Species Identification using MALDIquant

Sebastian Gibb[*] and Korbinian Strimmer [†]

October 13, 2014

Abstract

This vignette describes how to use MALDIquant for species identification.

Contents

[*] mail@sebastiangibb.de

[†] k.strimmer@imperial.ac.uk

1 Foreword

MALDIquant is free and open source software for the R (R Core Team, 2014) environment and under active development. If you use it, please support the project by citing it in publications:

> Gibb, S. and Strimmer, K. (2012). MALDIquant: a versatile R package for the analysis of mass spectrometry data. *Bioinformatics*, 28(17):2270–2271

If you have any questions, bugs, or suggestions do not hesitate to contact me (mail@sebastiangibb.de).
Please visit http://strimmerlab.org/software/maldiquant/.

2 Dataset

The dataset we use in this vignette was kindly provided by Dr. Bryan R. Thoma (bryanthoma@yahoo.com). It contains spectra of four different bacteria species. Each species is represented by eight individual samples and each sample has three technical replicates.

3 Analysis

3.1 Setup

First we need to install the necessary packages (you can skip this part if you have already done this). You can install MALDIquant (Gibb and Strimmer, 2012), MALDIquantForeign (Gibb, 2014), pvclust (Suzuki and Shimodaira, 2011), sda (Ahdesmäki and Strimmer, 2010) and crossval (Strimmer, 2014) directly from CRAN. To install this data package from http://github.com/sgibb/MALDIquantExamples you need the devtools (Wickham and Chang, 2014) package.

```r
install.packages(c("MALDIquant", "MALDIquantForeign", "pvclust",
                   "sda", "crossval", "pvclust", "devtools"))
library("devtools")
install_github("sgibb/MALDIquantExamples")
```

Next we load the packages.

```
library("MALDIquant")
library("MALDIquantForeign")
library("pvclust")
library("sda")
library("crossval")

library("MALDIquantExamples")
```

3.2 Import Raw Data

We use the `getPathSpecies` function to get the correct local file path to the spectra.

```
spectra <- import(getPathSpecies(), verbose=FALSE)
```

We do a basic quality control and test whether all spectra contain the same number of data points and are not empty.

3.3 Quality Control

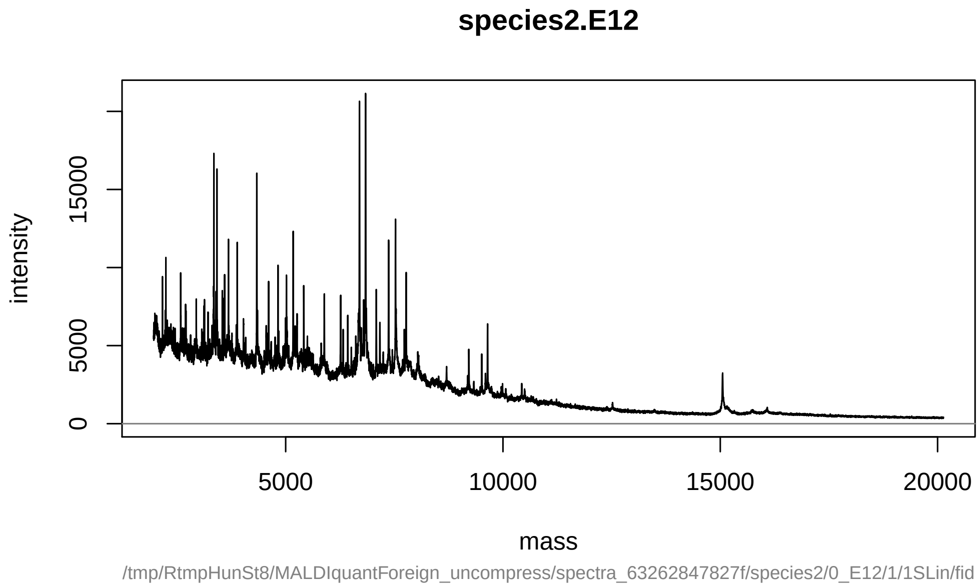
```
table(sapply(spectra, length))


20882
   96

any(sapply(spectra, isEmpty))

[1] FALSE

all(sapply(spectra, isRegular))

[1] TRUE
```

Subsequently we ensure that all spectra have the same mass range.

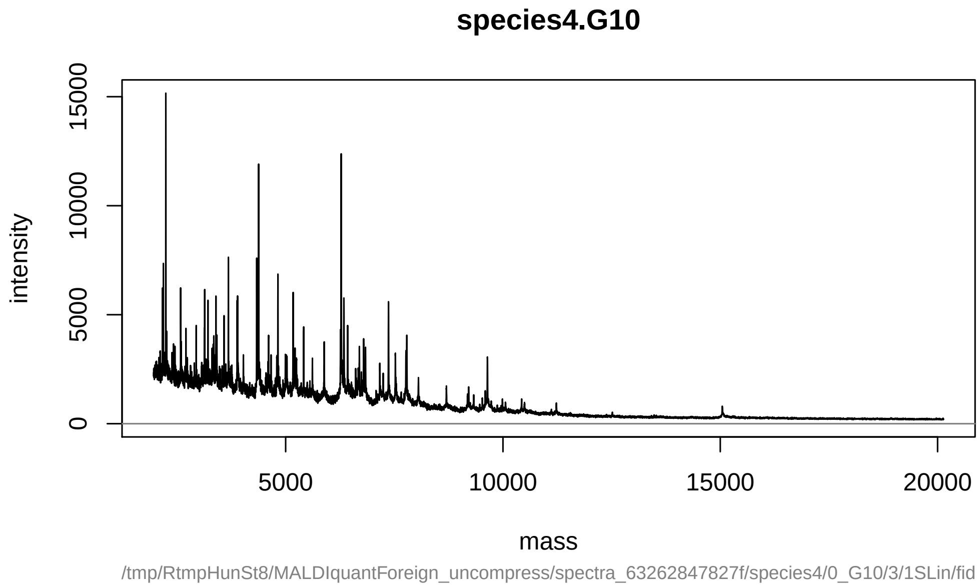
```
spectra <- trim(spectra)
```

Finally we draw some plots and inspect the spectra visually.

```
idx <- sample(length(spectra), size=2)
plot(spectra[[idx[1]]])
```



```
plot(spectra[[idx[2]]])
```



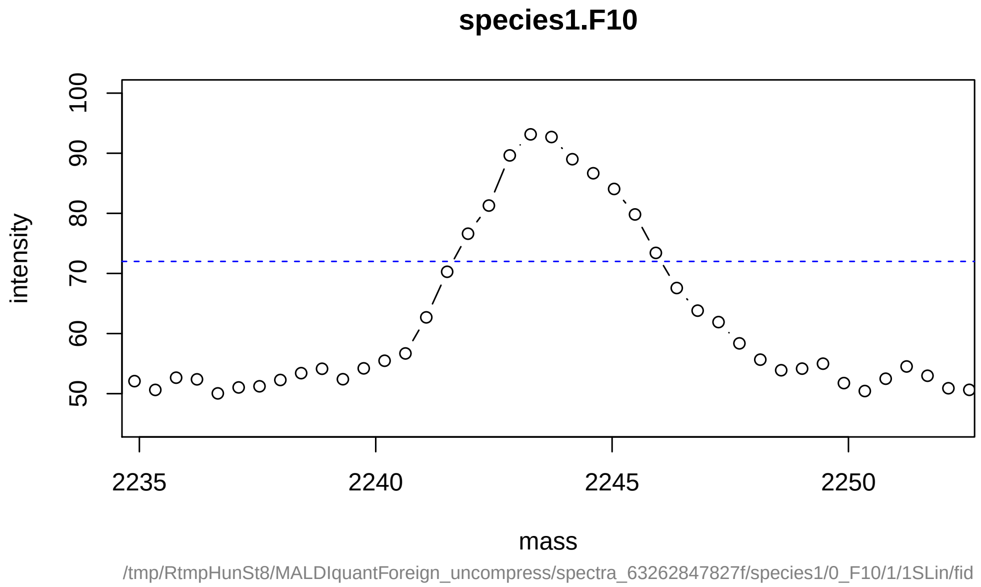
3.4 Transformation and Smoothing

We apply the square root transformation to simplify graphical visualization and to overcome the potential dependency of the variance from the mean.

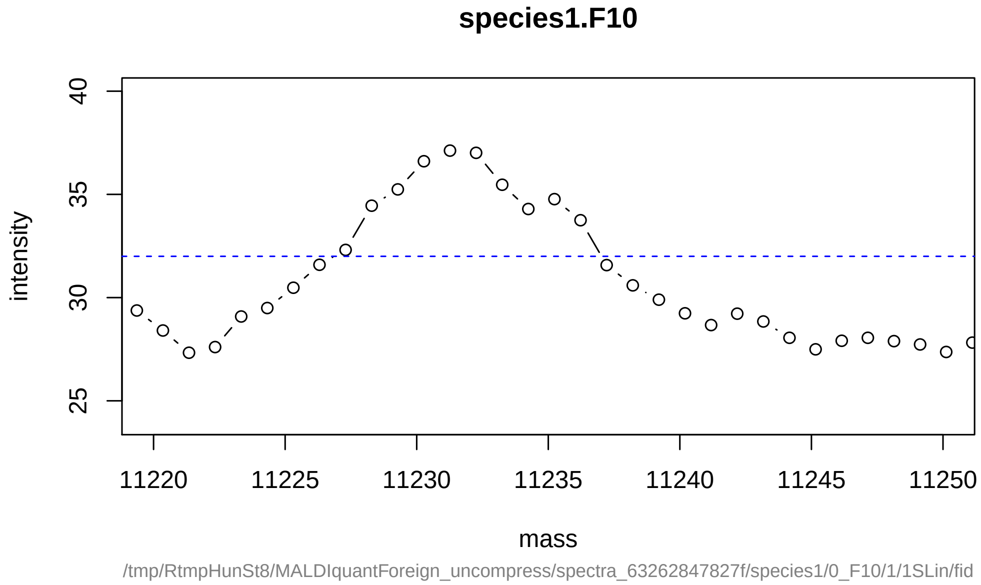
```
spectra <- transformIntensity(spectra, method="sqrt")
```

In the next step we want to smooth our spectra with the *Savitzky-Golay-*Filter (Savitzky and Golay, 1964). According to Bromba and Ziegler (1981) the best `halfWindowSize` should be smaller than the *FWHM* (full width at half maximum) of the peaks. We add the argument `type="b"` to the `plot` command to show both lines and data points in our plots. We count the data points in a few different regions of some spectra to estimate the average *FWHM* (of course this is not the most sophisticated method). In the figure below we consider all points above the dashed blue line and get a *FWHM* around 10-12 data points. We choose `halfWindowSize=10`.

```
plot(spectra[[1]], type="b",
     xlim=c(2235.3, 2252.0), ylim=c(45, 100))
abline(h=72, col=4, lty=2)
```



```
plot(spectra[[1]], type="b",
     xlim=c(11220, 11250), ylim=c(24, 40))
abline(h=32, col=4, lty=2)
```



Afterwards we apply a 21 (2*halfWindowSize+1) point *Savitzky-Golay*-Filter (Savitzky and Golay, 1964) to smooth the spectra.

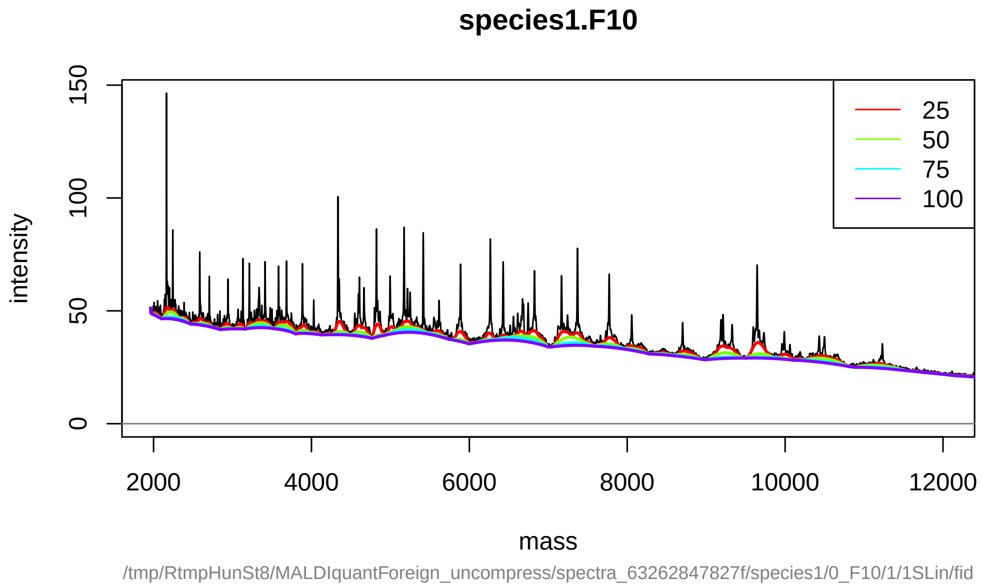
```r
spectra <- smoothIntensity(spectra, method="SavitzkyGolay",
                           halfWindowSize=10)
```

3.5 Baseline Correction

Matrix effects and chemical noise results in some background noise. That's why we have to apply a baseline correction. In this example we use the *SNIP* algorithm (Ryan et˜al., 1988) to correct the baseline.

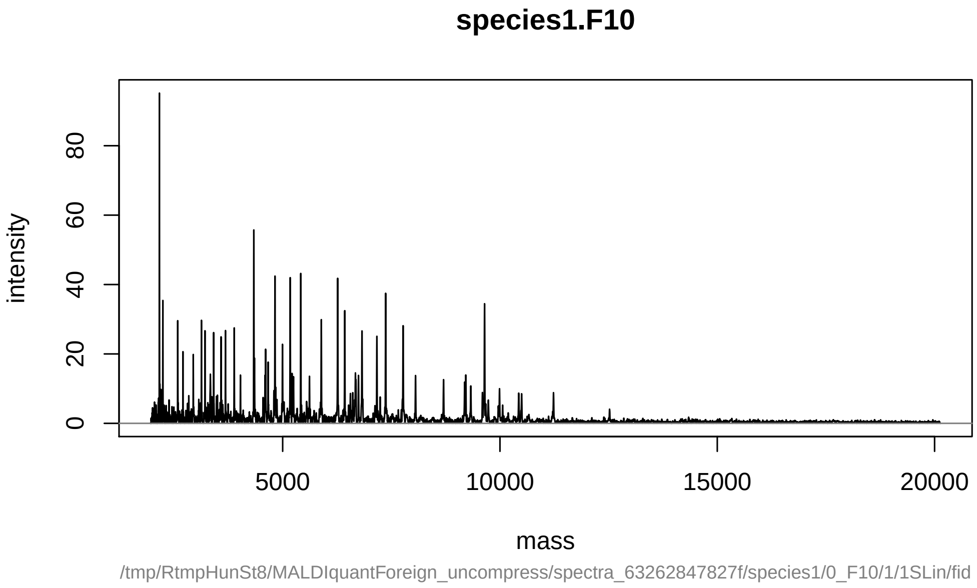
Similar to the problem of the halfWindowSize in section 3.4 we need to choose a halfWindowSize respectively number of iterations for the baseline correction algorithm as well. The baseline should be flexible enough to follow trends but must not reduce the high of the peaks. We simply try a few different numbers of iterations.

```r
## define iteration steps: 25, 50, ..., 100
iterations <- seq(from=25, to=100, by=25)
## define different colors for each step
col <- rainbow(length(iterations))

plot(spectra[[1]], xlim=c(2000, 12000))

## draw different baseline estimates
for (i in seq(along=iterations)) {
  baseline <- estimateBaseline(spectra[[1]], method="SNIP",
                               iterations=iterations[i])
  lines(baseline, col=col[i], lwd=2)
}

legend("topright", legend=iterations, col=col, lwd=1)
```



25 `iterations` are already very flexible but 50 is not flexible enough and the height of the peaks is not reduced very much. So we choose `iterations=25` for the baseline removal.

```
spectra <- removeBaseline(spectra, method="SNIP",
                          iterations=25)
plot(spectra[[1]])
```



species1.F10

/tmp/RtmpHunSt8/MALDIquantForeign_uncompress/spectra_63262847827f/species1/0_F10/1/1SLin/fid

3.6 Intensity Calibration

We perform the *Total-Ion-Current*-calibration (TIC; often called normalization) to equalize the intensities across spectra.

```
spectra <- calibrateIntensity(spectra, method="TIC")
```

3.7 Alignment

Next we need to (re)calibrate the mass values. Our alignment procedure is a peak based warping algorithm. `MALDIquant` offers `alignSpectra` as a wrapper around more complicated functions. If you need a finer control or want to investigate the impact of different parameters please use `determineWarpingFunctions` instead (see `?determineWarpingFunctions` for details).

```
spectra <- alignSpectra(spectra)
```

We want to average the technical replicates before we are looking for peaks. Our spectra are recorded thrice for each spot. That's why we average each spot. We get the spot information using the `metaData` method.

```
metaData(spectra[[1]])$spot

[1] "F10"
```

We collect all spots with a `sapply` call (to loop over all spectra) and use this information to create our average spectra. Because some species are measured in different runs on the same spot location we also add the species name to average only corresponding spectra.

```
spots <- sapply(spectra, function(x)metaData(x)$spot)
species <- sapply(spectra, function(x)metaData(x)$sampleName)
head(spots)

[1] "F10" "F10" "F10" "F11" "F11" "F11"

head(species)

[1] "species1" "species1" "species1" "species1" "species1" "species1"
```

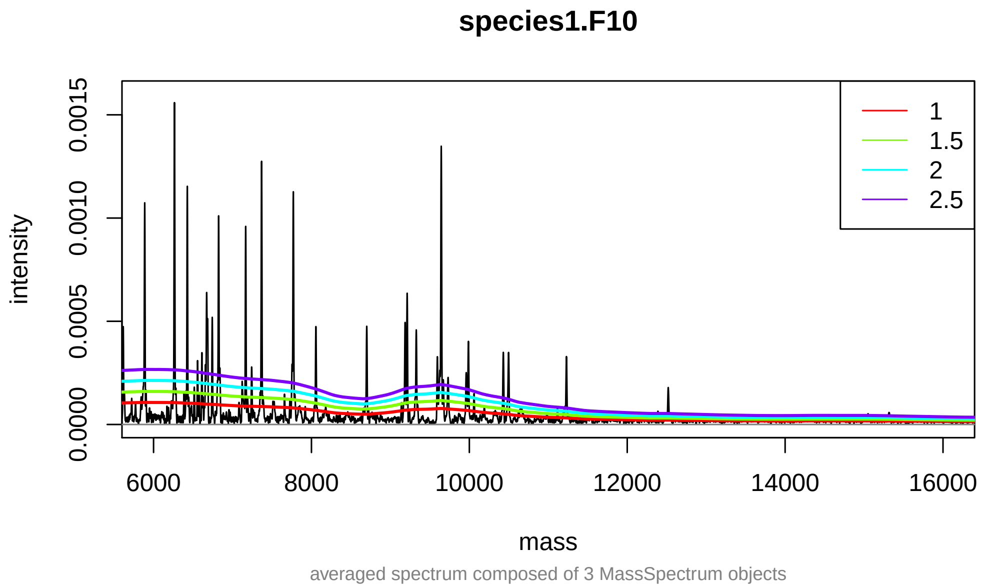
```
avgSpectra <-
  averageMassSpectra(spectra, labels=paste0(species, spots))
```

3.8 Peak Detection

The peak detection is the crucial feature reduction step. Before performing the peak detection we need to estimate the noise of some spectra to get a feeling for the *signal-to-noise ratio* (SNR). We use a similar approach as in section 3.5.

```
## define snrs steps: 1, 1.5, ... 2.5
snrs <- seq(from=1, to=2.5, by=0.5)
## define different colors for each step
```

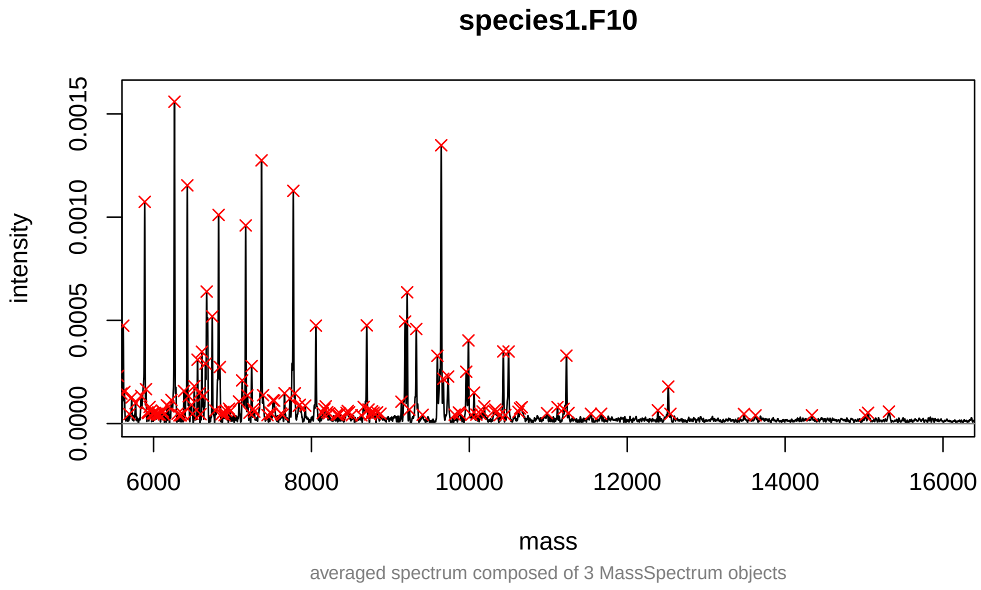
```
col <- rainbow(length(snrs))

## estimate noise
noise <- estimateNoise(avgSpectra[[1]],
                       method="SuperSmoother")

plot(avgSpectra[[1]],
     xlim=c(6000, 16000), ylim=c(0, 0.0016))

for (i in seq(along=snrs)) {
  lines(noise[, "mass"],
        noise[, "intensity"]*snrs[i],
        col=col[i], lwd=2)
}
legend("topright", legend=snrs, col=col, lwd=1)
```



2 or 2.5 look like a good compromise between sensitivity and specificity. We prefer a higher sensitivity and choose a *SNR* of 2 (blue line) for the peak detection. For the `halfWindowSize` we use a similar value as determined in section 3.4.

```
peaks <- detectPeaks(avgSpectra, SNR=2, halfWindowSize=10)
```

```
plot(avgSpectra[[1]], xlim=c(6000, 16000), ylim=c(0, 0.0016))
points(peaks[[1]], col="red", pch=4)
```



3.9 Post Processing

After the alignment the peak positions (mass) are very similar but not identical. The binning is needed to make similar peak mass values identical.

```
peaks <- binPeaks(peaks)
```

We chose a very low signal-to-noise ratio to keep as much features as possible. To remove some false positive peaks we remove peaks that appear in less than 25~% (because we have four groups) of all spectra.

```
peaks <- filterPeaks(peaks, minFrequency=0.25)
```

Finally we create the feature matrix and label the rows with the corresponding species and spot name. We need to recollect both information because we reduce the number of spectra in the average step (see section 3.7).

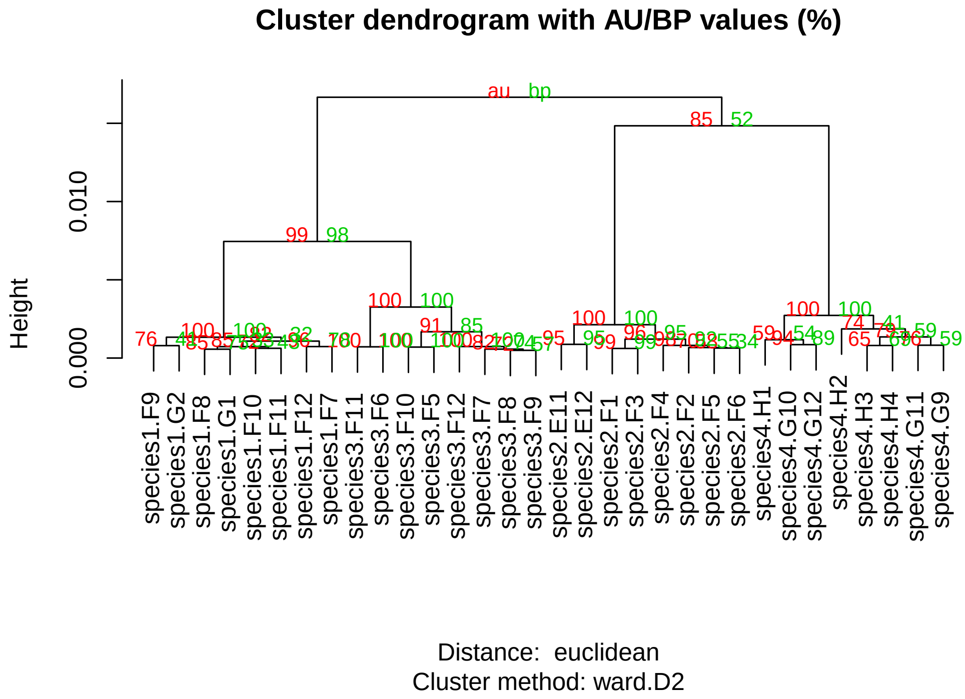
```
spots <- sapply(avgSpectra, function(x)metaData(x)$spot)
species <- sapply(avgSpectra, function(x)metaData(x)$sampleName)
species <- factor(species) # convert to factor
                           # (needed later in crossval)
```

```
featureMatrix <- intensityMatrix(peaks, avgSpectra)
rownames(featureMatrix) <- paste(species, spots, sep=".")
```

3.10 Clustering

Now we use the `pvclust` package (Suzuki and Shimodaira, 2011) to apply a hierarchical clustering analysis with bootstrapping.

```
pv <- pvclust(t(featureMatrix),
              method.hclust="ward.D2",
              method.dist="euclidean")

Bootstrap (r = 0.5)... Done.
Bootstrap (r = 0.6)... Done.
Bootstrap (r = 0.7)... Done.
Bootstrap (r = 0.8)... Done.
Bootstrap (r = 0.9)... Done.
Bootstrap (r = 1.0)... Done.
Bootstrap (r = 1.1)... Done.
Bootstrap (r = 1.2)... Done.
Bootstrap (r = 1.3)... Done.
Bootstrap (r = 1.4)... Done.

plot(pv, print.num=FALSE)
```

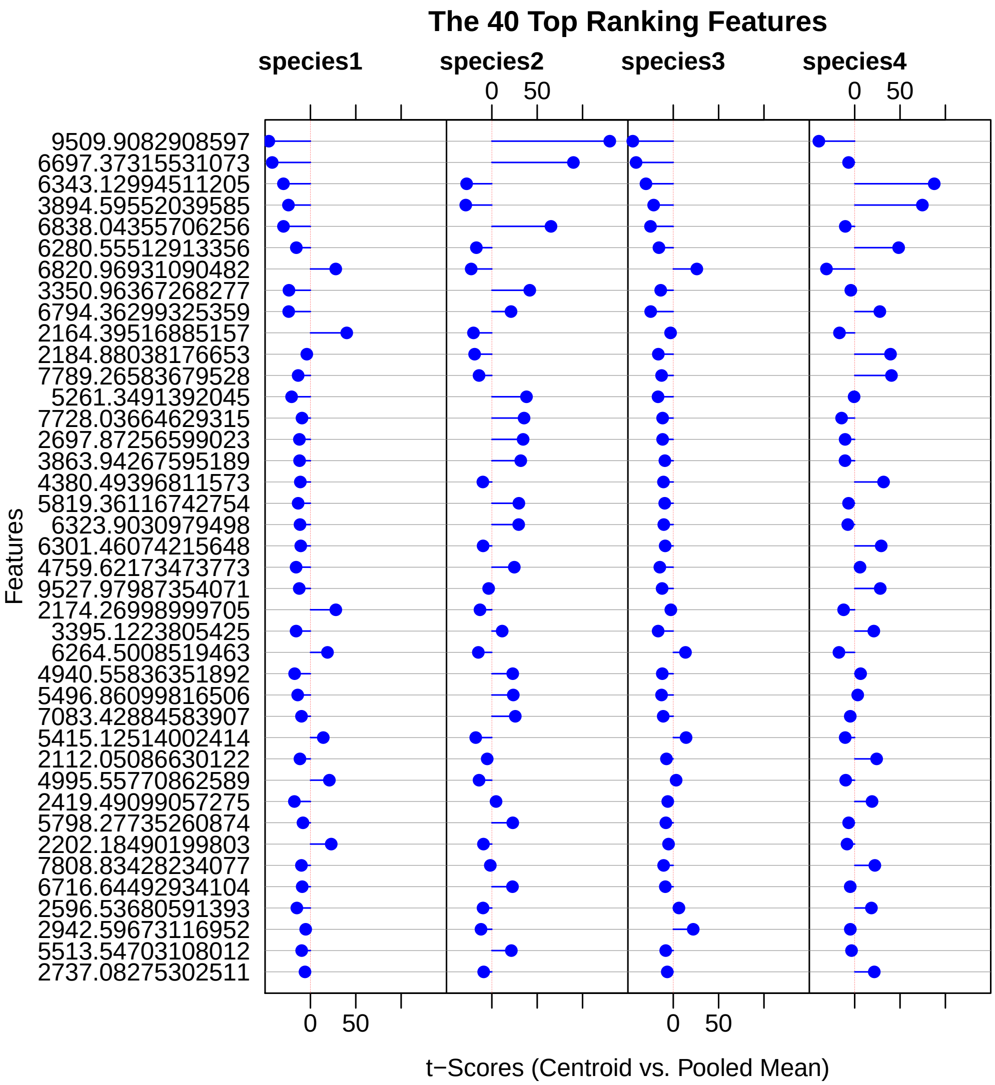


3.11 Diagonal Discriminant Analysis

We finish our analysis using the *diagonal discriminant analysis* (DDA) function of `sda` (Ahdesmäki and Strimmer, 2010) to find the peaks that are typical for a specific species.

```
ddar <- sda.ranking(Xtrain=featureMatrix, L=species,
                    fdr=FALSE, diagonal=TRUE)

Computing t-scores (centroid vs. pooled mean) for feature ranking

Number of variables: 448
Number of observations: 32
Number of classes: 4

Estimating optimal shrinkage intensity lambda.freq (frequencies): 1
Estimating variances (pooled across classes)
Estimating optimal shrinkage intensity lambda.var (variance vector): 0.1016
```

```
plot(ddar)
```

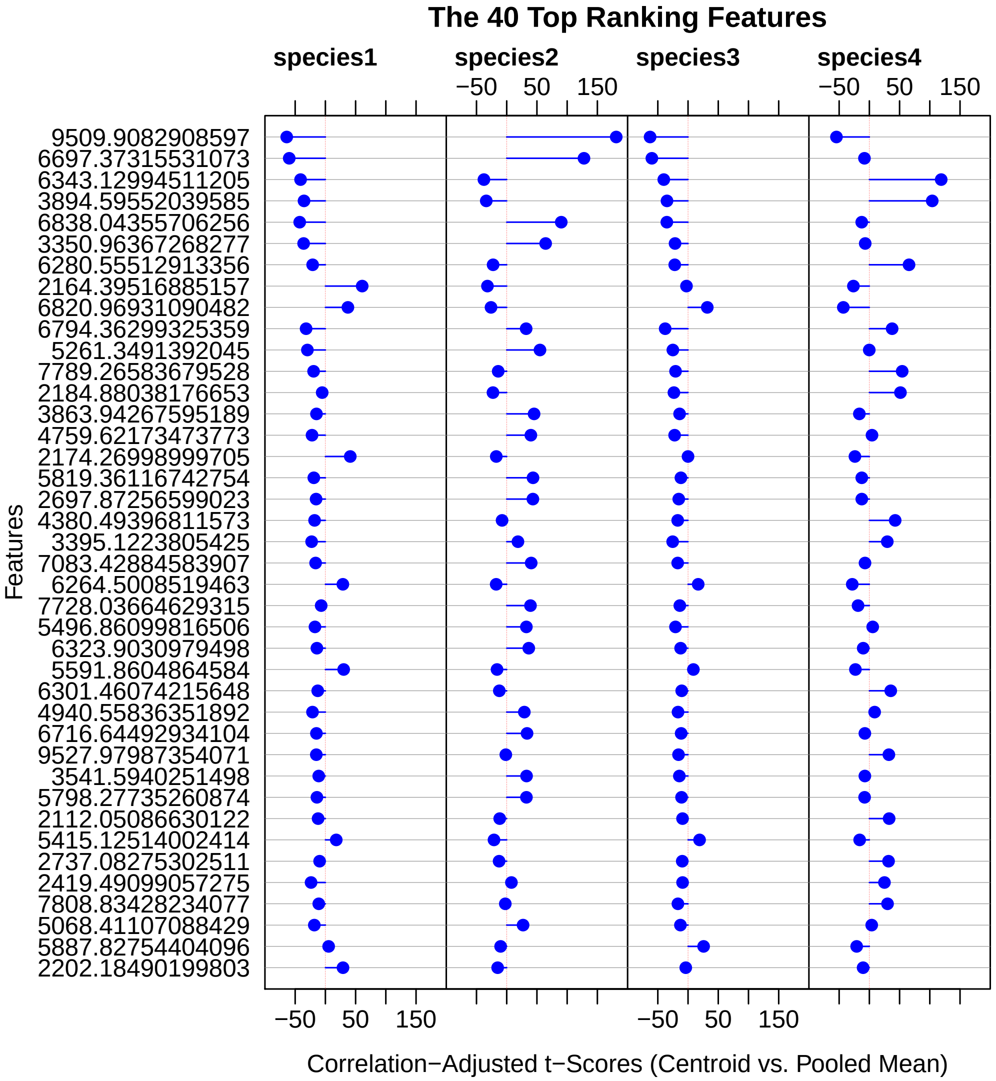


In the plot above we could see that the peak *m/z* 9509 seems to be typical for *species2*, *m/z* 6343 for *species4* and so on.

3.12 Linear Discriminant Analysis

We try the *linear discriminant analysis* (LDA), too (it is part of sda (Ahdesmäki and Strimmer, 2010) as well).

```
ldar <- sda.ranking(Xtrain=featureMatrix, L=species,
                    fdr=FALSE, diagonal=FALSE)

Computing cat scores (centroid vs. pooled mean) for feature ranking

Number of variables: 448
Number of observations: 32
Number of classes: 4

Estimating optimal shrinkage intensity lambda.freq (frequencies): 1
Estimating variances (pooled across classes)
Estimating optimal shrinkage intensity lambda.var (variance vector): 0.1016

Computing the square root of the inverse pooled correlation matrix
Estimating optimal shrinkage intensity lambda (correlation matrix): 0.4737

plot(ldar)
```



3.13 Variable Selection using Cross-Validation

In this section we want to apply cross-validation to find out, how many peaks and which ones we need to discriminate between the species.

We use the package `crossval` (Strimmer, 2014). This package provides the `crossval` function which needs a specific prediction function. The prediction function combines the model creation, the prediction and the comparison between the true and the predicted results.

```
predfun <- function(Xtrain, Ytrain, Xtest, Ytest,
                    numVars, diagonal=FALSE) {
  # estimate ranking and determine the best numVars variables
  ra <- sda.ranking(Xtrain, Ytrain,
                    verbose=FALSE, diagonal=diagonal, fdr=FALSE)
  selVars <- ra[,"idx"][1:numVars]

  # fit and predict
  sda.out <- sda(Xtrain[, selVars, drop=FALSE], Ytrain,
                 diagonal=diagonal, verbose=FALSE)
  ynew <- predict(sda.out, Xtest[, selVars, drop=FALSE],
                  verbose=FALSE)$class

  # compute accuracy
  acc <- mean(Ytest == ynew)

  return(acc)
}
```

We want to repeat the cross-validation 20 times and use 5 folds.

```
K <- 5  # number of folds
B <- 20 # number of repetitions
```

To test our cross-validation setup we want to determine the performance of DDA using the top 10 features (peaks) ranked by *t* scores.

```
set.seed(12345)
cv.dda10 <- crossval(predfun,
                     X=featureMatrix, Y=species,
                     K=K, B=B,
                     numVars=10, diagonal=FALSE,
                     verbose=FALSE)
cv.dda10$stat

[1] 1
```

In the next step we look for the optimal number of peaks (which is more interesting than calculating the performance for the top 10 features).

We calculate the performance of the top 1-15 (and all features) in a similar way as the top 10 features in the example above.

```
npeaks <- c(1:15, ncol(featureMatrix))  # number of peaks
```

First we use DDA.

```
# estimate accuracy for DDA
set.seed(12345)
cvsim.dda <- sapply(npeaks, function(i) {
  cv <- crossval(predfun,
                 X=featureMatrix, Y=species,
                 K=K, B=B, numVars=i, diagonal=TRUE,
                 verbose=FALSE)
  return(cv$stat)
})
```

The same using LDA (the only difference is `diagonal=FALSE`).

```
# estimate accuracy for LDA
set.seed(12345)
cvsim.lda <- sapply(npeaks, function(i) {
  cv <- crossval(predfun,
                 X=featureMatrix, Y=species,
                 K=K, B=B, numVars=i, diagonal=FALSE,
                 verbose=FALSE)
  return(cv$stat)
})
```

We combine the results and put them into a table.

```
result.sim <- cbind(nPeaks=npeaks,
                    "DDA-ACC"=cvsim.dda,
                    "LDA-ACC"=cvsim.lda)
```

We find out that LDA and DDA perform very similar and we need only 9 respectively 10 features (peaks) for a perfect discrimination of the species.

	nPeaks	DDA-ACC	LDA-ACC
1	1	0.772	0.772
2	2	0.877	0.889
3	3	0.901	0.897
4	4	0.911	0.922
5	5	0.962	0.970
6	6	0.967	0.992
7	7	0.975	0.988
8	8	0.984	0.991
9	9	0.997	1.000
10	10	1.000	1.000
11	11	1.000	1.000
12	12	1.000	1.000
13	13	1.000	1.000
14	14	1.000	1.000
15	15	1.000	1.000
16	448	1.000	1.000

3.14 Summary

We have shown how to identify species based on MALDI spectra using `MALDIquant` and `pvclust`. Additionaly we performed a variable selection using `sda` and `crossval` to find the minimal number of peaks for a perfect discriminant.

4 Session Information

- R version 3.1.1 (2014-07-10), `x86_64-pc-linux-gnu`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: MALDIquant~1.11.1, MALDIquantExamples~0.2.3, MALDIquantForeign~0.9, corpcor~1.6.7, crossval~1.0.1, entropy~1.2.0, fdrtool~1.2.12, knitr~1.7, pvclust~1.3-0, sda~1.3.4, xtable~1.7-4
- Loaded via a namespace (and not attached): XML~3.98-1.1, base64enc~0.1-2, digest~0.6.4, downloader~0.3, evaluate~0.5.5,

formatR~1.0, highr~0.3, readBrukerFlexData~1.8, readMzXmlData~2.8, stringr~0.6.2, tools~3.1.1

References

Ahdesmäki, M. and Strimmer, K. (2010). Feature selection in omics prediction problems using cat scores and false nondiscovery rate control. *The Annals of Applied Statistics*, 4(1):503–519.

Bromba, M. U.~A. and Ziegler, H. (1981). Application hints for savitzky-golay digital smoothing filters. *Analytical Chemistry*, 53(11):1583–1586.

Gibb, S. (2014). *MALDIquantForeign: Import/Export routines for MALDIquant*. R package version 0.7.

Gibb, S. and Strimmer, K. (2012). MALDIquant: a versatile R package for the analysis of mass spectrometry data. *Bioinformatics*, 28(17):2270–2271.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ryan, C.~G., Clayton, E., Griffin, W.~L., Sie, S.~H., and Cousens, D.~R. (1988). SNIP, a statistics-sensitive background treatment for the quantitative analysis of PIXE spectra in geoscience applications. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, 34:396–402.

Savitzky, A. and Golay, M. J.~E. (1964). Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Analytical Chemistry*, 36:1627–1639.

Strimmer, K. (2014). *crossval: Generic Functions for Cross Validation*. R package version 1.0.0.

Suzuki, R. and Shimodaira, H. (2011). *pvclust: Hierarchical Clustering with P-Values via Multiscale Bootstrap Resampling*. R package version 1.2-2.

Wickham, H. and Chang, W. (2014). *devtools: Tools to make developing R code easier*. R package version 1.5.