

# Further analysis of microarray data in Bioconductor

Mark Dunning

January 27, 2014

Previously in this course, we have presented examples of how to pre-process gene-expression data generated on a specific platform (Illumina or Affymetrix) and produce a list of genes that show statistical evidence for differential expression. In most modern microarray experiments this is rarely the end-point of the analysis, and here we will introduce other ways that we can interrogate our gene expression data, which can be more open-ended or exploratory. The methods that we will apply rely on the data being stored in a common format such as the *ExpressionSet* class using in Bioconductor. Hence, we can analyse any microarray technology using the same steps. Indeed, the data doesn't necessarily need to be expression data.

We will use example datasets from Gene Expression Omnibus and a Bioconductor data package. When importing such datasets, it is important to check the properties of the data to ensure that the distributional assumptions of our methods are valid, and that the appropriate sample meta data are available to allow comparisons to be made.

- Have the data been normalized? If so, what method was used?
- Has quality control been performed?
- Is there sufficient sample meta-data to perform the analysis?
- Are there any batch-effects remaining in the data?

Section 1 will illustrate unsupervised exploratory analysis techniques and also a Gene Ontology analysis. Section 2 will use a supervised approach to find a classifier for breast cancer samples.

## 1 Colon cancer data from GEO

We will consider GSE33126, which comprises nine paired tumor/normal colon tissues on Illumina HT12\_v3 gene expression Beadchips. These data were generated to inform a comparison of technologies for microRNA profiling. However, we will only use the mRNA data here. You should already have the raw GEO data on your computer in the `Downstream` folder. If not, it can be downloaded from the GEO website and then imported into R using `GEOquery`.

**Use Case:** Download the dataset with GEO ID GSE33126 and import the data using the `GEOquery` package.

```

1 > library(GEOquery)
2 > url <- "ftp://ftp.ncbi.nih.gov/pub/geo/DATA/SeriesMatrix/GSE33126/"
3 > filenm <- "GSE33126_series_matrix.txt.gz"
4 > if (!file.exists("GSE33126_series_matrix.txt.gz")) download.file(paste(url,
5 + filenm, sep = ""), destfile = filenm)
6 > colonData <- getGEO(filename = filenm)

```

We now need to explore the data to ensure that they have been normalised and are on an appropriate scale for analysis.

**Use Case:** Are the data on the  $\log_2$  scale? Are they normalised? Look at the phenotypic data stored with the object and find information about the sample groups in the study and patient IDs.

```

1 > head(exprs(colonData))
2 > exprs(colonData) <- log2(exprs(colonData))
3 > boxplot(exprs(colonData), outline = FALSE)
4 > pData(colonData)[1:2, ]
5 > SampleGroup <- pData(colonData)$source_name_ch1
6 > Patient <- pData(colonData)$characteristics_ch1.1

```

❏ The ArrayExpress package can be used in a similar way to download data from the Array-Express repository

## 1.1 Filtering the data

A first step in the analysis of microarray data is often to remove any uninformative probes. We can do this because typically only 50% of probes genes will be expressed, and even fewer than this will be differentially expressed. Including such non-informative genes in visualisation will obscure any biological differences that exist. The **genefilter** package contains a suite of tools for the filtering of microarray data. The **varFilter** function allows probes with low-variance to be removed from the dataset. The metric used to decide which probes to remove is the Inter-Quartile Range (*IQR*), and by default half of the probes are removed. Both the function used to do the filtering, and cut-off can be specified by the user.

**Use Case:** Remove probes with low-variance from the colon cancer dataset.

```

1 > library(genefilter)
2 > dim(colonData)
3 > varFiltered <- varFilter(colonData)
4 > dim(varFiltered)
5 > nrow(colonData)/nrow(varFiltered)

```

## 1.2 Calculating a distance matrix

The first step towards clustering the data is to construct a *distance matrix*. Each entry in this matrix is the pairwise distance between two samples. Note that for expression data we

have to transpose the standard ExpressionSet format, as clustering is designed to work on rows rather than columns. The standard function to make a distance matrix in R is `dist` which uses the *euclidean* metric. As the data entries in a distance matrix are symmetrical, it has a different representation to the default matrix (i.e. values are not repeated unnecessarily), and clustering algorithms are able to accept this representation as input.


**Use Case:** Construct a matrix of Euclidean distances for the variance-filtered dataset

```
1 > euc.dist <- dist(t(exprs(varFiltered)))
2 > euc.dist
```

For gene-expression data, it is common to use correlation as a distance metric rather than the Euclidean. *You should make sure that you know the difference between the two metrics.* The `cor` function can be used to calculate the correlation of columns in a matrix. Each row (or column) in the resulting matrix is the correlation of that sample with all other samples in the dataset. The matrix is symmetrical and we can transform this into a distance matrix by first subtracting 1 from the correlation matrix. Hence, samples with a higher correlation have a smaller 'distance'.

**Use Case:** Calculate a matrix of sample correlations and transform into a distance matrix

```
1 > corMat <- cor(exprs(varFiltered))
2 > corMat
3 > cor.dist <- as.dist(1 - corMat)
```

 The default method used to calculate correlations is the 'Pearson' method. If you wish to use the 'Spearman' (non-parametric) method you can specify `method="spearman"` in the call to `cor`.

### 1.3 Hierarchical clustering

Hierarchical clustering is the method by which samples with similar profiles are grouped together, based on their distances. The method for grouping samples can be specified, with the default being to use *complete linkage*. Different linkage methods can affect the properties of the clustering output. e.g. the 'ward' method tends to produce smaller, compact clusters. A popular way of displaying the results of clustering is a *dendrogram*, which arranges the samples on the x-axis and shows the distances between samples on the y-axis. It is important to note that the distance of samples on the x-axis has no meaning. The fact that two samples are displayed next to each other, does not mean that they are closest together. One has to use the y-axis to determine their distance

**Use Case:** Perform a hierarchical clustering on the filtered data using Euclidean distances. Are there any distinct groups in the data? Try using the correlation based similarity measure and complete linkage. Does the choice of method effect your clustering?

```
1 > par(mfrow = c(1, 2))
2 > clust.euclid = hclust(euc.dist)
```

```

3 > clust.cor = hclust(cor.dist)
4 > par(mfrow = c(1, 2))
5 > plot(clust.euclid, label = SampleGroup)
6 > plot(clust.cor, label = SampleGroup)

```

**Use Case:** Experiment with different methods of building the clustering; such as 'ward'.

```

1 > plot(hclust(euc.dist, method = "ward"))

```

## 1.4 Extracting data from the clustering

If we want to interpret the data presented in a clustering analysis, we need a way of extracting which samples are grouped together, or to determine the optimal grouping of samples. One intuitive way of assigning groups is to 'cut' the dendrogram at a particular height on the y-axis. We can do this manually on the plot, or use the `cutree` function to return the labels of samples that belong to the same group when the dendrogram is cut at the specified height,  $h$ . Alternatively, we can specify how many groups,  $k$ , that we want to create.

**Use Case:** 'Cut' the dendrogram at a height of 0.12. How many groups are formed? Now cut the dendrogram to obtain three groups. How well do these groups agree with the original sample groupings?

```

1 > library(cluster)
2 > par(mfrow = c(1, 1))
3 > plot(clust.cor)
4 > abline(h = 0.12, col = "red")
5 > cutree(clust.cor, h = 0.12)
6 > cutree(clust.cor, k = 2)
7 > table(cutree(clust.cor, k = 3), SampleGroup)

```

A *Silhouette plot* can be used to choose the optimal number of clusters. For each sample, we calculate a value that quantifies how well it 'fits' the cluster that it has been assigned to. If the value is around 1, then the sample closely fits other samples in the same cluster. However, if the value is around 0 the sample could belong to another cluster. In the silhouette plot, the values for each cluster are plotted together and ordered from largest to smallest. The number of samples belonging to each group is also displayed.

**Use Case:** Produce silhouette plots for the result of cutting the dendrogram to give 2,3,4 and 5 clusters. Which value of  $k$  seems to be the most sensible?

```

1 > par(mfrow = c(2, 2))
2 > plot(silhouette(cutree(clust.cor, k = 2), cor.dist),
3 +     col = "red", main = paste("k=", 2))
4 > plot(silhouette(cutree(clust.cor, k = 3), cor.dist),
5 +     col = "red", main = paste("k=", 3))
6 > plot(silhouette(cutree(clust.cor, k = 4), cor.dist),
7 +     col = "red", main = paste("k=", 4))
8 > plot(silhouette(cutree(clust.cor, k = 5), cor.dist),

```

```
9 + col = "red", main = paste("k=", 5))
```

If we have prior knowledge of how many clusters to expect, we could run the clustering in a supervised manner. The **Partition Around Mediods** method can be used to group samples into  $k$  clusters.

**Use Case:** Group the samples into two clusters using the `pam` method

```
1 > library(cluster)
2 > pam.clus <- pam(euc.dist, k = 2)
3 > clusplot(pam.clus)
4 > pam.clus$clustering
5 > table(pam.clus$clustering, SampleGroup)
```

## 1.5 Producing a heatmap

A heatmap is often used to visualise differences between samples. Each row represents a gene and each column is an array and coloured cells indicate the expression levels of genes. Both samples and genes with similar expression profile are clustered together. By default, euclidean distances are used with complete linkage clustering.

Drawing a heatmap in R uses a lot of memory and can take a long time, therefore reducing the amount of data to be plotted is usually recommended. Including too many non-informative genes can also make it difficult to spot patterns. Typically, data are filtered to include the genes which tell us the most about the biological variation. In an *un-supervised* setting, the selection of such genes is done without using prior knowledge about the sample groupings.

**Use Case:** Make a heatmap using the 100 most variable genes in the experiment according to their inter-quartile range (IQR). Can you see any structure in the data?

```
1 > IQRs = apply(exprs(varFiltered), 1, IQR)
2 > highVarGenes = order(IQRs, decreasing = T)[1:100]
3 > Symbols <- as.character(fData(colonData)$Symbol[highVarGenes])
4 > heatmap(as.matrix(exprs(varFiltered)[highVarGenes, ]),
5 + labCol = SampleGroup, labRow = Symbols)
```

The default options for the heatmap are to cluster both the genes (rows) and samples (columns). However, sometimes we might want to specify a particular order. For example, we might want to order the columns according to sample groups. We can do this by re-ordering the input matrix manually and setting the `Colv` to `NA`. This tells the `heatmap` function not to cluster the columns. It is important when choosing this option, we need to be careful that any column colourings or labels are set to the same ordering. Alternatively, a pre-calculated dendrogram could be used.

**Use Case:** Produce a new heatmap with columns ordered according to the tumour / normal status.

```

1 > colOrd <- order(SampleGroup)
2 > colOrd
3 > heatmap(as.matrix(exprs(varFiltered)[highVarGenes, colOrd]),
4 +       Colv = NA, labCol = SampleGroup[colOrd])

```

**Use Case:** Use the 'ward' method to cluster the samples and pass this to the heatmap function

```

1 > clus.ward <- hclust(cor.dist, method = "ward")
2 > heatmap(as.matrix(exprs(varFiltered)[highVarGenes, ]),
3 +       Colv = as.dendrogram(clus.ward), labCol = SampleGroup)

```

## 1.6 Customising the heatmap

The heatmap function can be customised in many ways to make the output more informative. For example, the `labRow` and `labCol` parameters can be used to give labels to the rows (genes) and columns (sample) of the heatmap. Similarly, `ColSideColors` and `RowSideColors` give coloured labels, often used to indicate different groups which are known in advance. See the help page for `heatmap` for more details.

**Use Case:** Define a colour label to distinguish tumour and normal samples.

```

1 > labs <- as.factor(SampleGroup)
2 > levels(labs) <- c("yellow", "blue")
3 > heatmap(as.matrix(exprs(varFiltered)[highVarGenes, ]),
4 +       labCol = Patient, ColSideColors = as.character(labs),
5 +       labRow = Symbols)

```

The colours used to display the gene expression values can also be modified. For this, we can use the `RColorBrewer` package which has functions for creating pre-defined palettes. The function `display.brewer.all` can be used to display the palettes available through this package.

**Use Case:** Change the colours to a red/blue scale

```

1 > library(RColorBrewer)
2 > display.brewer.all()
3 > hmcol <- brewer.pal(11, "RdBu")
4 > heatmap(as.matrix(exprs(varFiltered)[highVarGenes, ]),
5 +       ColSideColors = as.character(labs), labRow = Symbols,
6 +       col = hmcol)

```

**i** You should avoid using the traditional red / green colour scheme as it may be difficult for people with colour-blindness to interpret

A popular use for heatmaps is to take an existing gene list (e.g. genes found to be significant in a previous study, or genes belonging to a particular pathway) and produce an image of how they cluster the data for exploratory purposes. This can be achieved easily by selecting the correct rows in the data matrix.

**Use Case:** Produce a heatmap of all genes belonging to the cell-cycle pathway (KEGG 04110)

```
1 > library(illuminaHumanv3.db)
2 > pathwayGenes <- unlist(mget("04110", revmap(illuminaHumanv3PATH)))
3 > pathwayGenes <- pathwayGenes[pathwayGenes %in% featureNames(varFiltered)]
4 > symbols <- fData(varFiltered)[pathwayGenes, "Symbol"]
5 > heatmap(as.matrix(exprs(varFiltered)[pathwayGenes, ]),
6 +       ColSideColors = as.character(labs), labCol = Patient,
7 +       labRow = symbols, col = hmccl)
```

## 1.7 Principal Components Analysis

Principal components analysis (PCA) is a data reduction technique that allows us to simplify multidimensional data sets to 2 or 3 dimensions for plotting purposes and identify which factors explain the most variability in the data. We can use the `prcomp` function to perform a PCA and we have to supply it with a distance matrix. The resulting object contains information on the proportion of variance that is 'explained' by each component. Ideally, we want to see that the majority of the variance is explained by the first 2 or 3 components, and that these components are associated with a biological factor

**Use Case:** Perform a PCA on the correlation distances. How much variation is explained by the first two components?

```
1 > pca <- prcomp(cor.dist)
2 > plot(pca)
3 > summary(pca)
```

The `ggplot2` package is convenient for plotting principal components results as it allows many different covariates to be overlaid on the plot. See <http://ggplot2.org/> for more information on this package.

**Use Case:** Display the first two components using `ggplot2` and overlay the `SampleGroup` and cluster groupings that you obtained previously. Can you separate the tumours and normals on the plot?

```
1 > library(ggplot2)
2 > clusLabs <- cutree(clust.cor, k = 3)
3 > pcRes <- data.frame(pca$rotation, SampleGroup, Sample = rownames(pca$x),
4 +   Patient)
5 > ggplot(pcRes, aes(x = PC1, y = PC2, col = SampleGroup,
6 +   label = Patient, pch = as.factor(clusLabs))) + geom_point() +
7 +   geom_text(vjust = 0, alpha = 0.5)
```

**Use Case:** Compare the values of the first and second components for tumours and normals. Do they support the separation of sample groups?

```
1 > ggplot(pcRes, aes(x = SampleGroup, y = PC1, fill = SampleGroup)) +
```

```

2 +   geom_boxplot()
3 > ggplot(pcRes, aes(x = SampleGroup, y = PC2, fill = SampleGroup)) +
4 +   geom_boxplot()

```

## 1.8 Gene-Ontology analysis

In this section we give an example of how to find a list of relevant pathways / GO terms from a list of differentially-expressed genes. We will use the colon cancer data that we downloaded from GEO.

### 1.8.1 Non-specific filtering

We are now going to create a gene universe by removing genes for which will not contribute to the subsequent analysis. Such filtering is done without regarding the phenotype variables - hence a "non-specific" filter. An Illumina Human6 chip contains around 48,000 probes, but less than half of these have enough detailed information to be useful for a GO analysis. Therefore we restrict the dataset to only probes for which we have an Entrez ID. It is also recommended to select probes with sufficient variability across samples to be interesting; as probes with little variability will not be interesting to the question we are trying to answer. The interquartile-range of each probe across all arrays is commonly used for this with a cut-off of 0.5.

**Use Case:** Create the gene universe of all genes with Entrez ID and with sufficient variation across samples. How big is the universe?

```

1 > library(illuminaHumanv3.db)
2 > entrezIds = mget(rownames(exprs(colonData)), illuminaHumanv3ENTREZID,
3 +   ifnotfound = NA)
4 > haveEntrezId = names(entrezIds)[sapply(entrezIds, function(x) !is.na(x))]
5 > entrezSubset = exprs(colonData)[haveEntrezId, ]
6 > entrezIQR = apply(entrezSubset, 1, IQR)
7 > selected = entrezIQR > 0.5
8 > nsFiltered = entrezSubset[selected, ]
9 > universeIds = unlist(mget(rownames(nsFiltered), illuminaHumanv3ENTREZID,
10 +   ifnotfound = NA))

```

Remember that the size of the universe can have an effect on the analysis. If the universe is made artificially large by including too many uninformative probes, the p-values for the GO terms will appear more significant.

### 1.8.2 Selecting genes of interest and performing Hypergeometric test

We now test the genes in the universe to see which ones have significant differences between the two groups. For this, we use the `rowttests` function implemented in the `genefilter` package, which performs a t-test for each row with respect to a factor. The p-values of the test can be extracted, with one p-value given for each probe.

**Use Case:** Do a t-test for each probe between the two groups of samples we have identified. How many probes are significant at the 0.05 level? Define a list of genes to be used in the hypergeometric test by finding the Entrez IDs for these significant probes.



```

1 > library(GOstats)
2 > library(genefilter)
3 > fac = as.factor(SampleGroup)
4 > ttests = rowttests(as.matrix(nsFiltered), fac)
5 > smPV = ttests$p.value < 0.05
6 > pvalFiltered = nsFiltered[smPV, ]
7 > dim(pvalFiltered)
8 > selectedEntrezIds = unlist(mget(rownames(pvalFiltered),
9 +   illuminaHumanv3ENTREZID, ifnotfound = NA))

```

The `hyperGTest` function is used to do the hypergeometric test for GO terms. Rather than passing a long list of parameters to the function. An object of type *GOHyperGParams* is created to hold all the parameters we need to run the hypergeometric test. This object can then be passed to `hyperGTest` multiple times without having to re-type the parameters each time. The meanings of these parameters are as follows:

- `geneIds` - The list of identifiers for the genes that we have selected as interesting
- `universeGeneIds` - The list of identifiers resulting from non-specific filtering
- `annotation` - The name of the annotation package that will be used
- `ontology` - The name of the GO ontology that will be tested; either BP, CC or MF
- `pvaluecutoff` - p-value that we will use to select significant GO terms
- `testDirection` - Either "over" or "under" for over or under represented terms respectively
- `conditional` - A more sophisticated form of hypergeometric test, which takes the relationships between terms in the GO graph can be used if this is set to TRUE. For this practical we will keep `conditional = FALSE`

**Use Case:** Do a hypergeometric test to find which GO terms are over-represented in the filtered list of genes. How many GO terms are significant with a p-value of 0.05?

```

1 > params = new("GOHyperGParams", geneIds = selectedEntrezIds,
2 +   universeGeneIds = universeIds, annotation = "illuminaHumanv3",
3 +   ontology = "BP", pvalueCutoff = 0.05, conditional = FALSE,
4 +   testDirection = "over")
5 > hgOver = hyperGTest(params)
6 > hgOver

```

The `summary` function can be used to view the results of the test in matrix form. The rows of the matrix are arranged in order of significance. The p-value is shown for each GO term along with total number of genes for that GO term, number of genes we would be expected to appear in the gene list by chance and that number that were observed. A descriptive name is also given for each term. The results can also be printed out to a HTML report using `htmlReport`.

**Use Case:** View the results of the top 20 GO terms and create a HTML report.

```
1 > summary(hgOver)[1:20, ]
```

GStats also has the facility to test for KEGG pathways and chromosome bands which are over-represented. The procedure of creating a gene universe and set of selected genes is the same. However, we have to use a different object for the parameters, as not all

**Use Case:** Repeat the hypergeometric test for chromosome bands and KEGG pathways

```
1 > keggParams = new("KEGGHyperGParams", geneIds = selectedEntrezIds,
2 +   universeGeneIds = universeIds, annotation = "illuminaHumanv3",
3 +   pvalueCutoff = 0.05, testDirection = "over")
4 > keggHgOver = hyperGTest(keggParams)
5 > summary(keggHgOver)
6 > chrParams = new("ChrMapHyperGParams", geneIds = selectedEntrezIds,
7 +   universeGeneIds = universeIds, annotation = "illuminaHumanv3",
8 +   pvalueCutoff = 0.05, testDirection = "over", conditional = TRUE)
9 > chrHgOver = hyperGTest(chrParams)
10 > summary(chrHgOver)
```

## 1.9 A one-off test

Occasionally, we might want to check if a particular set of genes appear to be up- or down-regulated in an analysis. This can be checked using the `geneSetTest` function in `limma`, which computes a p-value to test the hypothesis that the selected genes have more extreme test-statistics than one might expect by chance. Moreover, separate tests can be performed to see if the selected genes are up-regulated (`alternative=up`) or down-regulated (`alternative=down`). The default is to test for extreme statistics regardless of sign.

**Use Case:** Do a `geneSetTest` to see if Cell Cycle (KEGG 04110) genes have extreme test-statistics. Are the genes significantly up-, or down-regulated?

```
1 > library(limma)
2 > ccGenes <- unlist(mget("04110", revmap(illuminaHumanv3PATH)))
3 > ccInd <- which(rownames(nsFiltered) %in% ccGenes)
4 > geneSetTest(index = ccInd, statistics = ttests$statistic)
5 > geneSetTest(index = ccInd, statistics = ttests$statistic,
6 +   alternative = "down")
7 > geneSetTest(index = ccInd, statistics = ttests$statistic,
8 +   alternative = "up")
9 > barcodeplot(ttests$statistic, index = ccInd)
10 > plot(density(ttests$statistic))
11 > lines(density(ttests$statistic[ccInd]), col = "red")
```

## 2 Using Bioconductor data packages

The Bioconductor project has a collection of example datasets. Often these are used as examples to illustrate a particular package or functionality, or to accompany the analysis presented in a publication. For example, several datasets are presented to accompany the `genefu`

package which has functions useful for the classification of breast cancer patients based on expression profiles.

An experimental dataset can be installed and loaded as with any other Bioconductor package. The data itself is saved as an object in the package. You will need to see the documentation for the package to find out the relevant object name. The full list of datasets available through Bioconductor can be found here <http://tinyurl.com/p5scw4a>.

**Use Case:** Install the `breastCancerVDX`, `breastCancerTRANSBIG` datasets. What microarray platform were they generated on? How many samples are in each dataset?

```
1 > library(breastCancerVDX)
2 > library(breastCancerTRANSBIG)
3 > data(vdx)
4 > data(transbig)
5 > dim(vdx)
6 > dim(transbig)
7 > annotation(vdx)
8 > annotation(transbig)
```

If we want any classifiers to be reproducible and applicable to other datasets, it is sensible to exclude probes that do not have sufficient annotation from the analysis. For this, we can use the `genefilter` package as before. The `nsFilter` function performs this annotation-based filtering as well as variance filtering. The output of the function includes details about how many probes were removed at each stage of the filtering.

**Use Case:** Filter the `vdx` dataset using to remove low-variances probes, and probes without sufficient annotation. How many probes are removed?

```
1 > library(genefilter)
2 > vdx.filt <- nsFilter(vdx)
3 > vdx.filt
4 > vdx.filt <- vdx.filt[[1]]
```

We will now attempt to build a classifier using the `pamr` (Predication analysis of Microarrays) <sup>1</sup> package.

**Use Case:** Format the `vdx` data for `pamr`, and train a classifier to predict ER status. For extra clarity in the results, it might be useful to rename the binary `er` status used in the data package to something more descriptive.

```
1 > library(pamr)
2 > dat <- exprs(vdx.filt)
3 > gN <- as.character(fData(vdx.filt)$Gene.symbol)
4 > gI <- featureNames(vdx.filt)
5 > sI <- sampleNames(vdx.filt)
6 > erStatus <- pData(vdx.filt)$er
7 > erStatus <- gsub(0, "ER-", erStatus)
8 > erStatus <- gsub(1, "ER+", erStatus)
```

---

<sup>1</sup>Not to be confused with the Partition Around Mediods method of clustering!

```

9 > train.dat <- list(x = dat, y = erStatus, genenames = gN,
10 +   geneid = gI, sampleid = sI)
11 > model <- pamr.train(train.dat, n.threshold = 100)
12 > model

```

We can perform cross-validation using the `pamr.cv` function. Printing the output of this function shows a table of how many genes were used at each threshold, and the number of classification errors. Both these values need to be taken into account when choosing a suitable threshold. The `pamr.plotcv` function can assist with this by producing a diagnostic plot which shows how the error changes with the number of genes. In the plot produced by this function there are two panels; the top one shows the errors in the whole dataset and the bottom one considers each class separately. In each panel, the x axis corresponds to the threshold (and number of genes at each threshold) whereas the y-axis is the number of misclassifications.

**Use Case:** Perform 10-fold cross-validation on the model and plot the results. Why is the maximum number of errors 135?

```

1 > model.cv <- pamr.cv(model, train.dat, nfold = 10)
2 > model.cv
3 > pamr.plotcv(model.cv)

```

**I** In the following sections, feel free to experiment with different values of the threshold (which we will call `Delta`)

The misclassifications can easily be visualised as a 'confusion table'. This simply tabulates the classes assigned to each sample against the original label assigned to the sample. e.g. Misclassifications are samples that we thought were 'ER+' but have been assigned to the 'ER-' group by the classifier, or 'ER-' samples assigned as 'ER+' by the classifier.

**Use Case:** Create a confusion table for your chosen threshold

```

1 > Delta <- 8
2 > pamr.confusion(model.cv, Delta)

```

A visual representation of the class separation can be obtained using the `pamr.plotcvprob` function. For each sample there are two circles representing the probability of that sample being classified ER- (red) or ER+ (green).

**Use Case:** Plot the cross-validated probabilities for each sample. Which samples seem to be easier to classify; ER positive, or ER negative?

```

1 > pamr.plotcvprob(model, train.dat, Delta)

```

There are a couple of ways of extract the details of the genes that have been used in the classifier. We can list their names using the `pamr.listgenes` function, which in our case these are just returns the microarray probe names. We can however, use these IDs to query the featureData stored with the original `vdX` object. We can also plot the expression values for each gene, coloured according to the class label.

**Use Case:** Extract the details of the probes used in the classifier and plot each one

```
1 > pamr.listgenes(model, train.dat, Delta)
2 > classifierGenes <- pamr.listgenes(model, train.dat, Delta)[,
3 +   1]
4 > fData(vdx.filt)[classifierGenes, ]
5 > pamr.geneplot(model, train.dat, Delta)
```

☞ You may get an error message `Error in plot.new(): Figure margins too large` when trying to produce the gene plot. If this occurs, try increasing the size of your plotting window, or decrease the number of genes by decreasing the threshold. Alternatively, the following code will write the plots to a pdf.

```
1 > pdf("classifierProfiles.pdf")
2 > for (i in 1:length(classifierGenes)) {
3 +   Symbol <- fData(vdx.filt)[classifierGenes[i], "Gene.symbol"]
4 +   boxplot(exprs(vdx.filt)[classifierGenes[i], ] ~ erStatus,
5 +     main = Symbol)
6 + }
7 > dev.off()
```

**Use Case:** Use the genes identified by the classifier to produce a heatmap to confirm that they separate the samples as expected.

```
1 > symbols <- fData(vdx.filt)[classifierGenes, "Gene.symbol"]
2 > heatmap(exprs(vdx.filt)[classifierGenes, ], labRow = symbols)
```

## 2.1 Testing the model

We can now test the classifier on an external dataset. We choose the *transbig* dataset for simplicity as it was generated on the same microarray platform

**Use Case:** Load the TRANSBIG breast cancer dataset and subset to the same genes as the filtered vdx dataset that we have just worked with.

```
1 > library(breastCancerTRANSBIG)
2 > data(transbig)
3 > pData(transbig)[1:4, ]
4 > transbig.filt <- transbig[featureNames(vdx.filt), ]
```

**Use Case:** Classify each sample in the Transbig dataset using the model generated from the vdx data. How well do the predictions agree with the original labels?

```
1 > predClass <- pamr.predict(model, exprs(transbig.filt),
2 +   Delta)
3 > table(predClass, pData(transbig)$er)
4 > boxplot(pamr.predict(model, exprs(transbig.filt), Delta,
5 +   type = "posterior")[, 1] ~ pData(transbig)$er)
```

**Use Case:** Make a heatmap of the transbig data using the genes involved in the vxd classifier

```
1 > erLab <- as.factor(pData(transbig)$er)
2 > levels(erLab) <- c("blue", "yellow")
3 > heatmap(exprs(transbig.filt)[classifierGenes, ], labRow = symbols,
4 +         ColSideColors = as.character(erLab))
```

## 3 Optional Extensions

### 3.1 Clustering and classification on the VDX datasets

**Use Case: Optional** If you have time, use some of the techniques from earlier in the practical to explore the VDX data

```
1 > pr <- prcomp(dist(t(exprs(vdx.filt))))
2 > pcRes <- data.frame(pr$rotation, SampleGroup = pData(vdx)$er,
3 +                     Sample = rownames(pr$x))
4 > ggplot(pcRes, aes(x = PC1, y = PC2, col = as.factor(SampleGroup))) +
5 +   geom_point()
6 > iqrs <- apply(exprs(vdx.filt), 1, IQR)
7 > topVar <- order(iqrs, decreasing = F)[1:50]
8 > erCol <- as.factor(pData(vdx)$er)
9 > levels(erCol) <- c("blue", "yellow")
10 > heatmap(exprs(vdx.filt)[topVar, ], ColSideColors = as.character(erCol))
```

### 3.2 Survival Analysis

An attractive feature of the vdx dataset is that it includes *survival* data for each breast cancer patient. We are not explicitly covering survival analysis in this course, but for your reference, here are the commands to create survival curves when patients are grouped by ER status and tumour grade.

```
1 > library(survival)
2 > par(mfrow = c(1, 2))
3 > plot(survfit(Surv(pData(vdx)$t.dmfs, pData(vdx)$e.dmfs) ~
4 +       pData(vdx)$er), col = c("cyan", "salmon"))
5 > plot(survfit(Surv(pData(vdx)$t.dmfs, pData(vdx)$e.dmfs) ~
6 +       pData(vdx)$grade), col = c("blue", "yellow", "orange"))
7 > survdiff(Surv(pData(vdx)$t.dmfs, pData(vdx)$e.dmfs) ~
8 +          pData(vdx)$er)
9 > survdiff(Surv(pData(vdx)$t.dmfs, pData(vdx)$e.dmfs) ~
10 +          pData(vdx)$grade)
```