Supplementary notes for **FIEmspro**

Wanchang Lin

October 16, 2007

1 Introduction

The package **FIEmspro** is a collection of R functions for the rapid analysis of flow injection fingerprints. **FIEmspro** is focusing on several aspects from fingerprint processing, data integrity checking, discrimination task and finally feature ranking. The note gives a brief explanation on how to use the main functions implemented in the package to carry out the feature ranking and classification using the provided methods.

It is aimed at supporting data analysis for researchers who already have some basic experience of R command-line usage. The package is updated regularly with regard to bug fixing, code improvement and new functions based on current research findings. For Windows and MacOS X platforms, straightforward installing can be realised via the binary packages available from the website. For any other platforms, source code can be downloaded and compiled appropriately.

Note that this document has been generated with Sweave and aims at reproducing the workflow with additional comments. In order to reduce computation time, subset of the original data signals and/or smaller number of resampling steps have used, resulting in slight differences between the outcome of the original the workflow.r and whose produced in this document.

To use the package, type in R console window:

```
>library(FIEmspro)
```

It also loads its dependent packages, for example, **randomForest**. The reference manual is supplying details on how to use the functions provided by the package, including a simple example for each function. Use the command `help` (or simply ? followed by the name of the function) to view the content about **FIEmspro** or any information regarding a function:

```
>help(package = FIEmspro)
>help(accest)
```

2 Load data into R

There are several possibilities to loading data into R:

- For dataset as an ASCII file, use `read.table` to import the data by selecting the appropriate separator in `sep`. For e.g. tab and comma separated files can be loaded this way:

```
>pos <- read.table(file = "./data/abr1/pos.txt",
+     sep = "\t")
>fact <- read.table(file = "./data/abr1/fact.csv",
+     sep = ",", header = T)
```

- For R binary dataset, typically with a `.RData` or `.rda` extension, use `load` instead:

  ```
  >load("./data/abr1.Rdata")
  ```

- When profiles are stored as ANDI NetCDF files (*.cdf), generate the fingerprint matrix according to specialized functions. Type `?fiems_lct_main` or `?fiems_ltq_main` for more information regarding the conversion protocol.

- In R, datasets can be loaded using `data`.

FIEmspro is providing a FIE-MS dataset `abr1` which is a list including two self-explanatory data frames, `pos` and `neg` corresponding the fingerprint recorded in the positive and negative ionisation modes, and a data frame, `fact` where meta-data are kept.

```
>data(abr1)
>names(abr1)

[1] "fact" "pos"  "neg"
```

A subset of the positive mode data and experimental factor matrix will enter further statistical treatments. To extract the required information, one can do the following:

```
>X <- abr1$pos[, 110:1930]
>Y <- abr1$fact
```

3 Preliminary data structure assessment

You can check `X` and `Y` dimensions using the function `dim` for matrices or `length` for vectors:

```
>dim(X)

[1]  120 1821

>dim(Y)

[1] 120   9

>length(Y$class)

[1] 120
```

Variable names in `X` can be printed out with `colnames`:

```
>colnames(X)
```

Ideally, the names should be meaningful to avoid confusions when interpreting feature ranking output for example. In this case, changing the names can be easily done with `paste`. In the following example, the name of first variable will be `pos110`, the second one, `pos111` etc...:

```
>colnames(X) <- paste("pos", 110:1100, sep = "")
```

It is also necessary to check the number of zero values in each variable. The following code sum up the number of times X=0 alongside the dimension 2 (i.e. columns):

```
>nzero <- apply(X == 0, 2, sum)
>plot(nzero, xlab = "Variable id", ylab = "Number of zeros")
```

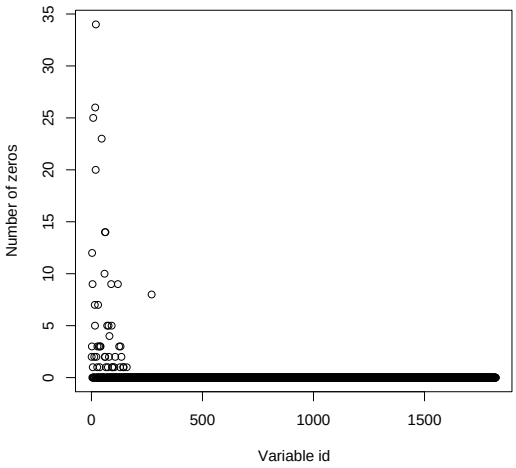

Figure 1: Number of zeros for each signals across samples.

One can keep only the variables with less than 20 zeros:

```
>X.noz <- X[, (nzero < 80)]
```

Function `table` builds a contingency table of the counts at each combination of factor levels (or discrete elements). It is possible to get some insights into the original design and interactions between to experimental factors. The distribution of 'day' and contingency table between 'rep' and 'day' can be done as follows:

```
>table(Y$day)

 1  2  3  4  5  H
20 20 20 20 20 20

>table(Y$day, Y$rep)
```

```
  1 2 3 4 5
1 4 4 4 4 4
2 4 4 4 4 4
3 4 4 4 4 4
4 4 4 4 4 4
5 4 4 4 4 4
H 4 4 4 4 4
```

To assess that factor ‘ day ’ is correctly randomised, one can plot the distribution of each level of ‘ day ’ according the order of injection (Figure 2) as well as print out, for each level, statistics related to the location and dispersion of the corresponding injection order (Table 1):

```
>plot(Y$injorder, Y$day)
>tmp <- tapply(Y$injorder, Y$day, summary)
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1	4.00	31.75	62.50	62.35	94.50	120.00
2	1.00	36.25	68.50	59.80	84.50	119.00
3	2.00	33.00	66.50	60.35	82.75	113.00
4	3.00	33.00	52.50	52.45	66.00	101.00
5	12.00	37.75	77.50	71.00	107.00	117.00
H	9.00	25.75	58.00	57.05	85.75	118.00

Table 1: Summary of injection order for each level of ’day’.

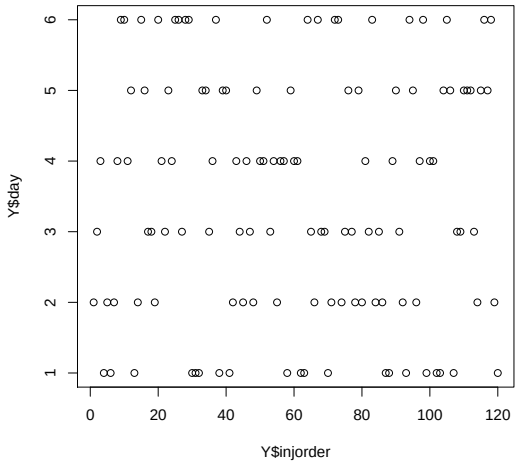

Figure 2: Position in the injection for each factor levels of ’day’.

4 Data assessment by means of Total Ion Count

By definition the total ion count of a spectrum is the sum of all the m/z signal intensities. As an easy diagnostic measure, the TIC can provide an estimation of factors that may affect the overall intensity of the run such as gradual instrument drift (e.g. resulting from loss of sensitivity of the ion source), or step changes in instrument characteristics after maintenance. For each sample, calculate the Total Ion Count (TIC) by summing up all signal intensities. This time, `apply` is summing up all the intensity for each row (dimension 1). In figure 3, sample TIC (y-axis) is plotted against sample position during the chemical analysis (x-axis):

```
>tic <- apply(X, 1, sum)
>plot(tic, xlab = "Order of Injection", ylab = "TIC")
```

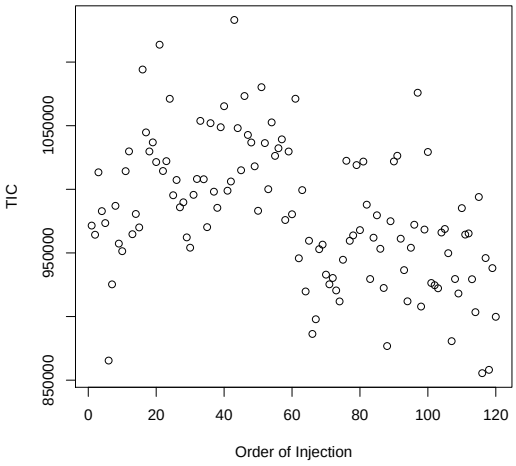

Figure 3: Sample TIC versus injection order.

A robust regression can be built to model the effect of the injection order on the TIC of each sample as shown in the function `ticstats`. As a conservative rule, any sample for which the residual deviates more than `thres=3` times from the median absolute deviation (MAD) of the residuals (e.g sample 6 in Figure 4) must be examined manually to identify the origin of the different intensity behaviour and then potentially removed before further statistical analysis if corrective measures do not improve the individual fingerprint. The list of potential outliers is given in `restic$lout`:

```
>restic <- ticstats(X, Y$injorder, thres = 3)
>print(restic$lout)
```

```
6
6
```

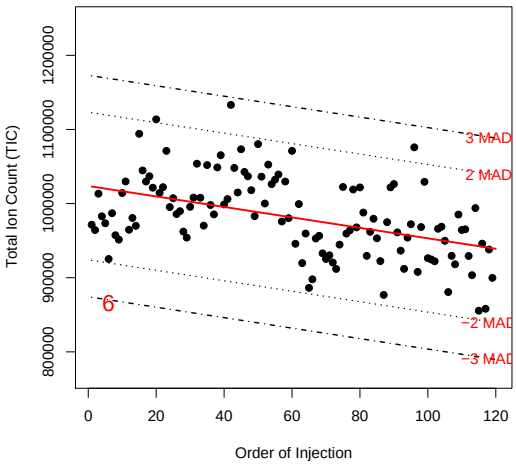

Figure 4: Detection of outlying samples by analysis of the sample TIC.

5 Baseline correction

. A simple consensual approach consists in fitting a monotone local minimum curve to each fingerprint by using FIEmspro functions `onebc` and `multibc`. Basically, the fingerprint is divided into equally spaced m/z intervals (`wsise`) and a local minimum intensity value ((`qtl`)) is returned as the baseline estimate for this region. Finally, the whole fingerprint baseline is computed by linear interpolation based on pairs made of the centre of the interval and its corresponding local minima. While it is not necessary in the context of `abr1`, the following example illustrates a baseline correction of sample `60` using the highest intensity in the lowest 10% intensities contained in window comprising `wsize=50` bins. The result of baseline correction is shown in Figure 5. For better visualisation, the maximum intensity is set to `maxy=4000`.

```
>res.onebc <- onebc(X[60, ], wsize = 50, qtl = 0.1,
+     maxy = 4000, title = F)
```

`multibc` performs baseline correction on multiple samples:

```
>resbc <- multibc(X, wsize = 50, qtl = 0.1, maxy = 4000)
>X.bc <- resbc$x
```

where `X.bc` is the new baseline corrected matrix. This function also provides two arguments to view visualization of baseline correction with a pause, `plotting` and `pause`. For example, one examine the baseline correction process for the first 10 samples with a delay of 0.5 seconds between each sample:

```
>resbc <- multibc(X[1:10, ], wsize = 50, qtl = 0.1,
+     maxy = 4000, plotting = T, pause = 0.5)
```

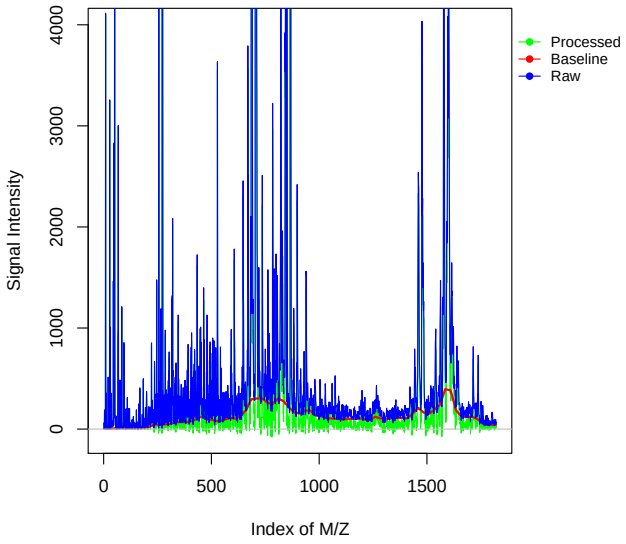

Figure 5: Illustration of the baseline correction

6 Imputation of low values

For many signals in the FIE-MS data matrix, zero values should typically concern no more than a maximum of roughly 3% of the total number of cells in the matrix. This being said, quite often imputation of very small values is more of a computational convenience (very few multivariate techniques can intrinsically accommodate missing data points) than a significant contribution to the improvement of model predictive abilities.A conservative strategy to improve data quality consists of calculating an intensity value representing a "limit of detection" signal to Ôin fillÕ cells with missing values to avoid the effect of zero values skewing further data analysis. For e.g, we can set the threshold in to 1 in the new matrix X.thr:

```
>X.thr <- abr1$pos[, 110:1930]
>X.thr[X.thr < 1] <- 1
```

An alternative solution is to adopted a non-parametric `k`-nearest neighbours based technique to impute low/missing information. To determine the efficiency of the method to obtain reasonable solutions, one can evaluate the error between imputed values obtained by artificially assigning random missing cells in the matrix and their original true values. The FIEmspro function `koptimp` is a wrapper function to determine this error for several values of `k` and several random assignment. In the code below (for illustration purposes only variables 100 to 300 are considered), `knn` based imputation is done as follows:

- data imputation is done on log transformed data and values < 1 are considered of low value.

- perform imputation with k from 1 to 10.
- 10% of low values are randomly generated to evaluate the RMSE.
- 5 iterations are performed.

```
>resimp <- koptimp(X[, 100:300], thres = 1, log.t = TRUE,
+     lk = 1:10, perc = 0.1, niter = 5)

Iteration (5): 1 2 3 4 5
```

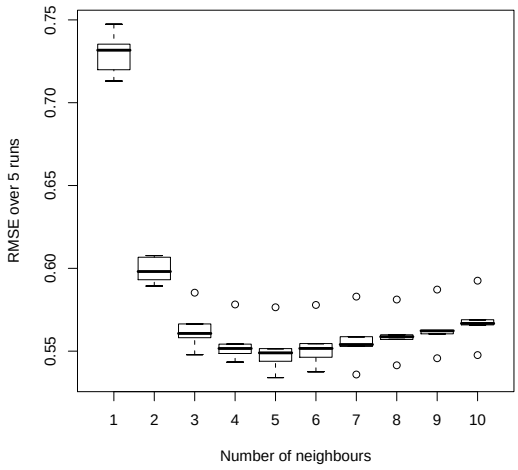

Figure 6: Imputation error for various number of neighbors.

Figure 6 depicts imputation error for various number of neighbors `k`. Characteristics of the graph elbow is the most convenient way to determine the optimum `k` (i.e. trade-off between under and overfitting). An optimum value is also calculated with `koptim` (`resimp$koptim`) to generate the imputed matrix in `resimp$x`:

```
>print(resimp$koptim)
>X.imp <- resimp$x
```

7 Logarithmic transformation

To perform a logarithmic transformation, **FIEmspro** function `preproc` with `method="log"` is equivalent to R function `log`:

```
>X <- abr1$pos[, 110:1930]
>X.log <- log10(X + 1)
>X.log <- preproc(X, method = "log10", add = 1)
```

The main advantage on using `preproc` lies several data transformation methods that can be used sequentially. For details, see `preproc` (`?preproc`).

For purposes of statistical analysis, the logarithmic transformation of signal intensity can be made in order to alleviate the dependency of the variance with the intensity. Essentially, log transformation converts multiplicative errors into additive errors so that, ideally, variance becomes more constant across the range of signal intensity on the logarithm scale and thus variance is stabilised. This illustrated by plotting the rank of median intensity against the rank of their dispersions for each signal.

Median and interquartile range ranks on the original data is computed as follows:

```
>iqr.raw <- apply(X, 2, IQR)
>med.raw <- apply(X, 2, median)
>iqr.raw <- rank(iqr.raw, na.last = T, ties.method = "random")
>med.raw <- rank(med.raw, na.last = T, ties.method = "random")
```

Idem for the log transformed data:

```
>iqr <- apply(X.log, 2, IQR)
>med <- apply(X.log, 2, median)
>iqr <- rank(iqr, na.last = T, ties.method = "random")
>med <- rank(med, na.last = T, ties.method = "random")
```

Figure 7 is produced as follows:

```
>opar <- par(mfrow = c(1, 2), pty = "m")
>xlab <- "Rank of m/z median intensity"
>ylab <- "Rank of m/z signal interquatitle range"
>plot(med.raw, iqr.raw, main = "Raw data", xlab = xlab,
+     ylab = ylab)
>plot(med, iqr, main = "Log 10 Tranformed data",
+     xlab = xlab, ylab = ylab)
>par(opar)
```

8 Normalisation to sample TIC

A common way to normalise metabolomics profile data is the use of one or more internal standards of "known" concentration, which is not possible in FIE-MS fingerprints and so the most widely used solution consists of a so called global normalisation by rescaling each measurement within a spectrum by a constant factor, such as the sum of all the spectra intensities (Total Ion Count). To do so, global normalisation to the sample TIC can be performed by `preproc` with `method="TICnorm"`.

```
>X.tic <- preproc(X, method = "TICnorm")
```

An assumption behind such normalization approaches is that the overall intensity captures a sort of average of both up and down changes in concentration related to biological treatment. To check that there is no dependency between sample TIC an a factor of interest, one can plot the sample TIC distributions for each factor levels, such day and rep:

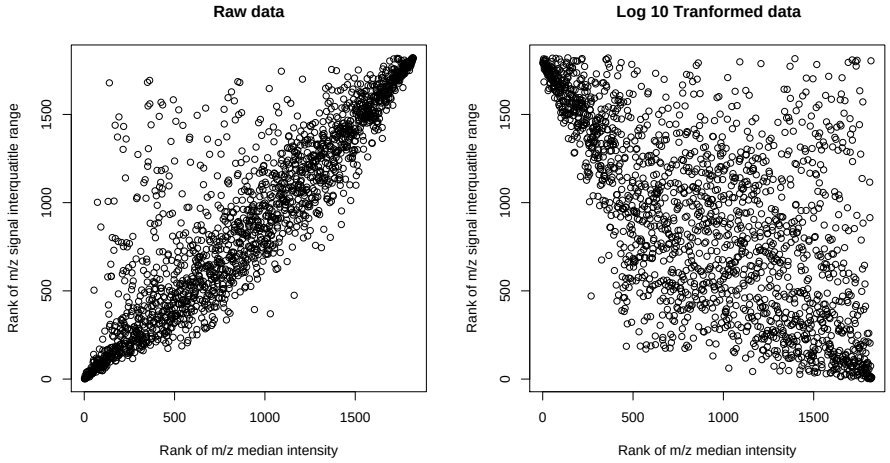

Figure 7: Effects of log transformation on the intensity dispersion dependency to its location.

```
>opar <- par(mfrow = c(1, 2), pty = "m")
>boxplot(apply(X, 1, sum) ~ Y$day, main = "Day")
>boxplot(apply(X, 1, sum) ~ Y$rep, main = "Rep")
>par(opar)
```

In this case as shown in Figure 8, yes. The normalisation needs to be corrected to ‘ day ’

```
>X.cltic <- preproc(X, method = "TICnorm", y = Y$day)
```

It does not look bad for factor rep but TIC dependency to day seems to be evident. In the following, we examine the effects of normalisation practices considering `day=3` and `day=4`. Here, we select samples corresponding to class 3 and 4 and perform no normalisation, global normalisation and class corrected normalisation:

```
>dat <- abr1$pos
>cl <- Y$day
>ind <- which(cl == 3 | cl == 4)
>cl <- cl[ind, drop = TRUE]
>levels(cl) <- c(0, 1)
>mat.1 <- dat[ind, , drop = FALSE]
>mat.2 <- preproc(mat.1, method = "TICnorm")
>mat.3 <- preproc(mat.1, method = "TICnorm", y = cl)
```

For each dataset, compute the Area under the ROC curve for discriminating classes 3 and 4:

```
>auc.1 <- apply(mat.1, 2, FIEmspro:::auc, cl)
>auc.2 <- apply(mat.2, 2, FIEmspro:::auc, cl)
>auc.3 <- apply(mat.3, 2, FIEmspro:::auc, cl)
```

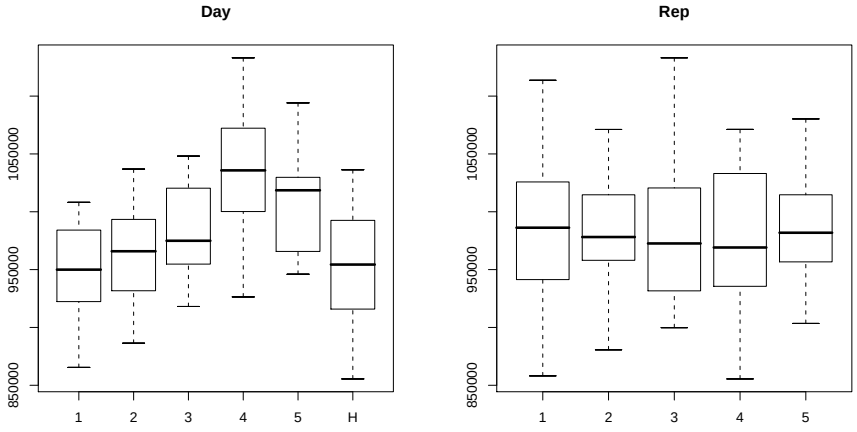

Figure 8: TIC normalisation

Figure 9 is generated as follows:

```
>opar <- par(mfrow = c(2, 2), pty = "m")
>plot(auc.1, auc.2, ylab = "AUC (normalised total data)",
+     xlab = "AUC raw data")
>plot(auc.1, auc.3, ylab = "AUC (class corrected normalised data)",
+     xlab = "AUC raw data")
>tic.1 <- apply(mat.1, 1, sum)
>tic.1 <- data.frame(tic.1[cl == 0], tic.1[cl ==
+     1])
>colnames(tic.1) <- c("3", "4")
>ylab <- "Raw Total Ion Count (TIC)"
>boxplot(tic.1, xlab = "Treatment class", ylab = ylab)
>tic.3 <- apply(mat.3, 1, sum)
>tic.3 <- data.frame(tic.3[cl == 0], tic.3[cl ==
+     1])
>colnames(tic.3) <- c("3", "4")
>ylab <- "Class corrected normalized TIC"
>boxplot(tic.3, xlab = "Treatment class", ylab = ylab)
>par(opar)
```

9 Carry out outlier detection

Despite an initial inspection of distribution of sample TIC, subsequent outliers may not be detected unless they exhibit gross intensity differences from the mean. Rather than concentrating on individual variables that could potentially lead to the rejection of all the samples, outlier detection can be approached from a different angle that consists of looking at the homogeneity of the samples in the main directions of variance. Standard methods are based on calculating

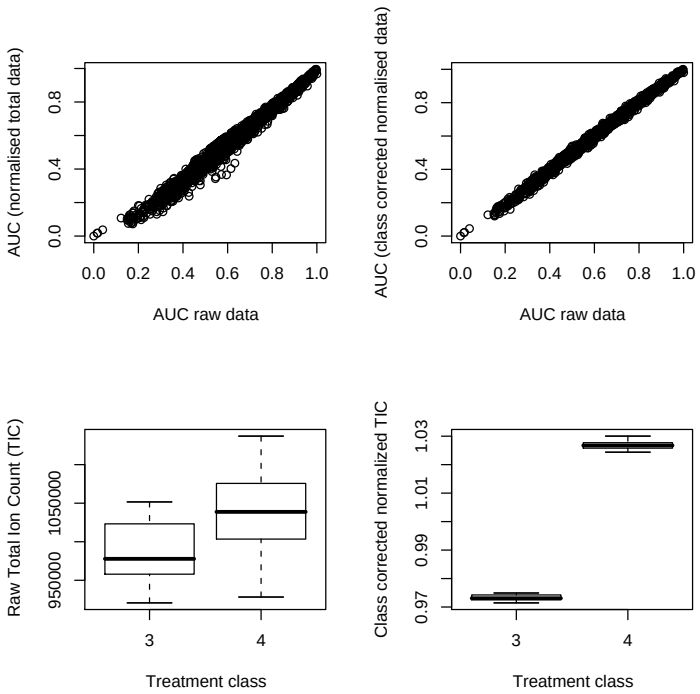

Figure 9: Effect of global and class dependant normalization on a subset of the original data.

(robust) Mahalanobis distances between each sample and the centre in a given reduced space such as that derived from Principal Component Analysis (PCA). `outl.det` is a function that output the robust square root distance from the center is displayed alongside a 2D mapping of the data and its confidence ellipse. In the following example, class 3 for outlier detection on the first two PCs: `outl.det`.

```
>X.2 <- log10(abr1$pos[, 110:1930] + 1)
>ind.y <- which(Y$day == 3)
>dat.y <- X.2[ind.y, , drop = FALSE]
>x.y <- prcomp(dat.y, scale = FALSE)$x
>resoutd.y <- outl.det(x.y[, c(1, 2)], method = "mcd",
+     dimen = c(1, 2), conf.level = 0.975)
>print(resoutd.y$outlier)

[1] 14
```

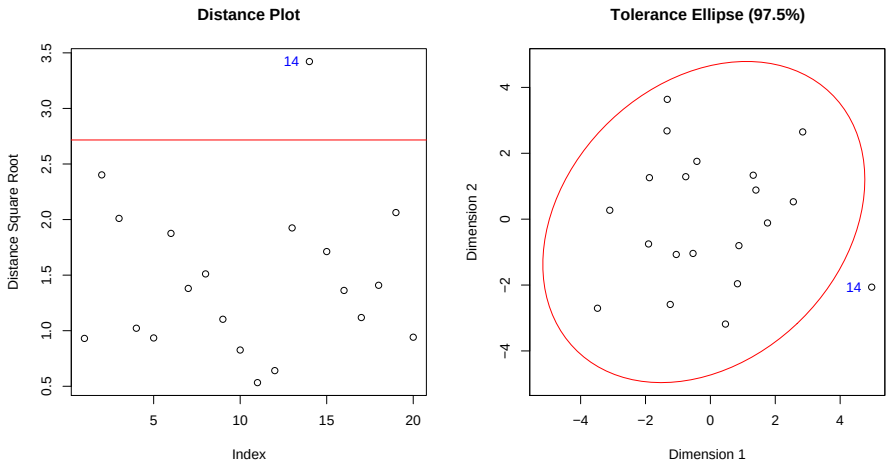

Figure 10: Sample outlier detection based on PCA analysis samples from day=3

The outlier, `resoutd.y$outlier`, is located above the cut-off line of distance plot shown in the left of Figure 10 and outside of confidential boundary of ellipse plot shown in the right of Figure 10.

10 Multivariate analysis on the training data

Principal Components Analysis. An initial inspection of the sample grouping alongside the axis of variability can be realised by Principal Components Analysis. the percentage of explained variance is calculated in the variable `vars` and `grpplot` creates a scatter plot by group. Figure 11 depicts the PC1 vs PC2 with percentage of variance and data points are coloured according the factor day: shown in the axies.

```
>X <- preproc(X, method = "TICnorm")
>X.pca <- prcomp(X, scale = F)
>vars <- X.pca$sdev^2
>vars <- vars/sum(vars)
>names(vars) <- colnames(X.pca$rotation)
>print(vars[1:5])

       PC1        PC2        PC3        PC4        PC5
0.64244972 0.20005760 0.04445056 0.02671619 0.01765645

>pc1 <- round(vars[1] * 100, 2)
>pc2 <- round(vars[2] * 100, 2)
>grpplot(X.pca$x[, 1:2], Y$day, legend.loc = "rightside",
+     ylab = paste("PC2", " (", pc2, "%)", sep = ""),
+     xlab = paste("PC1", " (", pc1, "%)", sep = ""))
```

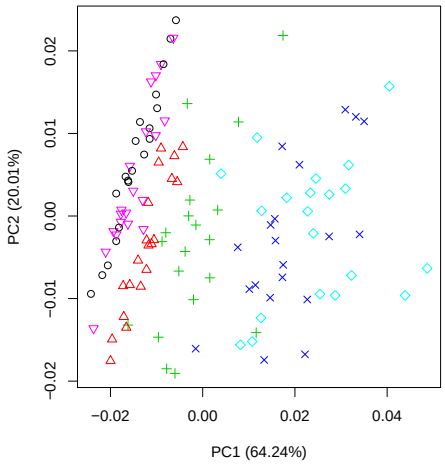

Figure 11: PCA analysis

Linear Discriminant Analysis. The function `nlda` in **FIEmspro** implements linear discriminant analysis for high dimensional problems that do not require any special parameter optimisation. Commands for building a model based on all the data for training and summarising NLDA associated statistics and training data confusion matrix are given below:

```
>ldamod <- nlda(X, Y$day)
>print(ldamod$st)

       Eig Perceig Cancor
DF1 12.840  71.902  0.963
DF2  2.779  15.564  0.858
```

```
DF3  1.728   9.674  0.796
DF4  0.321   1.798  0.493
DF5  0.190   1.062  0.399
```

```
>print(ldamod$conf)
```

```
   pred
cl   1  2  3  4  5  H
  1 17  0  0  0  0  3
  2  0 20  0  0  0  0
  3  0  0 20  0  0  0
  4  0  0  0 19  1  0
  5  0  0  0  1 19  0
  H  2  0  0  0  0 18
```

Score plots of every possible discriminant functions can be done by not supplying any argument `dimen`. In the following example, only DF 1 and 2 are considered.

```
>plot(ldamod)
```

or plot only the first two dimensions

```
>plot(ldamod, dimen = c(1, 2))
```

Plot the HCA between group centres.

```
>hca.nlda(ldamod)
```

Random Forest. A useful multivariate discrimination available in R is Random Forest form the package **ranfomForest**. The call for RF is quite simple:

```
>rfmod <- randomForest(X, Y$day, ntree = 1000)
```

A couple of meaningful metrics can be derived from the model built on all the data. First, the confusion matrix calculated from out of bag predictions is a fairly good approximation of the discrimination power:

```
>print(rfmod$confusion)
```

```
   1  2  3  4  5  H class.error
1 17  0  0  0  0  3        0.15
2  0 19  1  0  0  0        0.05
3  0  2 18  0  0  0        0.10
4  0  0  0 20  0  0        0.00
5  0  0  0  2 18  0        0.10
H  5  0  1  0  0 14        0.30
```

Secondly, a finer inspection of individual sample prediction margin can bring a greater insight into the confidence of the classification. In the following, one can check the class error at each iteration to verify that the property that RF does not overfit. In the case that errors do not converge, more trees must be built in the model (argument `ntree`). Sample margin of predictions are also plotted to identify misclassified or weakly discriminated samples.

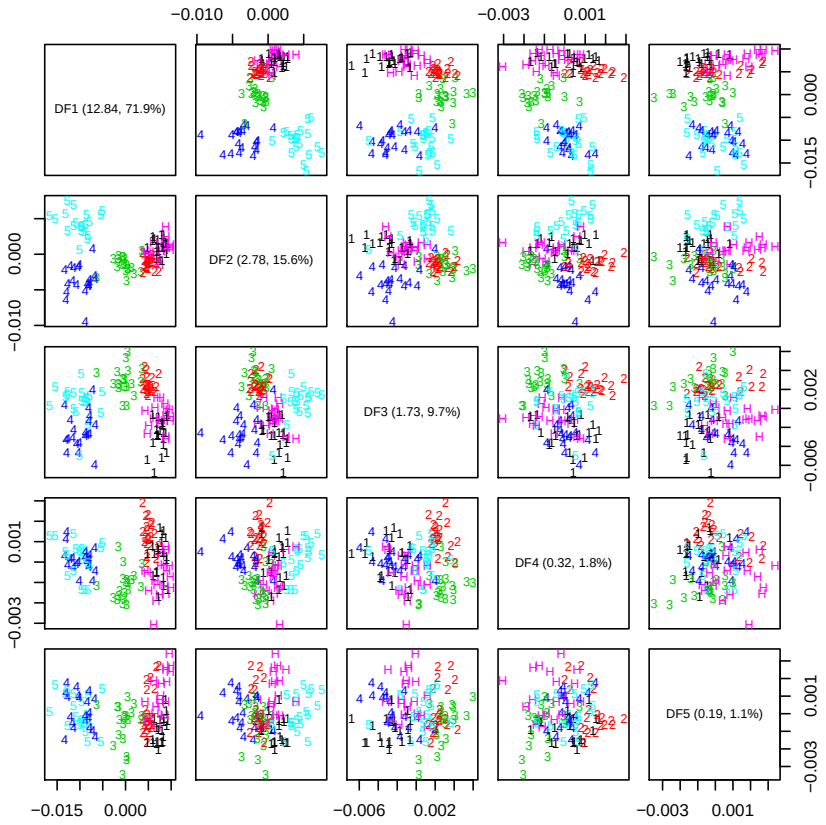

Figure 12: Plotting LDA scores

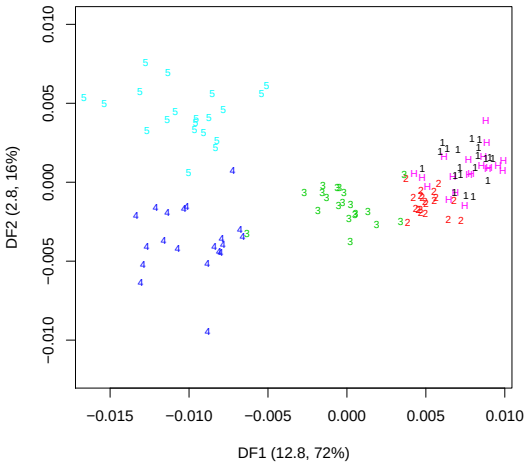

Figure 13: Plotting LDA scores in the first two dimensions

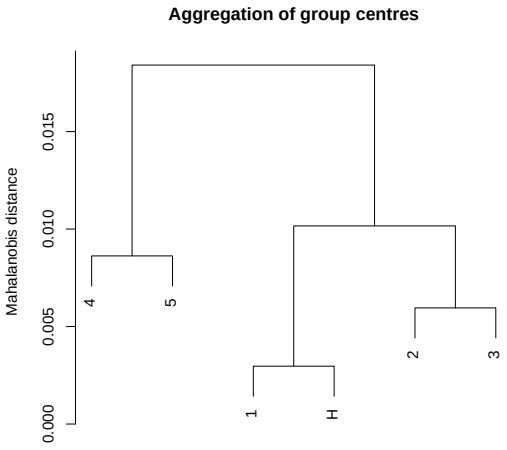

Figure 14: HCA of the distances between group centres.

```
>opar <- par(mfrow = c(1, 2), pty = "m")
>plot(rfmod, main = "Class Error in the Random Forest model")
>plot(margin(rfmod, Y$day), sort = FALSE, main = "Sample Margin in the RF model")
>print(mean(margin(rfmod, Y$day)))
```

```
[1] 0.3023432
```

```
>par(opar)
```

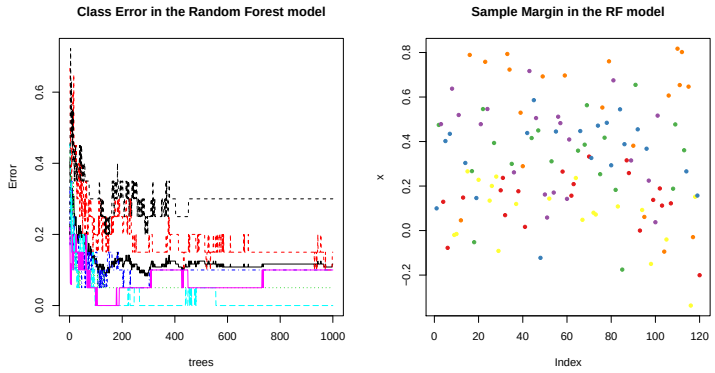

Figure 15: Plotting randomForest: error and margin

The following example is the binary comparison between class 1 and H, class 2 and 5, and class 3 and 4 using random forest classifier. Function `dat.sel` implemented in **FIEmspro** generates pairwise data set based on class labels.

```
>dat.sub <- dat.sel(X, Y$day, choices = list(c("1",
+     "H"), c("2", "5"), c("3", "4")))
>n <- nrow(dat.sub$com)
>com <- apply(dat.sub$com, 1, paste, collapse = "~")
>marg <- sapply(1:n, function(x) {
+     val <- randomForest(dat.sub$dat[[x]], dat.sub$cl[[x]],
+         ntree = 1000)
+     mean(margin(val, dat.sub$cl[[x]]))
+ })
>names(marg) <- com
>marg <- data.frame(Margin = marg)
>marg
```

```
       Margin
1~H 0.1076623
2~5 0.8876986
3~4 0.6302070
```

11 Multivariate discrimination results assessment

Two wrapper functions, `accest` and `accest.1` are available in **FIEmspro** to assess classification estimates based on re-sampling procedures. They can be em-

ployed to both multi-class and two-class discrimination. The former works with some generic classifiers that fulfil R standards to define predictive techniques such as the ones available in packages like **MASS** (`lda` and `qda`), **e1071**(`svm`) or **randomForest**(`randomForest`) and `nlda`. The latter works with user defined classifiers. Both of them require as input `valipars` and `trainind` objects that provide information regarding re-sampling strategy and the actual data partitioning.

An example of multiple class problem is given below. 10 stratified 5-fold stratified cross-validation (CV) resampling strategy is done as follows

```
>pars <- valipars(sampling = "cv", niter = 2, nreps = 4,
+     strat = T)
```

The index of training samples based on the re-sampling method is generated by:

```
>tr.idx <- trainind(Y$day, pars)
```

The main advantage of generating data partitioning that way is that `tr.idx` can be re-used with different classifiers or saved to be reused later for reproducibility purposes. The validation with re-sampling procedure, in which the classifier is SVM with linear kernel can be done as follows:

```
>svm.res <- accest(X, Y$day, clmeth = "svm", pars = pars,
+     tr.idx = tr.idx, kernel = "linear")

ACCEST.1 Iteration (2): 1 2

>svm.res

Method:                 svm
Arguments:         kernel=linear
Discrimination:         1~2~3~4~5~H

No. of iteration:         2
Sampling:                 cross validation
No. of replications:         4

Accuracy:                 0.9
Margin:                         0.435

Overall confusion matrix of training data:
      all.pred
all.cl  1  2  3  4  5  H
     1 35  0  0  0  0  5
     2  0 40  0  0  0  0
     3  0  4 35  1  0  0
     4  0  0  1 37  2  0
     5  0  0  0  3 37  0
     H  5  3  0  0  0 32
```

In a batch-process involving more than one classifier (for e.g `randomForest` and `nlda`), a simple set of code lines could be (note that as a partitioning is generated each time, results are not strictly comparable) :

```
>method <- c("randomForest", "nlda")
>res <- lapply(method, function(m) {
+     accest(X, Y$day, clmeth = m, pars = pars,
+         tr.idx = tr.idx, ntree = 1000)
+ })

ACCEST.1 Iteration (2): 1 2
ACCEST.1 Iteration (2): 1 2

>names(res) <- method
>res

$randomForest
Method:                 randomForest
Arguments:          ntree=1000
Discrimination:         1~2~3~4~5~H

No. of iteration:         2
Sampling:                 cross validation
No. of replications:         4

Accuracy:                 0.908
Margin:                         0.283

Overall confusion matrix of training data:
      all.pred
all.cl  1  2  3  4  5  H
     1 34  0  0  0  0  6
     2  0 38  2  0  0  0
     3  0  3 37  0  0  0
     4  0  0  0 40  0  0
     5  0  0  0  4 36  0
     H  6  1  0  0  0 33

$nlda
Method:                 nlda
Arguments:          ntree=1000
Discrimination:         1~2~3~4~5~H

No. of iteration:         2
Sampling:                 cross validation
No. of replications:         4

Accuracy:                 0.825
Margin:                         0.64

Overall confusion matrix of training data:
      all.pred
all.cl  1  2  3  4  5  H
     1 29  0  0  0  0 11
     2  0 33  6  0  0  1
```

```
3  0  3 33  2  2  0
4  0  0  1 36  3  0
5  0  0  0  2 38  0
H 10  1  0  0  0 29
```

For two class problem, both `accest` and `accest.1` will output additional information such as classifier AUC. The following code lines perform multivariate analysis on a three two-class problem using two classifiers. As a result, we are interested in both classifier AUC and accuracies (note that as a partitioning is generated each time, results are not strictly comparable) :

```
>val.rf.nlda <- do.call("cbind", lapply(method,
+     function(y) {
+         func.cl <- function(m) {
+             acc.auc <- sapply(c(1:n), function(x) {
+                 val <- accest(dat.sub$dat[[x]],
+                   dat.sub$cl[[x]], clmeth = m,
+                   pars = pars, ntree = 1000)
+                 return(c(val$acc, val$auc))
+             })
+             acc.auc <- t(acc.auc)
+             rownames(acc.auc) <- com
+             colnames(acc.auc) <- c(paste(m, ".acc",
+                 sep = ""), paste(m, ".auc", sep = ""))
+             acc.auc
+         }
+         func.cl(y)
+     }))

ACCEST.1 Iteration (2): 1 2
ACCEST.1 Iteration (2): 1 2
ACCEST.1 Iteration (2): 1 2
ACCEST.1 Iteration (2): 1 2
ACCEST.1 Iteration (2): 1 2
ACCEST.1 Iteration (2): 1 2

>print(val.rf.nlda)

    randomForest.acc randomForest.auc nlda.acc nlda.auc
1~H           0.8500             0.91    0.725    0.810
2~5           1.0000             1.00    1.000    1.000
3~4           0.9875             1.00    0.950    0.995
```

12 Feature Ranking

FIEmspro provides several feature selection or ranking methods, such as ANOVA, Random Forest and AUC. Some methods can be applied to both multiple and binary class problems, but some only to two-class problem. For more details, see `?fs.techniques`.

Simple feature ranking example. An example of feature selection by AUC based on the whole data is

```
>fs.1H <- fs.auc(dat.sub$dat[["1~H"]], dat.sub$cl[["1~H"]])
```

The top 20 features are

```
>fs.1H$fs.order[1:20]
```

```
 [1] P236  P274  P141  P371  P257  P130  P1812 P712  P232
[10] P187  P565  P615  P762  P210  P717  P166  P890  P966
[19] P536  P224
```

Re-sampling based feature ranking. The feature selection methods implemented in package **FIEmspro** can be validated by the re-sampling strategies. This is fullfiled by the function `feat.rank.se`. Just like using classification validation performed by `accest` or `accest.1`, data partition strategies are need to be prepared before using `feat.rank.se`.

The following is an example for batch-processing of feature ranking for binary comparison, `1 ~ H`, `2 ~ 5` and `3 ~ 4` using three methods, randomForest(`fs.rf`), AUC(`fs.auc`) and ANOVA (`fs.anova`). The re-sampling method is randomised validation(`rand`). The number of iteration is 1 and there are 30 repeats in each iteration.

```
>rank.method <- c("fs.rf", "fs.auc", "fs.anova")
>pars <- valipars(sampling = "rand", niter = 1,
+     nreps = 3)
>fs.all <- lapply(1:n, function(i) {
+     func.fs <- function(x, y) {
+         res <- lapply(rank.method, function(m) feat.rank.re(x,
+             y, method = m, pars = pars))
+         names(res) <- rank.method
+         fs.rank <- sapply(res, function(x) x$fs.rank)
+         fs.stats <- sapply(res, function(x) x$fs.stats)
+         fs.order <- sapply(res, function(x) x$fs.order)
+         list(fs.rank = fs.rank, fs.order = fs.order,
+             fs.stats = fs.stats)
+     }
+     func.fs(dat.sub$dat[[i]], dat.sub$cl[[i]])
+ })
```

```
Iteration (1): 1
Iteration (1): 1
Iteration (1): 1
Iteration (1): 1
Iteration (1): 1
Iteration (1): 1
Iteration (1): 1
Iteration (1): 1
Iteration (1): 1
```

```
>(names(fs.all) <- com)
```

```
[1] "1~H" "2~5" "3~4"
```

```
>names(fs.all$"1~H")
```

```
[1] "fs.rank"  "fs.order" "fs.stats"
```

where `fs.rank`, `fs.order` and `fs.stats` are the feature ranks, feature orders and feature statistics or values. Using these results, user can construct or generates some specific results. For example, to generate a ranking table based on feature statistics or values, it is done by

```
>fs.tab <- lapply(names(fs.all), function(x) {
+     feat.tab <- function(fs.stats) {
+         mz <- rownames(fs.stats)
+         rank.tab <- lapply(as.data.frame(fs.stats),
+             function(x) {
+                 x <- round(x, digits = 4)
+                 df <- data.frame(mz = mz, values = x)
+                 df <- df[order(df$values, decreasing = TRUE),
+                   ]
+                 rownames(df) <- 1:nrow(df)
+                 df
+             })
+         rank.tab <- do.call("cbind", rank.tab)
+         rank.tmp <- colnames(rank.tab)
+         rank.tmp <- sapply(rank.tmp, function(x) substr(x,
+             4, 100))
+         colnames(rank.tab) <- rank.tmp
+         return(rank.tab)
+     }
+     feat.tab(fs.all[[x]]$fs.stats)
+ })
>names(fs.tab) <- com
>fs.tab$"1~H"[1:20, ]
```

```
   rf.mz rf.values auc.mz auc.values anova.mz anova.values
1   P274    0.0076   P236     0.9201     P236      22.8921
2   P141    0.0044   P371     0.8803     P274      19.4212
3   P130    0.0037   P274     0.8764     P257      17.3141
4   P236    0.0032   P141     0.8683     P141      14.9097
5   P224    0.0027  P1409     0.8304    P1812      12.8630
6  P1812    0.0025   P257     0.8245    P1125      11.6568
7   P232    0.0021   P130     0.8130     P890      10.8452
8   P802    0.0021   P232     0.8108     P305      10.5780
9   P257    0.0020   P762     0.8067     P371      10.5714
10 P1440    0.0017   P717     0.8066    P1444      10.0517
11 P1055    0.0015   P210     0.8051    P1128       9.7809
12  P305    0.0013  P1930     0.8049     P712       9.6497
13 P1236    0.0012   P471     0.7986     P717       9.6298
14  P371    0.0011   P615     0.7953    P1492       9.3813
15 P1409    0.0011   P712     0.7947     P615       9.0500
```

```
16  P118    0.0010   P510     0.7944     P714       9.0029
17  P187    0.0010   P187     0.7929     P224       8.8891
18  P222    0.0009  P1812     0.7929     P438       8.7753
19 P1378    0.0009   P472     0.7913     P130       8.7100
20  P194    0.0008   P536     0.7913     P232       8.6489
```

Or plot the random forest importance score in order to decide a significance threshold for variables with adequate explanatory power via

```
>par(mfrow = c(1, 1))
>rf.1H <- sort(as.data.frame(fs.all[["1~H"]]$fs.stats)[["fs.rf"]],
+     decreasing = T)
>rf.25 <- sort(as.data.frame(fs.all[["2~5"]]$fs.stats)[["fs.rf"]],
+     decreasing = T)
>rf.34 <- sort(as.data.frame(fs.all[["3~4"]]$fs.stats)[["fs.rf"]],
+     decreasing = T)
>rf.scores <- data.frame(rf_1_H = rf.1H, rf_2_5 = rf.25,
+     rf_3_4 = rf.34)
>n <- ncol(rf.scores)
>oldpar <- par(mar = c(5, 6, 4, 2))
>if (require("plotrix", quietly = TRUE)) {
+     matplot(rf.scores[1:60, ], type = "n", col = 1:n,
+         lty = 1:n, pch = 1:n, xlab = "Variable rank",
+         ylab = "", las = 1)
+     gradient.rect(1, 0.001, 60, 0.003, col = gray(100:60/100),
+         gradient = "y", border = NA)
+     oldpar <- par(new = TRUE)
+ }
>matplot(rf.scores[1:60, ], type = "o", col = 1:n,
+     lty = 1:n, pch = 1:n, xlab = "Variable rank",
+     ylab = "", las = 1)
>legend("topright", inset = 0.02, c("1_H", "2_5",
+     "3_4"), lty = 1:n, col = 1:n, pch = 1:n, box.lty = 0)
>mtext(2, text = "RF importance scores", line = 4)
>par(oldpar)
```

In addition, **FIEmspro** provides two functions for post-processing the feature selection with re-sampling. One is `fs.mrpval` which computes the pseudo mrp-value and another is `fs.summary` for aggregating results obtained from the re-sampling based feature selection.

Here is the example of feature ranking by Welch T-test based on bootstrap

```
>pars <- valipars(sampling = "boot", niter = 1,
+     nreps = 10)
>wel.1H.re <- feat.rank.re(dat.sub$dat[["1~H"]],
+     dat.sub$cl[["1~H"]], "fs.welch", pars)
```

```
Iteration (1): 1
```

Compute stability mr-p value using the 75% worst features as irrelevant

```
>wel.1H.re.mrp <- fs.mrpval(wel.1H.re, 0.25)
```

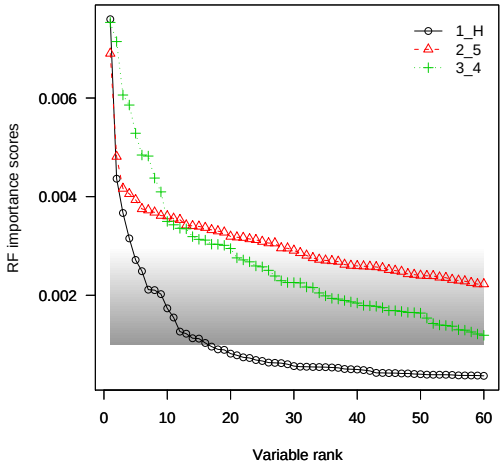

Figure 16: Random Forest importance scores

Summarise the re-sampling based ranking and correct the original p values by FDR

```
>wel.1H <- fs.summary(wel.1H.re, wel.1H.re.mrp,
+     padjust = "fdr")
```

The top twenty feature selected are listed in Table 2.

	fs.welch	OriRk	pval	pval.ad	AvgRk	SdevRk	mrp-0.25
P236	5.93	1	1.72e−06	3.13e−03	7.50	12.62	1.46e−04
P257	4.66	2	4.40e−05	3.27e−02	28.30	32.79	1.17e−03
P274	4.55	3	5.39e−05	3.27e−02	33.00	59.48	1.46e−03
P141	4.51	4	1.03e−04	3.37e−02	36.40	41.90	2.20e−03
P615	3.50	5	1.62e−03	1.74e−01	39.20	29.07	2.27e−03
P187	3.73	6	6.25e−04	1.42e−01	56.60	78.71	4.39e−03
P371	4.40	7	1.11e−04	3.37e−02	72.60	80.06	6.00e−03
P1444	2.93	8	5.64e−03	2.80e−01	72.90	77.36	6.00e−03
P1930	3.41	9	1.73e−03	1.75e−01	76.10	59.05	6.52e−03
P1812	4.33	10	1.07e−04	3.37e−02	78.20	75.69	6.73e−03
P305	3.36	11	2.06e−03	1.98e−01	84.70	80.42	7.83e−03
P438	3.26	12	2.40e−03	2.08e−01	90.70	75.89	9.15e−03
P1409	3.47	13	1.32e−03	1.63e−01	98.10	73.87	1.13e−02
P717	3.56	14	1.03e−03	1.63e−01	99.70	151.35	1.16e−02
P1492	3.02	15	4.55e−03	2.77e−01	108.30	106.67	1.34e−02
P427	2.96	16	5.33e−03	2.77e−01	113.70	140.42	1.44e−02
P445	2.96	17	5.69e−03	2.80e−01	114.60	103.71	1.47e−02
P802	2.97	18	5.22e−03	2.77e−01	115.40	114.73	1.49e−02
P472	3.03	19	4.62e−03	2.77e−01	117.20	115.71	1.52e−02
P1010	2.63	20	1.25e−02	3.82e−01	118.90	149.16	1.55e−02

Table 2: Results of feature selection using Welch t-test with bootstrap validation

Contents