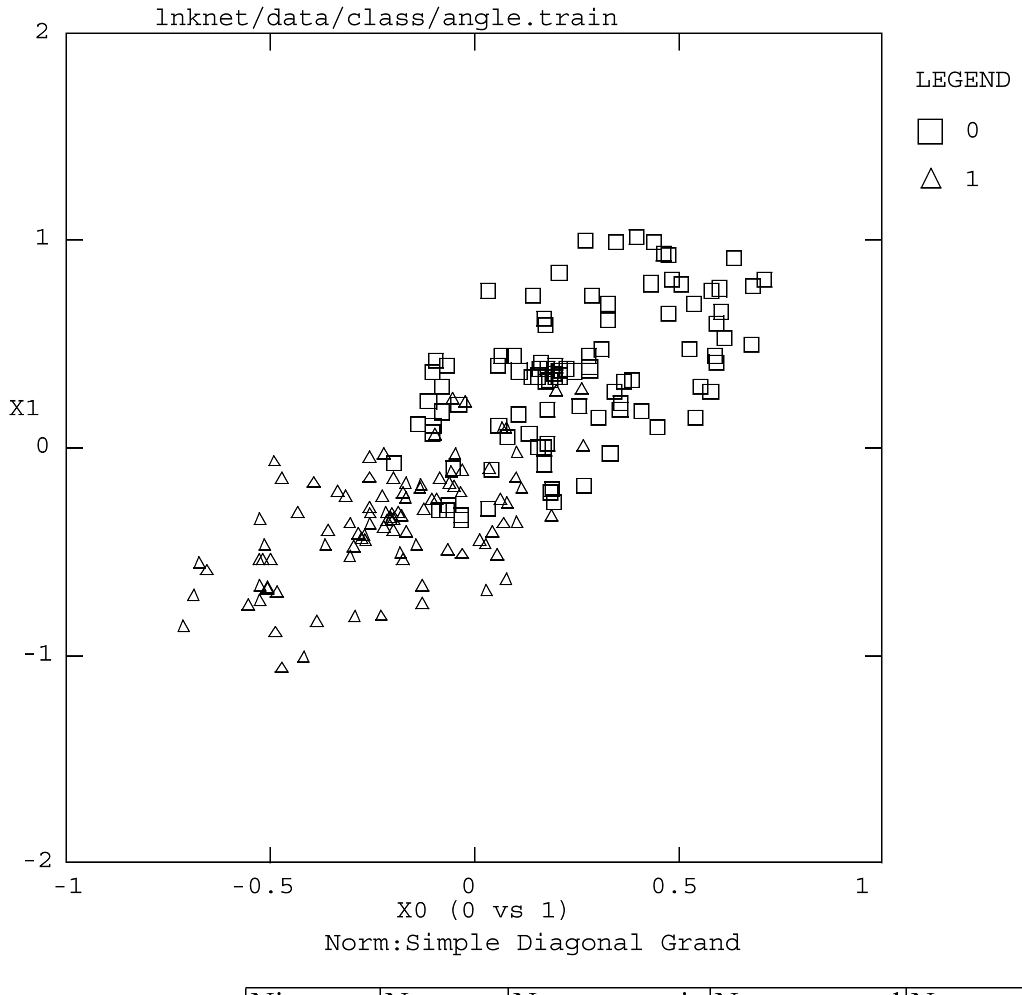
The decision region plot in Figure C.5 was produced using the program in Figure C.3 and Figure C.4. The outputs of the program were fed to a plotting package. Small squares represent points in the input space where class A is chosen by the classifier in Figure C.1. Small triangles represent points where class B is chosen. The large filled squares are the patterns from class A in the training file XOR.train. The large filled triangles are the patterns from class B.

FIGURE C.5 XOR.train Scatter Plot and Decision Region Plot generated by XORgauss.c



APPENDIX D Data Bases Included in LNKnet

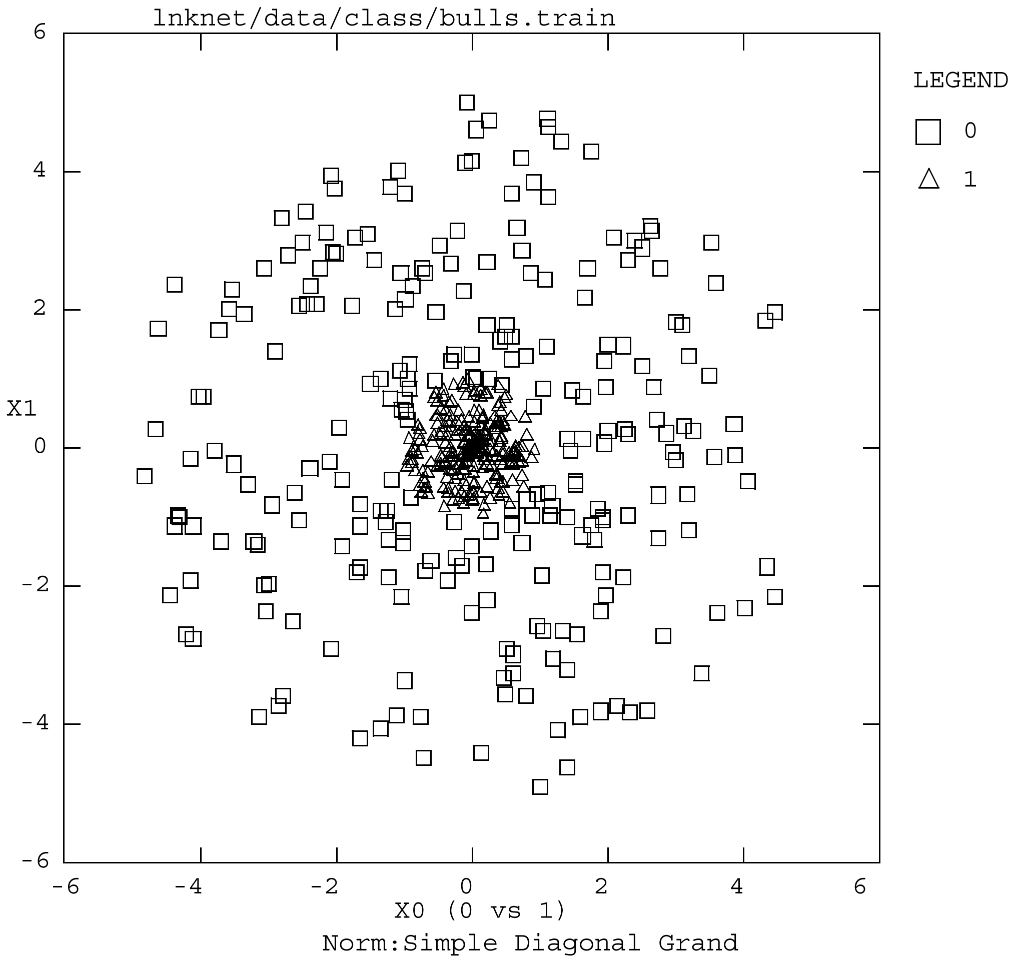
FIGURE D.1 angle



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	2	200	100	100

Angle is a generated data base with two classes. Each class is made of points uniformly sampled from a pair of ovals which are identical except for the positions of their centers. After the points were generated the data was rotated 60 degrees to put the classes at an angle to the origin.

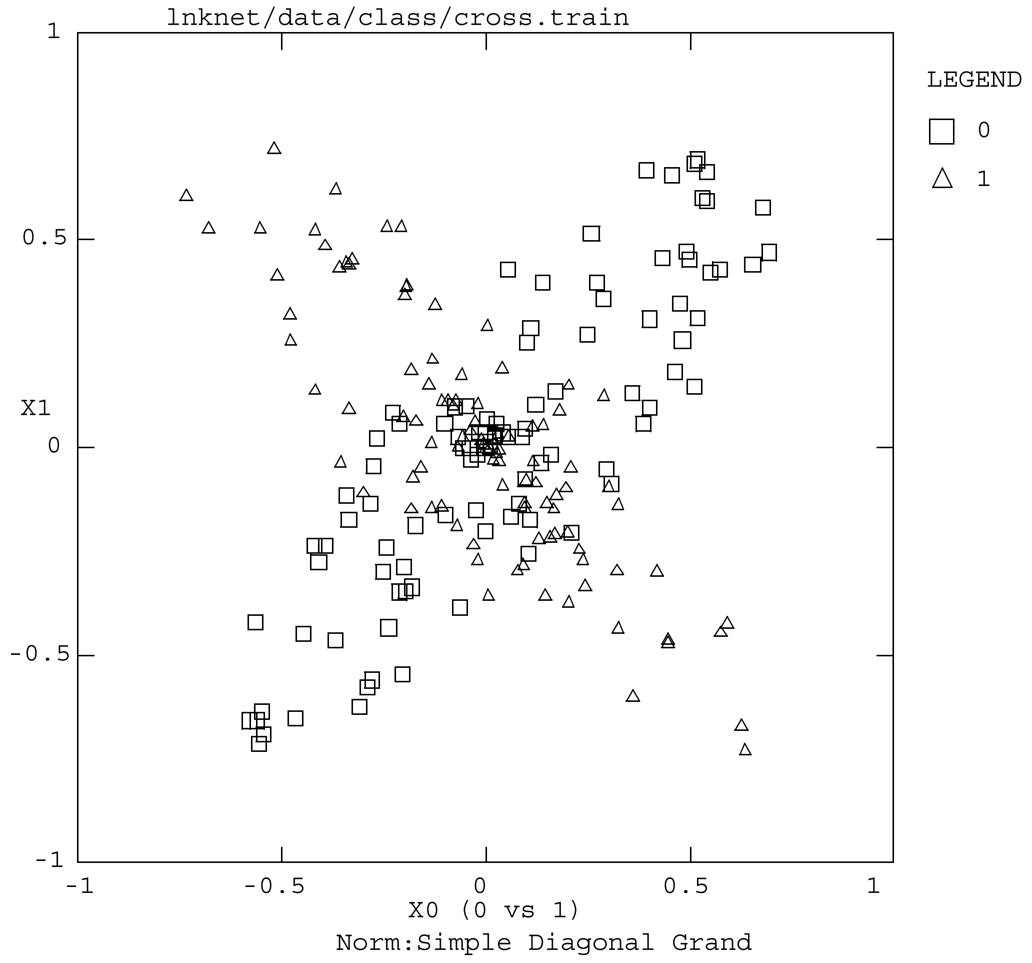
FIGURE D.2 bulls



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	2	500	250	250

This is bull's-eye data. There are two classes. One contains patterns uniformly distributed in a disk of radius 1.0 and the other class contains patterns uniformly distributed in an annulus with an inner radius of 1.0 and an outer radius of 5.0.

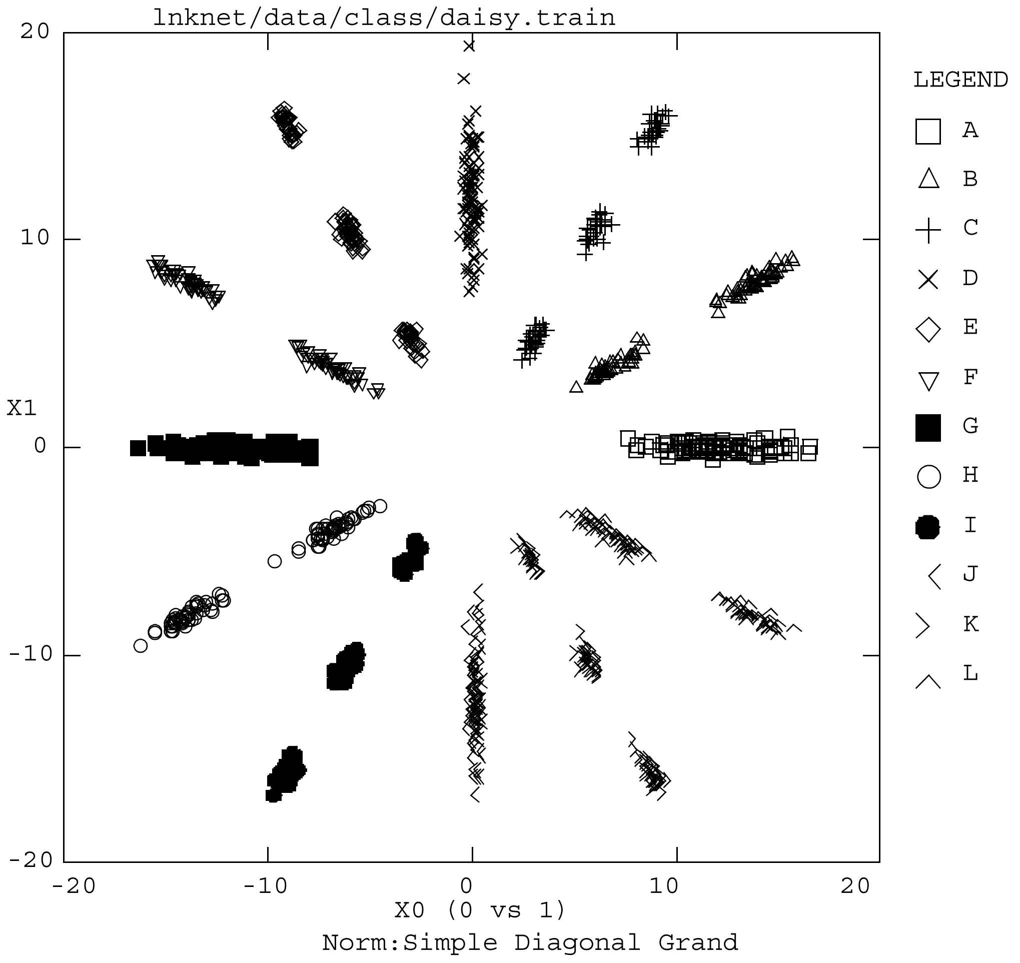
FIGURE D.3 cross



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	2	200	100	100

Class zero contains patterns from a 2-D Gaussian distribution with positively correlated features. Class one contains patterns from a similar 2-D Gaussian distribution with negatively correlated features. These distributions overlap and form a cross as shown.The data base was intended for use in testing the Gaussian classifier with full covariance matrices.

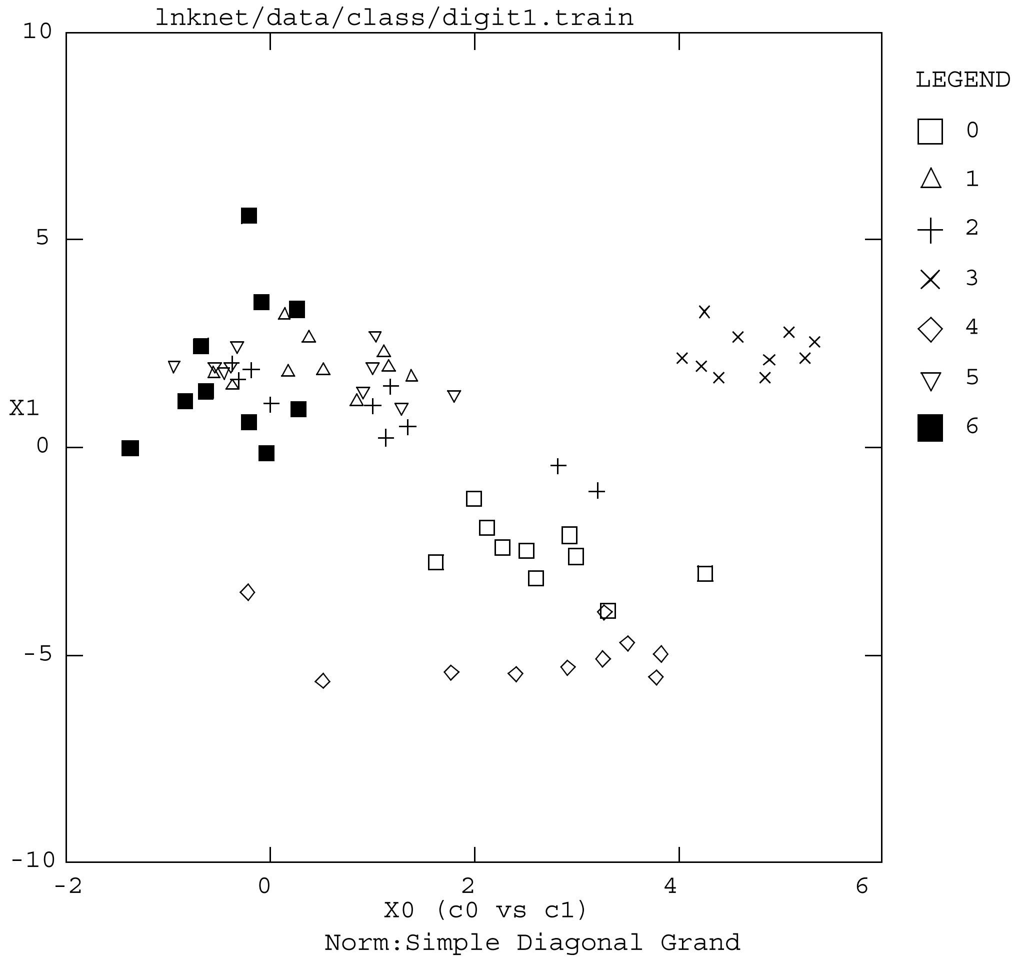
FIGURE D.4 daisy



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	12	1200	1200	1200

This data base has 12 classes. The points in each class are in Gaussian clusters which radiate out from the origin. There are one, two, or three Gaussians per class. This data base was generated to test the Gaussian Mixture Classifier.

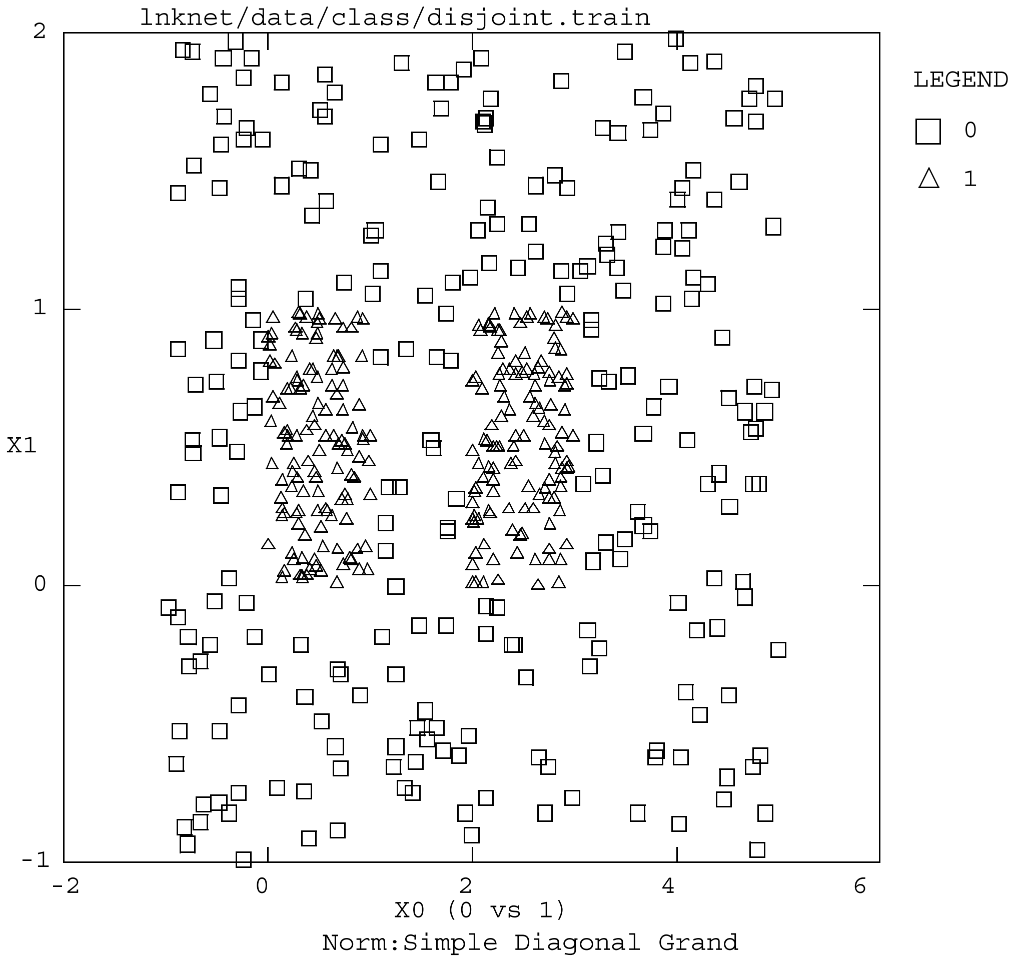
FIGURE D.5 digit1



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
22	7	70	56	56

The classes in this speech data base are the first seven monosyllabic digits from the TI digit data base. A version of the TI data base was sampled at 12kHz and processed to extract 15 mel cepstra from 10 msec frames. Eleven of the low cepstral values were used from two frames of each word. One frame was taken where the energy was highest and the other frame is from 30 msec before the highest energy frame. This data base was generated by Richard Lippmann of MIT Lincoln Laboratory [28].

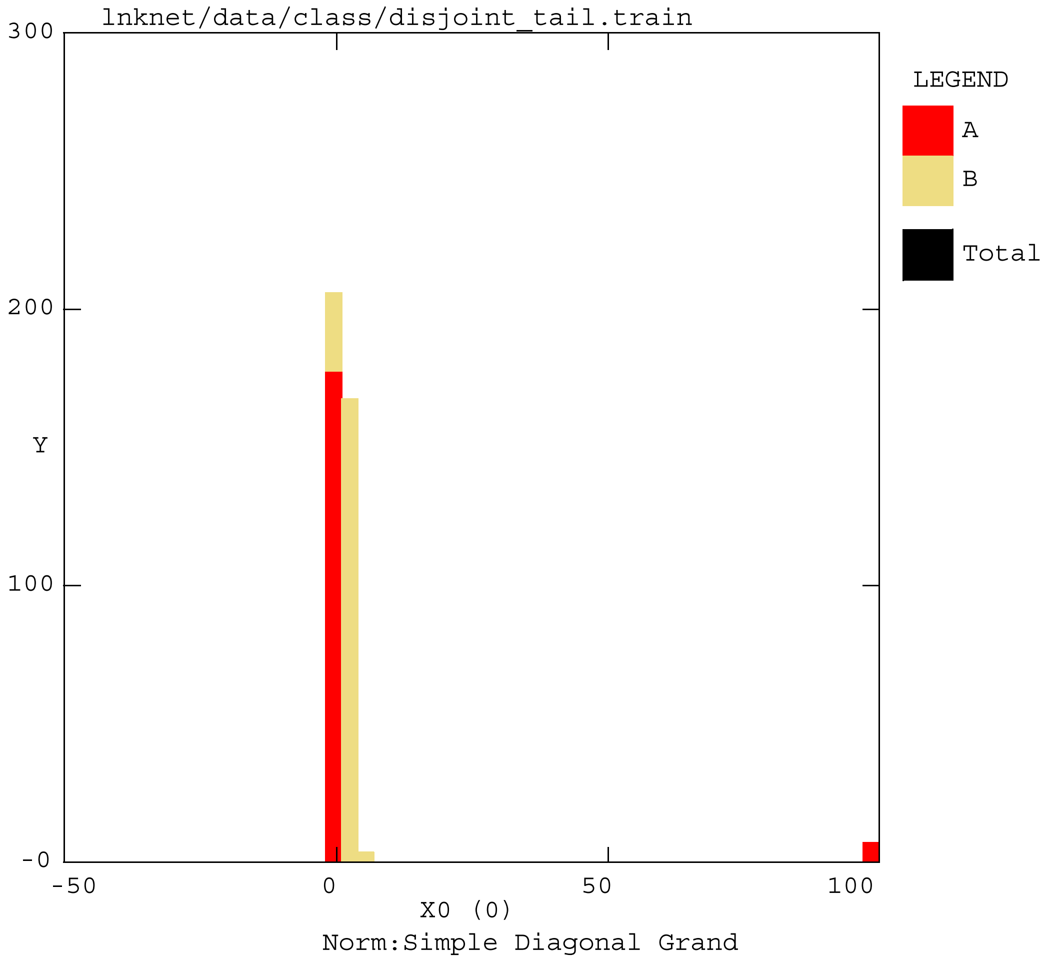
FIGURE D.6 disjoint



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	2	500	250	250

Class 0 contains patterns uniformly sampled from a 6 by 3 unit rectangle. There are two square regions where there are no patterns from class 0. The first square lies between (0,0) and (1,1). The second square lies between (2,0) to (3,1). Class 1 contains patterns uniformly sampled within these squares.

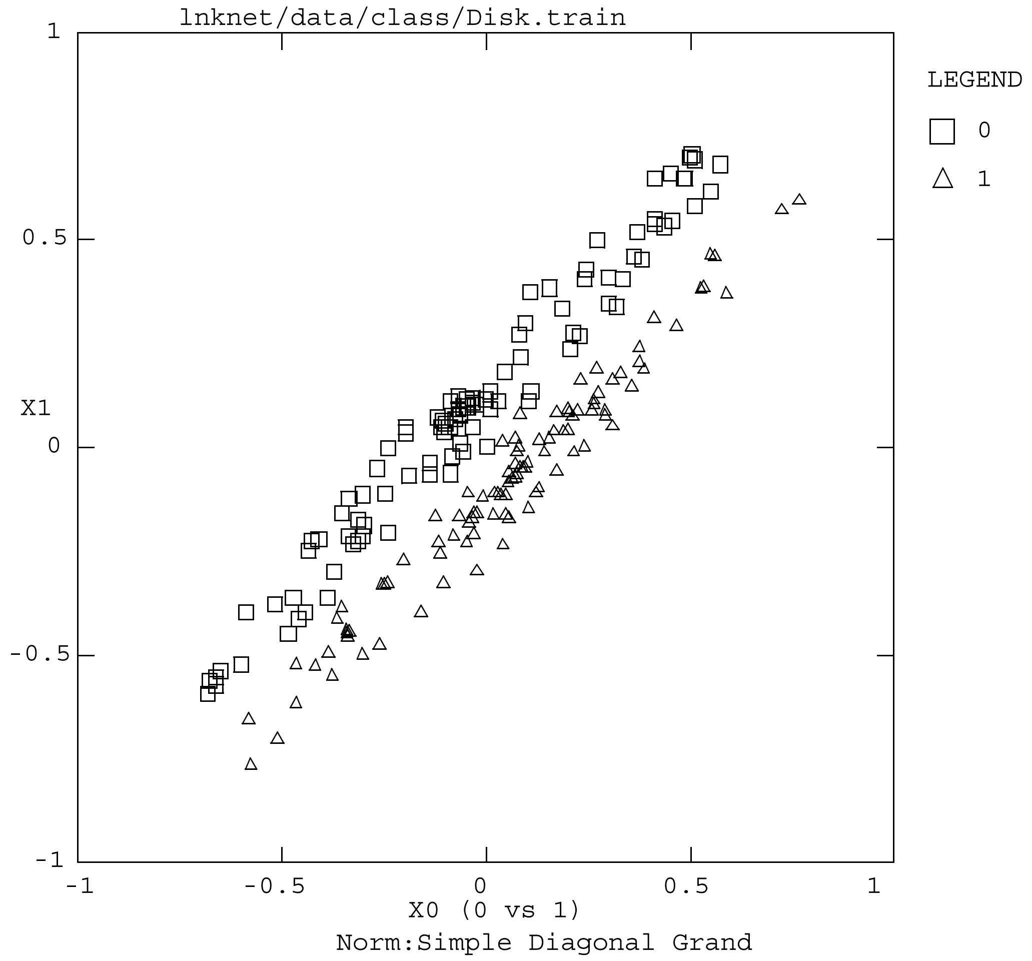
FIGURE D.7 disjoint_tail



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
1	2	400	200	200

Class A contains patterns with a bimodal distribution. Most of the patterns are found in a uniform distribution covering the range (-0.5,0.5). An additional 10% of the class A patterns are found in a second uniform distribution covering the range (99.5,100.5). The class B patterns have a Gaussian distribution with a mean at 2 and standard deviation of 1.0

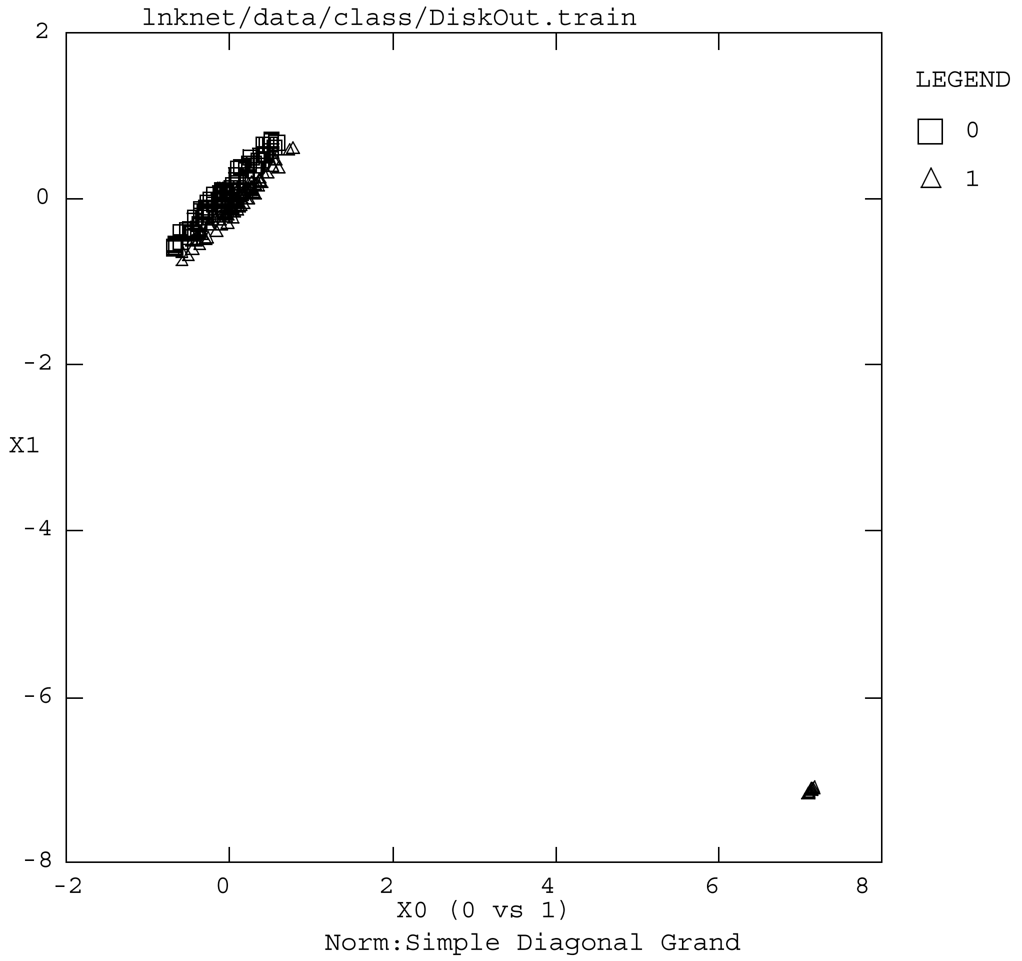
FIGURE D.8 Disk



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	2	200	100	100

This data base has two uniformly sampled ellipses which have been rotated to put them at a 45 degree angle to the input features.

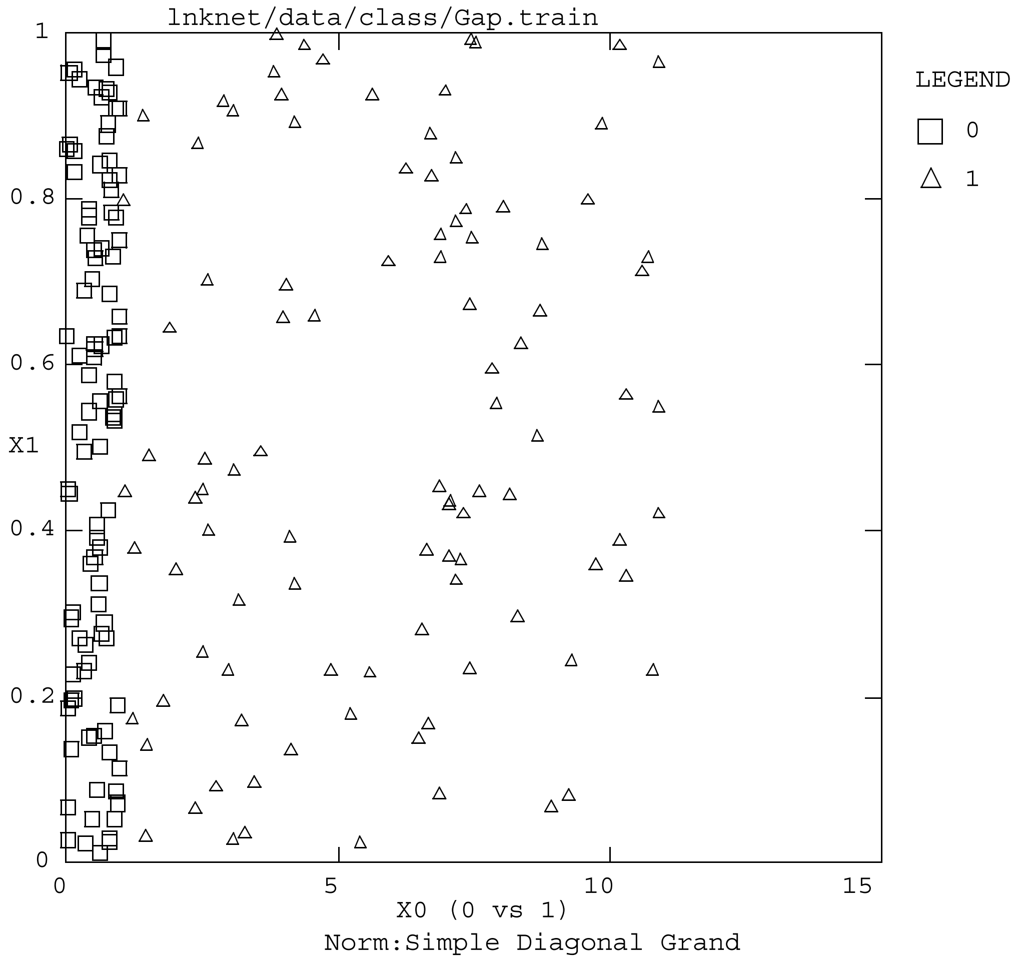
FIGURE D.9 DiskOut



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	2	210	105	105

This data base has two uniformly sampled ellipses which have been rotated to put them at a 45 degree angle to the input features. Class 1 has an additional set of patterns separated from the main ellipse by 10 units.

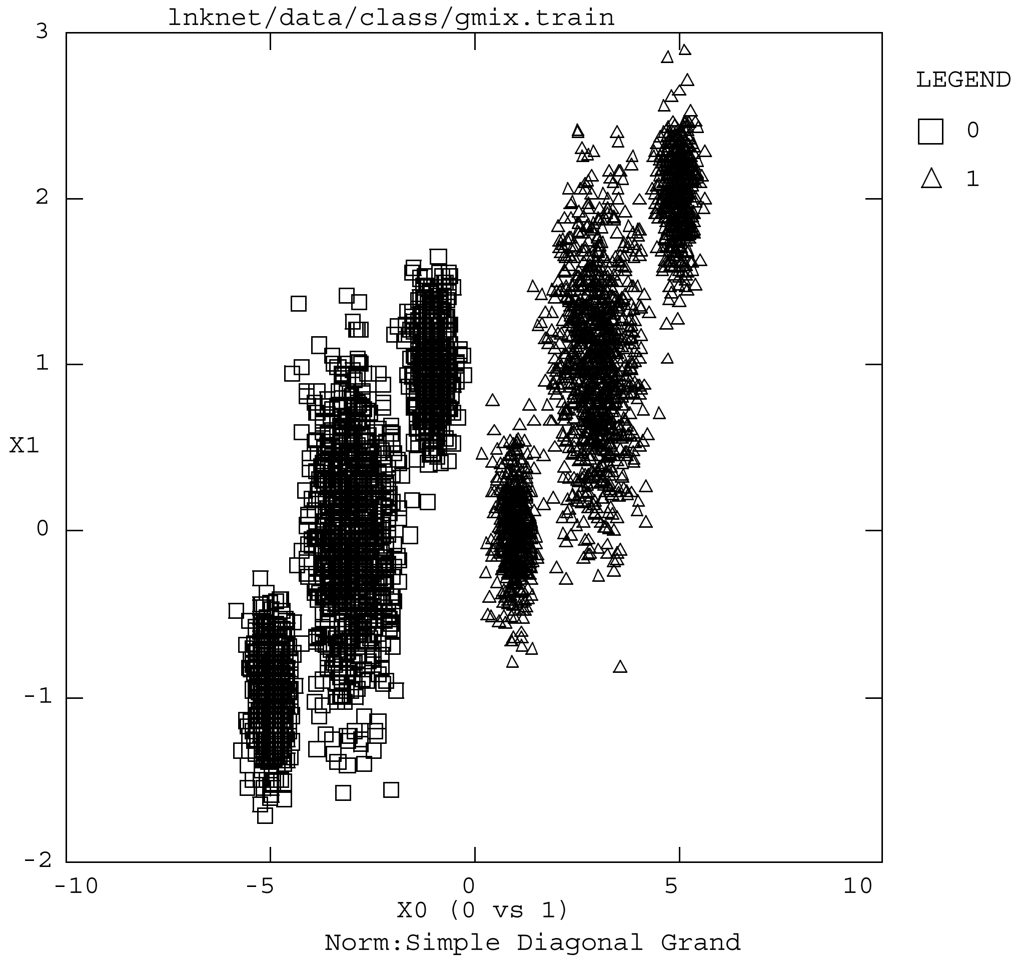
FIGURE D.10 Gap



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	2	200	100	100

The data for each class is uniformly sampled from a rectangle with height 1. The width of the rectangle for class 0 is 1, the width for class 1 is 10. Each class has the same number of patterns. This data base was generated while testing the Perceptron Convergence Procedure cost function of the MLP classifier.

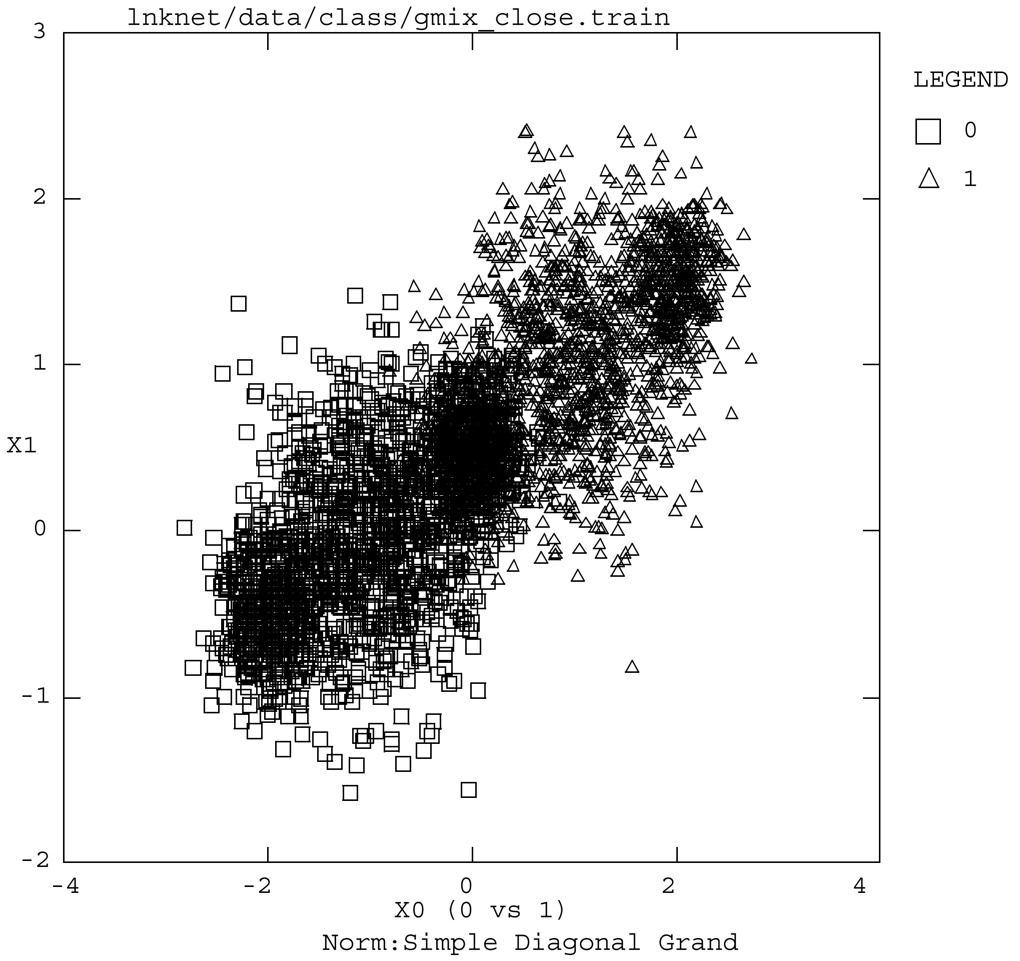
FIGURE D.11 gmix



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	2	4000	2000	2000

The patterns for each class are taken from Gaussian mixture distributions. Each Gaussian mixture distribution has three clusters, as shown, with one half of the patterns in the central cluster and one quarter of the patterns in each of the other two clusters. This data base was generated to test the Gaussian mixture classifier using diagonal covariance matrices.

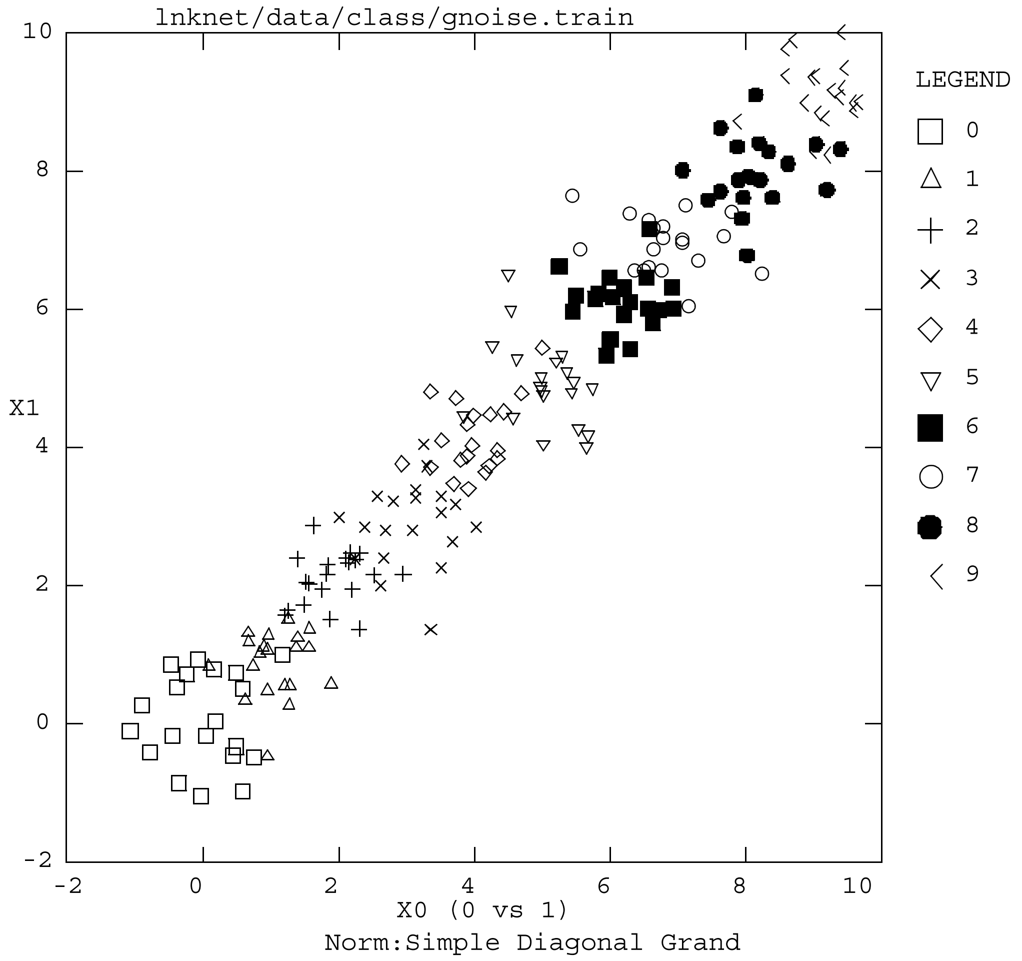
FIGURE D.12 gmix_close



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	2	4000	2000	2000

The patterns for each class are taken from Gaussian mixture distributions. Each Gaussian mixture distribution has three clusters as shown with one half of the patterns in the central cluster and one quarter of the patterns in each of the other two clusters. This data base is very similar to the gmix data base. The class distributions are considerably closer together, however and overlap.

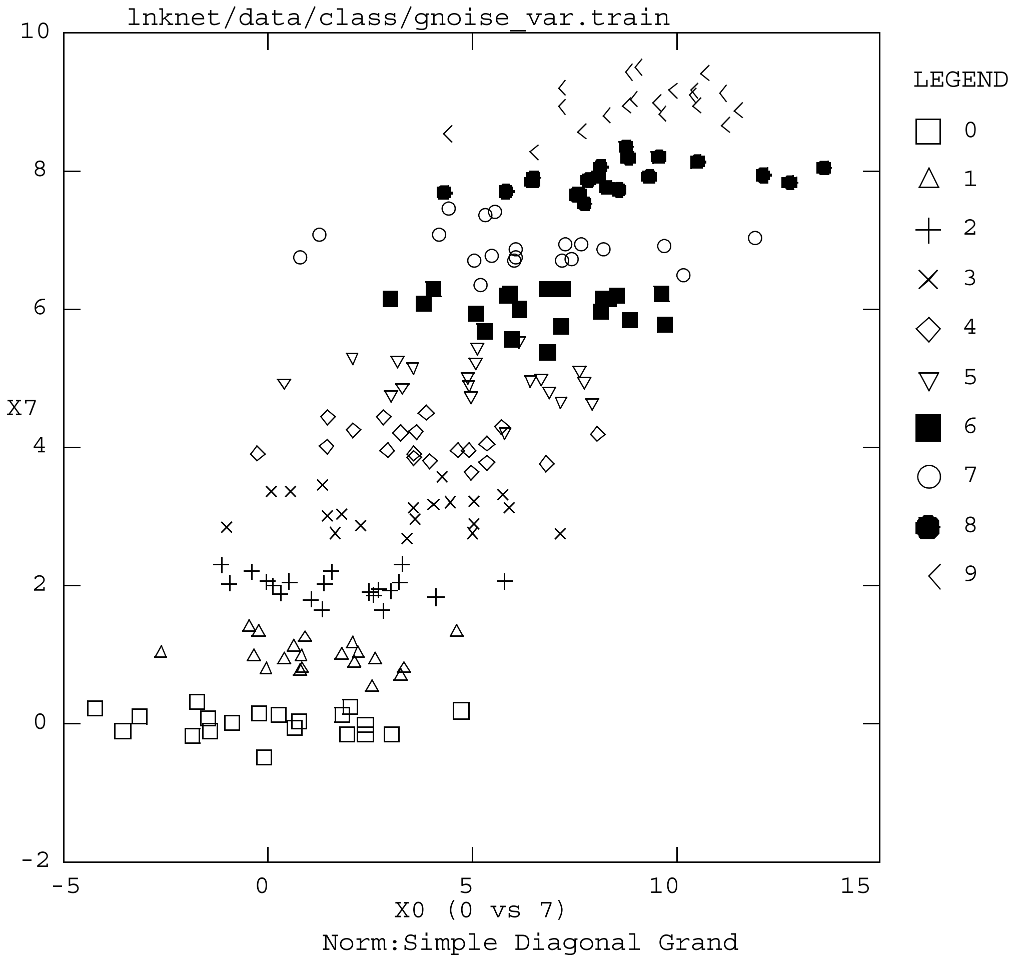
FIGURE D.13 gnoise



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
8	10	200	100	100

Each pattern was generated by adding Gaussian noise to the center of each class in all eight input dimensions. The standard deviation of the noise is 0.5. The class means are found along the line $x_d = j, (0 \le d \le 7), (0 \le j \le 9)$ where x_d is the value of the dth feature and j is the class.

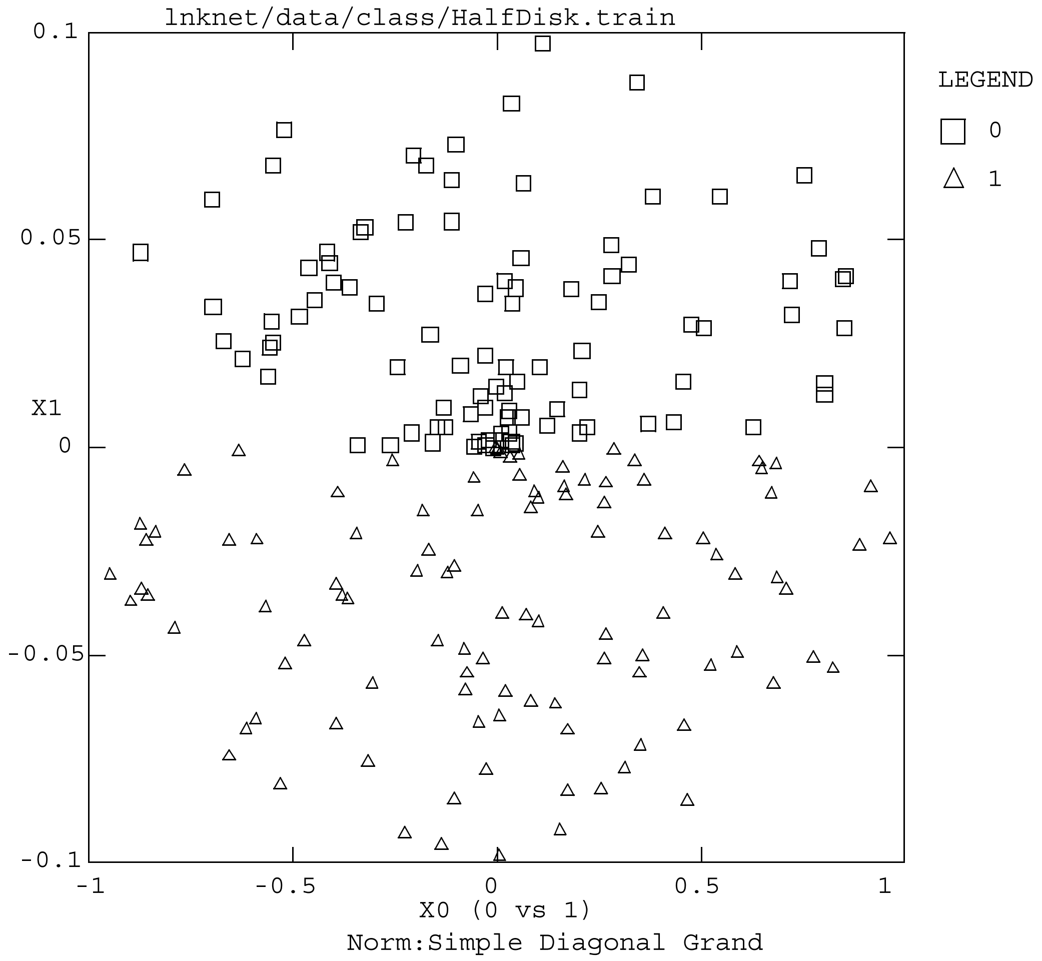
FIGURE D.14 gnoise_var



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
8	10	200	100	100

This is a modified version of gnoise where the variance is smaller for the higher input features. Each pattern was generated by adding Gaussian noise to the center of each class in all eight input dimensions. The standard deviation decreases as the dimensions increase. The standard deviation is $\sigma_d = (8 - d)0.25$ where d is the number of the dimension $(0 \le d \le 7)$. The class means are found along the line $x_d = j$, $(0 \le d \le 7)$, $(0 \le j \le 9)$. The plot shows the first, noisiest, dimension plotted against the last, most clean, dimension.

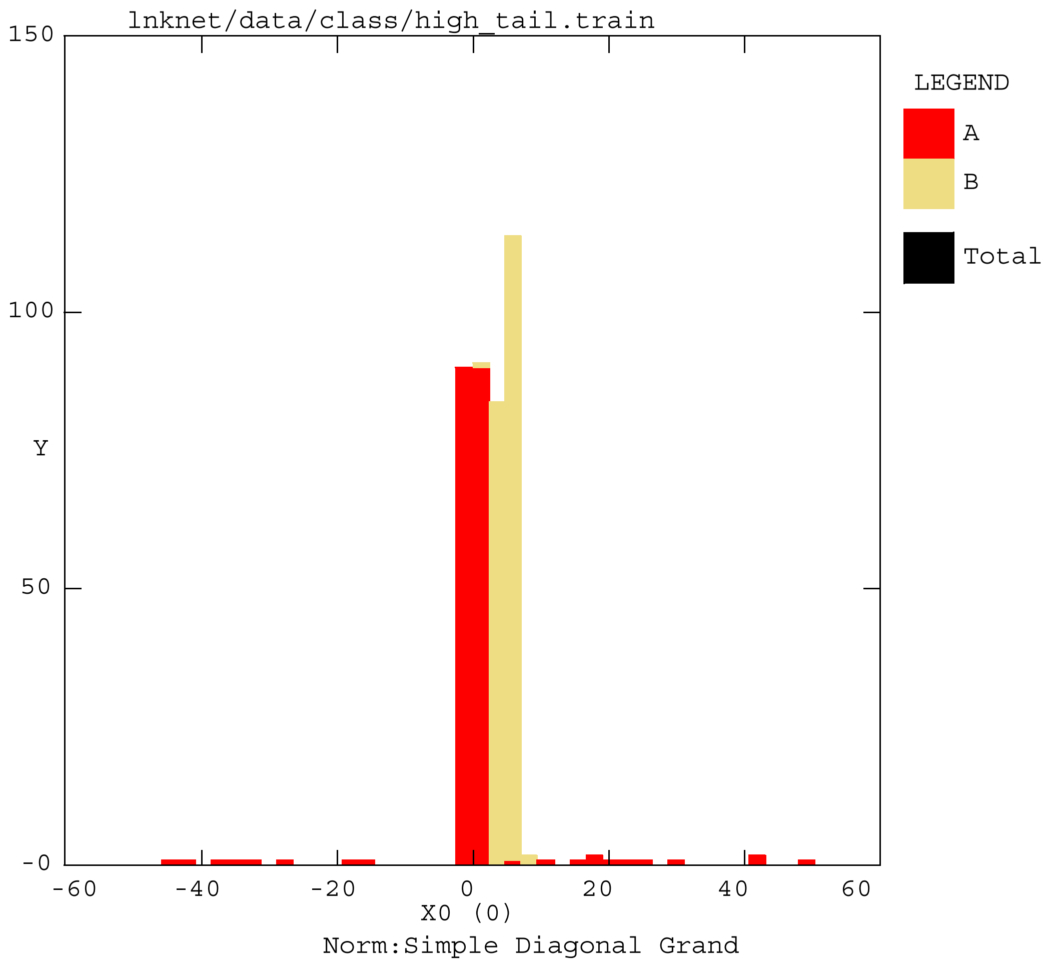
FIGURE D.15 HalfDisk



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	2	200	100	100

The patterns in this data base were all uniformly sampled from an ellipse which is ten times longer in the first direction than in the second. All the sampled patterns with $X1 > 0$ were assigned to class 0. All other patterns were assigned to class 1.

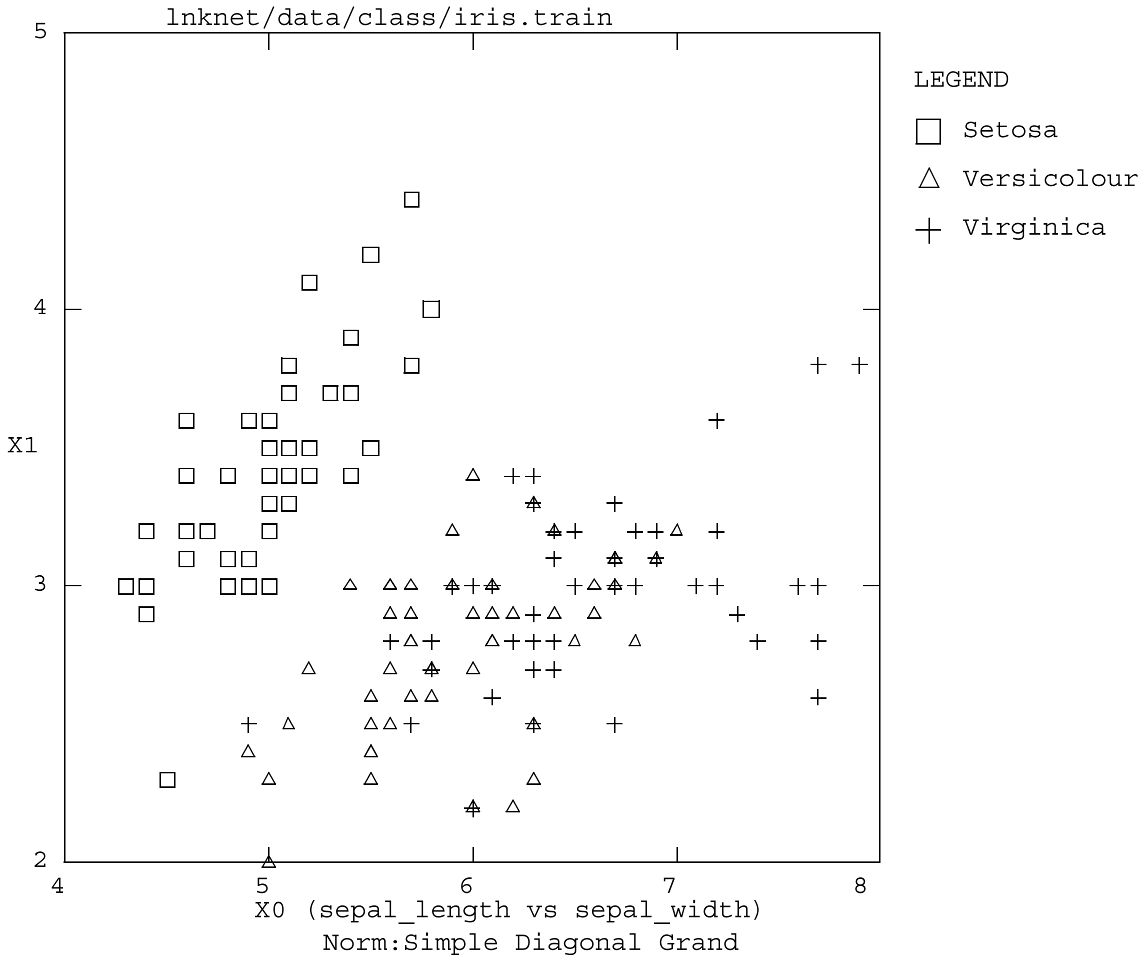
FIGURE D.16 high_tail



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
1	2	400	200	200

The first class was generated using a single Gaussian distribution with a mean of 5 and a variance of 1. The second class was generated by sampling two overlapping uniform distributions of differing length. The probability of a pattern in the second class being in the first segment is 0.9. The first segment covers the range (-0.5, 0.5). The second segment covers the range (-50,50).

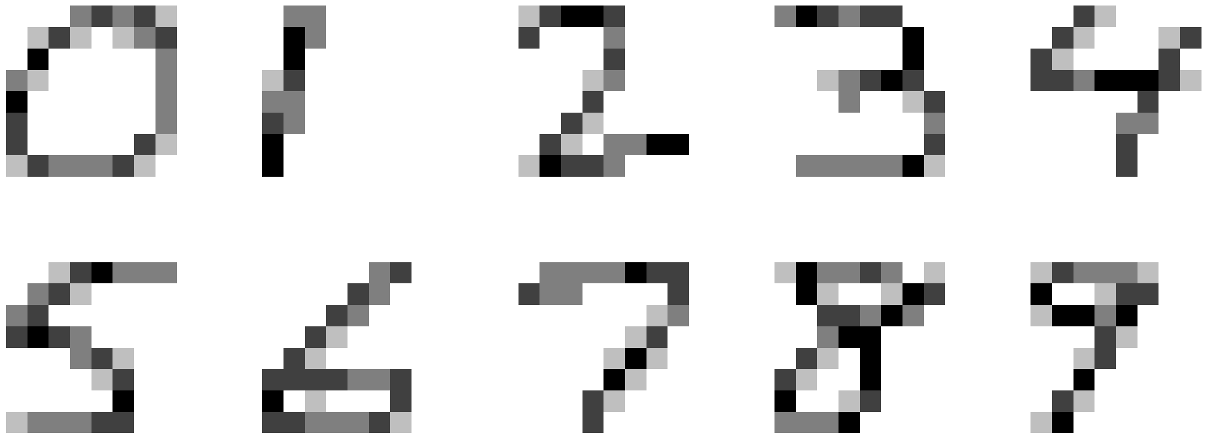
FIGURE D.17 iris



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
4	3	150		

This is R.A. Fisher's iris data [8]. The data set contains three classes with 50 patterns for each class. Each class is a type of iris plant. The inputs are the sepal length and width in centimeters and the petal length and width in centimeters.

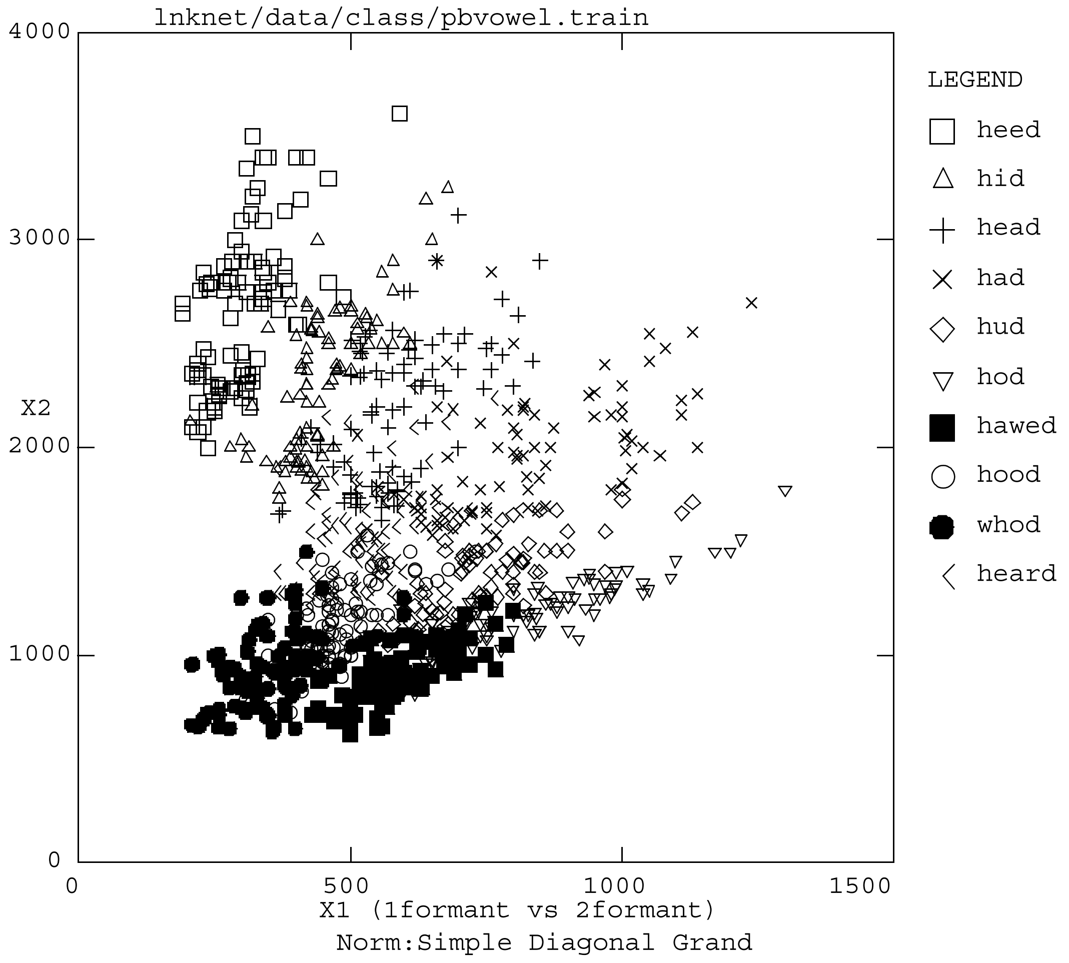
FIGURE D.18 ocrdigit



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
64	10	600		600

This is the little 1200 data base which was collected at AT&T Bell Laboratories by Isabelle Guyon[35] among her collaborators. Twelve people wrote the 10 digits several times each. The data was mapped to a 64 by 64 grid and then smoothed and fit into a 8 by 8 grid. This smoothed data was provided by John Hampshire.

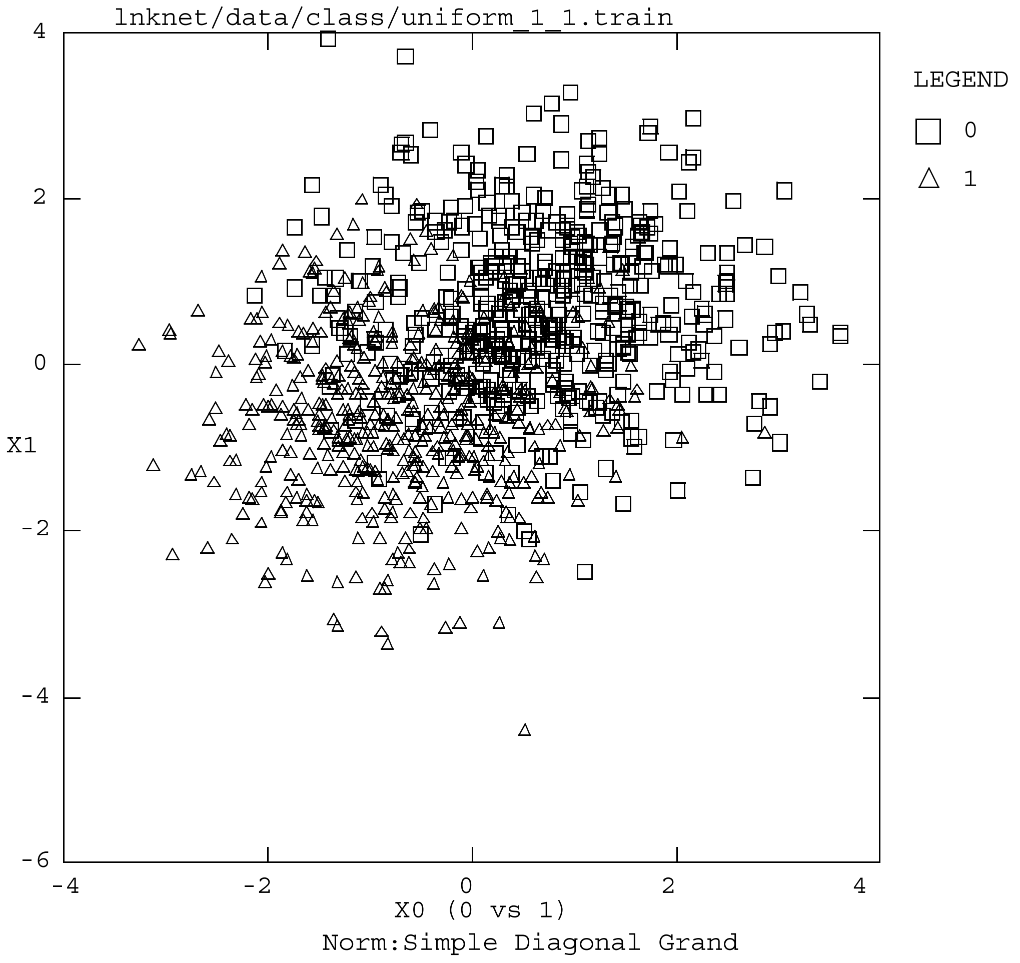
FIGURE D.19 pbvowel



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
5	10	896	298	300

This is most of the original Peterson and Barney[32] vowel data which was collected in the 1950's. The original data was collected from 67 speakers, each of whom said the 10 words given in the legend. The inputs are the pitch and first three formant frequencies of the vowel in each word and whether the speaker was a man, woman, or child.

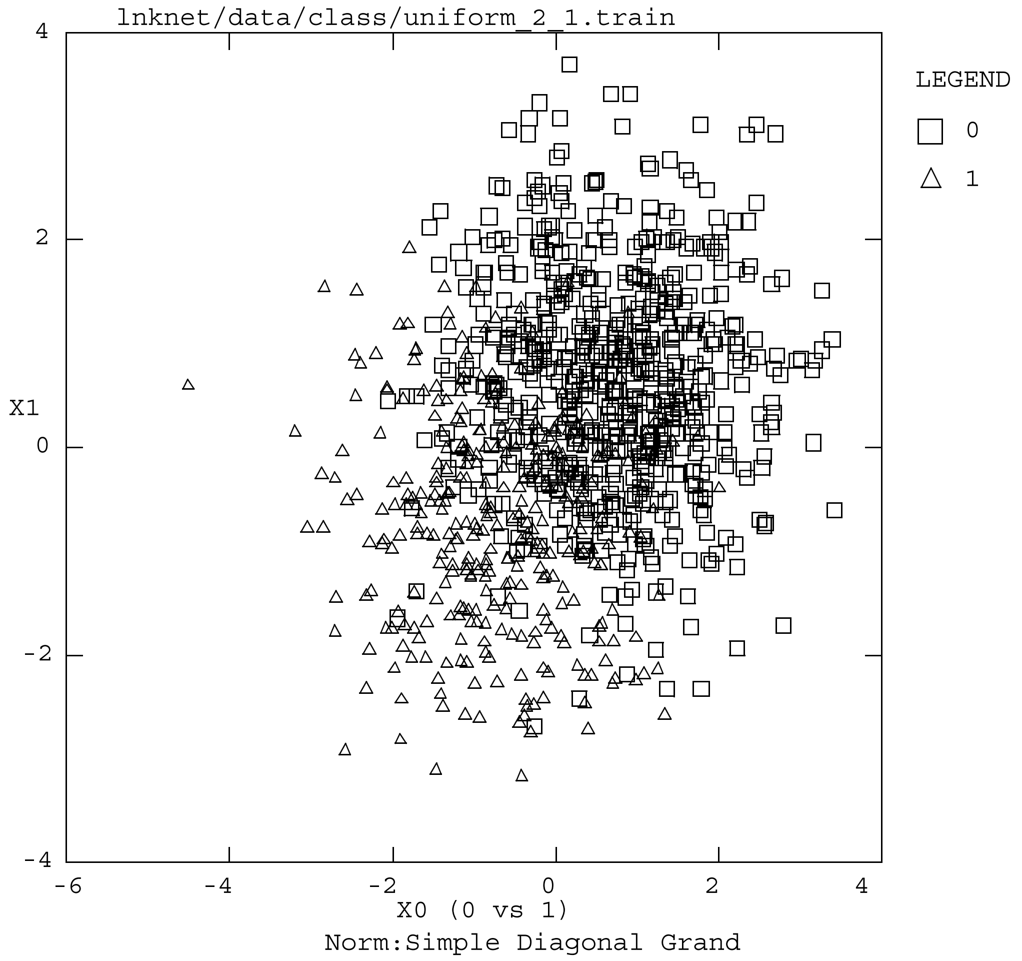
FIGURE D.20 uniform_1_1



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	2	1000	1000	

The three data bases, uniform_1_1, uniform_2_1, and uniform_10_1, were all generated by sampling the same pair of Gaussian distributions. The means of the Gaussians are one standard deviation apart along the line x=y. The differences among the three data bases are in the number of patterns sampled for each class. There ar 500 patterns in each class in uniform_1_1. Uniform_2_1 has 666 patterns from class 0 and 333 patterns from class 1, giving a 2 to 1 ratio in the *a priori* probabilities of the classes. Uniform_10_1 has 1000 patterns from class 0 and 100 from class 1, giving a 10 to 1 ratio in the class probabilities. These data bases were generated to test the *a priori* probability adjustment features of the LNKnet classifiers.

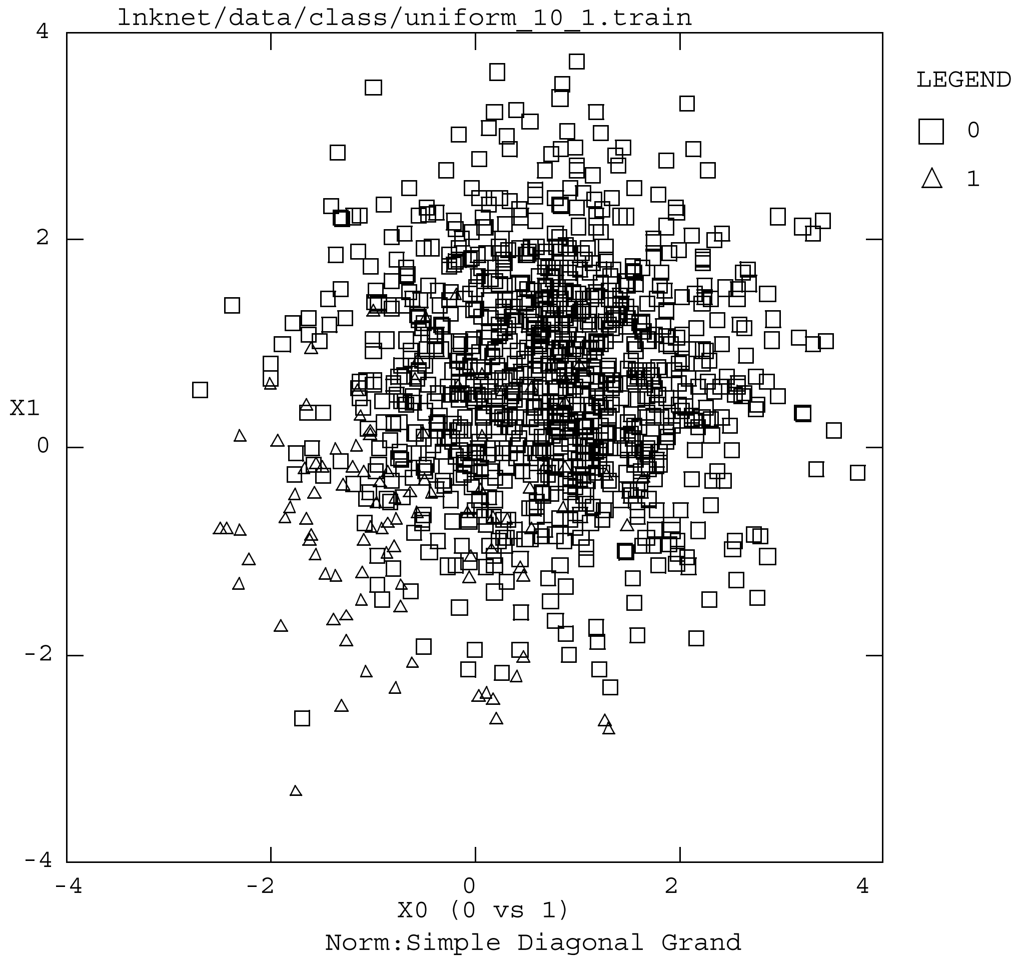
FIGURE D.21 uniform_2_1



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	2	999	999	

Data taken from two identical Gaussian distributions with means one standard deviation apart. There are twice as many patterns from class 0 as from class 1.

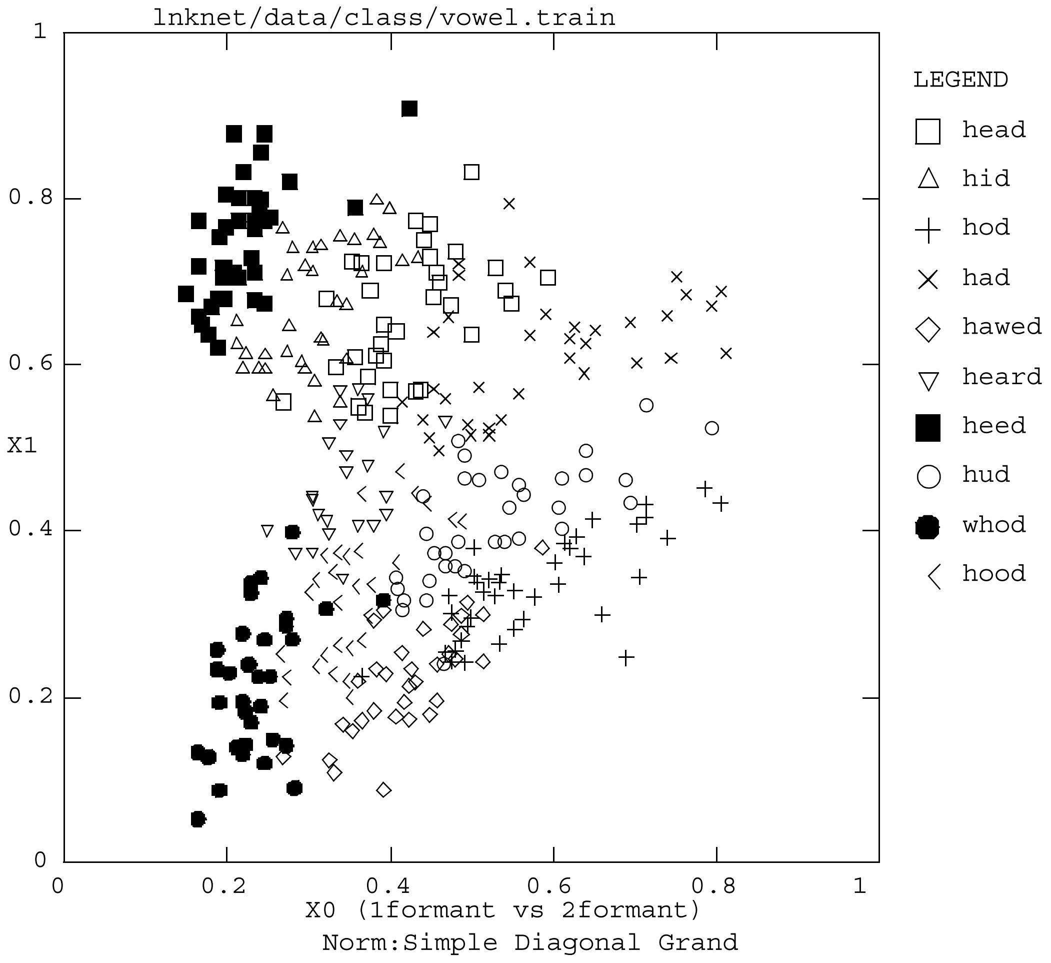
FIGURE D.22 uniform_10_1



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	2	1100	1100	

Data taken from two identical Gaussian distributions with means one standard deviation apart. There are 10 times as many patterns from class 0 as from class 1.

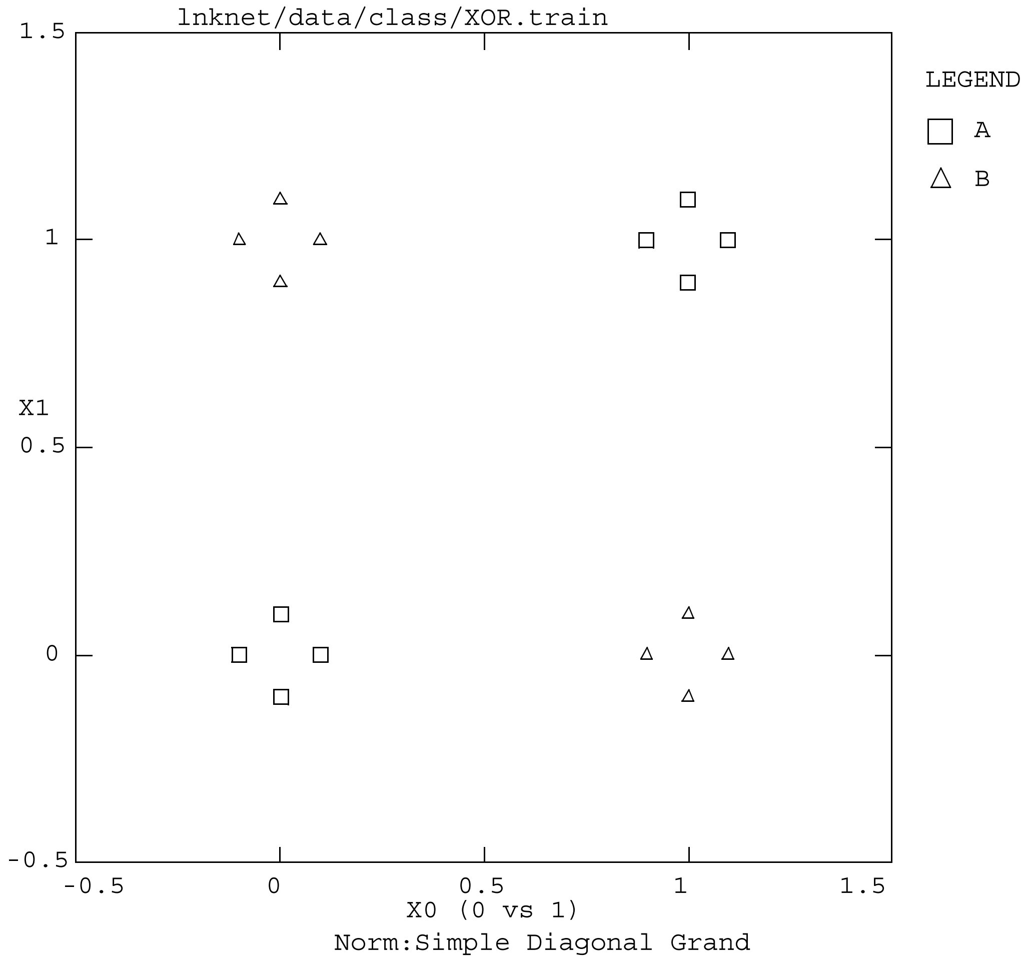
FIGURE D.23 vowel



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	10	338	166	167

This data is a normalized version of some of the Peterson and Barney[32] vowel data. The inputs are the first and second formant frequencies of the vowel in the 10 words given in the legend. The frequency data is normalized to be between 0 and 1.

FIGURE D.24 XOR



Ninputs	Noutputs	Npatterns train	Npatterns eval	Npatterns test
2	2	16		16

This data base has hand generated patterns from the XOR problem. It is intended for testing algorithms by hand to verify the outputs of calculations.

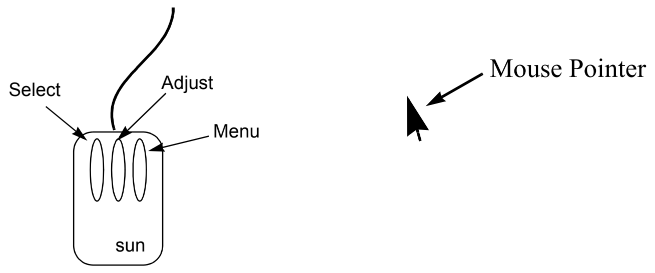
APPENDIX E Using OpenWindows

SUN has an excellent tutorial describing the use of the OpenLook window manager. The tutorial can be found in $OPENWINHOME/bin/helpopen. SUN also publishes an OpenWindows User Guide which provides information on the use of OpenLook style applications. If these sources of information are unavailable, this appendix covers those parts of OpenWindows that affect the use of LNKnet. This appendix assumes that you can already start a window manager. For more information on how to do this, contact your system administrator.

E.1 The Mouse

Most graphical window interfaces are built around a mouse. OpenWindows assumes that the mouse has three buttons. The settings of these buttons can be changed, but the default settings are as follows: The left button is the Select Button; The middle button is the Adjust Button; The right button is the Menu Button. In LNKnet, the select button is used for setting check boxes, selecting items from a scrolling list, and “pushing” buttons. The menu button is used for making selections from a pull down menu. The adjust button is not used. A SUN mouse is shown in Figure E.1. A SUN mouse comes with a metal mouse pad. The mouse must be on that pad to work.

FIGURE E.1 A SUN three button mouse and pointer

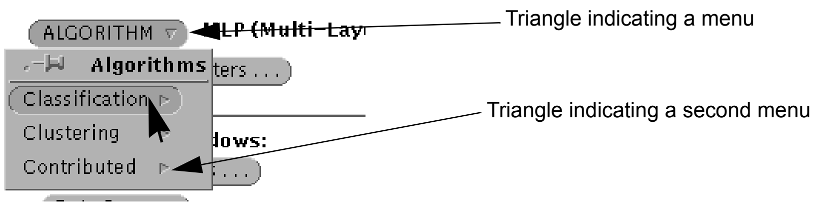


The mouse controls a pointer, also shown in Figure E.1. When the mouse is moved on its pad, the pointer moves on the screen. To select a button on a LNKnet window, you must first use the mouse to move the pointer over the button. It may also be necessary to “click” the mouse on the bar at the top of the window to bring the LNKnet window to the attention of the window manager. To do this, move the pointer to the bar at the top of the window. Press and release the select button on the mouse.

E.1.1 Menus

In OpenWindows, a menu is indicated by a small triangle on a button or a small box. To select something from a menu you must first move the mouse pointer to the triangle. Press and hold the menu button on the mouse. The menu attached to the triangle should now appear. Still holding down the mouse menu button, drag the pointer down the menu to the item you want to select, then let go. It is possible that the item you want also has a triangle next to it, indicating that there is another menu that must be selected from. If so, do not let go of the menu button. Drag the mouse pointer in the direction the triangle points to bring up the second menu and proceed as before.

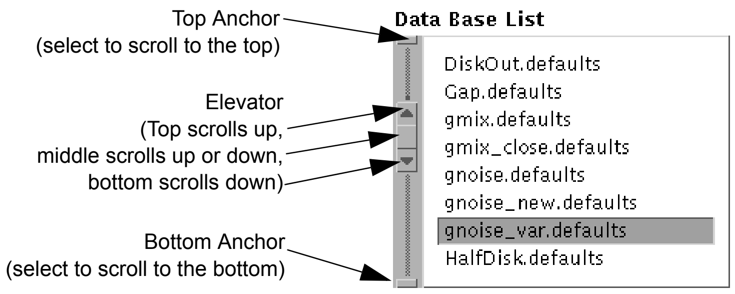
FIGURE E.2 Making a Menu Selection



E.1.2 Scrolling Lists

On the LNKnet file window, there is a scrolling list with the names of the standard data bases available in LNKnet. To make a selection from this list you must first move scroll the list up or down to show the item you want. Beside the list is a scroll bar. This bar has a anchors at the top and bottom and an “elevator” in the middle. The list scrolls up when the select button is pressed over the elevator. The list scrolls down when the select button is pressed under the elevator. The square in the middle lets you move the mouse while holding the select button to scroll the list up or down. Once the item you want is showing, select it with the select button on the mouse. Figure E.3 shows the scrolling list on the LNKnet file window. The gnoise_var data base has been selected.

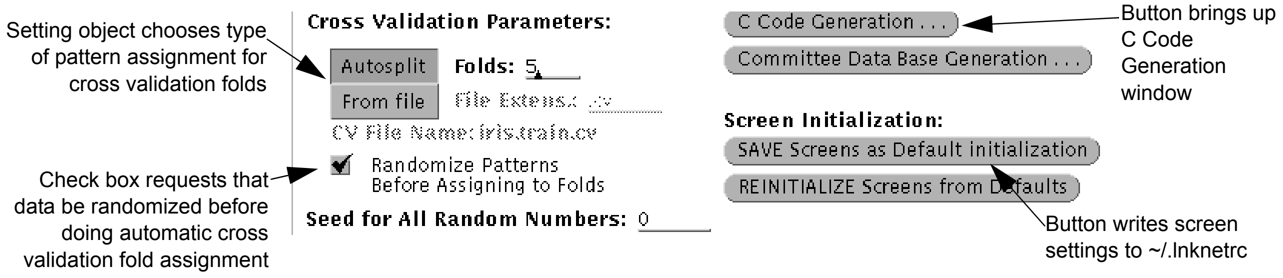
FIGURE E.3 Scrolling LIst



E.1.3 Buttons, Setters, and Check Boxes

Several other graphical controls are used by clicking on them with the select mouse button. LNKnet uses buttons, setting objects, and check boxes. The buttons do a variety of things. Some of them have menus attached to them. The others run experiments, bring up windows, or perform some function inside LNKnet. The setting objects set some LNKnet or algorithm variable to some setting from a short list. The check boxes are graphical binary flags. They turn on or off LNKnet features. Figure E.4 shows a set of LNKnet selection objects.

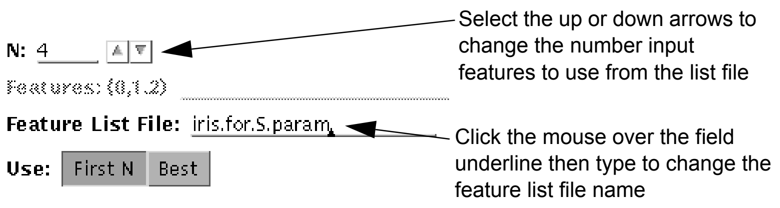
FIGURE E.4 LNKnet Objects that are Set using the Select Mouse Button


Setting object chooses type of pattern assignment for cross validation folds
Check box requests that data be randomized before doing automatic cross validation fold assignment
Button brings up C Code Generation window
Button writes screen settings to ~/.lnknetrc
Cross Validation Parameters:
Autosplit
From file
Folds: 5
File Extens.: .cv
CV File Name: iris.train.cv
Randomize Patterns Before Assigning to Folds
Seed for All Random Numbers: 0
C Code Generation . . .
Committee Data Base Generation . . .
Screen Initialization:
SAVE Screens as Default initialization
REINITIALIZE Screens from Defaults

E.2 The Keyboard

At least half the fields on most LNKnet windows are text fields. They are set by first selecting them with the mouse and then typing on the keyboard. When a text field is selected, a cursor, a small triangle, will appear on the line. Some text fields are associated with numbers. These numbers can be set by typing or by selecting the up and down arrows beside the text field. When you are finished typing a new setting for a text field, you must hit carriage return or tab. The change you have made will not take affect if you do not. Figure E.5 shows some text fields from the LNKnet file window.

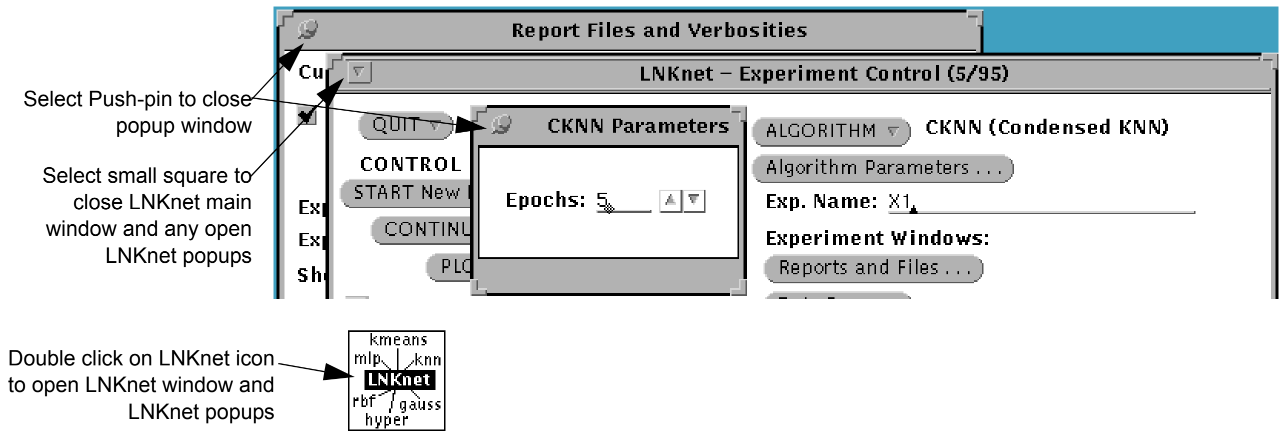
FIGURE E.5 LNKnet Text Fields


Select the up or down arrows to change the number input features to use from the list file
Click the mouse over the field underline then type to change the feature list file name
N: 4
Features: (0,1,2)
Feature List File: iris.for.S.param
Use:
First N
Best

E.3 Windows

LNKnet is built around many windows. There is a main window and many popup windows. When olwm is your window manager, these windows come up automatically. With some other window managers, each window must be placed when it is displayed. In olwm, the main difference between the main window and the popup windows is that the popup windows are displayed with a push pin in the upper left corner of the window. If you select the push pin, the popup window will disappear. It can be redisplayed by reselecting the button which brought up the window before. If you select the small square in the upper left of the main window, LNKnet will be iconified. That is, the LNKnet main window and all of its popup windows will disappear and a square icon will appear somewhere on your screen. Double clicking on the LNKnet icon will redisplay the main LNKnet window and its popups. Figure E.6 shows the bar at the top of the LNKnet main window and two popup windows as well as the LNKnet icon.

FIGURE E.6 LNKnet windows and icon


Select Push-pin to close popup window
Select small square to close LNKnet main window and any open LNKnet popups
Double click on LNKnet icon to open LNKnet window and LNKnet popups
Report Files and Verbosities
LNKnet – Experiment Control (5/95)
CKNN Parameters
QUIT
CONTROL
START New
CONTINU
PLO
Algorithm Parameters . . .
Experiment Windows:
Reports and Files . . .
kmeans
mlp
knn
LNKnet
rbf
gauss
hyper

Epochs: 5

ALGORITHM CKNN (Condensed KNN)

Exp. Name: X1

BIBLIOGRAPHY

1. Bruce G. Batchelor, ed. *Pattern Recognition: Ideas in Practice.* Plenum Press: New York, (1978).

2. Christopher M. Bishop, *Neural Networks for Pattern Classification*. Oxford: Clarendon Press, (1995).

3. Leo Breiman, Jerome H. Friedman, Richard A. Olshen, Charles J. Stone, *Classification and Regression Trees.* Belmont, California: Wadsworth, Inc., (1984).

4. Linde, Y., A. Buzo, and R.M. Gray, "An Algorithm for Vector Quantizer Design," IEEE Transactions on Communications, COM-28, 84-95, 1980.

5. N. Christiani and J. Shawe-Tayor, *An Introduction to Support Vector Machines*. Cambridge University Press (2000).

6. Eric I. Chang and Richard P. Lippmann, "Using Genetic Algorithms to Select and Create Features for Pattern Classification," MIT Lincoln Laboratory, Technical Report 892, 1991.

7. R.O. Duda, P.E. Hart, and David Stork, *Pattern Classification (Second Edition)*. New York: Wiley (2000).

8. R. A. Fisher, "The Use of Multiple Measurements in Taxonomic Problems," *Annals of Eugenics*, 7, 179-188, 1936.

9. K. Fukunaga, *Introduction to Statistical Pattern Recognition (Second Edition)*. New York, NY: Academic Press (1990).

10. John B. Hampshire and B. V. K. Vijaya Kumar. "Why Error Measures are Sub-Optimal for Training Neural Network Pattern Classifiers," in *IEEE Proceedings of the 1992 International Joint Conference on Neural Networks*. IEEE, 1992.

11. John B. Hampshire and Alexander H. Waibel, "A Novel Objective Function for Improved Phoneme Recognition Using Time-Delay Neural Networks," in *IEEE Transactions on Neural Networks*, 216-228, 1990.

12. J. A. Hartigan, *Clustering Algorithms*. New York: John Wiley and Sons (1975).

13. J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*. Addison-Wesley (1991).

14. Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *The Elements of Statistical Learning*,

New York, Springer (2001).

15. William Y. Huang and Richard P. Lippmann, “Comparisons Between Conventional and Neural Net Classifiers,” in *Proceedings of the 1st International Conference on Neural Networks*, IV-485, 1987.

16. Don R. Hush and Bill G. Horne, “Progress in Supervised Neural Networks,” IEEE Signal Processing Magazine, 10(1), 8-39, 1993.

17. Keerthi, S.S. and S.K. Shevade, “Improvements to Platt’s SMO algorithm for SVM classifier design,” Neural Computation Vol. 13, 2001, 637-649, http://guppy.mpe.nus.edu.sg/~mpessk/svm/smo_mod_nc.ps.gz.

18. Ruby Kennedy, Yuchun Lee, Benjamin Van Roy, C. Reed, and Richard Lippmann, *Solving Data Mining Problems through Pattern Recognition*. Prentice Hall (1997).

19. R. Kohavi, B. Becker, and D. Somerfield, Improving Simple Bayes, in *Proceedings European Conference on Machine Learning*, 1997.

20. L. Kukolich, R. P. Lippmann, “*Getting Started With LNKnet: A Quick Introduction*”, MIT Lincoln Laboratory, 1994.

21. Yuchun Lee, *Classifiers: Adaptive Modules in Pattern Recognition Systems*. Cambridge, MA: Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science (1989).

22. Yuchun Lee and Richard P. Lippmann, “Practical Characteristics of Neural Network and Conventional Pattern Classifiers on Artificial and Speech Problems,” in *Advances in Neural Information Processing Systems* 2, D.S. Touretzky, (Eds.), Morgan Kaufman: San Mateo, CA, 1990.

23. Richard P. Lippmann, “Pattern Classification Using Neural Networks,” in *IEEE Communications Magazine*, 47-54, 1989.

24. Richard P. Lippmann, “A Critical Overview of Neural Network Pattern Classifiers,” in *Neural Networks for Signal Processing, Proceedings of the 1991 IEEE Workshop*, B.H. Juang, S.Y. Kung, and C.A. Kamm, (Eds.), IEEE: Piscataway, N.J., 1991.

25. Richard P. Lippmann, “An Introduction to Computing with Neural Nets,” in *Neural Networks: Theoretical Foundations and Analysis*, C. Lau, (Eds.), IEEE Press: 1992.

26. R. P. Lippmann, “Neural Networks, Bayesian a posteriori Probabilities and Pattern Classification,” in *From Statistics to Neural Networks. Theory and Pattern Recognition Applications,* V. Cherkassky, J.H. Friedman, and H. Wechsler, Editors, Springer-Verlag, 1993.

27. R. P. Lippmann, L. Kukolich, and E. Singer, “LNKnet: Neural Network, Machine Learning, and Statistical Software for Pattern Classification”, Lincoln Laboratory Journal, Vol. 6, No. 2, pp 249-268, 1993.

28. Kenney Ng and Richard P. Lippmann, “A Comparative Study of the Practical Characteristics of Neural Network and Conventional Pattern Classifiers,” MIT Lincoln Laboratory, Technical Report 894, 1991.

29. Kenney Ng and Richard P. Lippmann, “A Comparative Study of the Practical Characteristics of Neural Network and Conventional Pattern Classifiers,” in *Neural Information Processing Systems 3*, R. Lippmann, J. Moody, and D. Touretzky, (Eds.), Morgan Kaufmann: San Mateo, California, 970-976, 1990.

30. Nils J. Nilsson, *Learning Machines*. McGraw Hill, N.Y. (1965).

31. T. W. Parsons, *Voice and Speech Processing*. New York: McGraw-Hill (1986).

32. Gorden E. Peterson and Harold L. Barney, “Control Methods Used in a Study of Vowels,” in *The Journal of the Acoustical Society of America*, 175-84, 1952.

33. Platt, J., “Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods,” in Advances in Large Margin Classifiers, A. Smola, et al., Editors. 2000, MIT Press, http://www.research.microsoft.com/users/jplatt/SVMprob.ps.gz.

34. Platt, J., “Fast Training of Support Vector Machines Using Sequential Minimal Optimization,” in *Advances in Kernel Methods - Support Vector Learning*, B. Scholkopf, C. Burges, and A. Smola, Editors. 1998, MIT Press, http://www.research.microsoft.com/~jplatt/smo.html.

35. Guyon, I. Poujand, et al., “Comparing Different Neural Network Architectures for Classifying Handwritten Digits,” in Proceedings International Joint Conference on Neural Networks, Washington DC, II.127-II.132, 1989.

36. Mike D. Richard and Richard P. Lippmann, “Neural Network Classifiers Estimate Bayesian a Posteriori Probabilities,” *Neural Computation*, 3, 461-483, 1992.

37. Elliot Singer and Richard P. Lippmann, “Improved Hidden Markov Model Speech Recognition Using Radial Basis Function Networks,” in *Neural Information Processing Systems 4*, J. Moody, S. Hanson, and R. Lippmann, (Eds.), Morgan Kaufmann: San Mateo, California, 1992.

38. Elliot Singer and Richard P. Lippmann. A Speech Recognizer Using Radial Basis Function Neural Networks in an HMM Framework. in *Proceedings International Conference on Acoustics Speech and Signal Processing*. San Francisco: IEEE, 1992.

39. Donald Specht *Probabilistic Neural Networks* in *Neural Networks, Volume 3* pp 109-118, Pergamon Press, 1990.

40. Sholom M. Weiss and Casimir A. Kulikowski, *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. San Mateo, California: Morgan Kaufmann (1991).

SUBJECT INDEX