# PCP - Pattern Classification Program, version 2.2
# User's Guide

Ljubomir J. Buturović
ljubomir@sfsu.edu

May 24, 2006

# Contents

# Chapter 1

# Introduction

PCP (Pattern Classification Program) is an open-source program for supervised classification of patterns. In the context of PCP, *pattern* is defined as a multidimensional vector of measurements (*features* or *attributes*), and pattern classification refers to the problem of assigning the vectors into one of a given number of categories (classes).

Note that sometimes different terms are used to refer to this field, for example *machine learning*, *pattern recognition*, *statistical learning*, etc. We use the term *pattern classification* to make it clear that PCP only supports classification of patterns, and not regression, data mining, image analysis or other related functions.

The principal purpose of PCP is to efficiently analyze potentially large datasets using established, state-of-the-art machine learning algorithms. This permits comparison with other methods, for example those which the user is developing, or other published methods not implemented in PCP.

PCP uses character-driven menus, and thus does not require graphics support or libraries. Input data is read from tab-delimited text (ASCII) files. Commands are issued interactively or supplied in a batch file. Results are presented in tabular form on the screen, and also saved in text files in easily parsable formats. The files can then be used for graphical display or further analyses by other programs.

This User's Guide provides detailed instructions for using PCP. It assumes that the reader is familiar with the concepts and purpose of pattern classification. Understanding of the internal workings of the algorithms, other than familiarity with the basic parameters, is not required for the use of PCP.

A brief overview of PCP has been published in journal Bioiformatics [29]. Please cite this article if you publish work which uses PCP.

The main features of PCP are:

- efficient, industrial-strength implementation of a variety of algorithms for pattern classification, feature extraction and feature selection, including support for very large datasets, limited only by the amount of available memory
- PCP consists of a single binary executable which does not depend on any special run-time environment, libraries or interpreters
- it allows easy binding of source code functions from many programming languages

- simple installation (no configuration required)
- interactive or batch processing
- unrestricted redistribution of the program in binary and source formats (including unrestricted incorporation in commercial products)

PCP implements the following machine learning algorithms:

- parametric classifiers (linear and quadratic)
- least-squares (pseudo-inverse) linear discriminant
- $k$-Nearest Neighbor ($k$-NN)
- neural networks (Multi-Layer Perceptron, MLP)
- Support Vector Machine (SVM) algorithm
- model selection for SVM, MLP and $k$-NN
- Bayes error estimation
- cross-validation
- bagging (committee) classification
- Fisher's linear discriminant
- dimensionality reduction using Singular Value Decomposition
- Principal Component Analysis
- feature subset selection using feature ranking, forward selection or backward elimination

# Chapter 2

# Getting Started

This chapter describes unpacking and installation of PCP and basic usage. PCP distribution contains executables for Linux and Windows (running under the free `Cygwin` environment).

## 2.1 Downloading and Installation

PCP home page is at `http://pcp.sourceforge.net`. To download the program, scroll down to *Downloading and Installing* section. The software is distributed in a single compressed archive. The file is named `pcp-ver.tar.gz` for GNU/Linux, and `pcp-ver.zip` for Windows, where `ver` is the current version number.

Windows version of PCP runs under the free `Cygwin` environment. Cygwin provides Linux-like utilities and API for Windows, and is available for free download at `http://www.cygwin.com`. Note that PCP does not actually need entire Cygwin distribution; only the API library file `cygwin1.dll` is required to run PCP.

The contents of the GNU/Linux and Windows archives is identical; only the compression method differs. To decompress and unpack the file on GNU/Linux, type:

```
% gunzip -c pcp-ver.tar.gz | tar xvf -
```

On Windows, double-click on the archive to start the unpacking.

The unpacking/decompression will create a directory `pcp-ver`. In the directory, file `Linux/pcp` is the PCP executable for GNU/Linux, and `Windows/pcp.exe` is the Windows executable. There is no configuration - upon unpacking the program is immediately ready for use.

## 2.2 Basic Usage

This section describes basic PCP usage by performing simple classification of the well-known IRIS dataset [1] (the data files are provided with the distribution). This dataset is widely used for algorithm demonstrations in the pattern recognition community. It represents four measurements of the petal and sepal dimensions for three species of IRIS flowers. The pattern classification task is to assign correct IRIS species for a given flower, using the four measurements. In this experiment, we will estimate the error rate of such assignment for nearest-neighbor classifier.

PCP is run from command line. To start PCP, type:

```
% pcp
```

at the command-line prompt, assuming the executable `pcp` is in your `PATH`. Otherwise, use the full path to the executable.

Upon startup, PCP displays the **Main Menu** shown in Fig. 2.1.

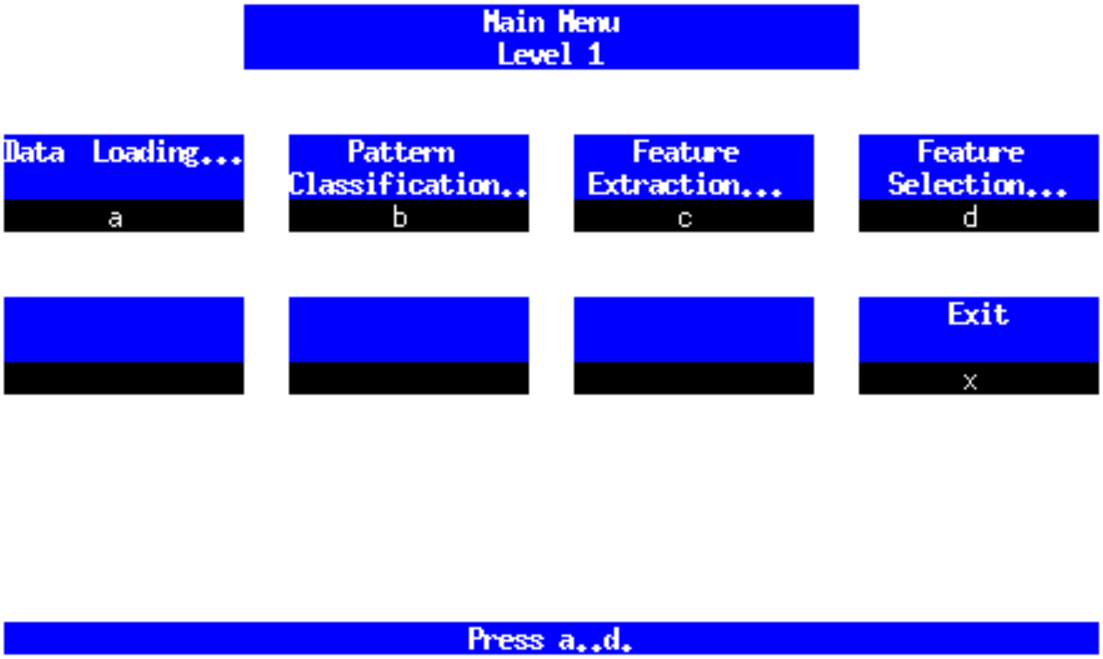


Figure 2.1: PCP Main Menu

The first step is to define the training dataset. Press **a** to enter **Data Loading** menu shown in Fig. 2.2. In the **Data Loading** menu, press **a**, and the data loading dialog will start.

To define the training dataset, answer the questions as follows. Note that PCP suggests a reasonable default value for most questions, in this and other menus. The default values are given in square brackets. To accept the default, press **Enter** in response to the question.

```
Load training (0) or test (1) dataset [0]:
0 (or Enter)
File format: header line (0: no; 1: yes) [0]:
1
File format: named rows (0: no; 1: yes) [0]:
0 (or Enter)
Enter number of classes:
3
Enter file name for class     1:
```


Figure 2.2: Data Loading menu

```
iris_setosa.dat
Enter file name for class     2:
iris_versicolor.dat
Enter file name for class     3:
iris_virginica.dat
```

Control returns to the **Data Loading** menu. Press **Enter** to return to the **Main Menu**. The information about the dataset is now stored in `pcp.sts` file and will be used in each subsequent session (invocation) of the program, until it is redefined.

In order to perform a simple supervised learning analysis of the IRIS data, press **b** to enter **Pattern Classification** menu, and **d** to enter **Nearest Neighbors Classifiers** menu shown in Fig. 2.3.

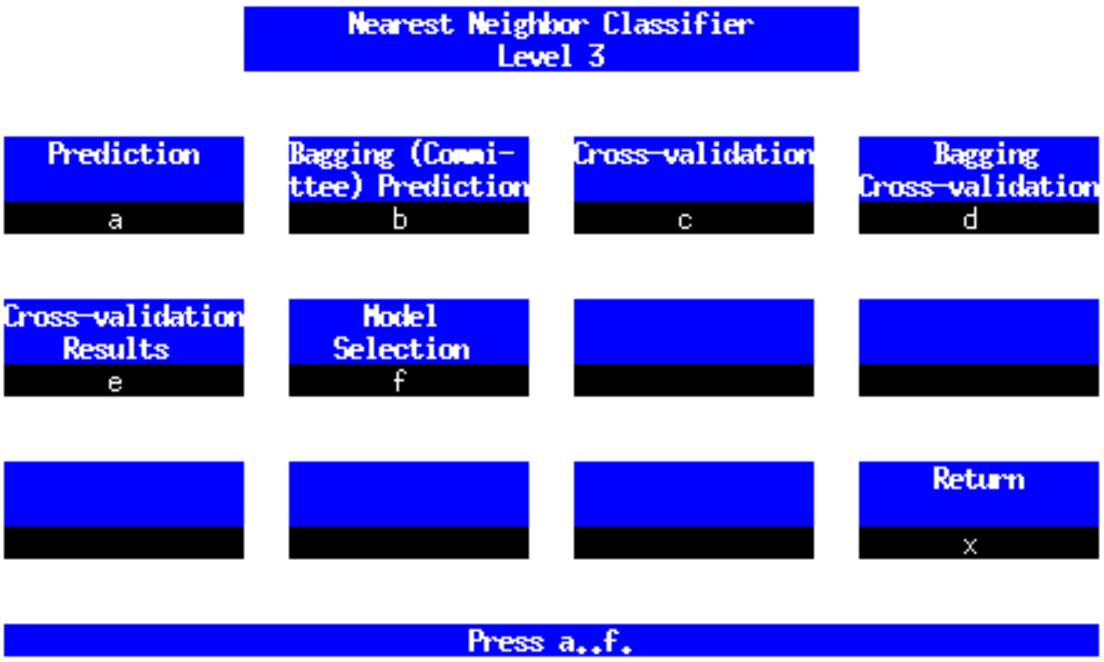


Figure 2.3: Nearest Neighbors Classifiers menu

To start cross-validation of the $k$-NN classifier, press **c** and press **Enter** in response to each of the following questions except **Enter number of experiments**, where you should enter 10:

```
Enter number of nearest neighbors (>= 1) [    1]:

Use Euclidean (1), city-block (2) or Mahalanobis distance (3) [1]:

Enter k-NN cross-validation file name [pcp.xnn]:
```

```
Enter number of cross-validation subsets (2..50) [   10]:

Enter seed for pseudo-random number generator [    1]:

Enter number of experiments [1]:
10
Use raw (0) or normalized data (1) [0]:

Choose dim. reduction method: none (0), FLD (2), PCA (3), EMAP (4), feature ra
nking (5), forward selection (6), or backward elimination (7) [0]:
```

Pressing **Enter** accepts the default choice for a parameter. In this example, the default choices are reasonable. The training will now start, and the results for individual cross-validation experiments will sequentially appear on the screen. Shortly, the program finishes the calculation and pauses. Press any key to return to the **Nearest Neighbors** menu, and press **e** to view the detailed results. Press **Enter** twice in response to the questions.

The results should look similar to Fig. 2.4. The table shows the cumulative and clas-conditional error rates and standard deviations for the 10 cross-validation experiments. The detailed results are saved in text file `pcp.xnn`.

Now press **Enter** repeatedly to leave PCP. This concludes the introduction to basic usage of PCP for pattern classification, which demonstrates that relatively elaborate pattern classification analysis can be completed with a few keystrokes.

```
+-----------------------------------------------------------------------+
|                                                                       |
|            C R O S S - V A L I D A T I O N   R E S U L T S             |
|                                                                       |
+-----------------------------------------------------------------------+
| Classifier:                          k-nearest neighbor               |
| Number of experiments:               10                               |
| Number of cross-validation subsets:  10                               |
| Normalization:                       no                               |
| Dimensionality reduction method:     none                             |
| Seed:                                1                                |
| Number of neighbors:                 1                                |
| Distance measure:                    1 (Euclidean)                    |
+-----------------------------------------------------------------------+
+-----------------------------------------------------------------------+
|            |                    Cumulative error rate                   |
+-----------------------------------------------------------------------+
|    Mean    |                        4.07                              |
| Std.  dev. |                        0.21                              |
+-----------------------------------------------------------------------+
|            |       Error rate for class iris_setosa                   |
+-----------------------------------------------------------------------+
|    Mean    |                        0.00                              |
| Std.  dev. |                        0.00                              |
+-----------------------------------------------------------------------+
|            |       Error rate for class iris_versicolor               |
+-----------------------------------------------------------------------+
|    Mean    |                        6.00                              |
| Std.  dev. |                        0.00                              |
+-----------------------------------------------------------------------+
|            |       Error rate for class iris_virginica                |
+-----------------------------------------------------------------------+
|    Mean    |                        6.20                              |
| Std.  dev. |                        0.63                              |
+-----------------------------------------------------------------------+
```

Figure 2.4: Cross-validation results for 1-NN classifier trained on IRIS data. All error rate values are given in percents.

# Chapter 3

# PCP Basics

## 3.1 Running PCP

Entire PCP is contained in a single executable file. The distribution contains two prebuilt binaries, for GNU/Linux and Windows/Cygwin operating systems. The program uses no configuration files. As shown earlier, to start PCP type

```
% pcp
```

from command line, assuming the location of the executable is in your **PATH** environment variable (otherwise, use the full path to the executable).

To get help on command-line options for PCP, use **-h** argument:

```
% pcp -h
pcp (Pattern Classification Program) version 2.2

Usage: pcp [-h] [-b batch_file] [-q] [-d]
       -h              display this message
       -b batch_file   run in batch mode, reading commands from batch_file
       -q              quiet mode (no screen output)
       -d              enable debug mode (produces pcp.dbg file)

To kill the program, use SIGINT (Ctrl-C) or SIGQUIT (Ctrl-\)
```

PCP has two modes of operation: interactive and batch. In batch mode, activated with **-b** command line option, a sequence of commands are read from a file previously created by the user. The program comes with a sample batch file `iris.bat` which can be used to load the IRIS dataset used as an example in Chapter 2. Batch processing is further explained in Section 3.5.

The debug mode produces a potentially large `pcp.dbg` file used for bug fixing and regression testing.

## 3.2 Platform Support

PCP runs under Linux and Windows/Cygwin operating systems on i386 architecture CPUs such as Intel Pentium or AMD Athlon. It uses GNU Autoconf package for building, and should therefore be easily portable to any UNIX-like platform.

PCP has been developed and tested on `RedHat Linux 9.0` distribution. It has also been tested on `SUSE Linux 9.1` and `Fedora Core 2` and verified to run on `Knoppix 3.7` distribution, as well as on Windows XP Home Edition and Windows 2000 Professional operating systems under Cygwin environment.

## 3.3 Navigation

For navigation and execution of commands, PCP uses uses a set of menus organized in a tree structure. Examples of menus have been shown previously in Figs. 2.1- 2.3. Each menu page has title field, six or nine action fields, and message field. The title field contains menu name and *level*, which is the depth of the menu within the tree hierarchy. The top level menu (level 1) is called **Main Menu**. The complete menu tree hierarchy is shown in Fig. 3.1.

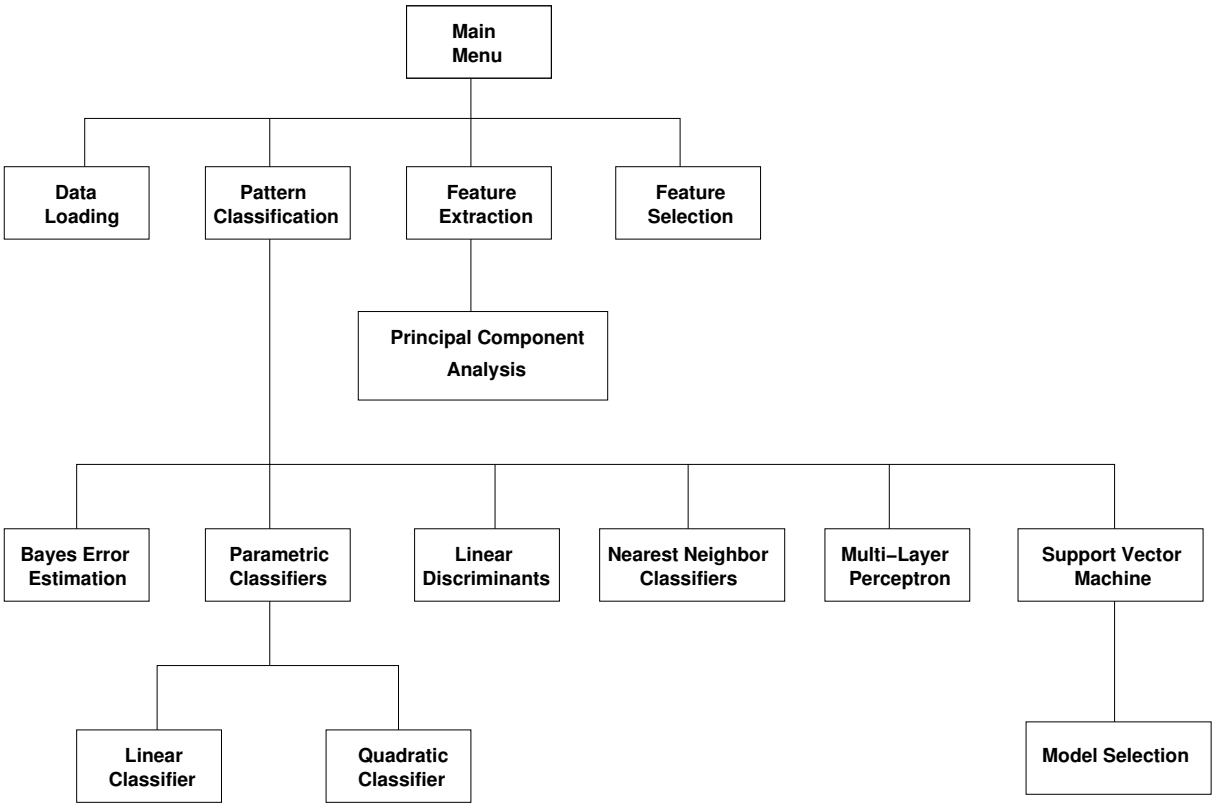


Figure 3.1: PCP menu hierarchy

Each menu option displays a letter indicating the keyboard key used to run the action associated with the option (the keys are case-insensitive). In all menus, the last active option (the rectangle

in the lower right corner of the menu display) is **Return** to the previous menu (except in the **Main menu**, where it is labeled **Exit**). **Enter** key is used to activate the **Return** option; as a convenience, **Escape** and **x** keys can also be used for the same purpose.

## 3.4 PCP Input File Formats

Datasets in PCP are modeled after the rigorous machine learning principles as defined in [7], Section 3.4.2, Model Selection via Resampling. See Section 5.1 for a detailed explanation. In this section, we describe the format of the required input data files.

From the user's point of view, PCP distinguishes two datasets: *training* and *test*. Training dataset is used to adjust parameters of a classifier (typically through a series of cross-validation experiments); test dataset is an independent data set used to evaluate performance of the previously built classifier. Most pattern classification menus have a *learning* and *prediction* function. Learning uses training dataset; prediction uses test dataset (an exception is $k$-NN algorithm, for which there is no training; the learning machine is defined by the training dataset itself).

The information about the currently defined datasets is stored in file `pcp.sts`. This file is used to maintain information between invocations of PCP. Like all files produced by PCP, it is a human-readable text file. Given that there can only be one `pcp.sts` file per directory, only one training/test dataset combination can be defined in a given directory. To analyze multiple datasets simultaneously, run PCP from different directories. Results are undetermined if multiple instances of PCP are simultaneously defining different PCP datasets in the same directory.

PCP uses text files for input data, and recognizes four file formats: *raw*, *named column*, *named row* and *named*. In all four cases, an input vector (pattern) is assumed to be stored in a single, tab- or space-delimited row. In *raw* format, input files contain the vector attribute values (floating-point numbers). In *named row* format, the first column in each row is the vector (row) name. The name can be an arbitrary string of non-whitespace characters. The vector names are used in various analyses reports to identify individual vectors. In *named column* format, the first (header) line in each input file contains column names. Again, the names may be arbitrary strings which may not contain whitespace characters. *named* format has both row and column names. In all four formats, all rows must have identical number of columns.

If row names are not provided, the vectors are referred to by their internal vector indexes. The indexes correspond to line numbers of the vectors within input files. If column names are not provided, the attributes are referred to by the corresponding column indexes in input files. The columns are numbered sequentially starting from 1.

The default input data format in PCP is the raw format.

Figs. 3.2-3.5 show first five lines of an input file for IRIS dataset in raw, named vector, named column and named formats, respectively.

For training dataset, each class must be stored in a separate file. The file name without suffix (extension) is used as class name in PCP, therefore it is helpful if the file names are mnemonic.

For test dataset, there is no requirement to put each class into a separate file, since the class labels of the test set patterns may not be known. In practice, however, the classification of patterns in the test dataset usually *is* known, since the test dataset is commonly used to estimate error rate of a classifier previously built using training dataset. If this is the case, the user has to signal somehow to PCP the class labels of the test set vectors. The convention in PCP is the same as for the training set: the test set vectors have to be divided into files corresponding to the classes.

```
5.1 3.5 1.4 0.2
4.9 3.0 1.4 0.2
4.7 3.2 1.3 0.2
4.6 3.1 1.5 0.2
5.0 3.6 1.4 0.2
```

Figure 3.2: Section of an IRIS dataset input file in *raw* format.

```
vec1   5.1 3.5 1.4 0.2
vec2   4.9 3.0 1.4 0.2
vec3   4.7 3.2 1.3 0.2
vec4   4.6 3.1 1.5 0.2
vec5   5.0 3.6 1.4 0.2
```

Figure 3.3: Section of an IRIS dataset input file in *named row* format.

| sepal_length | sepal_width | petal_length | petal_width |
|---|---|---|---|
| 5.1 | 3.5 | 1.4 | 0.2 |
| 4.9 | 3.0 | 1.4 | 0.2 |
| 4.7 | 3.2 | 1.3 | 0.2 |
| 4.6 | 3.1 | 1.5 | 0.2 |
| 5.0 | 3.6 | 1.4 | 0.2 |

Figure 3.4: Section of an IRIS dataset input file in *named column* format.

| rowID | sepal_length | sepal_width | petal_length | petal_width |
|---|---|---|---|---|
| vec1 | 5.1 | 3.5 | 1.4 | 0.2 |
| vec2 | 4.9 | 3.0 | 1.4 | 0.2 |
| vec3 | 4.7 | 3.2 | 1.3 | 0.2 |
| vec4 | 4.6 | 3.1 | 1.5 | 0.2 |
| vec5 | 5.0 | 3.6 | 1.4 | 0.2 |

Figure 3.5: Section of an IRIS dataset input file in *named* format.

Specifically, if the test dataset has more than one file, PCP will assume that each file corresponds to one class. Furthermore, PCP assumes that test dataset file 1 corresponds to training class 1, test dataset file 2 to training class 2, etc. Based on this information, the program is able to calculate and report test set error rates and other performance statistics. This is obviously impossible when the class labels for the test data are not available (i.e., when the test dataset is in a single file).

## 3.5 Batch Usage of PCP

PCP can also be run in batch mode, using a file listing the sequence of operations to be performed. Each line in the batch file corresponds to a navigation key used to activate PCP functions, or to keyboard input entered by user in response to PCP questions. A simple batch file which loads IRIS dataset into PCP is shown in Fig. 3.6. The data files used in the batch file are part of PCP distribution.

```
a                                       data loading
a                                       load dataset
0                                       training dataset
1                                       header line
0                                       no named rows
3                                       number of classes
iris_setosa.dat
iris_versicolor.dat
iris_virginica.dat
e                                       copy dataset
0                                       training to test
x                                       return
x                                       return
```

Figure 3.6: Sample batch file for importing the IRIS dataset into PCP.

Batch files are created by manually entering the sequence of steps into the file in a text editor. The command sequence should first be verified by running it interactively. Once the file is created, use the following command to execute it (optionally, you can also use the **-q** command-line argument to disable screen output):

```
% pcp -b batch_file
```

Comments in the batch file are optional, but helpful to track down errors. Note that the lines with file names should have no comments, otherwise the comment will also be considered part of the file name. Also, it is recommended to perform data analyses through the use of multiple smaller batch files, instead of a single large one, since this makes debugging easier.

## 3.6 Limits

PCP is written in C programming language (with the exception of an SVM external function, which is written in C++). PCP dynamically allocates memory for all functions, and thus the only known limit to the amount of data that can be processed is the available virtual memory in the computer.

## 3.7 Error Handling and Exit Codes

In case of error, PCP displays appropriate message in the message line on the screen, and appends it to file `pcp.err` along with a timestamp. The content of the file is never erased by PCP; all error messages are concatenated to an existing file.

Upon exit, PCP sets the exit code. Exit code 0 indicates successful completion; codes 1..127 are reserved for system error codes (defined in file `errno.h`), while codes 128..255 are PCP-proprietary error codes. The list of codes is given in Table 3.1. Note that exit code 0 does not necessarily mean that no errors were encountered during execution; indeed, the exit code only reflects the status of the *most recent* operation. In contrast, `pcp.err` contains *all* error messages. Therefore, if PCP is used within a script, the caller should verify the exit status *and* contents of the error file `pcp.err`. Of course, in interactive mode the status of each function is displayed in the message line of the corresponding menu after completion, and thus there is no need to additionally examine `pcp.err` (which is nevertheless updated).

Table 3.1: PCP exit error codes.

| *Exit status* | *Description* |
|---|---|
| 128 | The training dataset is not defined. |
| 129 | The test dataset is not defined. |
| 130 | The datasets are not defined. |
| 131 | The dataset is too large. |
| 132 | This operation only available for two classes. |
| 133 | At least two classes must be defined. |
| 134 | Each training class must have at least two samples. |
| 138 | The dataset is inconsistent with the MLP file. |
| 139 | Unrecognized MLP file format. |
| 140 | Unrecognized SVM file format. |
| 141 | The dimension of the data must be greater than the number of data points. |
| 142 | Data dimension must be less or equal the number of data points. |
| 143 | The dataset is inconsistent with the linear mapping. |
| 144 | Status file pcp.sts is inconsistent with input file |
| 145 | Unrecognized data file format. |
| 146 | Conjugate gradient unable to obtain descent direction. |
| 147 | Internal software error. |
| 148 | The dataset is inconsistent with the model. |
| 149 | The dataset is inconsistent with the linear model. |
| 150 | Singular covariance matrix. |
| 151 | Attempted inversion of a singular matrix. |
| 152 | Reached iteration limit in an iterative procedure. |
| 153 | Variable number of columns in an input file. |

# Chapter 4

# Data Loading

**Data Loading** menu is shown in Fig. 2.2. It implements data manipulation functions such as data input and display, status, copying of datasets, normalization, eigenanalysis and correlation/covariance estimation. To enter **Data Loading** menu, press **a** in the **Main Menu**.

Following is a description of the individual options.

- **Load Dataset** function is used to define training and test datasets
- **Status** displays basic information about the datasets
- **Display Dataset** displays actual data vectors in the chosen dataset
- **Copy Dataset** copies datasets (training into test, or vice versa). This is occassionally useful for diagnostic purposes.
- **Normalize Data** normalizes the dataset(s) and saves the results. The normalization transformation consists of shifting and scaling each feature to zero mean and unit standard deviation. The normalization parameters (feature means and standard deviations) are computed using the training dataset, and the resulting transformation is then applied to both datasets. The results are stored in output files. The output file names have string `'_nrm'` inserted just before the file extension. Thus, if the original file name is `iris.dat`, the normalized data will be saved in file `iris_nrm.dat`.

  Optionally, this function allows the user to replace the current data sets with their normalized counterparts.
- **Eigenvalues** computes and displays eigenvectors, eigenvalues and condition numbers of class-conditional covariance matrices. The condition numbers are of particular interest here, since too large a condition number indicates that the corresponding matrix is near-singular, which in turn may affect algorithms involving inversion of covariance matrices, for example $k$-NN algorithm using Mahalanobis distance (Section 5.5), and Bayes error estimation (Section 5.9).
- **Covariance and Correlation** computes and displays class-conditional covariance and correlation matrices. This may occasionally be useful for diagnostic purposes.

**Load Dataset** is the most commonly used function in this menu. An interactive session for loading a dataset into PCP has been shown earlier in Chapter 2.

# Chapter 5

# Pattern Classification

This chapter describes in detail the supervised pattern classification functionality available in PCP.

## 5.1 Statistical Learning Paradigm

The supervised learning in PCP is based on the following statistical learning paradigm (see [7], Section 3.4.2, *Model Selection via Resampling* for further details):

**Step 1.** divide the available data into training and test datasets

**Step 2.** select optimal parameters (for example, number of hidden nodes in a neural network, or cost $C$ and $\gamma$ parameters in SVMs) for a given algorithm. This step is known as *model selection*, and is performed using cross-validation and the training dataset. It involves the standard cross-validation technique of dividing the training dataset into $k$ subsets (where $k$ is a user-defined parameter), building the learning machine using $k-1$ subsets (the *learning set*) and calculating the prediction error using the remaining *validation* subset. The cross-validation procedure is repeated for different values of the classifier parameters. The optimal model uses parameters which give minimum error rate (or other performance criterion).

**Step 3.** build model using entire training dataset and the optimal parameters found in Step 2. Estimate the error rate of the optimal model (classifier) by comparing actual and predicted class labels for the samples in the test dataset.

Step 1 has to be performed by the user before submitting data to PCP. Clearly, the separation into training and test datasets may significantly reduce the amount of data available for training, but is required for a statistically rigorous approach to the design of learning machines ([7], [8]).

In addition, the prescribed procedure may result in large variance of the resulting estimate of prediction risk. The bagging (committee) learning described in Section 5.2 may be useful in dealing with this problem.

The cross-validation in Step 2 is completely automated within PCP. The program divides the training dataset into subsets, performs all necessary experiments and reports results. Internally, the $k-1$ subsets are called *learning set*, and the remaining subset is called *validation set*, consistent with the terminology in [7]. However, since the process is entirely automatic, these subsets are not normally visible to the end user.

The model selection in Step 2 is automated within PCP for Support Vector Machine algorithm, MLP and $k$-NN. For SVM, PCP automatically selects the optimal values of the SVM parameters which minimize cross-validation error rate, as described in Section 5.8.1. For MLP, PCP performs cross-validation for varying number of nodes in the hidden layer. The optimal number of nodes is the value giving the lowest cross-validation error rate. For $k$-NN, PCP performs cross-validation for varying number of nearest neighbors. The optimal $k$ is the value which achieves the lowest cross-validation error rate.

Step 3 is performed by building a model using the training dataset, and then applying it to the test dataset. PCP automatically calculates error rates and other performance criteria for the test dataset. For SVM, MLP and $k$-NN, this step is also automatic, and follows execution of Step 2.

As a result of this paradigm, the PCP supervised classification menus have a certain common general structure and learning parameters. They are described in the next section.

## 5.2 Menu Structure and Common Learning Parameters

**Pattern Classification** menu provides access to the implementations of the following popular supervised pattern classification algorithms: Multi-Layer Perceptron (MLP), $k$-Nearest Neighbor classifier, linear and quadratic parametric classifiers, linear discriminant classifier and Support Vector Machine (SVM) algorithm. In addition, is also implements **Bayes Error Estimation** algorithm, an important although not widely known method for estimating classification error due to Fukunaga [4].

Supervised pattern classification menus, with the exception of **Bayes Error Estimation**, have similar structure based on the principles of statistical learning. As an example of the menu options, Fig. 5.1 shows **Multi-Layer Perceptron** menu. The seven active options provide the following functionality:

- **Learning** builds a classifier (in this case Multi-Layer Perceptron) using the training dataset, and stores it in a model file.
- **Bagging (Committee) Learning** builds a *bagging* MLP classifier. Bagging (also known as *committee*) classifiers are built by combining multiple classifiers, each trained on a set created by pseudo-randomly resampling the original training dataset. The prediction is obtained by taking a majority vote of the individual classifiers. The PCP implementation of bagging classifiers is based on [13], Sections 2.2.2 and 2.3.3. The result of the learning is a committee classifier stored in the model file.
- **Prediction** uses a previously built classifier to assign class predictions to the test dataset vectors. In case the test set is split into files corresponding to the classes, as explained in Section 3.4, this function also computes and displays performance measures.
- **Cross-validation** performs the requested number of cross-validation experiments using the training dataset. The result of the analysis is a text file with detailed performance results for each experiment. The file format is described in Section 5.3.1.
- **Bagging Cross-validation** performs the cross-validation experiments using a bagging version of the classifier.

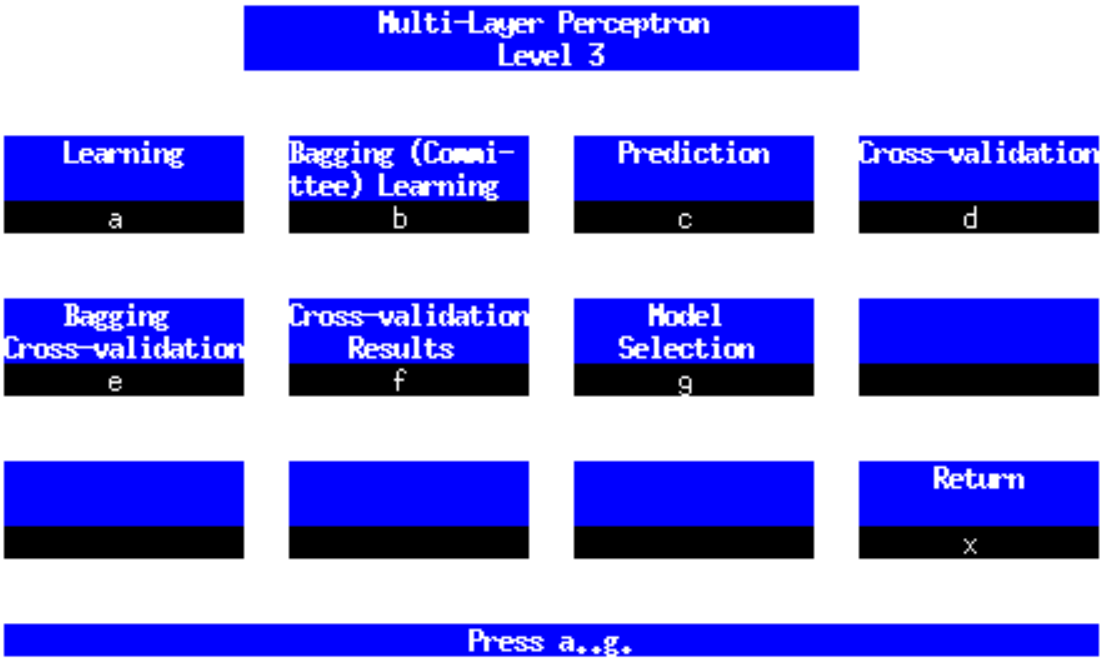


Figure 5.1: Multi-Layer Perceptron menu

- **Cross-validation Results** option displays summary of previously performed cross-validation experiments.
- **Model Selection** automatically chooses best classifier parameter(s), and is available for MLP, $k$-NN and SVM.

**Cross-validation**, **Bagging Cross-validation** and **Model Selection** correspond to Step 2 in the statistical learning paradigm; **Learning**, **Bagging** and **Prediction** correspond to Step 3.

The learning process described above is controlled by a set of user-defined parameters which can be divided in two groups: parameters common to all algorithms, and parameters specific to a particular algorithm. In the next section, we explain the common learning parameters by using the example of linear parametric classifier. The algorithm-specific parameters are described in detail in the pertaining sections.

### 5.2.1 Common Parameters for Supervised Learning

The examples in this section use IRIS dataset. To define the dataset in PCP, type:

```
% pcp -b iris.bat
```

and then restart PCP. To access linear parametric classifier menu, use **Parametric Classifiers...** menu option in **Pattern Classification** menu, and then choose **Linear Classifier...**. The menu is shown in Fig. 5.2.

#### Learning and Prediction

To start learning, press **a** key. The program asks:

```
Enter the classifier file name [pcp.plc]:
```

Press **Enter**, or supply the file name. PCP will store the classifier parameters in the specified file. In many situations, the default name is most convenient.

Upon learning, PCP displays error rates achieved on the training data set and pauses. Press any key to return to the menu.

To start bagging (committee) learning, press **b**. The program asks:

```
Enter the classifier file name [pcp.plc]:

Enter seed for pseudo-random number generator [    1]:

Enter number of classifiers to combine (>= 1) [   10]:
```

Again, PCP will store classifier parameters for the bagging classifier in the specified file. The seed initializes pseudo-random number generator used to resample the training dataset. Bagging classifier is constructed by combining the given number of classifiers.

To perform prediction using the IRIS test dataset, press **c**:

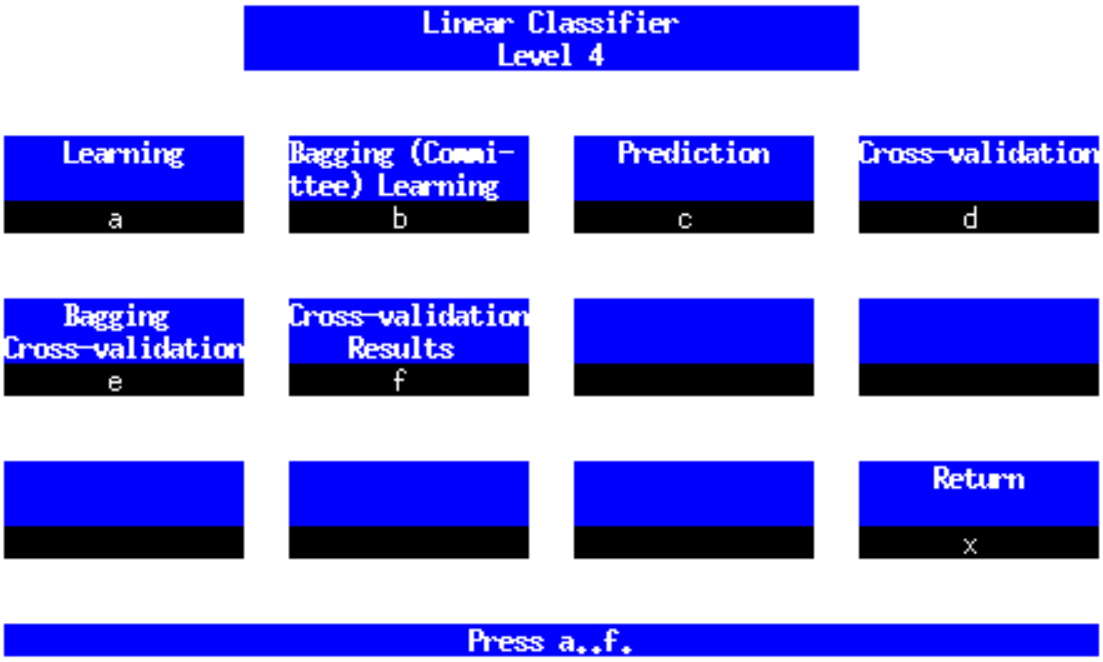


Figure 5.2: Parametric Linear Classifier menu

```
Enter parametric linear classifier file name [pcp.plc]:

Short (0) or long (1) output [0]:

+-------------------------------------------------------------------------+
|        Class         |  Actual/predicted card.  |        Error rate     |
+-------------------------------------------------------------------------+
|                      |         150/150          |    2.00% (     3/   150) |
|   1/iris_setosa      |          50/50           |    0.00% (     0/    50) |
|   2/iris_versicolor  |          50/49           |    4.00% (     2/    50) |
|   3/iris_virginica   |          50/51           |    2.00% (     1/    50) |
+-------------------------------------------------------------------------+
| Vector |          Actual  class          |  Parametric linear prediction  |
+-------------------------------------------------------------------------+
|     71 | iris_versicolor                | iris_virginica                 |
|     84 | iris_versicolor                | iris_virginica                 |
|    134 | iris_virginica                 | iris_versicolor                |
+-------------------------------------------------------------------------+
```

The **Prediction** is performed using classifier defined in the user-supplied file name. The default file name is `pcp.plc`, the same as the default output file name used in **Learning**. By choosing the default file name for learning and prediction, interaction with the program is simplified since there is no need to type any file names.

The first part of the output table shows cumulative and class-conditional actual and predicted cardinalities, and error rates (the cumulative actual and predicted cardinalities are, by definition, the same).

The lower part of the output shows predictions for individual vectors (patterns). In the short output format, the default, only the misclassified vectors are shown; in the long format, prediction results for all vectors in the test set are shown.

Note that the display slightly differs if the datasets contain two classes. The error rates and additional performance measures computed for two-class case are further described in Section 5.10. Also, when input files are in the *named vector* format, the individual vector names are displayed in the *Vector* column, instead of the vector indexes.

### Cross-validation

To launch cross-validation of linear parametric classifier using the currently active training dataset, press **d** key:

```
Enter the classifier file name [pcp.xpl]:

Enter number of cross-validation subsets (2..50) [   10]:

Enter seed for pseudo-random number generator [    1]:

Enter number of experiments [1]:
```

```
100
Use raw (0) or normalized data (1) [0]:

Choose dim. reduction method: none (0), FLD (2), PCA (3), EMAP (4), feature ra
nking (5), forward selection (6), backward elimination (7), or forward floatin
g search (8) [0]:
```

The training dataset is automatically split into the given number of cross-validation subsets (the number $k$ as detailed in Section 5.1). Seed is used to initialize the pseudo-random number generator to split the data. The reported performance measures are averages over the given number of experiments.

Dimensionality reduction method is used to select method for optionally mapping the training dataset into the lower dimension, where a more accurate classifier may be built. Description of dimensionality reduction algorithms available in PCP is given in Chapter 6. In case normalization is selected, the data is first scaled and shifted to zero-mean and unit-variance.

Note that both dimensionality reduction and normalization are performed in a rigorous cross-validation manner: the parameters for the operation are computed using the *learning* dataset, and are then applied to both *learning* and *validation* datasets.

Once cross-validation is completed, the results are available in the chosen output file, for display or analysis by external programs. To view the results, press **f**.

The bagging cross-validation (**e**) is very similar, except that it also requires that the user specifies the number of combined classifiers. Results are again available through the **Cross-validation Results** option.

The above dialog accurately represents the interaction with PCP in case no dimensionality reduction is requested; otherwise, the program requests additional parameters, described in Chapter 6.

## 5.3 Output File Formats

Pattern classifiers in PCP produce three different output files:

- cross-validation file
- model selection file
- prediction file

Cross-validation file contains results of cross-validation error estimation for a given classifier. Model selection performs repeated cross-validation simulations for different values of classifier parameters in order to choose optimal combination of the parameters. Prediction results file contains predictions and optionally prediction strengths for the test set samples, given a fixed classifier. The general formats of these files are described in the following sections; the details depend on the classifier and are further explained in the relevant classifier sections.

### 5.3.1 Cross-validation File

Cross-validation output file contains detailed performance results for individual experiments in a cross-validation analysis.

A typical file for multi-class cross-validation has the following format:

```
# classifier:                          parametric linear classifier
# number of models:                    1
# number of experiments:               10
# number of cross-validation subsets:  10
# number of classes:                   3
# normalization:                       no
# dimensionality reduction method:     none
# number of features:                  4
# seed:                                1
1       2.00   (3/150)  0.00   (0/50)   4.00   (2/50)   2.00   (1/50)
2       2.00   (3/150)  0.00   (0/50)   4.00   (2/50)   2.00   (1/50)
3       2.00   (3/150)  0.00   (0/50)   4.00   (2/50)   2.00   (1/50)
4       2.00   (3/150)  0.00   (0/50)   4.00   (2/50)   2.00   (1/50)
5       2.00   (3/150)  0.00   (0/50)   4.00   (2/50)   2.00   (1/50)
...
```

The comment lines (lines starting with # character) should be largely self-explanatory. Note that the *number of features* refers to the number of features after reduction of dimensionality. Each non-comment line corresponds to a single experiment and has the experiment ID in the first column. The second column is the cumulative error rate. It is followed by actual counts of misclassified vectors and total vectors in the training dataset, given in parentheses. Thus, in the above example, the first cross-validation experiment misclassified 3 vectors out of 150, for an error rate of 2%. The remaining columns have corresponding information for each class. In the example above, there are 3 classes, and thus six additional columns.

The two-class cross-validation introduces four additional performance measures (sensitivity, specificity, positive and negative predictive value) as shown in the following segment:

```
# classifier:                          SVM
# number of models:                    1
# number of experiments:               10
# number of cross-validation subsets:  10
# number of classes:                   2
# normalization:                       no
# dimensionality reduction method:     none
# number of features:                  1994
# seed:                                1
# SVM type:                            C-SVM
# kernel type:                         RBF
# gamma:                               0.01
# stopping criterion:                  0.001
```

```
# cost (C):                              1000
1        5.26   (2/38)   0.00   (0/27)  18.18   (2/11)  100.00  81.82   93.10   100.00
2        5.26   (2/38)   0.00   (0/27)  18.18   (2/11)  100.00  81.82   93.10   100.00
3        5.26   (2/38)   0.00   (0/27)  18.18   (2/11)  100.00  81.82   93.10   100.00
4        5.26   (2/38)   0.00   (0/27)  18.18   (2/11)  100.00  81.82   93.10   100.00
5        5.26   (2/38)   0.00   (0/27)  18.18   (2/11)  100.00  81.82   93.10   100.00
...
```

The first seven columns have the same meaning as the multi-class file format described above. The remaining four columns are sensitivity, specificity, positive and negative predictive value, respectively, which are important quantities in the analysis of two-class classifiers. Section 5.10 provides definition and detailed description of these performance indicators.

### 5.3.2 Model Selection File

The model selection file `pcp.msl` contains performance value for each combination of classifier parameters evaluated during the model selection search. A typical SVM model selection file for RBF kernel, applied to the IRIS dataset, may look like this:

```
   8.13          0.1           1e-08    1
   9.00          0.1      6.3096e-08    2
   8.60          0.1      3.9811e-07    3
   8.67          0.1      2.5119e-06    4
   8.67          0.1      1.5849e-05    5
   8.67          0.1          0.0001    6
...
```

Each line corresponds to a combination of parameters evaluated by the search. The first column is misclassification error rate (in percents). The next two columns are the values of SVM parameters $C$ or $\nu$, and $\gamma$, and the last column is iteration number.

In case of linear kernel, there is only one variable parameter ($C$ or $\nu$) and thus the model selection file has three columns.

Similarly, in case of MLP and $k$-NN model selection, the file has three columns. The first column is misclassification error rate, the second column is number of hidden nodes or number of nearest neighbors, respectively, and the last column is the iteration number.

The format is slightly more verbose for two-class problems, as follows (the example is for $k$-NN model selection):

```
   2.63            1    1        100.00   90.91   96.43  100.00
   5.26            2    2         96.30   90.91   96.30   90.91
   2.63            3    3        100.00   90.91   96.43  100.00
   5.26            4    4        100.00   81.82   93.10  100.00
   2.63            5    5        100.00   90.91   96.43  100.00
   7.89            6    6         96.30   81.82   92.86   90.00
   5.26            7    7        100.00   81.82   93.10  100.00
   5.26            8    8        100.00   81.82   93.10  100.00
```

```
   7.89            9    9        100.00   72.73   90.00  100.00
   7.89           10    10       100.00   72.73   90.00  100.00
...
```

The columns are:

- misclassification error rate
- number of nearest neighbors
- iteration number
- the last four columns are sensitivity, specificity, positive predictive value and negative prediction value in percentages

### 5.3.3 Prediction File

Prediction file `pcp.rcl` is a tab-delimited ASCII file which contains results of PCP prediction. Each line corresponds to a sample in the test set. For more than two classes, the file format is as follows:

```
1       ews_test        ews     1
2       ews_test        ews     1
3       ews_test        ews     1
4       ews_test        ews     1
5       ews_test        ews     1
6       ews_test        rms     0
7       ews_test        ews     1
8       rms_test        rms     1
...
```

The first column has vector ID, in case input data is in the raw format. If input data is in the *named* or *named row* format, the first column contains vector names. The second column is actual class, the third column is predicted class. The fourth column is 1 for correct prediction, 0 for incorrect. As explained in Section 3.4, this column is only present if the test set files (i.e., classes) correspond to the training set files/classes. This means that the number of test set files must be equal the number of training set files. PCP then assumes that test file 1 corresponds to class 1 of the training dataset, test file 2 corresponds to class 2 of the training dataset, etc. In the example above, test class 1 is named `ews_test`, after the file `ews_test.dat`, and the corresponding training class is named `ews`, after the file `ews.dat`. Test class 2 is named `rms_test`, after the file `rms_test.dat`, and training class 2 is named `rms`, after the file `rms.dat`. The classification for vector 1 is correct because the classifier predicts class 1, and the actual class is also class 1. In contrast, the classification for vector 6 is incorrect because the classifier predicts class 2 (`rms`), whereas the actual class is class 1 (`ews_test`).

For SVM classifier, the file has additional $c$ columns, where $c$ is the number of classes. The additional columns represent class probability estimates as computed by the LIBSVM library [30].

```
1       iris_setosa     iris_setosa     1       0.975769        0.0162202       0.00801093
2       iris_setosa     iris_setosa     1       0.961227        0.0286403       0.0101325
3       iris_setosa     iris_setosa     1       0.974465        0.0175656       0.00796911
4       iris_setosa     iris_setosa     1       0.957805        0.0310942       0.011101
5       iris_setosa     iris_setosa     1       0.978092        0.0142806       0.007627
...
```

For two classes, additional four columns (flags) are stored in the file. The columns are *true positive*, *false negative*, *false positive* and *true negative*, respectively. See Section 5.10 for definitions of these terms.

```
39 all_test all_train 1 1 0 0 0
40 all_test aml_train 0 0 1 0 0
42 all_test all_train 1 1 0 0 0
47 all_test all_train 1 1 0 0 0
48 all_test all_train 1 1 0 0 0
49 all_test all_train 1 1 0 0 0
...
```

Again, for SVM, PCP also saves class probability estimates:

```
39      all_test        all_train       1       1       0       0       0       0.955901        0.0440991
40      all_test        all_train       1       1       0       0       0       0.859633        0.140367
42      all_test        all_train       1       1       0       0       0       0.869194        0.130806
47      all_test        all_train       1       1       0       0       0       0.960011        0.039989
48      all_test        all_train       1       1       0       0       0       0.994483        0.00551723
49      all_test        all_train       1       1       0       0       0       0.932164        0.0678358
...
```

## 5.4 Parametric Linear and Quadratic Classifiers

Linear and quadratic classifier menus have the standard menu structure described previously. To access either of the menus, press **b** in **Pattern Classification** menu, and then press **a** or **b** to access the Linear or Quadratic classifier menu, respectively. The two algorithms implement Bayesian classifier for normal distributions with common or arbitrary class-conditional covariance matrices. For details of the algorithms, see for example [20].

The algorithms do not require any user-specified parameters, and thus the Section 5.2.1 completely describes the interaction.

## 5.5 $k$-Nearest Neighbor Classifier

The $k$-NN menu is slightly different from the standard described in Section 5.2. $k$-NN algorithm, as defined in any machine learning textbook (for example [20], [4]), assigns an unlabeled vector to the dominant class among the $k$ nearest vectors in the training set. Consequently, there is no learning mode, and this is reflected in the menu layout shown in Fig. 5.3.

In addition to the absence of learning mode, there is no file which stores the parameters of the classifier; in effect, the entire training dataset serves as the model file(s). Also, the user must

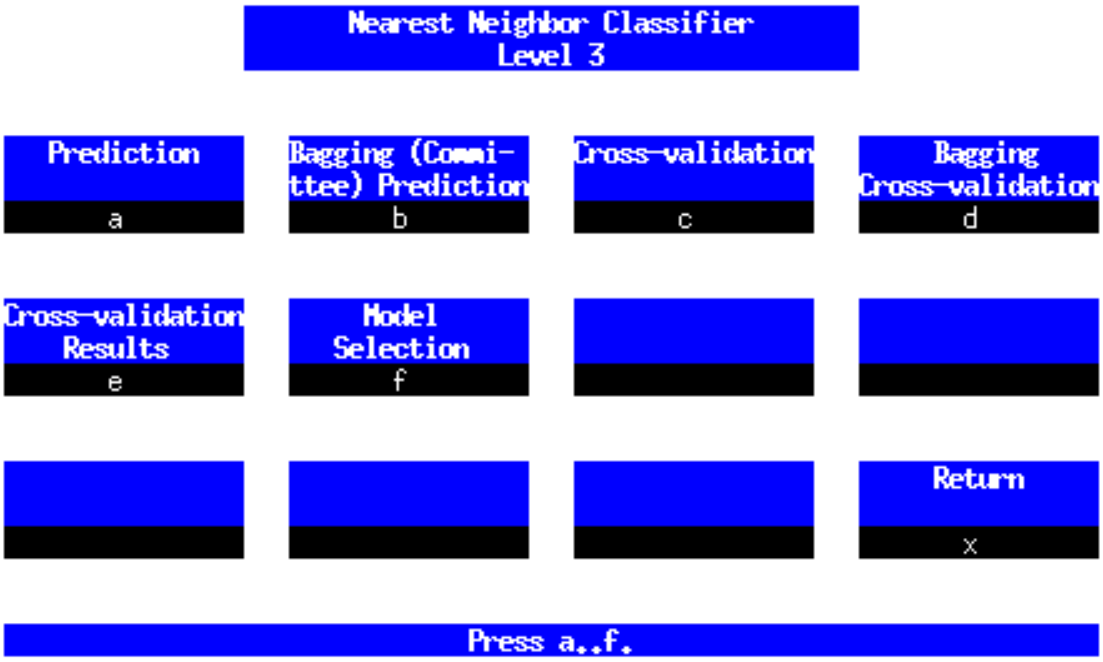


Figure 5.3: Nearest Neighbors Classifiers menu

specify the distance measure to be used for defining the neighbors. The **Prediction** function dialog proceeds as follows:

```
Enter number of nearest neighbors (1..50) [    1]:

Use Euclidean (1), city-block (2) or Mahalanobis distance (3) [1]:

Short (0) or long (1) output [0]:
```

The remaining functionality, except **Model Selection**, follows the general flow of processing described in Section 5.2.1. **Model Selection** is used to choose the optimal value of $k$ via cross-validation. To start **Model Selection**, press key **f**. A complete transaction using the default values of all parameters is shown as follows:

```
Use Euclidean (1), city-block (2) or Mahalanobis distance (3) [1]:

Enter number of cross-validation subsets (2..11) [   10]:

Enter number of experiments [1]:

Enter seed for pseudo-random number generator [    1]:

Use raw (0) or normalized data (1) [0]:

Choose dim. reduction method: none (0), SVD (1), EMAP (4), feature ranking (5)
, forward selection (6), or backward elimination (7) [0]:

Enter starting number of nearest neighbors (1..10) [1]:

Enter ending number of nearest neighbors (1..10) [10]:

Enter step (>= 1) [1]:

Experiment      1 cross-validation error rate:    2.63%
Iteration       1;  mean error rate:     2.63%
Current k:          1; error rate:    2.63%.
Optimal k:          1; error rate:    2.63%.

Experiment      1 cross-validation error rate:    5.26%
Iteration       2;  mean error rate:     5.26%
Current k:          2; error rate:    5.26%.
Optimal k:          1; error rate:    2.63%.

Experiment      1 cross-validation error rate:    2.63%
Iteration       3;  mean error rate:     2.63%
Current k:          3; error rate:    2.63%.
Optimal k:          1; error rate:    2.63%.

Experiment      1 cross-validation error rate:    5.26%
Iteration       4;  mean error rate:     5.26%
Current k:          4; error rate:    5.26%.
Optimal k:          1; error rate:    2.63%.
```

```
Experiment      1 cross-validation error rate:    2.63%
Iteration       5;  mean error rate:     2.63%
Current k:          5; error rate:    2.63%.
Optimal k:          1; error rate:    2.63%.

Experiment      1 cross-validation error rate:    7.89%
Iteration       6;  mean error rate:     7.89%
Current k:          6; error rate:    7.89%.
Optimal k:          1; error rate:    2.63%.

Experiment      1 cross-validation error rate:    5.26%
Iteration       7;  mean error rate:     5.26%
Current k:          7; error rate:    5.26%.
Optimal k:          1; error rate:    2.63%.

Experiment      1 cross-validation error rate:    5.26%
Iteration       8;  mean error rate:     5.26%
Current k:          8; error rate:    5.26%.
Optimal k:          1; error rate:    2.63%.

Experiment      1 cross-validation error rate:    7.89%
Iteration       9;  mean error rate:     7.89%
Current k:          9; error rate:    7.89%.
Optimal k:          1; error rate:    2.63%.

Experiment      1 cross-validation error rate:    7.89%
Iteration      10;  mean error rate:     7.89%
Current k:         10; error rate:    7.89%.
Optimal k:          1; error rate:    2.63%.

Applying the optimal classifier to the test dataset...

+-------------------------------------------------------------------------+
|        Class         |  Actual/predicted card.  |        Error rate        |
+-------------------------------------------------------------------------+
|                      |          34/34           |   11.76% (     4/    34) |
|   1/all_test         |          20/22           |    5.00% (     1/    20) |
|   2/aml_test         |          14/12           |   21.43% (     3/    14) |
+-------------------------------------------------------------------------+
| Sensitivity                                     |   95.00% (    19/    20) |
| Specificity                                     |   78.57% (    11/    14) |
| Positive Predictive Value                       |   86.36% (    19/    22) |
| Negative Predictive Value                       |   91.67% (    11/    12) |
+-------------------------------------------------------------------------+
| Vector               | Actual class             | k-NN prediction          |
+-------------------------------------------------------------------------+
| 40                   | all_test                 | aml_test                 |
| 52                   | aml_test                 | all_test                 |
| 54                   | aml_test                 | all_test                 |
| 60                   | aml_test                 | all_test                 |
+-------------------------------------------------------------------------+
```

As can be seen from this dialog, the procedure is consistent with the algorithm described in Section 5.1, PCP computes $k$-NN cross-validation error rates for $1 \leq k \leq 10$. Once the optimal

value (1, in this case) is found, it is used for the classification of the test set. The results of the model selection and test set prediction are saved in files `pcp.msl` and `pcp.rcl`, respectively.

## 5.6 Linear Discriminant Classifier

The implementation follows the standard, least-squares linear discriminant classifer algorithm as described in [15], Section 3.4.3 (note that the development in the reference assumes a non-linear transformation of input variables; for our purposes, the transformation is identity function, except for the bias term for which the corresponding transformation equals one).

The **Linear Discriminant Classifier** menu is shown in Fig. 5.4. The current implementation does not require any user-defined parameters, and all dialogs use only common supervised learning parameters defined in Section 5.2.1.

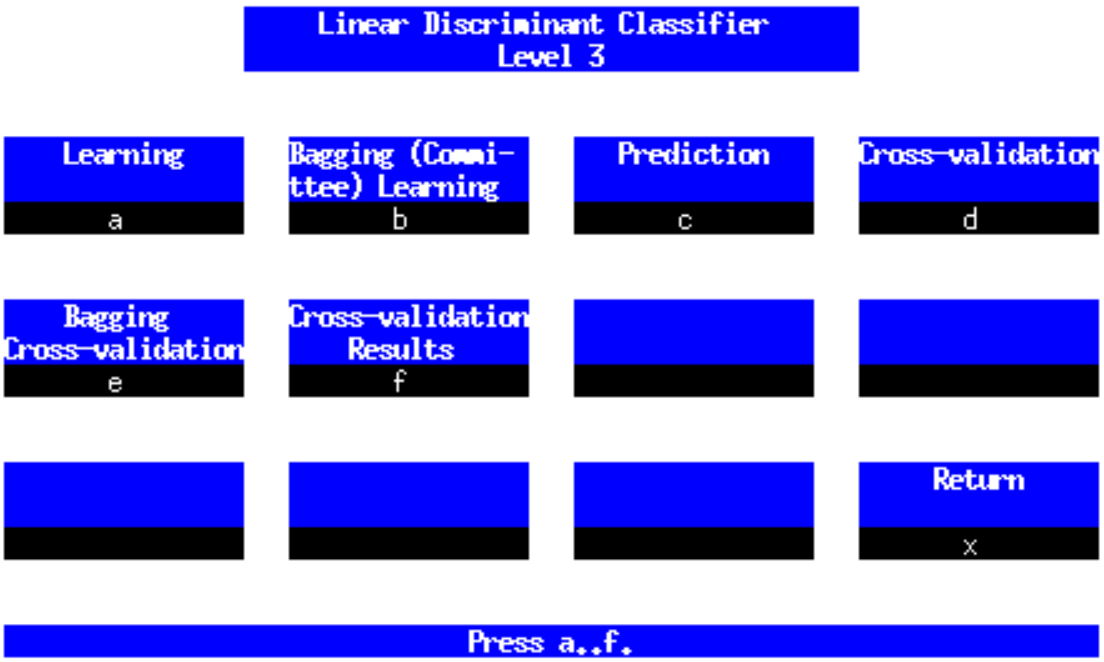


Figure 5.4: Linear Discriminant Classifier menu

## 5.7 Multi-layer Perceptron

The multi-layer perceptron (MLP) menu, shown in Fig. 5.1, provides the standard functions described in Section 5. In this section we describe the structure of MLP as used in PCP, parameters specific to this learning method, MLP file format, and model selection functionality. The section

assumes familiarity with the MLP model of machine learning; for an excellent introduction, see [15].

### 5.7.1 MLP Structure

Fig. 5.5 shows the structure of a single hidden layer MLP in PCP. As indicated in the picture, the first hidden layer is called *layer 0*, and further layers are labeled *layer 1*, *layer 2*, etc. Internally, all nodes in the network are indexed sequentially starting from 0. Thus, in Fig. 5.5, first (and only) hidden layer has four nodes labeled 0 – 3, and the output nodes are labeled 4 – 6. Each node implements sigmoid transfer function (5.1).

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{5.1}$$

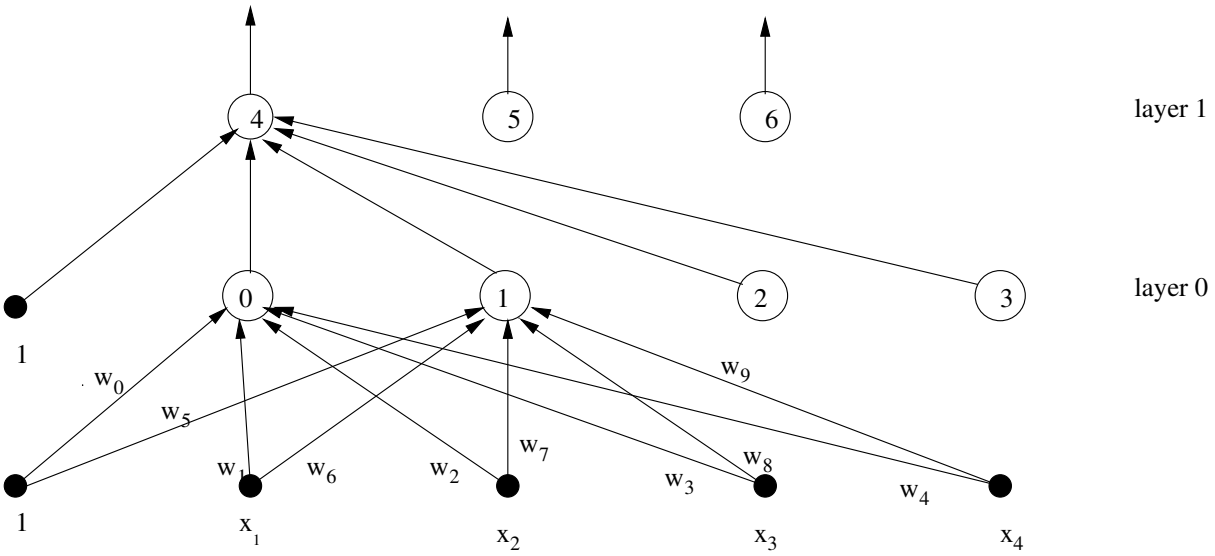


Figure 5.5: The MLP structure in PCP. Filled circles labeled $x_i$ represent the $d$ inputs, where $d$ is dimension of the input space. Filled circles labeled **1** correspond to the bias terms.

PCP also implements support for multi-layer perceptrons with multiple hidden layers. This functionality is not fully tested in the present release (see Section 8.1).

### 5.7.2 MLP Learning Parameters

The MLP implementation in PCP uses the *batch mode* learning. The following dialog is used to specify parameters for MLP learning:

```
Begin learning (0), or continue (1) [0]:

Enter MLP file name [pcp.mlp]:

Enter number of hidden layers (>= 1) [1]:
```

```
Number of inputs is     4.
Number of output nodes is     3.

Enter number of nodes in hidden layer     1:
5
Enter seed for pseudo-random number generator [    1]:

Enter amplitude of initial weights [ 1.00]:

Enter optimization method - conj. grad (1), grad. descent (2) [    1]:

Enter number of iterations (-1 for no iterations) [100]:
```

The following parameters are specific to MLP learning:

- number of hidden layers
- number of nodes in each of the hidden layers
- amplitude of initial weights
- optimization method
- number of learning iterations (also known as *epochs*)

In case user chooses a non-default optimization procedure (gradient descent), two additional parameters must be specified: `learning rate` and `momentum term`.

The `seed` is used to initialize the pseudo-random number generator which generates initial weights in the network. `range` is amplitude of the weights. The default value for `range` is 1.0, in which case the initial weights will be pseudo-random numbers in the interval $[-1.0, 1.0]$.

MLP learning in PCP implements two optimization procedures to adjust the weights of the network: a conjugate gradient method, and the standard gradient descent, sometimes misleadingly known as *backpropagation*.[1] The conjugate gradient algorithm is a C version of the FORTRAN package CG+, written by Guanghui Liu, J. Nocedal and R. Waltz [6], and is typically far more efficient than the gradient descent.

### 5.7.3 MLP Classifier File Format

Contents of an MLP file corresponding to perceptron in Fig. 5.5 is shown in Fig. 5.6.

The first line is used for bagging version of MLP; in that case, the MLP file contains multiple classifiers. The first number in the line is index of the individual network comprising the committee classifier, and the second number is the corresponding weight. In the case shown, no bagging was

[1] The *backpropagation* is actually a remarkably efficient method for computing gradient of the MLP criterion function, which is completely independent of the optimization method used to find the optimum of the function.

```
combined model / combined weight:       1 / 1.000000
iterations:                             100
optimization method:                    CG+ conjugate gradient
seed:                                   1
range of initial weights:               1.000000
misclassified samples:                  3
avg. error per node and sample (MSE):   0.037898
number of layers:                       2
nodes in layer 0:                       4
nodes in layer 1:                       3
number of inputs:                       4
number of weights:                      35
weights from inputs to node    0:       -2.031595 -1.036182 -1.129723 1.863392 1.789858
weights from inputs to node    1:       -0.663014 -0.649625 0.370059 -0.628752 0.062194
weights from inputs to node    2:       0.148260 1.240165 0.345430 0.403361 0.980632
weights from inputs to node    3:       0.586543 -0.495635 5.023008 -4.666823 1.263193
weights from layer 0 to node   4:       -0.465108 -11.170246 -0.277840 0.819177 1.552297
weights from layer 0 to node   5:       1.136494 -6.228109 -1.066593 2.439004 -5.501239
weights from layer 0 to node   6:       -3.874971 13.272957 -0.070897 -4.026823 -4.078823
```

Figure 5.6: MLP file for IRIS data

applied, thus there is only one classifier. The weight of all classifiers in the bagging model is equal to **1.0**; this parameter is reserved for future use in AdaBoost [24] learning algorithm.

The line `weights from inputs to node 0` corresponds to weights labeled $w_0 - w_4$ in Fig. 5.5. The first coefficient in the line is the weight associated with the bias term; the rest of the line lists weights from the $d$ inputs to node 0.

The next line, `weights from inputs to node 1` corresponds to weights labeled $w_5 - w_9$ in Fig. 5.5; again, the first coefficient in the line is the bias term.

The remaining weight lines can be interpreted accordingly.

The rest of the file should be self-explanatory.

### 5.7.4 MLP Model Selection

MLP **Model Selection** is used to determine the optimal number of hidden nodes. The dialog is similar to **Model Sekection** for $k$-nearest neighbor algorithm, described in Section 5.5, and is started by pressing key **g**:

```
Enter amplitude of initial weights [ 1.00]:

Enter optimization method - conj. grad (1), grad. descent (2) [    1]:

Enter number of iterations (-1 for no iterations) [100]:

Enter number of cross-validation subsets (2..50) [   10]:
```

```
Enter number of experiments [1]:

Enter seed for pseudo-random number generator [    1]:

Use raw (0) or normalized data (1) [0]:

Choose dim. reduction method: none (0), FLD (2), PCA (3), EMAP (4), feature ra
nking (5), forward selection (6), or backward elimination (7) [0]:

Enter starting number of nodes (>= 1) [5]:

Enter ending number of nodes (>= 5) [10]:

Enter step (>= 1) [1]:

Experiment      1 cross-validation error rate:    2.67%
Iteration       1;  mean error rate:     2.67%
Current number of hidden nodes:          5; error rate:    2.67%.
Optimal number of hidden nodes:          5; error rate:    2.67%.

Experiment      1 cross-validation error rate:    2.67%
Iteration       2;  mean error rate:     2.67%
Current number of hidden nodes:          6; error rate:    2.67%.
Optimal number of hidden nodes:          5; error rate:    2.67%.

Experiment      1 cross-validation error rate:    2.67%
Iteration       3;  mean error rate:     2.67%
Current number of hidden nodes:          7; error rate:    2.67%.
Optimal number of hidden nodes:          5; error rate:    2.67%.

Experiment      1 cross-validation error rate:    2.67%
Iteration       4;  mean error rate:     2.67%
Current number of hidden nodes:          8; error rate:    2.67%.
Optimal number of hidden nodes:          5; error rate:    2.67%.

Experiment      1 cross-validation error rate:    2.67%
Iteration       5;  mean error rate:     2.67%
Current number of hidden nodes:          9; error rate:    2.67%.
Optimal number of hidden nodes:          5; error rate:    2.67%.

Experiment      1 cross-validation error rate:    6.67%
Iteration       6;  mean error rate:     6.67%
Current number of hidden nodes:         10; error rate:    6.67%.
Optimal number of hidden nodes:          5; error rate:    2.67%.

Applying the optimal classifier to the test dataset...

Epoch:        0; avg. error per output node:  0.2607191; error rate:  66.67
Epoch:       10; avg. error per output node:  0.0618523; error rate:   4.67
Epoch:       10; avg. error per output node:  0.0585874; error rate:  15.33
Epoch:       10; avg. error per output node:  0.0580851; error rate:  10.00
Epoch:       10; avg. error per output node:  0.0580256; error rate:  12.00
Epoch:       20; avg. error per output node:  0.0156552; error rate:   2.00
Epoch:       20; avg. error per output node:  0.0152103; error rate:   2.00
```

```
Epoch:       20; avg. error per output node:  0.0145961; error rate:   2.00
...
Epoch:      100; avg. error per output node:  0.0090650; error rate:   2.00
Epoch:      100; avg. error per output node:  0.0090649; error rate:   2.00

+-----------------------------------------------------------------------------+
|        Class         |  Actual/predicted card.  |        Error rate        |
+-----------------------------------------------------------------------------+
|                      |          150/150         |    2.00% (     3/   150) |
|   1/iris_setosa      |           50/50          |    0.00% (     0/    50) |
|   2/iris_versicolor  |           50/47          |    6.00% (     3/    50) |
|   3/iris_virginica   |           50/53          |    0.00% (     0/    50) |
+-----------------------------------------------------------------------------+
| Vector |          Actual  class          |         MLP  prediction         |
+-----------------------------------------------------------------------------+
|     71 | iris_versicolor                | iris_virginica                  |
|     73 | iris_versicolor                | iris_virginica                  |
|     84 | iris_versicolor                | iris_virginica                  |
+-----------------------------------------------------------------------------+
```

In the above example, PCP estimates error rates of the MLP for $5 \leq h \leq 10$ hidden nodes, and determines that the lowest error rate is achieved for $h = 5$. Subsequently it builds an MLP with 5 hidden nodes using entire training dataset, and applies it to the test dataset. The results of model selection and test set prediction are saved in files `pcp.msl` and `pcp.rcl`, respectively.

## 5.8 Support Vector Machine Algorithm

Support Vector Machine (SVM) algorithm [10, 17, 22] is a recent outgrowth of a comprehensive theory of machine learning known as *Statistical Learning Theory*, developed over the past four decades primarily by Vladimir N. Vapnik of AT & T Labs. The method has achieved wide acceptance due to the following important properties which distinguish it from other pattern classification algorithms (in addition to the demonstrated good performance on real-life problems):

- SVM has been demonstrated in practice to be able to handle input vectors with thousands of attributes
- Statistical Learning Theory provides rigorous foundation for understanding and estimating the generalization ability of SVM
- SVM is formulated in terms of a quadratic optimization problem, for which a global minimum can efficiently be found

SVM can generally be used in the same way as other pattern classification algorithms, however some knowledge of the parameters is helpful in order to achieve good performance.

PCP incorporates a widely used open-source SVM implementation called LIBSVM [30]. The SVM menu is shown in Fig. 5.7.

In addition to the standard supervised pattern classification options, the menu has **Save Problem in LIBSVM Format**. This function can be used to save training dataset in the LIBSVM-compatible file format known as *sparse* file format, in case further analysis by LIBSVM is required.

Pressing **a** key launches the following dialog and starts SVM learning:

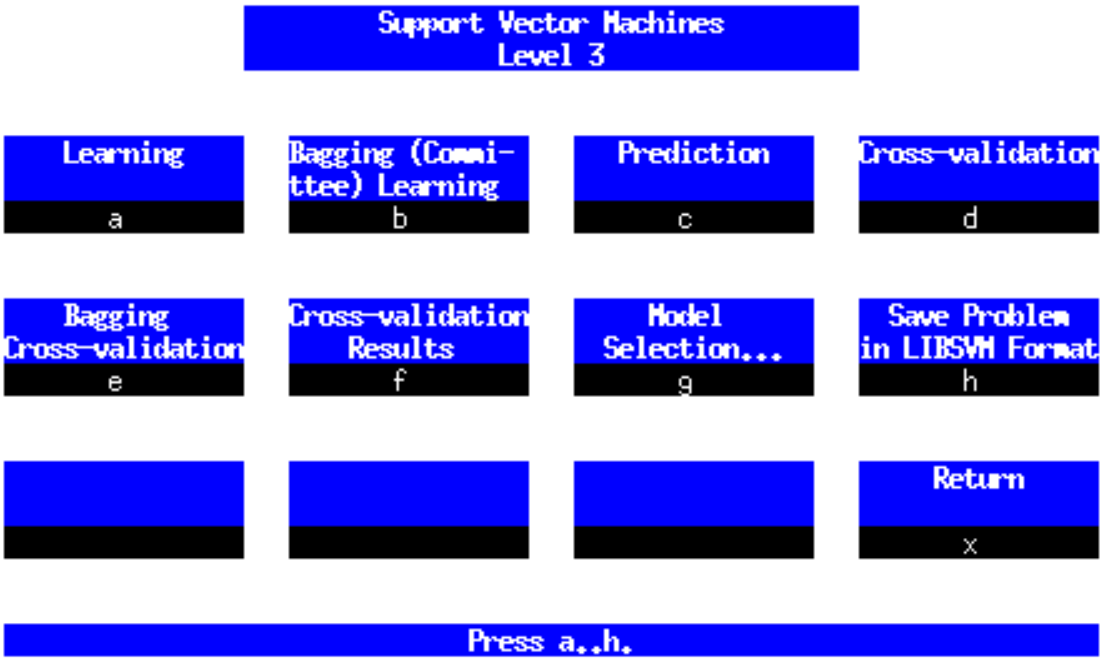


Figure 5.7: Support Vector Machines menu

```
Enter SVM type (1: NU-SVM; 2: C-SVM) [2]:

Enter kernel type (1: linear; 2: polynomial; 3: RBF; 4: sigmoid) [3]:

Enter cost parameter C [1000.00]:

Change class costs (1: yes; 0: no) [0]:

Enter gamma [    0.010000]:

Enter output model file name [pcp.svm]:
```

This dialog occurs if user chooses the default *C-SVM* SVM type, and *RBF* kernel. The parameter `C` (*cost*) is related to the penalty assigned to the input samples misclassified by the SVM. Presently there is no particular guidance as to the choice of value for `C`. The values in the range of 0.001 - 100000 have been used successfully, depending on the dataset.

The program enables the user to change the individual class costs. This may be useful to achieve a desired balance among class-conditional error rates.

Certain parameters are specific to the choice of SVM kernel function. PCP (through LIBSVM [30]) provides four kernel types (in the equations below, $\|X - Y\|$ is Euclidean distance between input vectors $X$ and $Y$, and $X \cdot Y$ is dot (scalar) product):

- RBF: $\kappa(X, Y) = \exp(-\gamma\|X - Y\|^2)$
- linear: $\kappa(X, Y) = X \cdot Y$
- polynomial: $\kappa(X, Y) = (\gamma X \cdot Y + c_0)^d$
- sigmoid: $\kappa(X, Y) = \tanh(X \cdot Y + c_0)$

The `gamma` ($\gamma$) parameter is specific to the choice of Radial Basis Function (RBF) or polynomial kernels. $c_0$ is specific to polynomial and sigmoid kernels, and $d$ is specific to polynomial kernel.

The recently developed `NU-SVM` algorithm [19] attempts to provide a more approachable alternative to the `C` parameter of `C-SVM`.

The `nu` ($\nu$) parameter of `NU-SVM` is a real number in the $[0, 1]$ interval, and it represents an upper bound on the training set error rate. In other words, when the SVM optimization algorithm converges, the fraction of misclassified samples in the training set will be at most $\nu$[2] (this doesn't say anything about the error rate on the independent test set).

Note that not all $[0, 1]$ values of $\nu$ are feasible. The corresponding `NU-SVM` problem does not have a solution if $\nu$ exceeds $\nu_{\max}$, defined for two-class case as:

$$\nu_{\max} = 2\frac{\min(n_1, n_2)}{n_1 + n_2} \tag{5.2}$$

[2]This statement is strictly true only if the SVM optimization problem is solved *exactly*. This, of course, can never be achieved due to, at least, floating-point round-off error, but in most practical cases it does appear to hold true.

where $n_1, n_2$ are the class cardinalities [25]. For multiple-class case, the condition $\nu \leq \nu_{\max}$ must be satisfied for each pair of classes.

The above interpretation, along with a bounded interval for the values of $\nu$, should provide a more intuitive means of controlling SVM generalization than the cost parameter `C`. For example, setting $\nu$ to 0.01 means that we tolerate up to 1% error rate on the training dataset, but at a cost of potentially poor performance on previously unseen examples. Conversely, setting $\nu$ to a higher value permits more errors during training, thus allowing a smoother discrimination function, and potentially better generalization performance.

A typical PCP SVM learning dialog involving the `NU-SVM` algorithm and RBF kernel might proceed as follows:

```
Enter SVM type (1: NU-SVM; 2: C-SVM) [2]:
1
Enter kernel type (1: linear; 2: polynomial; 3: RBF; 4: sigmoid) [3]:

Enter nu (max. allowed frac. of training errors) (0.00..0.58) [0.20]:
0.6
The value must be between 0.000000 and 0.580645.

Enter nu (max. allowed frac. of training errors) (0.00..0.58) [0.20]:
0.3
Enter gamma [    0.010000]:

Enter SVM model file name [pcp.svm]:
```

As shown in this example, the feasible region for $\nu$ for this particular dataset is $[0, 0.58]$. PCP does not accept input values outside the feasible region.

### 5.8.1 Model Selection

Model selection refers to the process of finding optimal values of classifier parameters, as previously defined in Section 5.1. For classifiers with a single, discrete-valued parameter, such as the number of nearest neighbors in $k$-NN or the number of hidden nodes in MLP, model selection is straightforward: perform cross-validation for a set of plausible values of the parameter, and choose the parameter value which minimizes a performance criterion.

Things are more complicated for SVM. The most commonly used configurations (C-SVM or NU-SVM with RBF kernel) have two continuous-valued parameters ($C$ or $\nu$, and $\gamma$). This makes it impractical to determine optimal combination by manually running cross-validation experiments. In addition, the performance criterion - typically cross-validation error rate - is a piecewise-constant function of the parameters, making it impossible to use gradient-based optimization procedures.

PCP provides automated way to determine the best SVM parameters through the **Model Selection** menu. The menu is accessed by pressing **g** key from **Support Vector Machines** menu, and is shown in Fig. 5.8.

PCP provides two algorithms for SVM model selection: *Simplex*, and *Grid Search*.

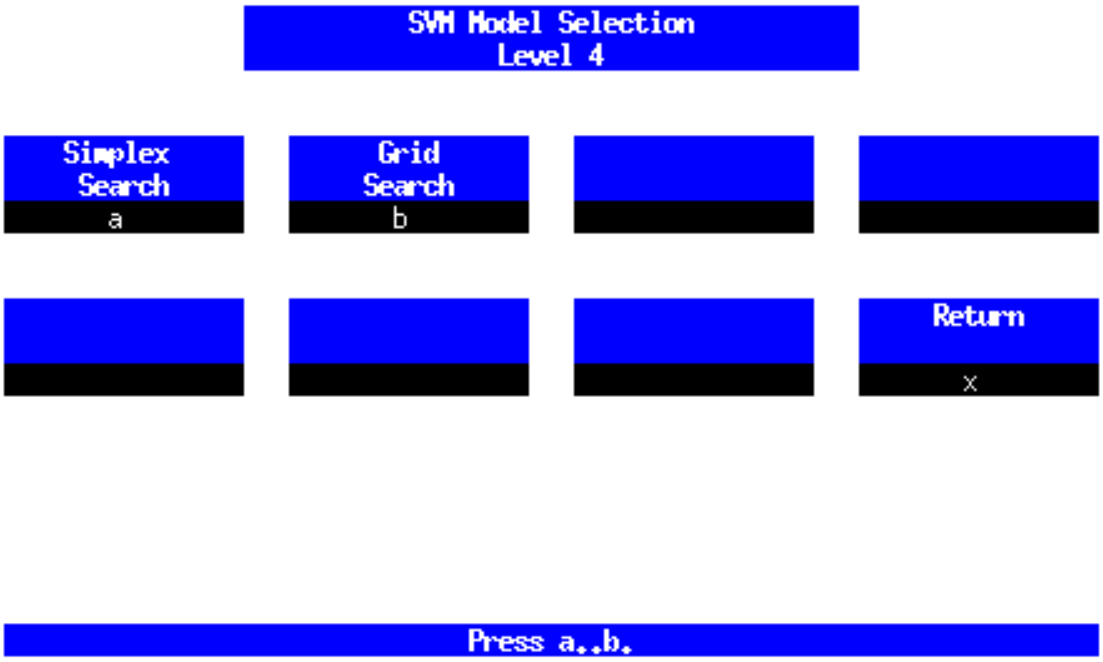


Figure 5.8: Model Selection Menu

The simpler method, Grid Search, computes cross-validation error rate on the training set over a 10 by 10 grid of parameter values (for a total of 121 $(C/\nu, \gamma)$ pairs) and reports the pair of values which give minimum error rate.

The Simplex search uses a heuristic based on *Simplex* algorithm for multivariate optimization without gradient ([5], [12]).

The searches are initiated by pressing the **a** or **b** key, respectively. The processing parameters are entered during the following dialog (the example applies to simplex search):

```
Enter SVM type (1: NU-SVM; 2: C-SVM) [2]:

Change class costs (1: yes; 0: no) [0]:

Enter number of cross-validation subsets (2..50) [   10]:

Enter number of experiments [1]:

Enter seed for pseudo-random number generator [    1]:

Use raw (0) or normalized data (1) [0]:

Choose dim. reduction method: none (0), FLD (2), PCA (3), EMAP (4), feature ra
nking (5), forward selection (6), backward elimination (7), or forward floatin
g search (8) [0]:

Enter maximum number of iterations [20]:
```

The only differences in comparison with cross-validation are fixed kernel type (Simplex model selection only supports RBF kernel, while Grid Search currently supports linear and RBF kernels), and lack of $C/\nu, \gamma$ parameters in the dialog. This is understanable since the purpose of the search is to find the optimal parameter values.

The optimal parameters and the corresponding error rate are displayed on screen at the conclusion of the search as follows:

```
Optimization completed in       4 iterations; minimum error rate:    2.00%.
Optimal parameters: C =       31.623; gamma =         0.01
```

As indicated previously (Section 5.3.2), detailed results of model selection are stored in file named `pcp.msl`. The file has the following format (the specific example is a result of Simplex search on IRIS dataset):

```
   8.13          0.1           1e-08    0
   8.13        10000           1e-08    0
   1.93       31.623            0.01    0
   6.60   3.1623e+06            0.01    1
   6.60   3.1623e+06            0.01    1
```

```
  64.13        10000           10000    2
   3.00        10000           1e-05    2
   3.00        10000           1e-05    2
   8.67          0.1           1e-05    3
   3.47        42170       0.0017783    3
   3.47        42170       0.0017783    3
   ...
```

The four columns are: error rate, $C$ or $\nu$, $\gamma$, and iteration number, respectively. The file records all parameter pairs examined by the program. To find the optimal parameters, sort the file as:

```
% sort -g pcp.svp
```

Simplex search starts with iteration 0 which serves to initialize vertices of the initial simplex (see [[5]] for details). Note that, unlike Grid Search, in Simplex algorithm a single iteration may involve multiple parameter pairs and function evaluations, hence multiple lines with identical iteration number in the above file segment. In Grid Search, iterations start with 1, and each iteration occupies exactly one line.

In our experience, Simplex is considerably faster and often reaches better optimum than Grid Search. On rare occasions, however, it gets trapped in a local minimum. A recommended procedure is to run Grid Search if Simplex produces an unexpectedly poor value of the performance criterion.

At the end of the parameter search, if the test set is defined, PCP will automatically build a classifier using entire training dataset and the optimal parameters, and apply it to the test dataset. Thus, a single function implements virtually entire Statistical Learning Paradigm of Section 5.1.

Last, but not least, model selection, as implemented in PCP, offers dramatic performance advantages when used with dimensionality reduction. It is easy to demonstrate that if a classifier being developed includes dimensionality reduction of a high-dimensional dataset (a routine requirement for such datasets), manual selection of SVM parameters is not only tedious, but virtually impossible to complete in reasonable time. This is further elaborated in Section 6.3.

## 5.9 Bayes Error Estimation

`Bayes Error Estimation` differs from the rest of the supervised classification algorithms. The menu is shown in Fig. 5.9, and is accessed by pressing **a** key in the **Pattern Classification** menu.

For an introduction to the notion of Bayes error rate in pattern classification, see any standard text ([20], [26]). In summary, it is the error rate achieved by the optimal pattern classification algorithm known as the *Bayes rule*. It is not attainable in practice, since the Bayes rule requires knowledge of class-conditional probability distributions, never available in real-life situations.

The theory of this elaborate algorithm has been developed by K. Fukunaga and collaborators ([3], [4]). The algorithm has not gained wide acceptance due to the following properties of the method:

- the algorithm uses Mahalanobis distance (5.3) for the $k$-NN method which lies at the heart of the algorithm; this in turn requires sufficiently large datasets in order to calculate reliable estimates of the covariance matrices.

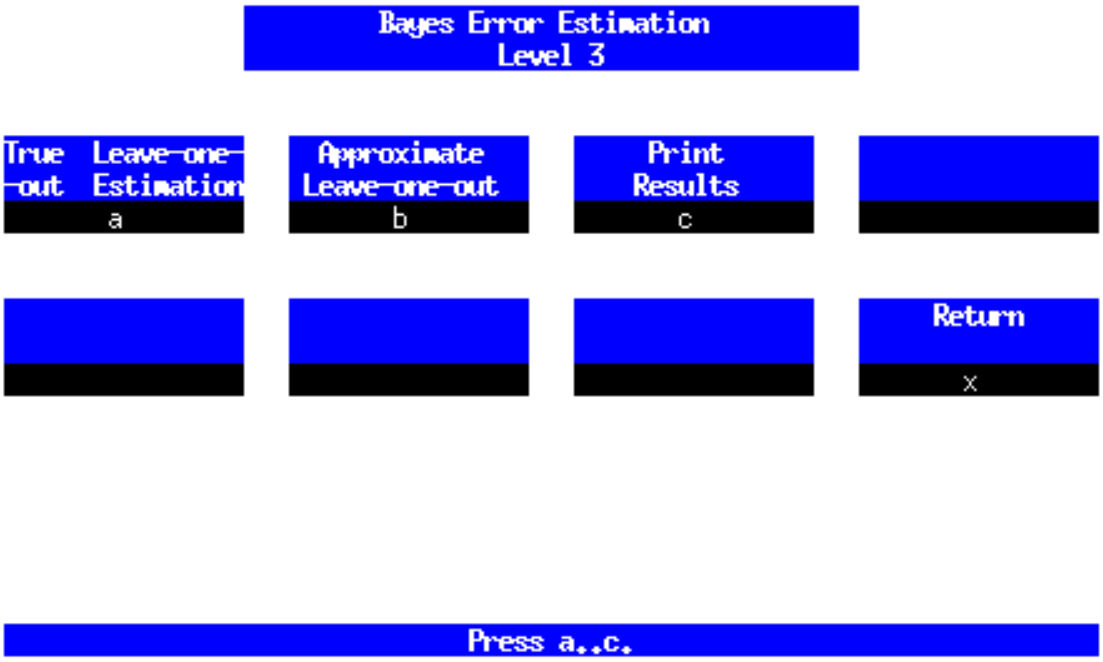


Figure 5.9: Bayes Error Estimation menu

Mahalanobis distance between a vector $X$, and vector $Y$ from class $i$ is defined as:

$$d(X,Y) = \sqrt{(X-Y)^T \Sigma_i^{-1} (X-Y)} \tag{5.3}$$

where $\Sigma_i$ is covariance matrix of class $i$ (the class to which $Y$ belongs). Clearly, the use of this distance measure requires that a covariance matrix estimate of each class be non-singular.

- computational complexity: error estimation for sizable datasets (thousands of vectors with hundreds of features) can easily take days on powerful computers
- difficult software implementation and testing

Nonetheless, in cases where it can be applied effectively, the Bayes error estimation can be exceptionally useful tool, since it may provide a solid estimate of the theoretical limit of the classification accuracy for the problem at hand, independent of the learning algorithm. If this quantity does not satisfy the requirements of the problem statement, the researcher/system designer has to collect additional features (measurements).

The menu contains two basic functions: estimation of Bayes error, and display of results. The third function, **Approximate Leave-one-out**, implements an approximation of the method which is generally interesting for diagnostic purposes. To estimate Bayes error using the training dataset, press **a** key and respond to the two questions:

```
Enter index of the first nearest neighbor [1]:

Enter index of the last nearest neighbor [   49]:
10
```

The numbers define the range of values for $k$ in the nearest neighbor procedures on which the algorithm is based. The results are stored in file `pcp.bee`.

To display the results, press **c**. A typical output for IRIS data may look like:

```
+-----------------------------------------------------------------------------+
|                                                                             |
|               B A Y E S   E R R O R   E S T I M A T I O N                   |
|                                                                             |
+-----------------------------------------------------------------------------+
|      k     | Leave-one-out error estimate  | Resubstitution error estimate |
+-----------------------------------------------------------------------------+
|          1 |                          4.00 |                          0.00 |
|          2 |                          5.33 |                          2.00 |
|          3 |                          2.67 |                          1.33 |
|          4 |                          4.00 |                          1.33 |
|          5 |                          2.00 |                          1.33 |
|          6 |                          2.67 |                          1.33 |
|          7 |                          2.00 |                          1.33 |
|          8 |                          2.00 |                          0.67 |
```

```
|          9 |                          2.67 |                          0.67 |
|         10 |                          2.00 |                          1.33 |
+-------------------------------------------------------------------------+
```

**leave-one-out** and **resubstitution** refer to the upper and lower limit of the Bayes error estimate for a given $k$.

Bayes error estimation is useful if the gap between the reported upper and lower bounds is relatively narrow. When number of samples is small compared with the input dimension, the bounds will be loose, and the resulting gap too wide to be useful.

In the above case, the bounds are very close to what is believed to be the best achievable error rate for this widely-analyzed dataset.

## 5.10 Measures of Classification Performance in PCP

In this section we describe the performance measures used to report supervised classification results in PCP.[3]

PCP distinguishes between two-class and multiple-class classification problems. For multi-class problems, overall and class-conditional error rates (defined shortly) are calculated and displayed; for two-class problems, sensitivity, specificity, positive and negative predictive value are also computed, saved and displayed.

For multi-class problems, class-conditional error rate for class $i$, $e_i$, is defined as the number of class $i$ samples assigned to another class ($E_i$), divided by the cardinality of the training data set for class $i$ ($n_i$). The overall error rate $e$ (also known as misclassification rate) equals the number of incorrectly assigned samples divided by the total number of samples used for learning ($n$). Hence we have:

$$e_i = \frac{E_i}{n_i} \tag{5.4}$$

$$e = \frac{\sum_{i=1}^{c} E_i}{n}, \quad n = \sum_{i=1}^{c} n_i \tag{5.5}$$

The sensitivity, specificity, positive and negative predictive value error measures are defined for two-class pattern classification problems, and assume that one of the classes is labeled *positive* ($P$), while the other is labeled *negative* ($N$) [23]. These terms are widely used in medical diagnostics, where such labels naturally correspond to diseased and healthy classification subjects (patients), respectively[4]; however the concepts apply to all two-class pattern classification problems. In PCP, it is assumed that class 1 has the positive samples, and class 2 has negative samples.

Introducing the notions of *true positive rate* $t_P$ as the fraction of correctly assigned class $P$ samples, *false negative rate* as the fraction of incorrectly assigned class $P$ samples, *true negative rate* $t_N$ as the fraction of correctly assigned class $N$ samples and *false positive rate* $f_P$ as the fraction of incorrectly assigned class $N$ samples, sensitivity and specificity can be defined as:

[3] All performance measures described in this section are fractions in the interval $[0, 1]$. PCP displays and saves the equivalent percentage values, i.e., the measures multiplied by 100.

[4] The term *positive* for an ill patient comes from the fact that the individual tested positive for the disease.

$$\text{sens} := t_P = \frac{T_P}{n_P} = 1 - f_N, \quad n_P := n_1 \tag{5.6}$$

$$\text{spec} := 1 - f_P = 1 - \frac{F_P}{n_N} = t_N, \quad n_N := n_2 \tag{5.7}$$

Using the previously defined error rates and noting that $F_P = E_N$ and $F_N = E_P$, these quantities can also be expressed as:

$$\text{sens} = \frac{n_P - F_N}{n_P} = 1 - \frac{F_N}{n_P} = 1 - e_P, \quad e_P := e_1 \tag{5.8}$$

$$\text{spec} = 1 - \frac{E_N}{n_N} = 1 - e_N, \quad e_N := e_2 \tag{5.9}$$

and, conversely,

$$e := \frac{E_P + E_N}{n} = \frac{(1 - \text{sens})n_P + (1 - \text{spec})n_N}{n_P + n_N} \tag{5.10}$$

Positive predictive value (PPV) is defined as the number of true positives divided by the sum of true positives and false positives:

$$\text{ppv} = \frac{T_P}{T_P + F_P} = \frac{1}{1 + \frac{n_N}{n_P}\frac{1-\text{spec}}{\text{sens}}} \tag{5.11}$$

Negative predictive value (NPV) is defined as the number of true negatives divided by the sum of true negatives and false negatives:

$$\text{ppv} = \frac{T_N}{T_N + F_N} = \frac{1}{1 + \frac{n_P}{n_N}\frac{1-\text{sens}}{\text{spec}}} \tag{5.12}$$

PPV and NPV quantify what fraction of the positive/negative predictions are correct[5], or, in other words, what is the confidence of positive/negative prediction. Ideally, of course, sensitivity, specificity, positive and negative predictive value should all equal 1. However, depending on the problem domain, much lower values may be acceptable [27].

In some applications PPV is known as *precision*, and sensitivity is known as *recall*.

These concepts are illustrated in the *confusion matrix* shown in Table 5.1.

Table 5.1: Confusion matrix for two-class problems.

| | | predicted | | | |
|---|---|---|---|---|---|
| | | class 1 *(positive)* | class 2 *(negative)* | $\Sigma$ | |
| actual | class 1 | $T_P$ | $F_N$ | $n_P$ | sens $= \frac{T_P}{n_P}$ |
| | class 2 | $F_P$ | $T_N$ | $n_N$ | spec $= \frac{T_N}{n_N}$ |
| | | ppv $= \frac{T_P}{T_P+F_P}$ | npv $= \frac{T_N}{T_N+F_N}$ | | |

PCP automatically computes and displays sensitivity, specificity, positive and negative predictive value when the number of classes is two.

[5] If all samples were assigned to class N (i.e., $T_P + F_P$ is 0), PPV is undefined. PCP sets it to 0. Likewise for NPV.

# Chapter 6

# Dimensionality Reduction

Dimensionality reduction is the process of transforming input data into a lower-dimensional space where a more efficient classifier can be built. In PCP, the dimensionality reduction methods are divided in two groups: *feature extraction*, which map input data using linear transformation (i.e., a transformation matrix), and *feature selection*, which performs the mapping by selecting a subset of the original features.

Feature extraction methods supported in PCP are Fischer's linear discriminant fuction, Principal Component Analysis, Singular Value Decomposition and EMAP algorithm. The feature subset selection methods implemented in PCP are feature ranking, forward selection and backward elimination.

All dimensionality reduction methods can be used in a stand-alone mode (through the dedicated menus), or as a part of the cross-validation analyses. The stand-alone mode may be used to explore the relationships among input features, and is described in Sections 6.1 and 6.2. However, it is not particularly well suited for classifier design since it may produce heavily biased error estimates. For classifier design, cross-validation should be applied to the combination of dimensionality reduction and classifier, as explained in Section 6.3.

## 6.1 Feature Extraction

Given input vectors $X$ of dimension $d$, feature extraction seeks to find an optimal $m \times d$ transformation matrix $W$ which maps $X$ into the $m$-dimensional space of vectors $Y$, where $m < d$:

$$Y = W_{\texttt{opt}} X, \quad W_{\texttt{opt}} = \arg\max_{W} J(W) \tag{6.1}$$

where $J(W)$ is the criterion used to evaluate the discriminatory potential of the selected subset of $m$ features.

Feature extraction methods differ in the choice of criterion $J(W)$. One popular criterion is the ratio of *between-class* and *within-class* scatter matrices $S_B$ and $S_W$ of the transformed vectors. Following [20], Section 3.8.3 *Multiple Discriminant Analysis*, the measure is defined as:

$$J(W) = \left| S_W^{-1} S_B \right| \tag{6.2}$$

This quantity, in effect, measures the relative average distance among class centers in the mapped space. In order for the mapping to ensure that an accurate classifier can be built in

the space of vectors $Y$, the average distance should be maximized. It can be shown [20] that the matrix $W$ which maximizes $J(W)$ of equation (6.2) can be computed by eigenanalysis of the scatter matrices in the input space, and the resulting transformation is known as *Fischer's linear discriminant*.

Another well-known approach to the problem is *Principal Component Analysis*, which identifies the mapping which best represents the original data in the least-squares sense. The resulting transformation matrix is again computed by eigenanalysis, this time of the scatter matrix $S_B$.

Finally, the transformation called **EMAP** [9] attempts to use a theoretically near-optimal criterion $J(W)$. Instead of applying heuristic criteria such as (6.2), the algorithm directly uses the criterion which lies at the heart of the pattern classification - probability of misclassification, as measured by the Bayes error estimate (see Section 5.9). This quantity is a piecewise-constant function of the transformation matrix parameters, and is thus generally not considered suitable for optimization. [9] however demonstrates that by using a non-gradient algorithm known as *Simplex* [5], effective minimization of such functions can be achieved.

The menu which implements these computations is shown in Fig. 6.1.

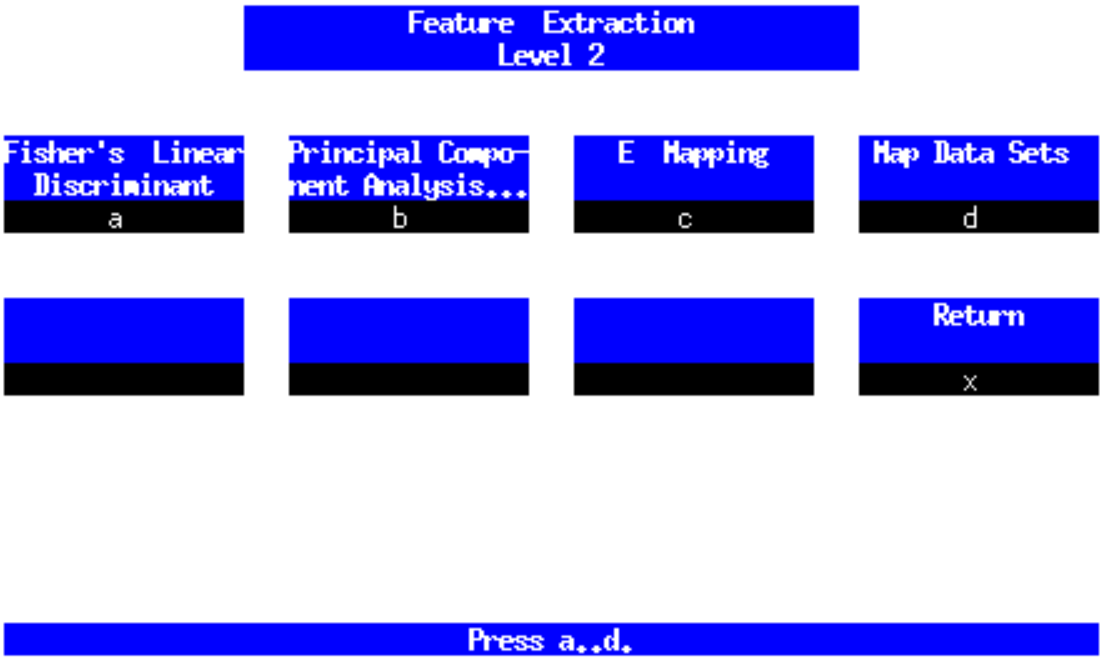


Figure 6.1: Feature Extraction menu

To calculate Fisher's linear discriminant, press **a** in the menu:

```
Enter dimension of transformed space: (1..2) [    2]:

Enter linear transformation output file name [pcp.lin]:
```

The output file has $m$ rows and $d$ columns, where each row corresponds to a feature in the output space.

The principal component analysis (PCA) has a submenu with two options, PCA and Singular Value Decomposition (SVD). SVD implementation follows [18], and is used in case the number of data points is less than the dimension of the input space. PCA employs identical dialog as the Fisher's linear discriminant. However, SVD adds one more input option. The transformation can be computed using both training and test sets, or just training set. The former approach is statistically legitimate since no class information is used in the process of computing SVD, and in our experience it can significantly improve performance.

A SVD dialog proceeds as follows:

```
Use training dataset (1) or both (0) [0]:

Enter dimension of transformed space: (1..1994) [   72]:

Enter linear transformation output file name [pcp.lin]:
```

SVD and PCA both produce output file in the same format as the Fisher's linear discriminant.

EMAP algorithm is launched by pressing **c** in the **Feature Extraction** menu:

```
Enter dimension of transformed space (1..4):
2
Enter linear transformation output file name [pcp.emp]:

Enter kmin [1]:

Enter kmax (1..49) [   24]:

Enter seed for pseudo-random number generator [    1]:

Enter maximum number of iterations [1000]:

Starting point: Fisher (2), PCA (3), random (12), or file (13) [3]:
```

The algorithm uses the Bayes error estimate (see Section 5.9) as the optimization (minimization) criterion. The `kmin`, `kmax` values is the range of $k$-nearest neighbors used for the error estimation. seed is used to initialize the pseudo-random number generator used to resolve ties. The maximum number of iterations controls the Simplex optimization algorithm. Starting point determines the initial vertex which defines the simplex [5]. It is recommended that the algorithm should start from a point defined by one of the other algorithms, for example PCA or Fisher's linear discriminant (as offered in the above dialog), however pseudo-random starting point, or starting point defined in a file are also possible.

EMAP can realistically be used only on smaller datasets (smaller in the number of data points *and* dimension of input space); its' memory complexity is $O((m \times d)^2 * M)$, where $M$ is memory required to store the nearest neighbors involved in Bayes error estimation. The computational complexity is very high, given that calculating *each* of the $m \times d$ vertices of the simplex involves a Bayes error estimate in the transformed space, already a demanding task. However, the method does demonstrate that it is technically feasible to use the empirical error rate as a learning criterion.

Feature extraction methods described above create a linear mapping and store it in a file. The feature extraction menu shown in Fig. 6.1 provides an option to map (transform) the current datasets (training and/or test) into low-dimensional space using the transformation stored in the file. To map dataset(s), press **Map Datasets** (**d** key):

```
Enter transformation matrix file name [pcp.lin]:

Replace current dataset(s) (1) or not (0) [0]:
1
```

Upon completion of the dialog, the currently defined dataset(s) will be mapped into low-dimensional space, using the specified mapping file. The mapped data is stored in files with string `_map` inserted in file name just before the extension suffix. For example, if training data for class 2 is stored in file `training_2.dat`, the mapped data will be written to `training_2_map.dat`.

The second question gives an option to replace the current training dataset(s). In case a replacement is chosen (answer 1 to the question), these files automatically become the training and/or test datasets.

## 6.2 Feature Selection

The goal of feature selection is to choose an optimal subset (according to some criterion) of cardinality $m$ among the $d$ input features. The benefit of feature selection is that, once a reliable subset has been identified, only $m$ measurements need to be collected to predict class labels for new samples. In contrast, feature extraction always requires the full complement of $d$ features. This difference may be a significant factor if obtaining the measurements is costly.

The definition of a feature selection method requires specification of the following two components:

- search algorithm determines which subsets are evaluated
- the definition of a criterion for evaluating the fitness of each subset

The optimal subset maximizes the value of the chosen criterion.

The search problem is clearly NP-complete, since evaluation of all possible subsets of cardinality $m$ requires $\binom{d}{m}$ evaluations of the criterion, if $m$ is known. If $m$ is unknown, which is normally the case, the number of evaluations to determine the optimal $m$ *and* the optimal subset of size $m$ jumps to $2^d$. Both numbers are astronomically large for most realistic data sets.

Given that finding the optimal solution is generally not feasible, several search heuristics exist. The PCP **Feature Selection** menu implements the following established algorithms [26], [31] (the descriptions below assume that the feature selection criterion is to be maximized):

- *feature ranking*. Each feature is evaluated individually according to the chosen criterion, and the values are then sorted; the $m$ features with the best value of the criterion are retained for classification. The method is conceptually simple and computationally attractive, but is only optimal in the unlikely case of independent features.

- *forward selection*. Start with an empty subset and add one feature at a time. The added feature is one which optimizes the value of the criterion for the newly formed subset. The process stops when $m$ features have been addeed.

- *backward elimination*. Start with all $d$ features and remove one feature at a time. The subtracted feature minimally reduces the value of the feature selection criterion for the new subset. The process stops when $m$ features remain.

In addition to solving the search problem, feature selection requires the choice of a criterion for evaluating the fitness of a particular feature subset. A variety of criteria have been proposed and used with these and other feature selection algorithms. When considering the criteria, a distinction must be made between feature ranking and feature subset selection because individual feature ranking can use specialized criteria not available for more than one feature.

PCP implements the following criteria:

- feature ranking:
  - Euclidean distance between vector of the feature values and corresponding integer class labels
  - Pearson correlation coefficient. The measure computes the Pearson correlation coefficient between two random variables: the feature being evaluated, and class label.
  - 1-NN criterion (cross-validation error rate of a nearest neighbor classifier constructed using the feature being evaluated)
  - Bayes criterion (Bayes error estimate for the feature being evaluated, see Section 5.9)
  - Golub criterion (defined for two classes [14])

- feature subset selection:
  - 1-NN criterion (cross-validation error rate of a nearest neighbor classifier constructed using the subset of features being evaluated)
  - Bayes criterion (Bayes error estimate (see Section 5.9) for the subset of features being evaluated)
  - inter-intra distance, defined as trace $\left\{S_W^{-1} S_B\right\}$

**Feature Selection** menu is shown in Fig. 6.2.

The first menu option **Select Feature Subset** is used to select optimal subset of features, specifying a feature evaluation criterion and a search algorithm. Pressing **a** triggers the following dialog (the example uses the two-class Leukemia dataset described in Chapter 7; in case of a dataset with more than two classes, Golub criterion would not be offered):

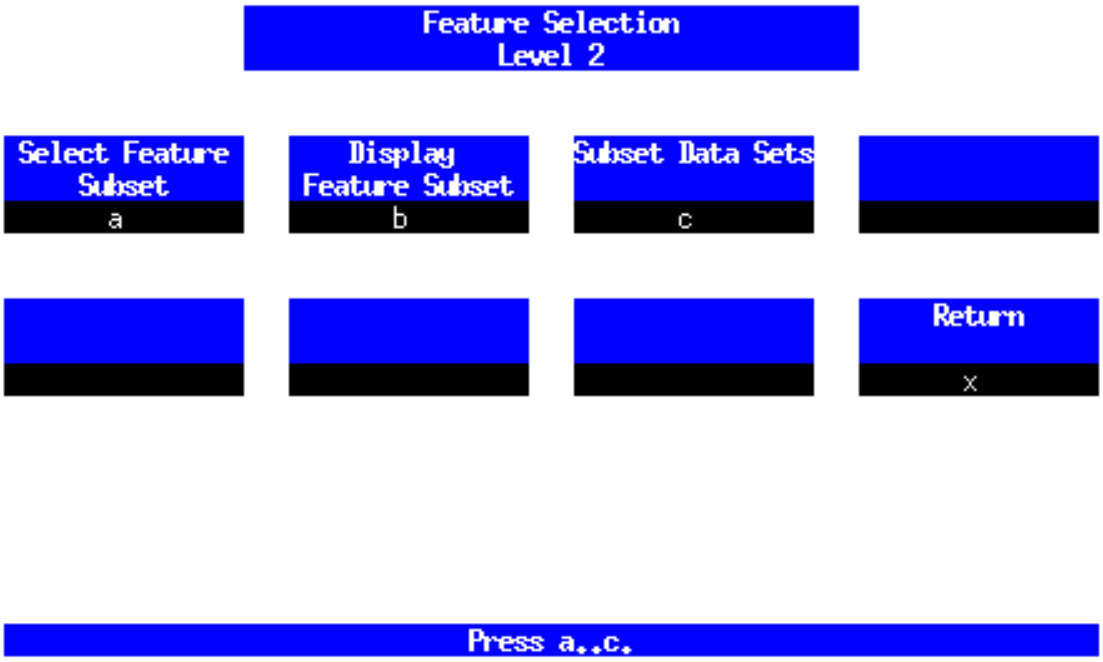


Figure 6.2: Feature Selection menu

```
Enter feature selection method: feature ranking (5), forward selection (6), or
 backward elimination (7) [5]:

Enter feature rank output file name [pcp.rnk]:

Use Euclidean (1), Pearson (2), Golub (3), 1-NN (4) or Bayes (5) criterion [2]
:
```

In this case, the default method (feature ranking) has been chosen. The ranking of features is stored in the output file in the following format:

```
971     0.808293        U50136_rna1_at
557     0.798445        M23197_at
1657    0.782888        U22376_cds2_s_at
1471    0.743460        Y12670_at
631     0.729944        M63138_at
...
```

The first column is the original input feature index, equal to the column number of the corresponding feature in the input data files. The second column is value of the criterion for the corresponding feature, sorted in the decreasing order. The optional third column has the feature name if the input files have named columns.

Option **Display Feature Subset** displays the user-defined number of best features:

```
Enter feature rank file name [pcp.rnk]:

Enter number of features to select [5]:
```

In case of feature subset selection using a forward selection method, the PCP interaction proceeds as follows:

```
Enter feature selection method: feature ranking (5), forward selection (6), or
 backward elimination (7) [5]:
6
Enter feature subset output file name [pcp.set]:

Use 1-NN (1), inter-intra distance (2) or Bayes criterion (3) [1]:

Enter number of features to select (1..1994) [2]:
5
Added feature    1674 (   1/   5); criterion value: 92.105263
Added feature     697 (   2/   5); criterion value: 97.368421
Added feature    1980 (   3/   5); criterion value: 100.000000
Added feature    1991 (   4/   5); criterion value: 100.000000
...
```

The display shows the index of feature being added and the value of the criterion of the resulting subset (after the feature has been added). In this example, the 1-NN classifier accuracy reaches 100% already with three features. The default feature subset file name is `pcp.set` and the file format is:

```
527     100.000000      M16038_at
1924    100.000000      M31523_at
1983    100.000000      X01677_f_at
1989    100.000000      L78833_cds4_at
1994    100.000000      X83863_at
...
```

The first and optional third column list the features which comprise the optimal subset. The second column is the value of the criterion for the subset. The value is the same in all rows, since it is associated with the subset, not with any particular feature.

The **Subset Datasets** function extracts the best features from the current datasets and saves them in output files in the standard PCP format. These output files could then be further analyzed in another PCP session. Optionally, this function replaces the current datasets (similar to **Map Data Sets** in **Feature Extraction**), in which case the selected subsets become the current training and test datasets. If this option is chosen, the additional analyses can continue in the same PCP session. The output file names have string `_sel` inserted before the extension in the input file names. For example, given an input file `iris.dat`, the selected subset of features would be stored in a file `iris_sel.dat`.

In practice, feature ranking performs well on many datasets, even though it ignores feature correlations. This may perhaps be due to well-established criteria (such as the Pearson correlation coefficient) for evaluating individual features, rather than to the power of the method itself.

## 6.3 Dimensionality Reduction in Cross-validation

In many applications, the learning machine consists of two stages: dimensionality reduction and classification. The structure is shown in Fig. 6.3.

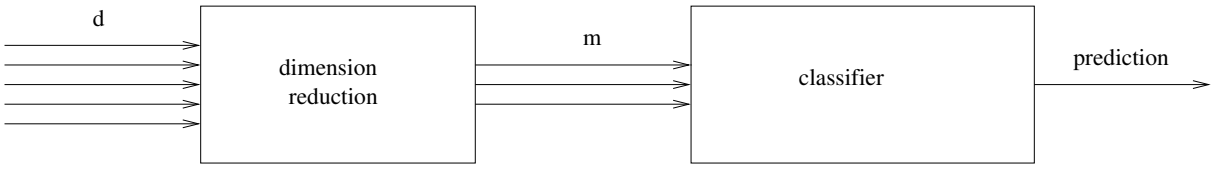


Figure 6.3: The combined learning machine consisting of dimensionality reduction and classification steps. $d$ is dimensionality of input vectors, $m$ the dimension of transformed vectors. Multiple arrows are meant to convey the effect of reducing the dimensionality of the input data.

This section describes how PCP incorporates dimensionality reduction in the design and error estimation of classifiers.

Dimensionality reduction as a stand-alone process independent of the design of the classifier stage leads to over-fitting. The reason is as follows. Independent feature subset selection/mapping

is performed using the training dataset. If the resulting features are then used to cross-validate a classifier (which is again done on the training dataset), the same data would have been used for training (the dimensionality reduction stage), and testing (the cross-validation of the classifier). This is over-fitting, and may produce heavily biased error estimates.

In contrast, PCP supports building dimensionality reduction and classifier stages together, and simultaneously estimating the error rate of both stages. In this scenario, each cross-validation *learning* dataset will first be used to compute the dimensionality reduction parameters, and then the computed transformation will be applied to the learning and *validation* datasets. This is followed by building a classifier using the mapped learning dataset, and testing it using the mapped validation dataset. Thus, validation subset is never used for either dimensionality reduction nor classifier learning, but exclusively for testing. This is a statistically rigorous approach to machine learning which avoids over-fitting.

As an example of the joint design of dimensionality reduction and classifier stages, the following dialog reflects calculation of the cross-validation error estimate of a combined classifier consisting of the two stages (the example uses the Leukemia dataset described in 7). The dimensionality reduction stage is Singular Value Decomposition feature extraction method, whereas the classifier is SVM in the in the 33-dimensional transformed space. To perform the calculation, start **Pattern Classification → Support Vector Machine → Cross-validation** (as usual, the empty response lines indicate acceptance of the default parameter value):

```
Enter SVM type (1: NU-SVM; 2: C-SVM) [2]:

Enter kernel type (1: linear; 2: polynomial; 3: RBF; 4: sigmoid) [3]:

Enter gamma [    0.010000]:

Enter tolerance of termination criterion [0.00100]:

Enter cost parameter C [1000.00]:

Enter number of cross-validation subsets (2..11) [   10]:

Enter SVM cross-validation file name [pcp.xsv]:

Enter seed for pseudo-random number generator [    1]:

Enter number of experiments [1]:

Dimension reduction method - none (0), Golub (1), SVD (3), EMAP (6) [0]:
3
Enter dimension of transformed space (1..33) [33]:

Use raw (0) or normalized data (1) [0]:
```

Since SVD has been chosen as the dimension reduction method, the dialog requests input of

additional parameter, the dimension of the transformed space. The SVM classifier is constructed in the lower-dimensional space.

The next example, using the same dataset, illustrates calculation of cross-validation error rate of a feature selection using feature ranking, followed by MLP classifier in the 10-dimensional space. To perform the calculation, start **Pattern Classification → Multi-Layer Perceptron → Cross-validation**:

```
Enter MLP cross-validation file name [pcp.xmp]:

Enter number of hidden layers (>= 1) [1]:

Number of inputs is  1994.
Number of output nodes is     2.

Enter number of nodes in hidden layer     1:
5
Enter seed for pseudo-random number generator [    1]:

Enter amplitude of initial weights [ 1.00]:

Enter optimization method - conj. grad (1), grad. descent (2) [    1]:

Enter number of iterations (-1 for no iterations) [100]:

Enter number of cross-validation subsets (2..11) [   10]:

Enter number of experiments [1]:

Use raw (0) or normalized data (1) [0]:

Choose dim. reduction method: none (0), FLD (2), PCA (3), EMAP (4), feature ra
nking (5), forward selection (6), or backward elimination (7) [0]:
5
Use Euclidean (1), Pearson (2), Golub (3), 1-NN (4) or Bayes (5) criterion [2]
:

Enter dimension of transformed space (1..1994) [1994]:
10
```

The specification of feature ranking as a dimensionality reduction stage triggered the input of two additional parameters: the number of features to select, and the choice of criterion for evaluating the features. The MLP is constructed in the space of the selected features.

This functionality performs cross-validation error estimation of the combined dimensionality reduction/classifier. Once the estimate is obtained, it can be used for model selection, as described in Section 6.4.

## 6.4 Dimensionality Reduction in Model Selection

Statistically valid model selection (see Section 5.8.1) becomes a computationally challenging process in the presence of dimensionality reduction, particularly for methods (such as SVM) with multiple continuous classifier parameters. It is easy to recognize this by considering the following:

- model selection involves finding the parameter values which give the lowest value of the chosen criterion (Section 5.1)
- frequently, error rate is used as a criterion, and cross-validation is used to estimate the error rate
- each cross-validation experiment has to perform dimensionality reduction followed by learning of the classifier, for each cross-validation subset (Section 6.3)

For example, for 10-fold cross-validation, this means that for each parameter value combination, the dimensionality reduction has to be performed 10 times for each possible combination of parameter values. This has to be multiplied by the number of parameter value combinations to be evaluated. For SVM, the number of $(C/\nu, \gamma)$ pairs can be in the hundreds for grid search, therefore the total number of dimensionality reduction computations easily reaches thousands.[1] Given the complexity of dimensionality reduction algorithms themselves, the total computational burden may quickly become impractical even for small datasets.

PCP solves this problem by an efficient design of model selection incorporating the dimensionality reduction. The implementation is currently available for Support Vector Machine algorithm, where the issue is the most challenging. The implementation is based on the idea that the dimensionality reduction computations are exactly the same for each pair $(C/\nu, \gamma)$ of SVM parameter values. Thus, it can be performed once, and then reused for each new combination of parameter values.[2] As a result, the effective required number of dimensionality reduction calculations equals the number of cross-validation subsets multiplied by the number of cross-validation experiments, a much more manageable complexity. This enables the rigorous use of dimensionality reduction algorithms within SVM model selection.

The dimensionality reduction is performed for each cross-validation subset, therefore the feature subsets (in case of feature selection dimensionality reduction) will differ slightly between the subsets. PCP stores the chosen feature subsets for each cross-validation subset in file `pcp.xsf`. A sample file recording selection of feature subsets of cardinality five is shown in Table 6.1. The file can be used to analyze which features get selected most frequently.

[1] In order to reduce bias of the cross-validation error estimates, each cross-validation experiment may optionally be performed multiple times (the `Number of experiments` parameter in cross-validation fuctionality). This increases the number of dimensionality reduction computations by at least another order of magnitude.

[2] The suggestion is due to Sasha Jaksic of San Francisco State University.

Table 6.1: Sample `pcp.xsf` file. Each line lists feature subset selected in a particular step of the parameter selection. For example, line 1 has the feature subset selected in experiment 1, cross-validation subset 1; line 2 has the feature subset select in experiment 1, cross-validation subset 2, etc. The feature names correspond to the Leukemia sample dataset 7.

| | | | | |
|---|---|---|---|---|
| U50136_rna1_at | U46751_at | M23197_at | U22376_cds2_s_at | Y12670_at |
| U50136_rna1_at | M23197_at | U22376_cds2_s_at | M92287_at | M55150_at |
| M23197_at | U50136_rna1_at | U22376_cds2_s_at | Y12670_at | D26308_at |
| U50136_rna1_at | M23197_at | U22376_cds2_s_at | M63138_at | Y12670_at |
| ... | | | | |

# Chapter 7

# Sample Datasets

PCP is accompanied by a set of data files and batch files used to demonstrate the functionality. Table 7.1 lists the names and properties of the datasets.

Table 7.1: Sample datasets. The column *Batch file* has the name of the accompanying batch file which can be used to load the data into PCP.

| *Name* | *Classes* | *Features* | *Training samples* | *Test samples* | *Batch file* | *Reference* |
|---|---|---|---|---|---|---|
| IRIS | 3 | 4 | 150 | 0 | `iris.bat` | [1] |
| Leukemia | 2 | 1994 | 38 | 34 | `al.bat` | [14] |
| SRBCT | 4 | 2308 | 63 | 25 | `srbct.bat` | [21] |
| LANDSAT | 6 | 36 | 4856 | 2000 | `landsat.bat` | [11] |

The following batch file, `srbct_svm.bat` serves as an example of the Support Vector Machine classifier learning using the SRBCT data set:

```
b                                   pattern classification
f                                   Support Vector Machine menu
a                                   learning
2                                   SVM type: C-SVM
2                                   kernel type: polynomial
1000                                cost
0                                   default class costs
3                                   polynomial degree
0.01                                gamma
0.0                                 constant term in kernel function
pcp.svm
x                                   return
x                                   return
x                                   return
```

To run the experiment, first load the dataset, then run the learning:

```
% pcp -b srbct.bat
% pcp -b srbct_svm.bat
```

After the learning is finished and control returns to the command prompt, start PCP again, and go to **Support Vector Machine**. Press **c** to view the SVM predictions, and press **Enter** twice in response to the questions. You should see the classifier predictions as:

```
+-------------------------------------------------------------------------+
|        Class         |  Actual/predicted card.  |        Error rate        |
+-------------------------------------------------------------------------+
|                      |          25/25           |    8.00% (     2/    25) |
|   1/ews_test         |           7/8            |    0.00% (     0/     7) |
|   2/rms_test         |           8/8            |   12.50% (     1/     8) |
|   3/nb_test          |           7/6            |   14.29% (     1/     7) |
|   4/bl_test          |           3/3            |    0.00% (     0/     3) |
+-------------------------------------------------------------------------+
| Vector |          Actual  class          |         SVM  prediction         |
+-------------------------------------------------------------------------+
|     13 | rms_test                        | ews_test                        |
|     22 | nb_test                         | rms_test                        |
+-------------------------------------------------------------------------+
```

The lower part of the table lists the samples misclassified by the SVM. To view the SVM parameters, type or print file `pcp.svm`. In this particular case, 3-rd degree polynomial kernel has been used, with $\gamma = 0.01$ and $C = 1000$.

# Chapter 8

# Bugs, Issues and Enhancements

This chapter lists known bugs, issues and possible enhancements as of the current release.

## 8.1 Bugs

- Support for multiple hidden layers in multi-layer perceptrons has not been thoroughly tested.

## 8.2 Issues

- There is an issue involving KDE `Konsole` terminal emulator. PCP uses inverse video to display most results. KDE `Konsole`, for some reason, renders entire screen in inverse video, once the first screenful of content has scrolled off the window. This problem is not present in other VT100 terminal emulators, for example GNOME terminal emulator, and the stock `xterm` program. I would consider this a `Konsole` issue, especially given that it does not perform as well as the other emulators on the VT100 compatibility test program (`http://freshmeat.net/projects/vttest`). In any case, the defect is not serious, the results are perfectly readable, but may look slightly different than in other terminal emulators.

  WORKAROUND: I recommend using `xterm` terminal emulator, or switching to GNOME.

- Same problem plagues Windows Command Prompt terminal. I recommend using Cygwin `xterm` instead (you have to download Cygwin anyway, in order to run Windows version of PCP).

- Linear discriminant classifier performance for multi-class case is poor. This issue is under investigation.

- The MLP conjugate gradient learning algorithm frequently reports a warning message `Line search failure in optimization procedure`, and yet practice shows that it is often able to find a reasonable minimum of the criterion function. In my experience, in case the learning produces satisfactory minimum, the message can be ignored; if it doesn't, retry with a different set of learning parameters.

## 8.3 Enhancements

These are the potential enhancements being considered for future releases of PCP (in no particular order). Implementation depends on user feedback and available resources.

- Add GUI (Graphical User Interface).
- Expand Model Selection functionality to other SVM kernels.
- Add graphical output. Presently all results are displayed in the terminal window, and saved in files.
- Implement Decision Tree classification algorithm.
- Add Help.
- Build PCL (Pattern Classification Library), a library of pattern classification functions based on the PCP functionality.

# Acknowledgements

This work has its origins in a project named PARIS, designed with Prof. Milan Milosavljević during my graduate studies at the School of Electrical Engineering, University of Belgrade, Serbia. The author is grateful for Prof. Milosavljević's support throughout the years. Additionally, the author wishes to acknowledge inspiring collaborations with Dr. Milan Marković, Prof. Srdjan Stanković and Prof. Mladen Veinović.

Sasha Jaksic of San Francisco State University contributed to the feature selection code.

The author acknowledges and wishes to thank authors of the following open-source software libraries used in PCP:

- Support Vector Machine library LIBSVM by Chih-Chung Chang and Chih-Jen Lin [30]
- hash code from Kazlib library by Kaz Kylheku [33]
- LAPACK library [16]
- FORTRAN package for unconstrained function optimization CG+ (version 1.1), written by G. Liu, J. Nocedal and R. Waltz [6] (PCP uses C version of the package, translated by the author)
- sorting functions by Ariel Faigon [34]

# Bibliography


[1] T. W. Anderson, *An Introduction to Multivariate Statistical Analysis*. New York: Wiley, 1958.

[2] M. R. Anderberg, *Cluster Analysis for Applications*. Academic Press, 1973.

[3] K. Fukunaga and D. M. Hummels, "Bayes Error Estimation Using Parzen and k-NN Procedures," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-9, pp. 634-643, 1987.

[4] K. Fukunaga, *Introduction to Statistical Pattern Recognition, Second Edition*. Boston: Academic Press, 1990.

[5] W. H. Press, P. Flannery, A. Teukolsky, W. T. Vetterling, *Numerical Recipes, Second Edition*. Cambridge, UK: Cambridge University Press, 1992.

[6] J. C. Gilbert and J. Nocedal, "Global Convergence Properties of Conjugate Gradient Methods for Optimization," *SIAM J. on Optimization*, vol. 2, No. 1, 1992.

[7] V. Cherkassky, F. Mulier, *Learning From Data: Concepts, Theory and Methods*. New York: John Wiley & Sons, 1998.

[8] Friedman, J. H., "An overview of predictive learning and function approximation," in V. Cherkassky, J. H. Friedman, and H. Wechsler (eds.), *From Statistics to Neural Networks*, NATO ASI Series F, 136. New York: Springer Verlag, 1994.

[9] Lj. J. Buturović, "Towards Bayes-Optimal Linear Dimension Reduction," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-16, No. 4, pp. 420-424, April 1994.

[10] V. N. Vapnik, *The Nature of Statistical Learning Theory, Second Edition*. New York: Springer-Verlag, Inc., 1995.

[11] Blake, C.L., Merz, C.J, UCI Repository of machine learning databases [http://www.ics.uci.edu/ mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998.

[12] M. H. Wright, "Optimization Methods For Base Station Placement In Wireless Applications," Proceedings of 1998 Vehicular Technology Conference, Vol. 89, pp. 11513-11517, 1998.

[13] A. J. C. Sharkey (Ed.), *Combining Artificial Neural Nets*. London: Springer-Verlag Limited, 1999.

[14] T. Golub, D. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. Mesirov, H. Coller, M. Loh, J. Downing, M. Caligiuri, C. Bloomfield, and E. S. Lander, "Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring," Science, 286(5439):531-537, October 1999.

[15] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York: Oxford University Press, Inc., 1999.

[16] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, *LAPACK Users' Guide, Third Edition*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[17] Nello Cristianini, John Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[18] O. Alter, P. O. Brown, D. Botstein, "Singular value decomposition for genome-wide expression data processing and modeling," *PNAS*, vol. 97, no. 18, pp. 10101-10106, August 29, 2000.

[19] C.-C. Chang, C.-J. Lin, "Training $\nu$-Support Vector Classifiers: Theory and Algorithms," *Neural Comp.*, vol. 13, pp. 2119-2147, 2001.

[20] R. O. Duda, P. E. Hart, D. G. Stork, *Pattern Classification, Second Edition*. New York: John Wiley & Sons, 2001.

[21] J. Khan, J. S. Wei, M. Ringnér, L. H. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. R. Antonescu, C. Peterson, P. S. Meltzer, "Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks," *Nat. Med*, June 2001, vol. 7, no. 6, pp. 673-679.

[22] B. Schölkopf, A. Smola, *Learning with Kernels*. MIT Press, 2002.

[23] D. Pearl, "Proteomic patterns in serum and identification of ovarian cancer," *The Lancet*, vol. 360, pp. 169-170, 2002.

[24] Robert E. Schapire, "The boosting approach to machine learning: An overview." In *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

[25] P.-H. Chen, C.-J. Lin, B. Schölkopf, "A Tutorial on $\nu$-Support Vector Machines," http://www.csie.ntu.edu.tw/~cjlin/papers/nusvmtutorial.pdf, 2003.

[26] S. Theodoridis, K. Koutroumbas, *Pattern Recognition, Second Edition*. Academic Press, 2003.

[27] S. G. Baker, "The Central Role of Receiver Operating Characteristic (ROC) Curves in Evaluating Tests for the Early Detection of Cancer," *Journal of the National Cancer Institute*, vol. 95, No. 7, pp. 511-515, April 2, 2003.

[28] W. Noble, "Support Vector Machine Applications in Molecular Biology." in B. Schölkopf, K. Tsuda and J.-P. Vert (eds.), *Kernel Methods in Computational Biology*. MIT Press, 2004, pp. 71-92.

[29] Lj. J. Buturović, "PCP: a program for supervised classification of gene expression profiles," Bioinformatics Advance Access published online on November 8, 2005. *Bioinformatics*, doi:10.1093/bioinformatics/bti760

[30] C.-C. Chang, C.-J. Lin, "LIBSVM: a Library for Support Vector Machines," http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf, August 12, 2004.

[31] S. Jaksic, "Gene Selection for Supervised Learning of Microarray Profiles." M. Sc. thesis, San Francisco State University, Computer Science Department, 2005.

[32] C.-W. Hsu, C.-C. Chang, C.-J. Lin, "A Practical Guide to Support Vector Classification, " http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[33] K. Kylheku, Kazlib library, http://users.footprints.net/ kaz/kazlib.html

[34] A. Faigon, "A library of internal sorting routines," http://www.yendor.com/programming/sort.