Prediction Analysis of Microarrays for R: Installation Guide and Manual

Robert J. Tibshirani, Trevor J. Hastie, Balasubramanian Narasimhan, and Gilbert Chu

$Revision: 1.3 $ of $Date: 2004/07/17 05:21:06 $.

This document describes the installation and use of the PAM for R package. This document is always kept up-to-date at The PAM Page.

---

1. Introduction

The package `pamr` provides R functions for carrying out sample classification from gene expression data, by the method of nearest shrunken centroids.

- PAM is a simple, accurate and fast classifier, providing intrepretable results for the biologist.
- It has many novel features, including *heterogeneity analysis* for discriminating a diseased group from a normal one.
- PAM works under Windows and Unix/Linux and Mac. Under Windows, one may use the R package direct or use a graphical user interface. The latter is an Excel front-end called PAM For Excel.
- The methodology is described in the paper Diagnosis of multiple cancer types by shrunken centroids of gene expression by Tibshirani, Hastie, Narasimhan and Chu (May 14, 2002).

1.1 About the package name

There are already some functions in R called *pam* for partitioning around medoids. To avoid confusion, we have called our R package *pamr*. All functions provided by our package are named with a `pamr` prefix.

2. Installing `pamr`

1. You first need to install a recent version of the R statistical package. This is free, and can be found at The R Project.

   Follow the instructions. Click on http://cran.r-project.org/mirrors.html under Download. Pick a mirror closest to you. You probably want a pre-compiled version. For windows, choose the **Windows (95 or later)** link, select `base`, and download `SetupR.exe`. Installation takes 5-10 minutes.

   R is a great package, and is worth knowing about!

2. Download the appropriate Unix/Linux version or Windows version for your platform from the PAM R distribution site. Often, browsers offer you a choice of opening the file or saving it. Elect to save it and remember the name of the folder where you saved it.
3. For Unix/Linux type

   ```
   R CMD INSTALL -l mylib pamr_1.28.tar.gz
   ```

   In Windows, click on the R icon on your desktop, pull down the `Packages` menu item and select `Install packages from local zip file`. A file chooser dialog will allow you to select the file `pamr_1.28.zip` that you just saved in the previous step.

The above concludes the installation process and needs to be done just once.

3. Using `pamr`

For Unix/Linux, start R and type

```
library(pamr,lib.loc="mylib")
```

In Windows, while in R pull down the `Packages` menu item and select `Load package`. Select `pamr`. In Windows you will find it helpful to go the `Misc` menu and turn off buffered output. This forces R output to be immediately written to the console.

You are now ready to use PAM.

4. Summary of functions

- pam.knnimpute A function to impute missing expression data
- pamr.adaptthresh A function to adaptive choose threshold scales, for use in `pamr.train`
- pamr.batchadjustA function to mean-adjust microarray data by batches
- pamr.cv A function to cross-validate the nearest shrunken centroid classifier
- pamr.fdr A function to estimate false discovery rates for the nearest shrunken centroid classifier
- pamr.plotfdr A function to plot the false discovery rates for the nearest shrunken centroid classifier
- pamr.from.excel A function to read in a text file saved from Excel
- pamr.makeclasses A function to interactively define classes from a clustering tree
- pamr.menu A function that interactively leads the user through a PAM analysis
- pamr.plotcen A function to plot the shrunken class centroids, from the nearest shrunken centroid classifier
- pamr.plotcv A function to plot the cross-validated error curves from the nearest shrunken centroid classifier
- pamr.plotcvprob A function to plot the cross-validated sample probabilities from the nearest shrunken centroid classifier
- pamr.predict A function producing predicted information, from a nearest shrunken centroid fit
- pamr.to.excel A function to write out a data object into a tab-delimited text file
- pamr.train A function to train a nearest shrunken centroid classifier
- pamr.outliers A function to find potentially outlying samples
- pamr.indeterminate A function to classify samples, allowing for an indeterminate (doubt) category

To get detailed help on any function in R, use the command `help(function)`. For example, `help(pamr.train)`.

5. A Sample Session

Please note that in what follows, code section lines starting with "#" are comments.

5.1 Reading in data

The first thing to do is to read in some data. For an example, download the sample dataset Khan data and save it as the text file khan.txt in your current directory. When reading in the data, remember to put quotes around each genename in your Excel spreadsheet. In Unix, you do this as follows.

```
## Read in sample dataset: khan data, 2308 genes, 65 columns.
khan.data <- pamr.from.excel("khan.txt", 65, sample.labels=TRUE)
```

On Windows, use the following.

```
khan.data <- pamr.from.excel("khan.txt", 65, sample.labels=TRUE)
```

5.2 PAM Analysis

To do a pam analysis, you can either run the various commands (e.g. `pamr.train`) one at a time, or use the function `pamr.menu` which interactively leads the user through a typical analysis. The individual commands used in a non-interactive analysis are documented below. *The "non-interactive" analysis offers more control of input options*.

To start the interactive analysis, type

```
pamr.menu(khan.data)
```

This produces the following menu:

```
1:pamr.train
2:pamr.cv
3:pamr.plotcv
4:pamr.plotcen
5:pamr.confusion
6:pamr.plotcvprob
7:pamr.geneplot
8:pamr.listgenes
9:pamr.train with heterogeneity analysis
10:Exit
Selection:
```

The standard procedure is to begin by typing `1` to select `pamr.train`, and then after that computation is done, you would pick `2` for `pamr.cv`. Typically, you would go through steps `3` through `8`, to generate plots and gene lists. Along the way, in some of the steps you are asked for a threshold value: this value you choose visually from the plot created by `pamr.plotcv`. Menu Choice `9` is optional.

Here are the steps of a typical non-interactive analysis.

```
## Train the classifier
khan.train <- pamr.train(khan.data)

## Type name of object to see the results
khan.train

Call:
pamr.train(data = khan.data)

   threshold nonzero errors
1  0.000     2308    2
2  0.262     2289    1
3  0.524     2145    1
4  0.786     1878    0
5  1.048     1494    0
6  1.309     1137    0
etc.

## Cross-validate the classifier
khan.results<- pamr.cv(khan.train, khan.data)
khan.results

Call:
pamr.cv(a = khan.train, data = khan.data)
   threshold nonzero errors
1  0.000     2308    2
2  0.262     2289    2
3  0.524     2145    2
4  0.786     1878    2
5  1.048     1494    2
6  1.309     1137    1
7  1.571      853    1
etc.

## Plot the cross-validated error curves
pamr.plotcv(khan.results)

## Compute the confusion matrix for a particular model (threshold=4.0)
pamr.confusion(khan.results, threshold=4.0)

## Plot the cross-validated class probabilities by class
pamr.plotcvprob(khan.results, khan.data, threshold=4.0)

## Plot the class centroids
pamr.plotcen(khan.train, khan.data, threshold=4.0)

## Make a gene plot of the most significant genes
pamr.geneplot(khan.train, khan.data, threshold=5.3)


# Estimate false discovery rates and plot them
fdr.obj<- pamr.fdr(khan.train, khan.data)
```

```
pamr.plotfdr(fdr.obj)


## List the significant genes
pamr.listgenes(khan.train, khan.data, threshold=4.0)

## Try heterogeneity analysis, with class "BL" taken to be the normal group
khan.train2 <- pamr.train(khan.data,hetero="BL")
khan.results2 <-  pamr.cv(khan.train2, khan.data)


## Look for better threshold scalings
khan.scales <- pamr.adaptthresh(khan.train)
khan.train3 <- pamr.train(khan.data, threshold.scale=khan.scales)
khan.results3 <-  pamr.cv(khan.train3, khan.data)
```

5.3 Missing data, Batch Adjustment

If you have *missing expression data*, you will first want to impute the missing values, before running `pamr.train` etc. For example, if `khan.data` had missing values---in reality, it does not---you would do

```
khan.data2 <- pamr.knnimpute(khan.data)
```

and proceed to use `khan.data2` in all the analyses.

Suppose `khan.data` had **batch labels**---in reality, it does not, but see discussion of how to input batch labels below---then you may want to mean-adjust genes by batches before further analysis.

```
khan.data3 <- pamr.batchadjust(khan.data2)
```

If you want to write this adjusted data to a text file, suitable for reading into Excel:

```
pamr.to.excel(khan.data3, file="khan3.txt")
```

6. Notes on data entry

PAM expects the data in an object with components *x* (the expression matrix of genes by samples) and *y*, a vector of class labels. Optionally the object can also contain a geneid component and a genenames component.

You can create this data object (read in your data) any way you'd like. For example if the expression data were in a tab-delimited text file `foo.txt`, one row per gene, you could say

```
data$x <- read.table("foo.txt",sep="\t")
```

Similarly if the class labels were in a tab-delimited text file `foo2.txt` you could say

```
data$y <- factor(scan("foo2.txt",sep="\t",what=""))
```

To make things easier for users of SAM, we have provided the function `pamr.from.excel`, which reads a dataset in SAM format, that has been saved as a text file. It unpacks the *x*, *y*, geneid, and genenames components automatically.

A SAM spreadsheet has one row of expression values per gene. In addition there is one information row and two information columns. The first row has class labels for each of the samples. The first column had gene identifiers, and the second column has gene names.

Here is an example:

```
empty cell          empty cell          EWS              EWS           EWS            EWS
GENE1   " ""\""catenin (cadherin-a"" "  0.773343723     -0.078177781  -0.084469157    0.965614087
GENE2   " ""farnesyl-diphosphate""  "   -2.438404816    -2.415753791  -1.649739209   -2.3805466343
etc.
```

In the above "empty cell" is an empty cell in Excel. There are tabs between each entry.

7. Gene Names

You should put quotes (") around the all genenames in the Excel spreadsheet. Otherwise PAM can get confused with long genenames that wrap over the end of a line. In Excel, this is done as follows (thanks to Brian Zing).

1. Select the column
2. Select **Format**Cells/Number/Custom/ and then in the text box under **Type** enter: \"@\"
3. Click OK and the column should have quotes around all the text.

In addition, if `sample.labels=TRUE` is specified in the call to `pamr.from.excel`, the data file is assumed to have an additional row at the top, consisting of two blank cells followed by a sample labels for each of the columns. If available, these sample labels are used by various plotting routines.

For example, here is the first part of the file `khan.txt`, supplied with the PAM distribution:

```
empty cell          empty cell           "sample1"     "sample2"       "sample3"       "sample4"
empty cell          empty cell            EWS           EWS             EWS             EWS
GENE1   " ""\""catenin (cadherin-a"" "  0.773343723     -0.078177781  -0.084469157    0.965614087
GENE2   " ""farnesyl-diphosphate""  "   -2.438404816    -2.415753791  -1.649739209   -2.3805466343
etc.
```

This is the first part of the file `khan.txt`, suppiled with this software. It is a tab-delimited text file, saved from an Excel spreadsheet.

Finally, one can also include a row of batch labels after the row of sample labels. Indicate batch.labels=TRUE in the call to `pamr.from.excel`. Example of input text file:

```
empty cell          empty cell           "sample1"     "sample2"      "sample3"      "sample4"
empty cell          empty cell           "batch1"      "batch2"       "batch3"       "batch4"
empty cell          empty cell            EWS           EWS             EWS             EWS
GENE1   " ""\""catenin (cadherin-a"" "  0.773343723     -0.078177781  -0.084469157    0.965614087
GENE2   " ""farnesyl-diphosphate""  "   -2.438404816    -2.415753791  -1.649739209   -2.3805466343
etc.
```

Or you could have batch labels but no sample labels in the file. Then would specify `batch.labels=TRUE sample.labels=FALSE` in the call to `pamr.from.excel`.

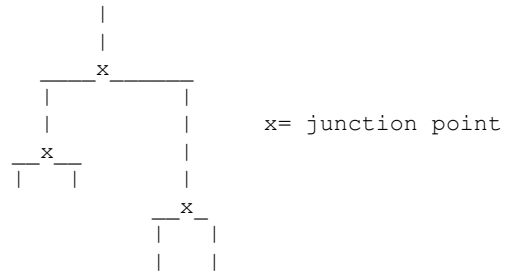
8. Defining new classes: the function `pamr.makeclasses`

This function allows one to interactively define classes from a clustering tree, for later use in `pamr.train`, `pamr.cv` etc. This is useful for creating classes from unlabelled data, or defining new groups from labelled data. Eg. given classes 1,2,3,4, one can define new groups [1,3] and [2,4].

One starts by typing

```
khan.data$newy <- pamr.makeclasses(khan.data)
```

This causes a clustering tree to be drawn, via the the R function `hclust`, and any arguments for `hclust` can be passed to it `pamr.makeclasses`. See `help(hclust)` for details on the options



Using the left button, the user clicks at the junction point defining the class 1. More groups can be added to class 1 by clicking on further junction points. The user ends the definition of class 1 by clicking on the rightmost button. [Under Windows, an additional menu appears; choose `Stop`] . This process is continued for classes 2,3 etc. Note that some samples may be left out of the new classes. Two consecutive clicks of the right button ends the definition for all classes.

At the end, the clustering is redrawn, with the new class labels shown.

The function returns a vector of class labels 1,2,3..., and these are assigned to component `newy` of `khan.data` by the command above. If a component is `NA` (missing), then the sample is not assigned to any class.

Then the user calls `pamr.train`, `pamr.cv` etc. in the usual way:

```
khan.train <- pamr.train(khan.data) khan.results <- pamr.cv(khan.train, khan.data)
```

Note that `pamr.train`, `pamr.cv` and all functions use the class labels in the component `newy` if they are present. Otherwise they use the data labels `y`.

There is one optional argument called `sort.by.class`. If `TRUE`, the clustering tree is forced to put all samples in the same class (as defined by the class labels `$y` in the data object) together in the tree. This is useful if a regrouping of classes is desired. Eg: given classes 1,2,3,4 you want to define new classes [1,3] vs [2,4] or 2 vs [1,3]. The default is `sort.by.class=FALSE`.

Note: This function is "fragile". The user must click close to the junction point, to avoid confusion with other junction points. Classes 1,2,3.. cannot have samples in common (if they do, an Error message will appear). If the function is confused about the desired choices, it will complain and ask the user to rerun pamr.makeclasses. The user should also check that the labels on the final redrawn cluster tree agrees with the desired classes.

IMPORTANT WARNING! Suppose you start with unlabelled data, and use `pamr.makeclasses` to define classes. You then run `pamr.train` and `pamr.cv`. Then the cross-validated misclassification rates given by `pamr.plotcv` will tend to be *biased downward*. This is because the same training data was used to define the class labels and and to train and cross-validate the classfier. Hence the absolute levels of misclassification rates (eg 5%) cannot be trusted. But the relative levels can still be useful. Eg. if you find that 100 genes give misclassification error lower that 50 or 500 genes, this is valid information.

9. How does nearest shrunken centroid classification work?

PAM uses the nearest shrunken centroid methodology described in: Diagnosis of multiple cancer types by shrunken centroids of gene expression by Tibshirani, Hastie, Narasimhan and Chu (May 14, 2002). Also see: Talk slides in Postscript or Talk slides in PDF.

Briefly, the method computes a standardized centroid for each class. This is the average gene expression for each gene in each class divided by the within-class standard deviation for that gene.

Nearest centroid classification takes the gene expression profile of a new sample, and compares it to each of these class centroids. The class whose centroid that it is closest to, in squared distance, is the predicted class for that new sample.

Nearest shrunken centroid classification makes one important modification to standard nearest centroid classification. It *shrinks* each of the class centroids toward the overall centroid for all classes by an amount we call the *threshold*. This shrinkage consists of moving the centroid towards zero by *threshold*, setting it equal to zero if it hits zero. For example if *threshold* was 2.0, a centroid of 3.2 would be shrunk to 1.2, a centroid of -3.4 would be shrunk to -1.4, and a centroid of 1.2 would be shrunk to zero.

After shrinking the centroids, the new sample is classified by the usual nearest centroid rule, but using the shrunken class centroids.

This shrinkage has two advantages:

1. It can make the classifier more accurate by reducing the effect of noisy genes
2. It does automatic gene selection.

In particular, if a gene is shrunk to zero for all classes, then it is eliminated from the prediction rule. Alternatively, it may

be set to zero for all classes except one, and we learn that high or low expression for that gene characterizes that class.

The user decides on the value to use for *threshold*. Typically one examines a number of different choices. To guide in this choice, PAM does K-fold cross-validation for a range of threshold values. The samples are divided up at random into K roughly equally sized parts. For each part in turn, the classifier is built on the other K-1 parts then tested on the remaining part. This is done for a range of threshold values, and the cross-validated misclassification error rate is reported for each threshold value. Typically, the user would choose the threshold value giving the minimum cross-validated misclassification error rate.

What one gets from this is a (typically) accurate classifier, that is simple to understand.