Appendix D

Introduction to PRTools

David M.J. Tax and Robert P.W. Duin

D.1 Introduction

The more than 100 routines offered by PRTools in its present state represent a basic set covering largely the area of statistical pattern recognition. In order to make the evaluation and comparison of algorithms more easy a set of data generation routines is included, as well as a small set of standard real world datasets.

This manual treats just the basic objects and constructs in the PRTools toolbox. It does not give an exhaustive explanation of all possibilities in the toolbox. To be more explicit, this manual *does not* treat:

- feature selection,
- error estimation techniques (cross-validation etc.),
- combining classifiers,
- clustering,
- applications on images.

Note: the manual tries to show the basic constructs and ideas behind the toolbox. In the actual application of the toolbox in practice, the user should frequently use the `help`. For each function defined in the PRTools toolbox an extensive help text is present, explaining the syntax, the arguments, the function and the possible default values for arguments.

D.2 PRTools

PRTools deals with sets of labeled objects and offers routines for generalising such sets into functions for data mapping and classification. An object is a `k`-dimensional vector of feature values. It is assumed that for all objects in a problem *all* values of the same set of features are given (and that there are no missing values). The space defined by the actual set of features is called the *feature space*. Objects are represented as points or vectors in this space.

A classification function assigns labels to new objects in the feature space. Usually, this is not done directly, but in a number of stages in which the initial feature space is successively mapped into intermediate stages, finally followed by a classification. The concept of mapping spaces and dataset is thereby important and constitutes the basis of many routines in the toolbox.

PRTools makes use of the possibility offered by Matlab 5 to define 'classes' and 'objects'. These programmatic concepts should not be confused with the classes and objects as defined in Pattern Recognition. Two classes have been defined: `prdataset` and `mapping`. A large number of operators (like `*`, `[]`) and Matlab commands have been overloaded and have thereby a special meaning when applied to a dataset and/or a mapping.

In the coming sections it is explained how `prdataset`s and `prmapping`s can be created, modified and applied.

D.3 Dataset

The central data structure of PRTools is the `prdataset`. It primarily consists of a set of objects represented by a matrix of feature vectors. Attached to this matrix is a set of labels, one for each object and a set of feature names. Moreover, a set of a priori probabilities, one for each class, is stored. In most help files of PRTools, a dataset is denoted by `a`. In almost any routine this is one of the inputs. Almost all routines can handle multi-class object sets.

prdataset	
`prdataset`	Define dataset from data matrix and labels
`getdata`	Retrieve data from dataset
`getlabels`	Retrieve object labels from dataset
`getfeat`	Retrieve feature labels from dataset
`genlab`	Generate dataset labels
`renumlab`	Convert labels to numbers

Sets of objects may be given externally or may be generated by one of the data generation routines of PRTools. Their labels may also be given externally or may be the result of a cluster analysis.

A dataset containing 10 objects with 5 random measurements can be generated by:

```
>> data = rand(10,5);
>> a = prdataset(data)
10 by 5 dataset with 0 classes: []
```

In the previous example no labels were supplied, therefore one class is detected. Labels can be added to the dataset by:

```
>> data = rand(10,5);
>> labs = ['A'; 'A'; 'A'; 'B'; 'A'; 'A'; 'B'; 'B'; 'B'; 'B'];
>> a = prdataset(data,labs)
10 by 5 dataset with 2 classes: [5  5]
```

In most cases the objects in the dataset are ordered (such that the first `n1` objects belong to class `A`, the next `n2` to class `B`, etc). In this case it is easier to use the `genlab` routine for the generation of the labels:

```
>> data = rand(10,5);
>> labs = genlab([5 5],['A';'B']);
>> a = prdataset(data,labs)
10 by 5 dataset with 2 classes
```

Next to class labels, it is also possible to supply feature labels. To set this parameter, the keyword `featlab` should be given before:

```
>> data = rand(10,3);
>> labs = genlab([5 5],['A';'B']);
>> feats = ['Feat1';'Feat2';'Feat3'];
>> a = prdataset(data,labs,'featlab',feats)
10 by 3 dataset with 2 classes: [5  5]
```

There are numberous other parameters which can be defined inside the dataset definition. The class prior probabilities can be defined, the dataset name, cost matrices can be given, it can be indicated whether features are numerical or categorical, etc. etc. See for more information about the possible parameters `help prdataset`.

All the values can be set by their respective commands. For instance for setting the prior:

```
>> a = setprior(a,[0.4 0.6]);
```

Analogous you can use `setcost`, `setname` or `setfeatdom`.

Various items stored in a dataset can be retrieved by the `get` series of commands:

```
>> name = getname(a);
>> prob = getprior(a);
>> nlab = getnlab(a);
>> lablist = getlablist(a);
>> [m,k,c] = getsize(a);
```

in which `nlab` are numeric labels for the objects `(1, 2, 3, ...)` referring to the true labels stored in the rows of `lablist`. The size of the dataset is `m` by `k`, `c` is the number of classes (equal to `max(nlab)`).

The original data matrix can be retrieved by `double(a)` or by `+a`. The labels in the objects of `a` can be retrieved `labels = getlab(a)`, which is equivalent to `labels = lablist(nlab,:)`. The feature labels can be retrieved by `featlist = getfeat(a)`.

Note: In many respects a `prdataset` object can be compared with a normal MATLAB matrix. Normal matrix operations like `a.^2`, `abs(a)`, `a(:,3)`, `a(1,:)`, `size(a)` etc, can directly be applied to `prdataset` objects. In some cases MATLAB routines cannot cope with the `prdataset` object. In these cases the user has to use `+a` instead of `a`.

Another way to inspect a dataset `a` is to make a scatterplot of the objects in the dataset. For this the function `scatterd` is supplied. This plots each object in the dataset `a` in a 2D graph, using a coloured marker when class labels are supplied. When more than 2 features are present in the dataset, the first two are used. When you want to make a scatterplot of two other features, you have to explicitly extract them by `scatterd(a(:,[2 5]))`.

`prdataset` generation	
`gauss`	Generation of multivariate Gaussian distributed data
`gendatb`	Generation of banana shaped classes
`gendatc`	Generation of circular classes
`gendatd`	Generation of two difficult classes
`gendath`	Generation of Higleyman classes
`gendatl`	Generation of Lithuanian classes
`gendatm`	Generation of many Gaussian distributed classes
`gendats`	Generation of two Gaussian distributed classes

Exercise D.1 Generate an artificial dataset (one from the `prdataset` generation table) containing 3 objects per class.

Exercise D.2 Inspect the data matrix in the `prdataset` object by using the `+` operator and make a scatterplot of the dataset.

Exercise D.3 Can you point out which object is plotted where in the the scatterplot? Check if each object is coloured according to their class label.

Datasets can be combined by `[a; b]` if `a` and `b` have equal numbers of features and by `[a b]` if they have equal numbers of objects.

Exercise D.4 Make a dataset containing 2 classes with 5 objects per class and call it `a1` (for instance from `gendats`). Make a second dataset containing 2 classes with 5 objects per class and call it `a2` (for instance from `gendatb`).

Combine the two datasets using

```
>> b = [a1; a2]
```

and make a scatterplot.

Next combine the two datasets using

```
>> c = [a1 a2]
```

and again make a scatterplot. In what do the two datasets differ? Check your opinions by using `size(b)` and `size(c)`.

Creating subsets of datasets can be done by `a(I,J)` in which `I` is a set of indices defining the desired objects and `J` is a set of indices defining the desired features. In all these examples the a priori probabilities set for `a` remain unchanged.

Exercise D.5 Use the dataset `c` from the previous exercise to extract feature 2 and 4. Make a scatterplot and compare it with the data matrix. Where is the first object in the data matrix plotted in the scatterplot?

There is a large set of routines for the generation of arbitrary normally distributed classes (`gauss`), and for various specific problems (`gendatc`, `gendatd`, `gendath`, `gendatm` and

`prdataset` manipulation	
`gendat`	Extraction of subsets of a given data set
`gendatk`	Nearest neighbour data generation
`gendatp`	Parzen density data generation
`gendat`	Extraction of test set from given dataset
`prdata`	Read data from file and convert into a dataset

`gendats`). There are two commands for enriching classes by noise injection (`gendatk` and `gendatp`). These are used for the general test set generator `gendatt`. A given dataset can be split randomly into a training set and a test set using `gendat`.

Exercise D.6 Generate a dataset using `gendatb` containing 5 objects per class.

Enlarge this dataset by making a new dataset using `gendatk` and `gendatp` and appending this dataset to your original dataset.

Make a scatterplot of the original and the second dataset, to check that it worked.

Note: have a look at the `help` if it is not clear how you should use a function!

Exercise D.7 Make a dataset with 100 objects per class. Generate two new datasets `trainset` and `testset` with 60 and 40 objects per class respectively.

Make two scatterplots of the datasets. Do they differ much?

D.4 Datafile

The above described `prdataset` class refers to data that are stored in the computer memory and not on disk. It has thereby its limitations w.r.t. the numbers of objects and features that can be used. Moreover, it is assumed that objects are already represented by a fixed number of vector elements (features, dissimilarities).

The recently introduced (PRTools version 4.1) class `prdatafile` offers several ways to start with objects stored in files as raw images, time signals or mat-files. There is thereby no limitation to the size. a `prdatafile` is a special type of `prdataset` by which many operations defined on a `prdataset` are supported. In addition there are tools to define several types of preprocessing filters and feature measurement commands. At some moment all objects in a `prdatafile` are represented by the same features and it can be converted to a `prdataset`. After that all procedures for dimension reduction, cluster analysis, classification and evaluation can be used.

An important difference between a `prdatafile` and a `prdataset` is that operations on the first are just stored in the `prdatafile` as a defined processing which is only executed at the moment it is converted to a `prdataset`. Operations on a `prdataset` are directly executed.

D.5 Mapping

Data structures of the class `mapping` store trained classifiers, feature extraction results, data scaling definitions, nonlinear projections, etc. They are usually denoted by `w`.

mapping	
`prmapping`	Define mapping
`getlab`	Retrieve labels assigned by mapping

A mapping `w` is often created by training a classifier on some data. For instance, the nearest mean classifier `nmc` is trained on some data `a` by:

```
>> a = gendatb(20);
>> w = nmc(a)
Nearest Mean, 2 to 2 trained classifier --> affine
```

`w` by itself, or `display(w)` lists the size and type of a classifier as well as the routine which is used for computing the mapping `a*w`.

When a mapping is trained, it can be applied to a dataset, using the operator `*`:

```
>> b = a*w
Banana Set, 20 by 2 dataset with 2 classes: [7  13]
```

The result of the operation `a*w` is again a dataset. It is the classified, rescaled or mapped result of applying the mapping definition stored in `w` to `a`.

`mapping`s and classifiers	
`classc`	Converts a mapping into a classifier
`labeld`	General classification routine for trained classifiers
`testc`	General error estimation routine for trained classifiers

All routines operate in multi-class problems. For mappings which change the labels of the objects (so the mapping is actually a classifier) the routines `labeld` and `testc` are useful. `labeld` and `testc` are the general classification and testing routines respectively. They can handle any classifier from any routine.

Linear and polynomial classifiers	
`klldc`	Linear classifier by KL expansion of common cov matrix
`loglc`	Logistic linear classifier
`fisherc`	Fisher's discriminant (minimum least square linear classifier)
`ldc`	Normal densities based linear classifier (Bayes rule)
`nmc`	Nearest mean classifier
`nmsc`	Scaled nearest mean classifier
`perlc`	Linear classifier by linear perceptron
`pfsvc`	Pseudo-Fisher support vector classifier
`qdc`	Normal densities based quadratic (multi-class) classifier
`udc`	Uncorrelated normal densities based quadratic classifier

Nonlinear classifiers	
`knnc`	k-nearest neighbour classifier (find k, build classifier)
`mapk`	k-nearest neighbour mapping routine
`testk`	Error estimation for k-nearest neighbour rule
`parzenc`	Parzen density based classifier
`parzenml`	Optimization of smoothing parameter in Parzen density estimation.
`parzen_map`	Parzen mapping routine
`testp`	Error estimation for Parzen classifier
`edicon`	Edit and condense training sets
`treec`	Construct binary decision tree classifier
`tree_map`	Classification with binary decision tree
`bpxnc`	Train feed forward neural network classifier by backpropagation
`lmnc`	Train feed forward neural network by Levenberg-Marquardt rule
`rbnc`	Train radial basis neural network classifier
`neurc`	Automatic neural network classifier
`rnnc`	Random neural network classifier
`svc`	Support vector classifier

D.6 Training and testing

There are many commands to train and use mappings between spaces of different (or equal) dimensionalities. For example:

if `a` is an `m` by `k` dataset (`m` objects in a `k`-dimensional space)
and `w` is a `k` by `n` mapping (map from `k` to `n` dimensions)
then `a*w` is an `m` by `n` dataset (`m` objects in a `n`-dimensional space)

Mappings can be linear (e.g. a rotation) as well as nonlinear (e.g. a neural network). Typically they can be used for classifiers. In that case a `k` by `n` mapping maps an `k`-feature data vector on the output space of an `n`-class classifier (exception: 2-class classifiers like discriminant functions may be implemented by a mapping to a 1D space like the distance to the discriminant, `n = 1`).

Mappings are of the data type `mapping`, have a size of `[k,n]` if they map from `k` to `n` dimensions. Mappings can be instructed to assign labels to the output columns, e.g. the class names. These labels can be retrieved by

`labels = getlab(w);` before the mapping, or
`labels = getlab(a*w);` after the dataset `a` is mapped by `w`.

Mappings can be learned from examples, (labeled) objects stored in a dataset `a`, for instance by training a classifier:

`w3 = ldc(a);` the normal densities based linear classifier
`w2 = knnc(a,3);` the 3-nearest neighbor rule
`w1 = svc(a,'p',2);` the support vector classifier based on a 2$^{\text{nd}}$ order polynomial kernel

OPTIONAL

The mapping of a test set `b` by `b*w1` is now equivalent to `b*(a*v1)` or even, irregularly but very handy to `a*v1*b` (or even `a*ldc*b`). Note that expressions are evaluated from left to right, so `b*a*v1` may result in an error as the multiplication of the two datasets `(b*a)` is executed first.

END OPTIONAL

D.7 Example

In this example a 2D Highleyman dataset `A` is generated, 100 objects for each class. Out of each class 20 objects are generated for training, `C` and 80 for testing, `D`. Three classifiers are computed: a linear one and a quadratic one, both assuming normal densities (which is correct in this case) and a Parzen classifier. Note that the data generation use the random generator. As a result they only reproduce if they use the original seed. After computing and displaying classification results for the test set a scatterplot is made in which all classifiers are drawn.

```
%PREX_PLOTC  PRTools example on the dataset scatter and classifier plot
help prex_plotc
echo on
              % Generate Higleyman data
A = gendath([100 100]);
              % Split the data into the training and test sets
[C,D] = gendat(A,[20 20]);
              % Compute classifiers
w1 = ldc(C);        % linear
w2 = qdc(C);        % quadratic
w3 = parzenc(C);    % Parzen
w4 = lmnc(C,3);      % neural net
              % Compute and display errors
              % Store classifiers in a cell
W = w1,w2,w3,w4;
              % Plot errors
disp(D*W*testc);
              % Plot the data and classifiers
figure
              % Make a scatter-plot
scatterd(A);
              % Plot classifiers
plotc(w1,w2,w3,w4);
echo off
```

Appendix E

Introduction to datafiles

Robert P.W. Duin and David M.J. Tax

E.1 Introduction

Since the introduction of PRTools 4.1 the toolbox is extended to avoid some of the limitations that were imposed by the use of the `prdataset` object. The first limitation of the dataset object is that all objects in a dataset should have the same (number of) features. In particular when one starts with raw data, for instance when one is using images or timeseries with different sizes, the processing has to be done outside PRTools. A second limitation is that datasets can become very large. So large in fact, that the dataset object cannot be stored in the memory. To remove these limitations the `prdatafile` object is introduced.

A datafile in PRTools is a generalization of the dataset concept, and they inherit most of the properties of a dataset. The datafile allows the distribution of a large dataset over a set of files, stored on disk. Datafiles are mainly an administration about the files and directories in which the objects are stored. A datafile is, like a dataset, a set consisting of M objects, each described by k features. k may be unknown and varying over different objects, in which case it is set to zero.

Almost all operations defined for datasets are also defined for datafiles, with a few exceptions. Operations on datafiles are possible as long as they can be stored (e.g. filtering of images for raw datafiles, or object selection by `gendat`). Furthermore, commands that are able to process objects sequentially, like `nmc` and `testc` can be executed on datafiles. The use of untrained mappings in combination with datafiles is a problem, as they have to be adapted to the sequential use of the objects. Mappings that can handle datafiles are indicated in the Contents file.

Different types of datafiles are defined:

raw Every file is interpreted as a single object in the dataset. These files may, for instance, be images of different size. It is assumed that objects from different classes are stored each in a different directory. The directory name is used as the class label.

cell All files should be mat-files containing just a single variable being a cell array. Its elements are interpreted as objects. The file names will be used as labels during construction. This may be changed by the user afterwards.

pre-cooked In this case the user should supply a command that reads the file and converts it to a dataset. More than one object may be stored in a single file. Furthermore, the command should define the class labels for the objects stored in the dataset.

half-baked All files are mat-files, containing a single labeled dataset.

mature This is a datafile created by PRTools, using the `savedatafile` command after execution of all preprocessings defined for the datafile.

The most natural way for a user to create a datafile, is using the option **raw**.

Whenever a raw datafile is sufficiently defined by pre- and postprocessing it can be converted into a dataset. If this is still a large dataset, not suitable for the available memory, it should be stored by the `savedatafile` command and is ready for later use. If the dataset is sufficiently small it can be directly converted into a dataset by `prdataset`.

E.2 Operations on datafiles

The main commands specific for datafiles are:

`prdatafile`	constructor. It defines a datafile on a directory.
`addpreproc`	adds preprocessing commands (low level command)
`addpostproc`	adds postprocessing commands (low level command)
`filtm`	user interface to add preprocessing to a datafile.
`savedatafile`	executes all defined pre- and postprocessing and stores the result as a dataset in a set of matfiles.
`prdataset`	conversion to dataset

Most functions that were defined in MeasTools are also available as mappings that operate on datafiles. In particular, all examples as they were defined in the MeasTools appendix work in an idential way:

```
% load digits as measurement variables
>> a = load_nist([5,9],[1:20]); figure; show(a)
% add rows and columns to create square figure (aspect ratio 1)
>> b = im_box(a,[],1);          figure; show(b)
% resample by 16 x 16 pixels
>> c = im_resize(b,[16,16]);    figure; show(c)
% compute means
>> d = im_mean(c);
% convert to PRTools dataset and display
>> d = prdataset(d,[],'featlab',char('mean-x','mean-y'));
>> scatterd(x,'legend');
```

E.3 Image Processing

Before measurering features, images might have to be preprocessed. PRTools contains a series of `mapping` routines to facilitate this. If the desired routine is not available, they can easily be built from standard Matlab or `DipImage` routines using `filtm`.

mappings for image processing	
im_gray	Convert any image to a gray-value image
im_invert	Invert an image by subtracting it from its maximum
im_gaussf	Gaussian filter
im_minf	Minimum filter
im_maxf	Maximum filter
im_stretch	Grey-value stretching
im_hist_equalize	Histogram equalization
im_threshold	Thresholding
im_berosion	Binary erosion
im_bdilation	Binary dilation
im_bpropagation	Binary propagation
im_label	Label binary image
im_select_blob	Selection of largest blob in a binary image
im_box	Bounding box for binary image
im_center	Center binary image in center of gravity
im_resize	Resize images
im_scale	Scale binary image
im_rotate	Rotate image
im_fft	Image FFT
im_cols	Conversion of full color images to 3 grey value images

A number of more specific routines for measuring features in images are available:

prfilters for features of images	
im_moments	Computes various image moments, central, Hu and Zernike.
im_mean	Computes just the centre of gravity.
im_profile	Computes the horizontal and vertical image profiles.
im_measure	Measures various object features.

The last routine, `im_measure`, computes a single set of features. If the image contains a set of objects, it has first to be split by `im2meas` into images containing a single object, in order to obtain a `measurement` set in which each object refers to a single physical object. The `im_measure`-routine makes use of the `DipImage` package. The user has to take care that this is loaded.