# Using *pRoloc* for spatial proteomics data analysis

***Laurent Gatto and Lisa M. Breckels***

Computational Proteomics Unit, University of Cambridge

**April 7, 2017**

## Abstract

This tutorial illustrates the usage of the *pRoloc* *R* package for the analysis and interpretation of spatial proteomics data. It walks the reader through the creation of *MSnSet* instances, that hold the quantitative proteomics data and meta-data and introduces several aspects of data analysis, including data visualisation and application of machine learning to predict protein localisation.

## Package

pRoloc 1.14.6

Report issues on https://github.com/Bioconductor/pRoloc/issues

Ask questions on https://support.bioconductor.org/

# Contents

1 Introduction . . . . . 6
1.1 Spatial proteomics . . . . . 6
1.2 About *R* and *pRoloc* . . . . . 6
2 Data structures . . . . . 8
2.1 Example data . . . . . 8
2.2 Importing and loading data . . . . . 8
2.2.1 The original data file . . . . . 8
2.2.2 From `csv` files to *R* data . . . . . 10
2.3 A shorter input work flow . . . . . 12
2.3.1 The *MSnSet* class . . . . . 14
2.4 *pRoloc*'s organelle markers . . . . . 17
2.5 Data processing . . . . . 18
3 Data visualisation . . . . . 20
3.1 Profile plots . . . . . 20
3.2 Sub-cellular cluster dendrogram . . . . . 21
3.3 Dimensionality reduction . . . . . 22
3.4 Features of interest . . . . . 25
3.5 Interactive visualisation . . . . . 27
4 Assessing sub-cellular resolution . . . . . 28
5 Data analysis . . . . . 28
5.1 Unsupervised ML . . . . . 29
5.2 Supervised ML . . . . . 33
5.2.1 Classification algorithm parameters optimisation . . . . . 33
5.2.2 Classification . . . . . 35
5.3 Semi-supervised ML . . . . . 43
5.4 Following up on novelty discovery . . . . . 43

6 Conclusions . . . . . 46

# Foreword

*MSnbase* and *pRoloc* are under active developed; current functionality is evolving and new features are added on a regular basis. This software is free and open-source software. If you use it, please support the project by citing it in publications:

> Gatto L. and Lilley K.S. *MSnbase - an R/Bioconductor package for isobaric tagged mass spectrometry data visualization, processing and quantitation.* Bioinformatics 28, 288-289 (2011).
>
> Gatto L, Breckels LM, Wieczorek S, Burger T, Lilley KS. *Mass-spectrometry-based spatial proteomics data analysis using pRoloc and pRolocdata.* Bioinformatics. 2014 Feb 5.

If you are using the `phenoDisco` function, please also cite

> Breckels L.M., Gatto L., Christoforou A., Groen A.J., Kathryn Lilley K.S. and Trotter M.W. *The effect of organelle discovery upon sub-cellular protein localisation.* J Proteomics, S1874-3919(13)00094-8 (2013)

For an introduction to spatial proteomics data analysis:

> Gatto L, Breckels LM, Burger T, Nightingale DJ, Groen AJ, Campbell C, Nikolovski N, Mulvey CM, Christoforou A, Ferro M, Lilley KS. *A foundation for reliable spatial proteomics data analysis.* Mol Cell Proteomics. 2014 Aug;13(8):1937-52. doi:10.1074/mcp.M113.036350.

The *pRoloc* package contains additional vignettes and reference material:

- *pRoloc-tutorial*: pRoloc tutorial.
- *pRoloc-ml*: Machine learning techniques available in pRoloc.
- *pRoloc-transfer-learning*: A transfer learning algorithm for spatial proteomics.
- *pRoloc-goannotations*: Annotating spatial proteomics data.
- *HUPO_2011_poster*: HUPO 2011 poster: *pRoloc – A unifying bioinformatics framework for organelle proteomics*

- *HUPO_2014_poster*: HUPO 2014 poster: *A state-of-the-art machine learning pipeline for the analysis of spatial proteomics data*

# Questions and bugs

You are welcome to contact me directly about *pRoloc*. For bugs, typos, suggestions or other questions, please file an issue in our tracking system[1] providing as much information as possible, a reproducible example and the output of `sessionInfo()`.

[1] https://github.com/lgatto/pRoloc/issues

If you wish to reach a broader audience for general questions about proteomics analysis using `R`, you may want to use the Bioconductor support site: https://support.bioconductor.org/.

# 1 Introduction

## 1.1 Spatial proteomics

Spatial (or organelle) proteomics is the study of the localisation of proteins inside cells. The sub-cellular compartment can be organelles, i.e. structures defined by lipid bi-layers,macro-molecular assemblies of proteins and nucleic acids or large protein complexes. In this document, we will focus on mass-spectrometry based approaches that assay a population of cells, as opposed as microscopy based techniques that monitor single cells, as the former is the primary concern of *pRoloc*, although the techniques described below and the infrastructure in place could also be applied the processed image data. The typical experimental use-case for using *pRoloc* is a set of fractions, originating from a total cell lysate. These fractions can originate from a continuous gradient, like in the LOPIT [1] or PCP [2] approaches, or can be discrete fractions. The content of the fractions is then identified and quantified (using labelled or un-labelled quantitation techniques). Using relative quantitation of known organelle residents, termed organelle markers, organelle-specific profiles along the gradient are determined and new residents are identified based on matching of these distribution profiles. See for example [3] and references therein for a detailed review on organelle proteomics.

It should be noted that large protein complexes, that are not necessarily separately enclosed within their own lipid bi-layer, can be detected by such techniques, as long as a distinct profile can be defined across the fractions.

## 1.2 About *R* and *pRoloc*

*R* [4] is a statistical programming language and interactive working environment. It can be expanded by so-called packages to confer new functionality to users. Many such packages have been developed for the analysis of high-throughput biology, notably through the Bioconductor project [5]. Two packages are of particular interest here, namely *MSnbase* [6] and *pRoloc*. The former provides flexible infrastructure to

store and manipulate quantitative proteomics data and the associated meta-data and the latter implements specific algorithmic technologies to analyse organelle proteomics data.

Among the advantages of *R* are robust statistical procedures, good visualisation capabilities, excellent documentation, reproducible research[2], power and flexibility of the *R* language and environment itself and a rich environment for specialised functionality in many domains of bioinformatics: tools for many omics technologies, including proteomics, biostatistics, gene ontology and biological pathway analysis, ... Although there exists some specific graphical user interfaces (GUI), interaction with *R* is executed through a command line interface. While this mode of interaction might look alien to new users, experience has proven that after a first steep learning curve, great results can be achieved by non-programmers. Furthermore, specific and general documentation is plenty and beginners and advanced course material are also widely available.

[2]The content of this document is compiled (the code is executed and its output, text and figures, is displayed dynamically) to generate the pdf file.

Once *R* is started, the first step to enable functionality of a specific packages is to load them using the `library` function, as shown in the code chunk below:

```
library("MSnbase")
library("pRoloc")
library("pRolocdata")
```

*MSnbase* implements the data containers that are used by *pRoloc*. *pRolocdata* is a data package that supplies several published organelle proteomics data sets.

As a final setup step, we set the default colour palette for some of our custom plotting functionality to use semi-transparent colours in the code chunk below (see `?setStockcol` for details). This facilitates visualisation of overlapping points.

```
setStockcol(NULL) ## reset first
setStockcol(paste0(getStockcol(), 70))
```

# 2 Data structures

## 2.1 Example data

The data used in this tutorial has been published in [7]. The LOPIT technique [1] is used to localise integral and associated membrane proteins in *Drosophila melanogaster* embryos. Briefly, embryos were collected at 0 – 16 hours, homogenised and centrifuged to collect the supernatant, removing cell debris and nuclei. Membrane fractionation was performed on a iodixanol gradient and fractions were quantified using iTRAQ isobaric tags [8] as follows: fractions 4/5, 114; fractions 12/13, 115; fraction 19, 116 and fraction 21, 117. Labelled peptides were then separated using cation exchange chromatography and analysed by LS-MS/MS on a QSTAR XL quadrupole-time-of-flight mass spectrometer (Applied Biosystems). The original localisation analysis was performed using partial least square discriminant analysis (PLS-DA). Relative quantitation data was retrieved from the supplementary file `pr800866n_si_004.xls` (http://pubs.acs.org/doi/suppl/10.1021/pr800866n/suppl_file/pr800866n_si_004.xls) and imported into *R* as described below. We will concentrate on the first replicate.

## 2.2 Importing and loading data

This section illustrates how to import data in comma-separated value (csv) format into an appropriate *R* data structure. The first section shows the original `csv` (comma separated values) spreadsheet, as published by the authors, and how one can read such a file into *R*using the `read.csv` function. This spreadsheet file is similar to the output of many quantitation software.

In the next section, we show 2 `csv` files containing a subset of the columns of original `pr800866n_si_004-rep1.csv` file and another short file, created manually, that will be used to create the appropriate *R* data.

### 2.2.1 The original data file

```
## The original data for replicate 1, available
## from the pRolocdata package
f0 <- dir(system.file("extdata", package = "pRolocdata"),
          full.names = TRUE,
          pattern = "pr800866n_si_004-rep1.csv")
csv <- read.csv(f0)
```

The three first lines of the original spreadsheet, containing the data for replicate one, are illustrated below (using the function `head`). It contains 888 rows (proteins) and 16 columns, including protein identifiers, database accession numbers, gene symbols, reporter ion quantitation values, information related to protein identification, ...

```
head(csv, n=3)
```

```
##   Protein.ID        FBgn Flybase.Symbol
## 1    CG10060 FBgn0001104    G-ialpha65A
## 2    CG10067 FBgn0000044         Act57B
## 3    CG10077 FBgn0035720        CG10077
##   No..peptide.IDs Mascot.score No..peptides.quantified
## 1               3       179.86                       1
## 2               5       222.40                       9
## 3               5       219.65                       3
##   area.114 area.115 area.116 area.117
## 1 0.379000 0.281000 0.225000 0.114000
## 2 0.420000 0.209667 0.206111 0.163889
## 3 0.187333 0.167333 0.169667 0.476000
##   PLS.DA.classification Peptide.sequence
## 1                    PM
## 2                    PM
## 3
##   Precursor.ion.mass Precursor.ion.charge pd.2013
## 1                                              PM
## 2                                              PM
## 3                                         unknown
##   pd.markers
## 1    unknown
## 2    unknown
## 3    unknown
```

### 2.2.2 From `csv` files to *R* data

There are several ways to create the desired *R* data object, termed *MSnSet*, that will be used to perform the actual sub-cellular localisation prediction. Here, we illustrate a method that uses separate spreadsheet files for quantitation data, feature meta-data and sample (fraction) meta-data and the `readMSnSet` constructor function, that will hopefully be the most straightforward for new users.

```
## The quantitation data, from the original data
f1 <- dir(system.file("extdata", package = "pRolocdata"),
          full.names = TRUE, pattern = "exprsFile.csv")
exprsCsv <- read.csv(f1)
## Feature meta-data, from the original data
f2 <- dir(system.file("extdata", package = "pRolocdata"),
          full.names = TRUE, pattern = "fdataFile.csv")
fdataCsv <- read.csv(f2)
## Sample meta-data, a new file
f3 <- dir(system.file("extdata", package = "pRolocdata"),
          full.names = TRUE, pattern = "pdataFile.csv")
pdataCsv <- read.csv(f3)
```

**`exprsFile.csv`** containing the quantitation (expression) data for the 888 proteins and 4 reporter tags.

```
head(exprsCsv, n=3)
```

```
##          FBgn     X114     X115     X116     X117
## 1 FBgn0001104 0.379000 0.281000 0.225000 0.114000
## 2 FBgn0000044 0.420000 0.209667 0.206111 0.163889
## 3 FBgn0035720 0.187333 0.167333 0.169667 0.476000
```

**`fdataFile.csv`** containing meta-data for the 888 features (here proteins).

```
head(fdataCsv, n=3)
```

```
##          FBgn ProteinID FlybaseSymbol NoPeptideIDs
## 1 FBgn0001104   CG10060   G-ialpha65A           3
## 2 FBgn0000044   CG10067        Act57B           5
```

```
## 3 FBgn0035720   CG10077       CG10077           5
##   MascotScore NoPeptidesQuantified PLSDA
## 1      179.86                    1    PM
## 2      222.40                    9    PM
## 3      219.65                    3
```

`pdataFile.csv` containing samples (here fractions) meta-data. This simple file has been created manually.

```
pdataCsv
```

```
##   sampleNames Fractions
## 1        X114       4/5
## 2        X115     12/13
## 3        X116        19
## 4        X117        21
```

A self-contained data structure, called *MSnSet* (defined in the *MSnbase* package) can now easily be generated using the `readMSnSet` constructor, providing the respective `csv` file names shown above and specifying that the data is comma-separated (with `sep=","`). Below, we call that object `tan2009r1` and display its content.

```
tan2009r1 <- readMSnSet(exprsFile = f1,
                        featureDataFile = f2,
                        phenoDataFile = f3,
                        sep = ",")
tan2009r1
```

```
## MSnSet (storageMode: lockedEnvironment)
## assayData: 888 features, 4 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: X114 X115 X116 X117
##   varLabels: Fractions
##   varMetadata: labelDescription
## featureData
##   featureNames: FBgn0001104 FBgn0000044 ...
##     FBgn0001215 (888 total)
```

```
##   fvarLabels: ProteinID FlybaseSymbol ... PLSDA
##     (6 total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation:
## - - - Processing information - - -
##  MSnbase version: 2.0.2
```

## 2.3 A shorter input work flow

The `readMSnSet2` function provides a simplified import pipeline. It takes a single spreadsheet as input (default is `csv`) and extract the columns identified by `ecol` to create the expression data, while the others are used as feature meta-data. `ecol` can be a `character` with the respective column labels or a numeric with their indices. In the former case, it is important to make sure that the names match exactly. Special characters like `'-'` or `'('` will be transformed by *R*into `'.'` when the `csv` file is read in. Optionally, one can also specify a column to be used as feature names. Note that these must be unique to guarantee the final object validity.

```
ecol <- paste("area", 114:117, sep = ".")
fname <- "Protein.ID"
eset <- readMSnSet2(f0, ecol, fname)
eset
```

```
## MSnSet (storageMode: lockedEnvironment)
## assayData: 888 features, 4 samples
##   element names: exprs
## protocolData: none
## phenoData: none
## featureData
##   featureNames: CG10060 CG10067 ... CG9983 (888
##     total)
##   fvarLabels: Protein.ID FBgn ... pd.markers (12
##     total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
```

```
## Annotation:
## - - - Processing information - - -
##  MSnbase version: 2.0.2
```

The `ecol` columns can also be queried interactively from *R* using the `getEcols` and `grepEcols` function. The former return a character with all column names, given a splitting character, i.e. the separation value of the spreadsheet (typically `","` for `csv`, `"\t"` for `tsv`, ...). The latter can be used to grep a pattern of interest to obtain the relevant column indices.

```
getEcols(f0, ",")
```

```
##  [1] "\"Protein ID\""
##  [2] "\"FBgn\""
##  [3] "\"Flybase Symbol\""
##  [4] "\"No. peptide IDs\""
##  [5] "\"Mascot score\""
##  [6] "\"No. peptides quantified\""
##  [7] "\"area 114\""
##  [8] "\"area 115\""
##  [9] "\"area 116\""
## [10] "\"area 117\""
## [11] "\"PLS-DA classification\""
## [12] "\"Peptide sequence\""
## [13] "\"Precursor ion mass\""
## [14] "\"Precursor ion charge\""
## [15] "\"pd.2013\""
## [16] "\"pd.markers\""
```

```
grepEcols(f0, "area", ",")
```

```
## [1]  7  8  9 10
```

```
e <- grepEcols(f0, "area", ",")
readMSnSet2(f0, e)
```

```
## MSnSet (storageMode: lockedEnvironment)
## assayData: 888 features, 4 samples
##   element names: exprs
## protocolData: none
```

```
## phenoData: none
## featureData
##   featureNames: 1 2 ... 888 (888 total)
##   fvarLabels: Protein.ID FBgn ... pd.markers (12
##     total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation:
## - - - Processing information - - -
##  MSnbase version: 2.0.2
```

The `phenoData` slot can now be updated accordingly using the replacement functions `phenoData<-` or `pData<-` (see `?MSnSet` for details).

### 2.3.1 The *MSnSet* class

Although there are additional specific sub-containers for additional meta-data (for instance to make the object MIAPE compliant), the feature (the sub-container, or slot `featureData`) and sample (the `phenoData` slot) are the most important ones. They need to meet the following validity requirements (see figure 1):

- the number of row in the expression/quantitation data and feature data must be equal and the row names must match exactly, and
- the number of columns in the expression/quantitation data and number of row in the sample meta-data must be equal and the column/row names must match exactly.

It is common, in the context of *pRoloc* to update the feature meta-data (described in section 5) by adding new columns, without breaking the objects validity. Similarly, the sample meta-data can also be updated by adding new sample variables. A detailed description of the `MSnSet` class is available by typing `?MSnSet` in the *R* console.

The individual parts of this data object can be accessed with their respective accessor methods:

- the quantitation data can be retrieved with `exprs(tan2009r1)`,
- the feature meta-data with `fData(tan2009r1)` and

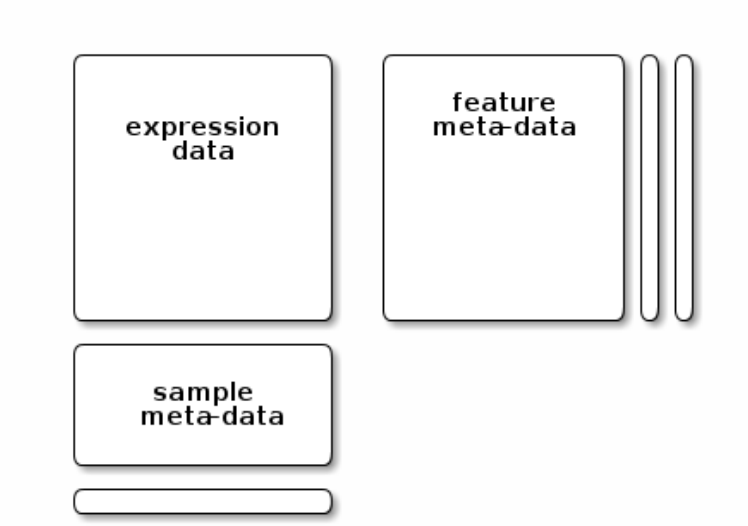


**Figure 1: Dimension requirements for the respective expression, feature and sample meta-data slots.**

- the sample meta-data with `pData(tan2009r1)`.

The advantage of this structure is that it can be manipulated as a whole and the respective parts of the data object will remain compatible. The code chunk below, for example, shows how to extract the first 5 proteins and 2 first samples:

```
smallTan <- tan2009r1[1:5, 1:2]
dim(smallTan)
```

```
## [1] 5 2
```

```
exprs(smallTan)
```

```
##                 X114     X115
## FBgn0001104 0.379000 0.281000
## FBgn0000044 0.420000 0.209667
## FBgn0035720 0.187333 0.167333
## FBgn0003731 0.247500 0.253000
## FBgn0029506 0.216000 0.183000
```

Several data sets, including the 3 replicates from [7], are distributed as *MSnSet* instances in the *pRolocdata* package. Others include, among others, the *Arabidopsis thaliana* LOPIT data from [1] (`dunkley2006`)

and the mouse PCP data from [2] (`foster2006`). Each data set can be loaded with the `data` function, as show below for the first replicate from [7].

```
data(tan2009r1)
```

The original marker proteins are available as a feature meta-data variables called `markers.orig` and the output of the partial least square discriminant analysis, applied in the original publication, in the `PLSDA` feature variable. The most up-to-date marker list for the experiment can be found in `markers`. This feature meta-data column can be added from a simple `csv` markers files using the `addMarkers` function - see `?addMarkers` for details.

The organelle markers are illustrated below using the convenience function `getMarkers`, but could also be done manually by accessing feature variables directly using `fData()`.

```
getMarkers(tan2009r1, fcol = "markers.orig")
```

```
## organelleMarkers
##            ER         Golgi            PM
##            20             6            15
## mitochondrion       unknown
##            14           833
```

```
getMarkers(tan2009r1, fcol = "PLSDA")
```

```
## organelleMarkers
##      ER/Golgi            PM mitochondrion
##           235           180            74
##       unknown
##           399
```

**Important** As can be seen above, some proteins are labelled `"unknown"`, defining non marker proteins. This is a convention in many *pRoloc* functions. Missing annotations (empty string) will not be considered as of unknown localisation; we prefer to avoid empty strings and make the absence of known localisation explicit by using the `"unknown"` tag. This information will be used to separate marker and

non-marker (unlabelled) proteins when proceeding with data visualisation and clustering (sections 3 and 5.1) and classification analysis (section 5.2.2).

## 2.4 *pRoloc*'s organelle markers

The *pRoloc* package distributes a set of markers that have been obtained by mining the *pRolocdata* datasets and curation by various members of the Cambridge Centre for Proteomics. The available marker sets can be obtained and loaded using the `pRolocmarkers` functions:

```
pRolocmarkers()
```

```
## 7 marker lists available:
## Arabidopsis thaliana [atha]:
##  Ids: TAIR, 543 markers
## Drosophila melanogaster [dmel]:
##  Ids: Uniprot, 179 markers
## Gallus gallus [ggal]:
##  Ids: IPI, 102 markers
## Homo sapiens [hsap]:
##  Ids: Uniprot ID, 205 markers
## Mus musculus [mmus]:
##  Ids: Uniprot, 937 markers
## Saccharomyces cerevisiae [scer_sgd]:
##  Ids: SGD, 259 markers
## Saccharomyces cerevisiae [scer_uniprot]:
##  Ids: Uniprot Accession, 259 markers
```

```
head(pRolocmarkers("dmel"))
```

```
##       Q7JZN0       Q7KLV9       Q9VIU7       P15348
##         "ER" "Proteasome"         "ER"    "Nucleus"
##       Q7KMP8       O01367
## "Proteasome"    "Nucleus"
```

```
table(pRolocmarkers("dmel"))
```

```
##
##  Cytoskeleton            ER         Golgi
##             7            24             7
```

```
##      Lysosome       Nucleus            PM
##             8            21            25
##    Peroxisome    Proteasome  Ribosome 40S
##             4            14            22
##  Ribosome 60S mitochondrion
##            32            15
```

These markers can then be added to a new *MSnSet* using the `addMarkers` function by matching the marker names (protein identifiers) and the feature names of the *MSnSet*. See `?addMarkers` for examples.

## 2.5 Data processing

The quantitation data obtained in the supplementary file is normalised to the sum of intensities of each protein; the sum of fraction quantitation for each protein equals 1 (considering rounding errors). This can quickly be verified by computing the row sums of the expression data.

```
summary(rowSums(exprs(tan2009r1)))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.9990  0.9999  1.0000  1.0000  1.0000  1.0010
```

The `normalise` method (also available as `normalize`) from the *MSnbase* package can be used to obtain relative quantitation data, as illustrated below on another iTRAQ test data set, available from *MSnbase*. Several normalisation `methods` are available and described in `?normalise`. For many algorithms, including classifiers in general and support vector machines in particular, it is important to properly per-process the data. Centering and scaling of the data is also available with the `scale` method, described in the `scale` manual.

In the code chunk below, we first create a test *MSnSet* instance[3] and illustrate the effect of `normalise(...,method="sum")`.

```
## create a small illustrative test data
data(itraqdata)
itraqdata <- quantify(itraqdata, method = "trap",
```

[3] Briefly, the `itraqdata` raw iTRAQ4-plex data is quantified by trapezoidation using *MSnbase* functionality. See the `MSnbase-demo` vignette for details.

```
                      reporters = iTRAQ4)
## the quantification data
head(exprs(itraqdata), n = 3)

##     iTRAQ4.114 iTRAQ4.115 iTRAQ4.116 iTRAQ4.117
## X1   1347.6158  2247.3097  3927.6931  7661.1463
## X10   739.9861   799.3501   712.5983   940.6793
## X11 27638.3582 33394.0252 32104.2879 26628.7278

summary(rowSums(exprs(itraqdata)))

##      Min.   1st Qu.    Median      Mean   3rd Qu.
##     59.06   5638.00  15340.00  38010.00  42260.00
##      Max.      NA's
## 305700.00         1

## normalising to the sum of feature intensitites
itraqnorm <- normalise(itraqdata, method = "sum")
processingData(itraqnorm)

## - - - Processing information - - -
## Data loaded: Wed May 11 18:54:39 2011
## Updated from version 0.3.0 to 0.3.1 [Fri Jul  8 20:23:25 2016]
## iTRAQ4 quantification by trapezoidation: Fri Apr  7 18:03:24 2017
## Normalised (sum): Fri Apr  7 18:03:24 2017
##  MSnbase version: 1.1.22

head(exprs(itraqnorm), n = 3)

##     iTRAQ4.114 iTRAQ4.115 iTRAQ4.116 iTRAQ4.117
## X1  0.08875373  0.1480074  0.2586772  0.5045617
## X10 0.23178064  0.2503748  0.2232022  0.2946424
## X11 0.23077081  0.2788287  0.2680598  0.2223407

summary(rowSums(exprs(itraqnorm)))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1       1       1       1       1       1
##    NA's
##       1
```

Note above how the processing undergone by the *MSnSet* instances `itraqdata` and `itraqnorm` is stored in another such specific sub-container, the `processingData` slot.

The different features (proteins in the `tan2009r1` data above, but these could also represent peptides or $MS^2$ spectra) are characterised by unique names. These can be retrieved with the `featureNames` function.

```
head(featureNames(tan2009r1))
```

```
## [1] "P20353" "P53501" "Q7KU78" "P04412" "Q7KJ73"
## [6] "Q7JZN0"
```

If we look back at section 2.2.2, we see that these have been automatically assigned using the first columns in the `exprsFile.csv` and `fdataFile.csv` files. It is thus crucial for these respective first columns to be identical. Similarly, the sample names can be retrieved with `sampleNames(tan2009r1)`.

# 3 Data visualisation

The following sections will focus on two closely related aspects, data visualisation and data analysis (i.e. organelle assignments). Data visualisation is used in the context on quality control, to convince ourselves that the data displays the expected properties so that the output of further processing can be trusted. Visualising results of the localisation prediction is also essential, to control the validity of these results, before proceeding with orthogonal (and often expensive) *dry* or *wet* validation.

## 3.1 Profile plots

The underlying principle of gradient approaches is that we have separated organelles along the gradient and by doing so, generated organelle-specific protein distributions along the gradient fractions. The most natural visualisation is shown on figure 2, obtained using the subsetting functionality of *MSnSet* instances and the `plotDist` function, as illustrated below.

```
## indices of the mito markers
j <- which(fData(tan2009r1)$markers.orig == "mitochondrion")
```

```
## indices of all proteins assigned to the mito
i <- which(fData(tan2009r1)$PLSDA == "mitochondrion")
plotDist(tan2009r1[i, ],
         markers = featureNames(tan2009r1)[j])
```

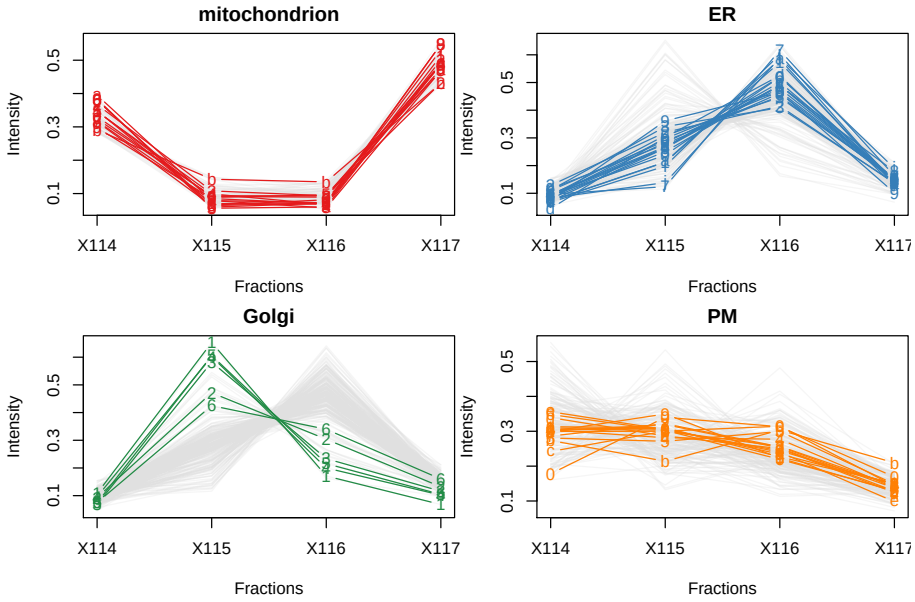


**Figure 2: Distribution of protein intensities along the fractions of the separation gradient for 4 organelles: mitochondrion (red), ER/Golgi (blue, ER markers and green, Golgi markers) and plasma membrane (purple).**

## 3.2 Sub-cellular cluster dendrogram

To gain a quick overview of the distance/similarity between the sub-cellular clusters, it can useful to compare average marker profiles, rather than all profiles, as presented in the profile plots above. The `mrkHClust` calculates average class profiles and generates the resulting dendrogram.

On figure 3, we see that the lysosome and the ribosome 60S are separated by the smallest distance. The advantage of this representation is that it provides a quick snapshot of the average similarity between organelles using the complete profiles (as opposed to a PCA plot, discussed in the next section). The main drawback is that it ignores any

```
mrkHClust(tan2009r1)
```

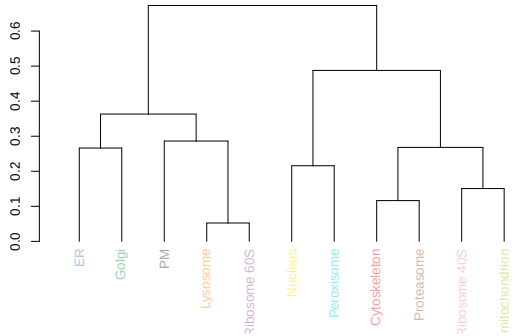


**Figure 3: Average distance between organelle classes.**

variability in individual markers (cluster thighness). It is however a good guide for a more thorough exploration of the data, as described in the next sections.

Note the colours of the labels on the dendrogram (figure 3), which match the colours used to annotate PCA plots, described in the next section (figure 4). These colours are defined at the session level (see `getStockcol` and `setStockcol`) and re-used throughout *pRoloc* for consistent annotation.

## 3.3 Dimensionality reduction

Alternatively, we can combine all organelle groups in one single 2 dimensional figure by applying a dimensionality reduction technique such as Principal Component Analysis (PCA) using the `plot2D` function (see figure 4). The protein profile vectors are summarised into 2 values that can be visualised in two dimensions, so that the variability in the data will be maximised along the first principal component (PC1). The second principal component (PC2) is then chosen as to be orthogonal to PC1 while explaining as much variance in the data as possible, and so on for PC3, PC4, etc.

Using a PCA representation to visualise a spatial proteomics experiment, we can easily plot all the proteins on the same figure as well a many sub-cellular clusters. These clusters are defined in a feature

meta-data column (slot `featureData`, accessed as a `data.frame` with the `fData` function) that is declared with the `fcol` argument (default is `"markers"` which contains the most current known markers for this experiment, while the original markers published with the data can be found in the slot `"markers.orig"`).

```
plot2D(tan2009r1, fcol = "markers")
addLegend(tan2009r1, fcol = "markers", cex = .7,
          where = "bottomright", ncol = 2)
```

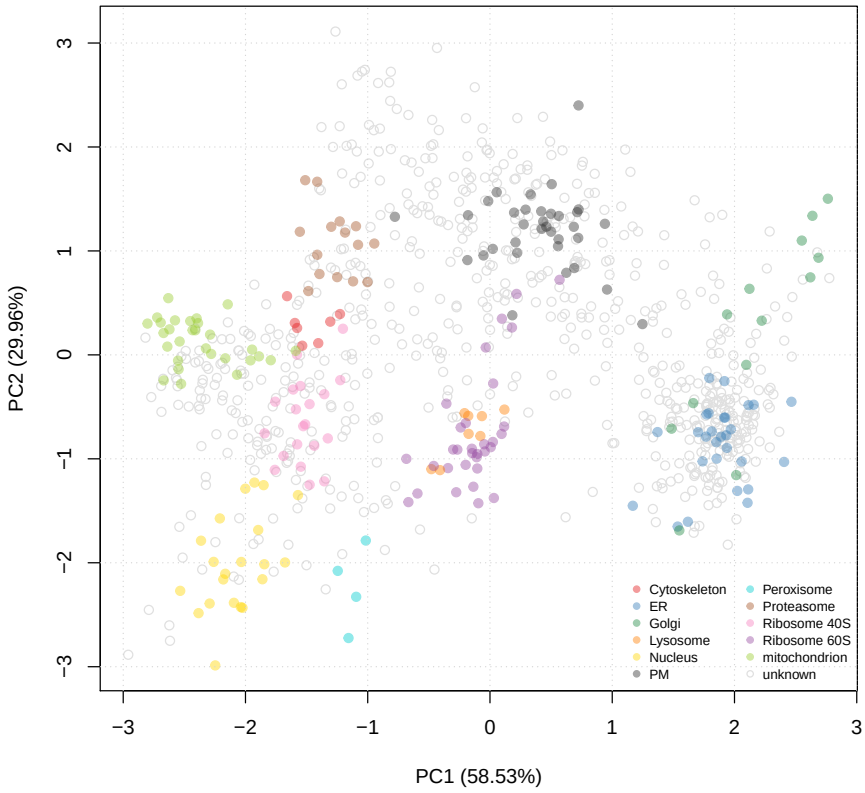


**Figure 4: PCA plot.** Representation of the 888 protein of `tan2009r1` after reduction of the 4 reporter quantitation data to 2 principal components.

As the default value for the `fcol` argument is `"markers"`, it is not necessary to specify it. It is however mandatory to specify other annotation feature variables, such as to visualise the set of markers described in the original publication.

```
plot2D(tan2009r1, fcol = "markers.orig")
addLegend(tan2009r1, fcol = "markers.orig", where = "bottomright")
```

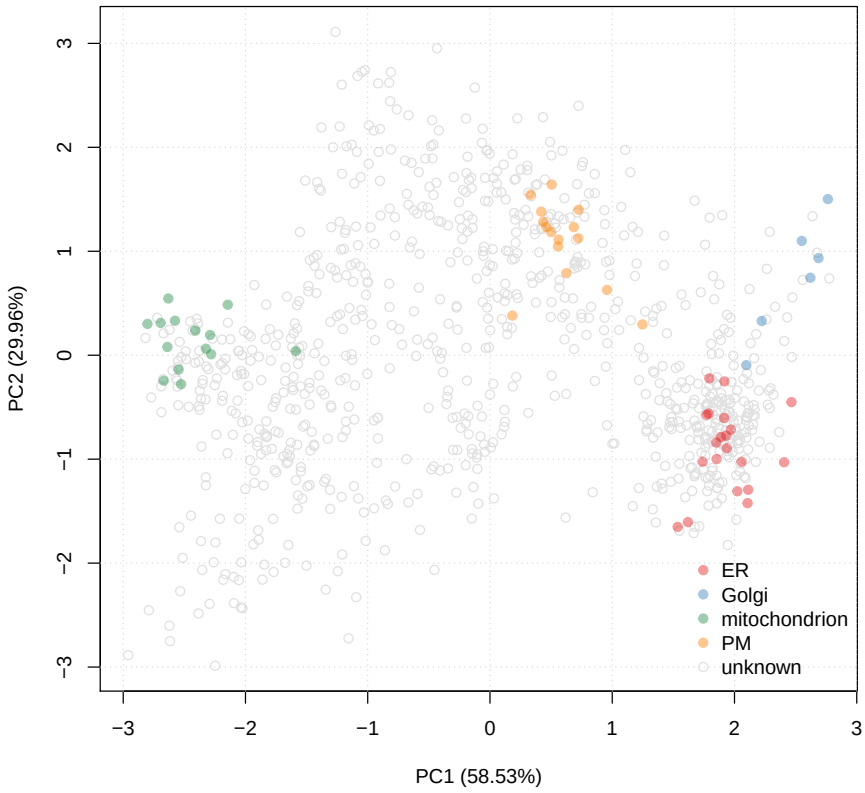


**Figure 5: PCA plot.** Reduced set of markers for the `tan2009r1` data projected onto 2 principal components.

It is also possible to visualise the data along 3 dimensions using the `plot3D` function, which works like the 2 dimension version (figure ref-fig:plot3D). The resulting figure can be rotated by the user.

```
plot3D(tan2009r1)
```

As can be seen on the figures 4 and 6, we indicate on the axis labels the percentage of total variance explained by the individual PCs. It is not unusual to reach over 75% along the first two PCs, even for experiments with several tens of fractions. One can calculate this information for

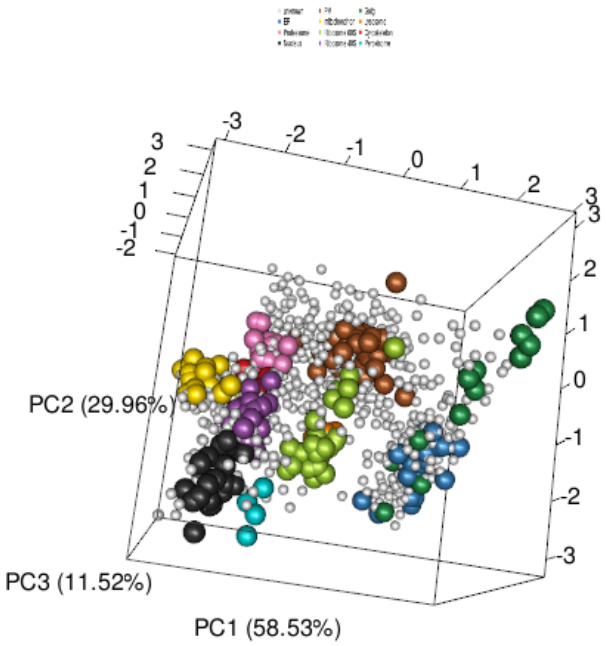


**Figure 6: Snapshot of the 3-dimensional PCA plot.** The `tan2009r1` data is represented along the first 3 principal components.

all PCs by setting `method="scree"` in `plot2D`. On figure 7, we see that the four PCs in the `tan2009r1` data account for 58.53, 29.96, 11.52 and 0 percent of the total variance.

## 3.4 Features of interest

In addition to highlighting sub-cellular niches as coloured clusters on the PCA plot, it is also possible to define some arbitrary *features of interest* that represent, for example, proteins of a particular pathway or a set of interaction partners. Such sets of proteins are recorded as *FeaturesOfInterest* instances, as illstrated below (using the ten first features of our experiment):

```
plot2D(tan2009r1, method = "scree")
```

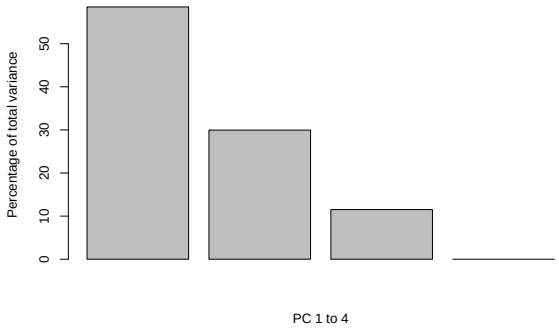


**Figure 7:** Percentage of variance explained.

```
foi1 <- FeaturesOfInterest(description = "Feats of interest 1",
                           fnames = featureNames(tan2009r1[1:10]))
description(foi1)

## [1] "Feats of interest 1"

foi(foi1)

##  [1] "P20353" "P53501" "Q7KU78" "P04412" "Q7KJ73"
##  [6] "Q7JZN0" "Q7KLV9" "Q9VM65" "Q9VCK0" "Q9VIU7"
```

Several such features of interest can be combined into collections:

```
foi2 <- FeaturesOfInterest(description = "Feats of interest 2",
                           fnames = featureNames(tan2009r1[880:888]))
foic <- FoICollection(list(foi1, foi2))
foic

## A collection of 2 features of interest.
```

*FeaturesOfInterest* instances can now be overlaid on the PCA plot with the `highlightOnPlot` function. The `highlightOnPlot3D` can be used to overlay data onto a 3 dimensional figure produced by `plot3D`.

See `?FeaturesOfInterest` and `?highlightOnPlot` for more details.

```
plot2D(tan2009r1, fcol = "PLSDA")
addLegend(tan2009r1, fcol = "PLSDA",
          where = "bottomright",
          cex = .7)
highlightOnPlot(tan2009r1, foi1,
                col = "black", lwd = 2)
highlightOnPlot(tan2009r1, foi2,
                col = "purple", lwd = 2)
legend("topright", c("FoI 1", "FoI 2"),
       bty = "n", col = c("black", "purple"),
       pch = 1)
```

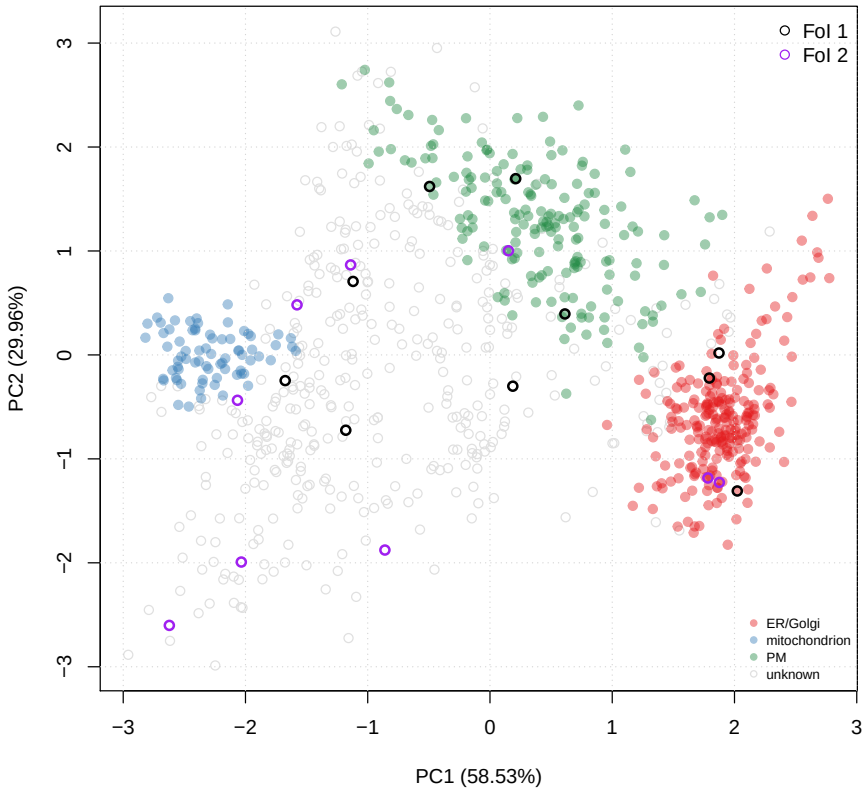


**Figure 8:** **Adding features of interest on a PCA plot.**

## 3.5 Interactive visualisation

The *pRolocGUI* application allows one to explore the spatial proteomics data using an interactive, web-based *shiny* interface [9]. The package is available from Bioconductor and can be installed and started as follows:

```
library("BiocInstaller")
biocLite("pRolocGUI")
library("pRolocGUI")
pRolocVis(tan2009r1)
```


**Figure 9: Screenshot of the *pRolocGUI* interface.** The GUI is also available as an online app for the hyperLOPIT experiment from [10] at https://lgatto.shinyapps.io/christoforou2015/.

More details are available in the vignette that can be started from the application by clicking on any of the question marks, by starting the vignette from *R* with `vignette("pRolocGUI")` or can be accessed online[4].

[4] http://bioconductor.org/packages/devel/bioc/vignettes/pRolocGUI/inst/doc/pRolocGUI.html

# 4 Assessing sub-cellular resolution

TODO

# 5 Data analysis

Classification of proteins, i.e. assigning sub-cellular localisation to proteins, is the main aspect of the present data analysis. The principle is the following and is, in its basic form, a 2 step process. First, an

algorithm learns from the known markers that are shown to him and models the data space accordingly. This phase is also called the training phase. In the second phase, un-labelled proteins, i.e. those that have not been labelled as resident of any organelle, are matched to the model and assigned to a group (an organelle). This 2 step process is called machine learning (ML), because the computer (machine) learns by itself how to recognise instances that possess certain characteristics and classifies them without human intervention. That does however not mean that results can be trusted blindly.

In the above paragraph, we have defined what is called supervised ML, because the algorithm is presented with some know instances from which it learns (see section 5.2.2). Alternatively, un-supervised ML does not make any assumptions about the group memberships, and uses the structure of the data itself to defined sub-groups (see section 5.1). It is of course possible to classify data based on labelled and unlabelled data. This extension of the supervised classification problem described above is called semi-supervised learning. In this case, the training data consists of both labelled and unlabelled instances with the obvious goal of generating a better classifier than would be possible with the labelled data only. The *phenoDisco* algorithm, will be illustrated in that context (section 5.3).

## 5.1 Unsupervised ML

The `plot2D` can also be used to visualise the data on a PCA plot omitting any marker definitions, as shown on figure 10. This approach avoids any bias towards marker definitions and concentrate on the data and its underlying structure itself.

Alternatively, *pRoloc* also gives access to *MLInterfaces*'s `MLearn` unified interface for, among others, unsupervised approaches using k-means (figure 11 on page 30), hierarchical (figure 12 on page 31) or partitioning around medoids (figure 13 on page 32), clustering.

```
plot2D(tan2009r1, fcol = NULL)
```

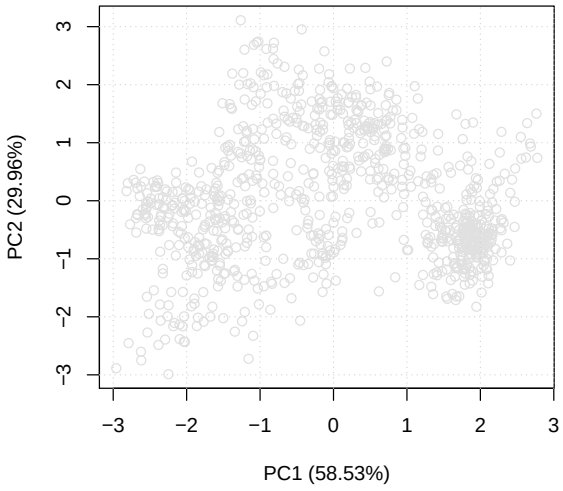


**Figure 10:** Plain PCA representation of the `tan2009r1` data.

```
kcl <- MLearn( ~ ., tan2009r1,  kmeansI, centers=5)
plot(kcl, exprs(tan2009r1))
```

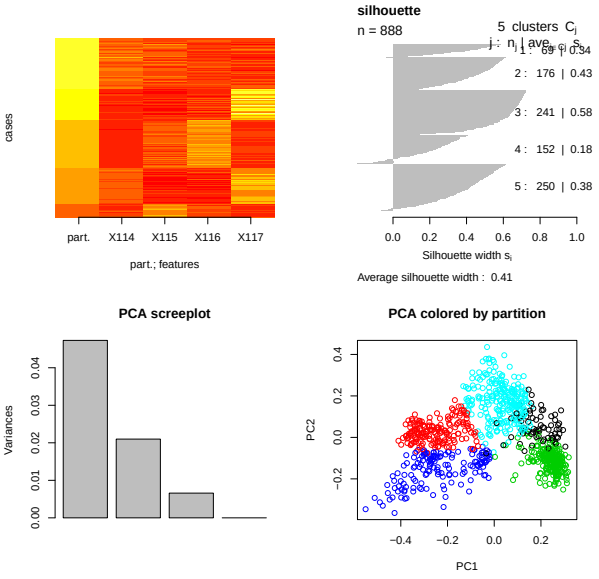


**Figure 11:** k-means clustering on the `tan2009r1` data.

```
hcl <- MLearn( ~ ., tan2009r1,
               hclustI(distFun = dist,
                       cutParm = list(k = 5)))
plot(hcl, labels = FALSE)
```

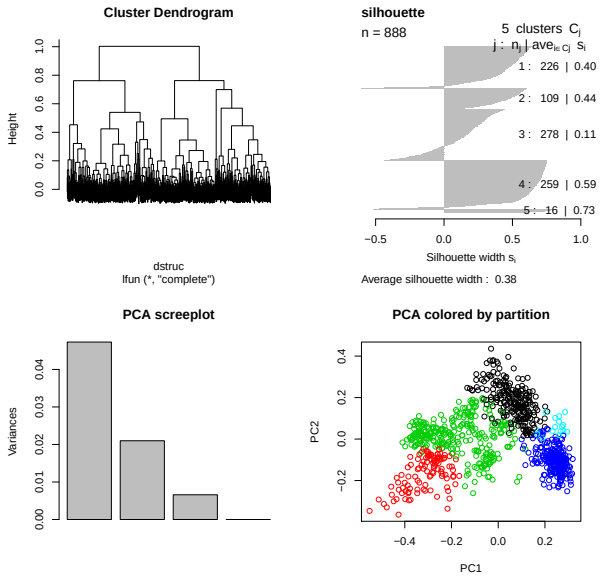


**Figure 12: Hierarchical clustering on the `tan2009r1` data.**

```
pcl <- MLearn( ~ ., tan2009r1,  pamI(dist), k = 5)
plot(pcl, data = exprs(tan2009r1))
```

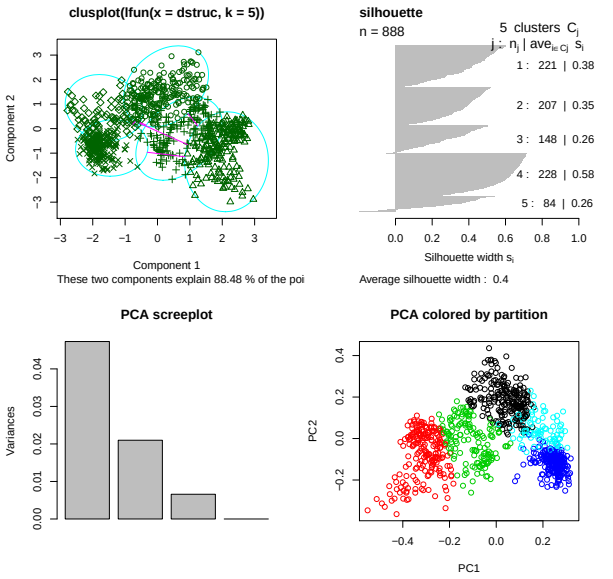


**Figure 13:** Partitioning around medoids on the `tan2009r1` data.

## 5.2 Supervised ML

In this section, we show how to use *pRoloc* to run a typical supervised ML analysis. Several ML methods are available, including k-nearest neighbour (knn), partial least square discriminant analysis (plsda), random forest (rf), support vector machines (svm), ... The detailed description of each method is outside of the scope of this document. We will use support vector machines to illustrate a typical pipeline and the important points that should be paid attention to. These points are equally valid and work, from a *pRoloc* user perspective, exactly the same for the other approaches.

### 5.2.1 Classification algorithm parameters optimisation

Before actually generating a model on the new markers and classifying unknown residents, one has to take care of properly setting the model parameters. Wrongly set parameters can have a very negative impact on classification performance. To do so, we create testing (to model) and training (to predict) subsets using known residents, i.e. marker proteins. By comparing observed and expected classification prediction, we can assess how well a given model works using the macro F1 score (see below). This procedure is repeated for a range of possible model parameter values (this is called a grid search), and the best performing set of parameters is then used to construct a model on all markers and predict un-labelled proteins. For this parameter optimisation procedure to perform well and produce useful results, it is essential to run it with a reasonable amount of markers. In our experience, 15 such marker proteins are necessary.

Model accuracy is evaluated using the F1 score, $F1 = 2\,\frac{precision \times recall}{precision + recall}$, calculated as the harmonic mean of the precision ($precision = \frac{tp}{tp+fp}$, a measure of *exactness* – returned output is a relevant result) and recall ($recall = \frac{tp}{tp+fn}$, a measure of *completeness* – indicating how much was missed from the output). What we are aiming for are high generalisation accuracy, i.e high $F1$, indicating that the marker proteins in the test data set are consistently correctly assigned by the algorithms.

In order to evaluate how well a classifier performs on profiles it was not exposed to during its creation, we implemented the following schema. Each set of marker protein profiles, i.e. labelled with known organelle

association, are separated into *training* and *test/validation* partitions by sampling 80% of the profile corresponding to each organelle (i.e. stratified) without replacement to form the training partition $S_{tr}$ with the remainder becoming the test/validation partition $S_{ts}$. The svm regularisation parameter $C$ and Gaussian kernel width $sigma$ are selected using a further round of stratified five-fold cross-validation on each training partition. All pairs of parameters $(C_i, sigma_j)$ under consideration are assessed using the macro F1 score and the pair that produces the best performance is subsequently employed in training a classifier on all training profiles $S_{tr}$ prior to assessment on all test/validation profiles $S_{ts}$. This procedure is repeated $N$ times (in the example below 10) in order to produce $N$ macro F1 estimated generalisation performance values (figure 14). This procedure is implemented in the `svmOptimisation`. See `?svmOptimisation` for details, in particular the range of $C$ and $sigma$ parameters and how the relevant feature variable is defined by the `fcol` parameters, which defaults to `"markers"`. Note that here, we demonstrate the function with only perform 10 iterations[5] (parametrised with `times=10`), which is enough for testing, but we recommend 100 (which is the default value) for a more robust analysis.

[5] In the interest of time, the optimisation is not executed but loaded from `dir(system.file("extdata", package="pRoloc"),full.names=TRUE,pattern="params.rda")`.

```
params <- svmOptimisation(tan2009r1, fcol = "markers.orig",
                          times = 10, xval = 5,
                          verbose = FALSE)
```

```
params
```

```
## Object of class "GenRegRes"
## Algorithm: svm
## Hyper-parameters:
##  cost: 0.0625 0.125 0.25 0.5 1 2 4 8 16
##  sigma: 0.01 0.1 1 10 100 1000
## Design:
##  Replication: 10 x 5-fold X-validation
##  Partitioning: 0.2/0.8 (test/train)
## Results
##  macro F1:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8889  0.8889  1.0000  0.9556  1.0000  1.0000
##  best sigma: 0.1 1
```

```
##  best cost: 0.5 1
```

In addition to the plots on figure 14, `f1Count(params)` returns, for each combination of parameters, the number of best (highest) F1 observations. One can use `getParams` to see the default set of parameters that are chosen based on the executed optimisation. Currently, the first best set is automatically extracted, and users are advised to critically assess whether this is the most wise choice.

```
f1Count(params)
```

```
##     0.5 1
## 0.1   1 0
## 1    NA 5
```

```
getParams(params)
```

```
## sigma  cost
##   0.1   0.5
```

*Important: It is essential to emphasise that the accuracy scores obtained during parameter optimisation are only a reflection of the classification performance on a set of distinct and ideally separated spatial clusters. Here, we assume that the data is characterised by good separation of the various spatial niches which are reflected by sub-cellular markers. Quality control of the data and the markers using the visualisation described in section 3 is essential and subsequent analyses are doomed to fail in the absence of such separation.*
*These classification scores are not representative of the reliability of the final classification (described in section 5.2.2), in particular along the boundaries separating the different sub-cellular niches. High scores at the optimisation stage are a requirement to proceed with the analysis, but are by no means indicative of the false positive rate in the final sub-cellular assignment of non-marker proteins.*

### 5.2.2 Classification

We can now re-use the result from our parameter optimisation (a best cost/sigma pair is going to be automatically extracted, using the `getParams` method, although it is possible to set them manually), and use

```
plot(params)
levelPlot(params)
```

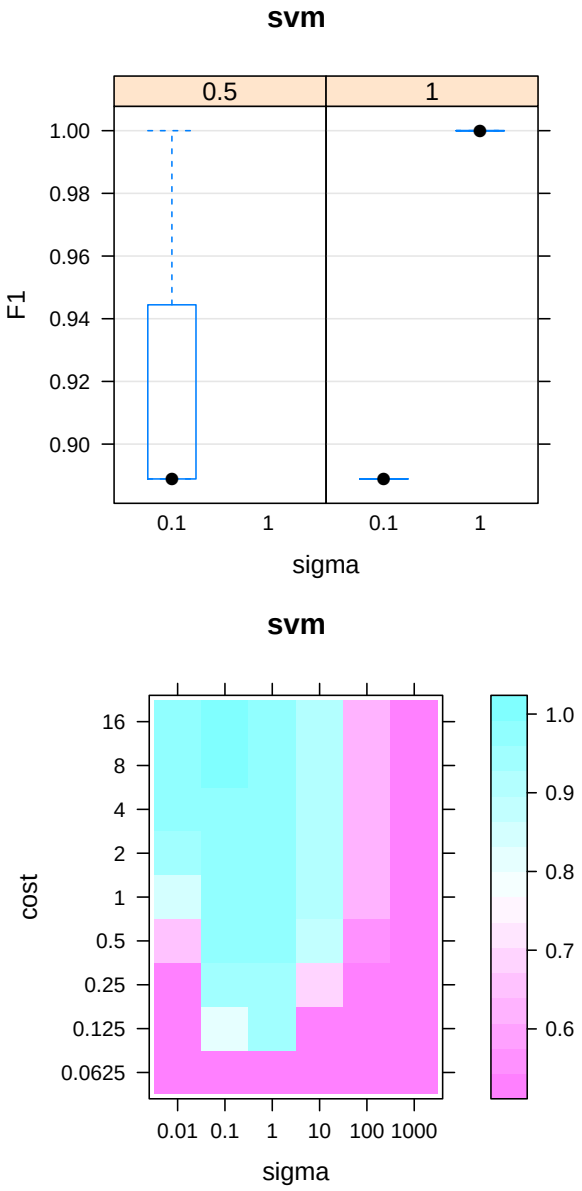


**Figure 14: Assessing parameter optimisation.** On the left, we see the respective distributions of the 10 macro F1 scores for the best cost/sigma parameter pairs. See also the output of `f1Count` in relation to this plot. On the right, we see the averaged macro F1 scores, for the full range of parameter values.

them to build a model with all the marker proteins and predict unknown residents using the `svmClassification` function (see the manual page for more details). By default, the organelle markers will be defined by the `"markers"` feature variables (and can be defined by the `fcol` parameter) e.g. here we use the original markers in `"markers.orig"` as a use case. New feature variables containing the organelle assignments and assignment probabilities, called scores hereafter, are automatically added to the `featureData` slot; in this case, using the `svm` and `svm.scores` labels.

*Important: The calculation of the classification probabilities is dependent on the classification algorithm. These probabilities are not to be compared across algorithms; they do* ***not*** *reflect any* ***biologically relevant*** *sub-cellular localisation probability but rather an algorithm-specific classification confidence score.*

```
## manual setting of parameters
svmres <- svmClassification(tan2009r1, fcol = "markers.orig",
                            sigma = 1, cost = 1)
```

```
## using default best parameters
svmres <- svmClassification(tan2009r1, fcol = "markers.orig",
                            assessRes = params)
processingData(svmres)
```

```
## - - - Processing information - - -
## Added markers from  'mrk' marker vector. Thu Jul 16 22:53:44 2015
## Performed svm prediction (sigma=0.1 cost=0.5) Fri Apr  7 18:03:28 2017
##  MSnbase version: 1.17.12
```

```
tail(fvarLabels(svmres), 4)
```

```
## [1] "markers"    "markers.tl" "svm"        "svm.scores"
```

The original markers, classification results and scores can be accessed with the `fData` accessor method, e.g. `fData(svmres)$svm` or `fData(svmres)$svm.scores`. Two helper functions, `getMarkers` and `getPredictions` are available and add some level of automation and functionality, assuming that the default feature labels are used. Both (invisibly) return the corresponding feature variable (the markers or assigned classification) and print a

summary table. The `fcol` parameter must be specified for `getPredictions`. It is also possible to defined a classification probability below which classifications are set to `"unknown"`.

```
p1 <- getPredictions(svmres, fcol = "svm")
```

```
## ans
##  Cytoskeleton            ER         Golgi
##             7           236            39
##      Lysosome       Nucleus            PM
##             8            21           285
##    Peroxisome    Proteasome  Ribosome 40S
##             4            15            20
##  Ribosome 60S mitochondrion
##            32           221
```

```
p1 <- fData(p1)$svm.pred
minprob <- median(fData(svmres)$svm.scores)
p2 <- getPredictions(svmres, fcol = "svm", t = minprob)
```

```
## ans
##  Cytoskeleton            ER         Golgi
##             7           167            21
##      Lysosome       Nucleus            PM
##             8            21           149
##    Peroxisome    Proteasome  Ribosome 40S
##             4            15            20
##  Ribosome 60S mitochondrion       unknown
##            32           109           335
```

```
p2 <- fData(p2)$svm.pred
table(p1, p2)
```

```
##                p2
## p1              Cytoskeleton  ER Golgi Lysosome
##   Cytoskeleton             7   0     0        0
##   ER                       0 167     0        0
##   Golgi                    0   0    21        0
##   Lysosome                 0   0     0        8
##   Nucleus                  0   0     0        0
##   PM                       0   0     0        0
##   Peroxisome               0   0     0        0
##   Proteasome               0   0     0        0
```

```
##   Ribosome 40S             0   0     0        0
##   Ribosome 60S             0   0     0        0
##   mitochondrion            0   0     0        0
##                p2
## p1              Nucleus  PM Peroxisome Proteasome
##   Cytoskeleton        0   0          0          0
##   ER                  0   0          0          0
##   Golgi               0   0          0          0
##   Lysosome            0   0          0          0
##   Nucleus            21   0          0          0
##   PM                  0 149          0          0
##   Peroxisome          0   0          4          0
##   Proteasome          0   0          0         15
##   Ribosome 40S        0   0          0          0
##   Ribosome 60S        0   0          0          0
##   mitochondrion       0   0          0          0
##                p2
## p1              Ribosome 40S Ribosome 60S
##   Cytoskeleton             0            0
##   ER                       0            0
##   Golgi                    0            0
##   Lysosome                 0            0
##   Nucleus                  0            0
##   PM                       0            0
##   Peroxisome               0            0
##   Proteasome               0            0
##   Ribosome 40S            20            0
##   Ribosome 60S             0           32
##   mitochondrion            0            0
##                p2
## p1              mitochondrion unknown
##   Cytoskeleton              0       0
##   ER                        0      69
##   Golgi                     0      18
##   Lysosome                  0       0
##   Nucleus                   0       0
##   PM                        0     136
##   Peroxisome                0       0
##   Proteasome                0       0
```

```
##   Ribosome 40S              0       0
##   Ribosome 60S              0       0
##   mitochondrion           109     112
```

To graphically illustrate the organelle-specific score distributions, we can use a boxplot and plot the scores for the respective predicted svm classes, as shown on figure 15. As can be seen, different organelles are characterised by different score distributions. Using a unique threshold (`minprob` with value 0.78 above) results in accepting 71 % of the initial ER predictions and only 49 % of the mitochondrion predictions. The `getPredictions` function also accepts organelle-specific score thresholds. Below, we calculate organelle-specific median scores.

```
boxplot(svm.scores ~ svm, data = fData(svmres),
        ylab = "SVM scores")
abline(h = minprob, lwd = 2, lty = 2)
```

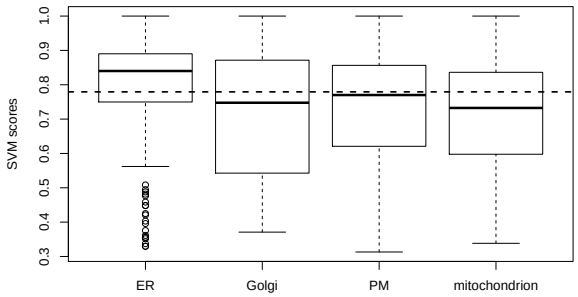


**Figure 15: Organelle-specific SVM score distributions.**

```
ts <- orgQuants(svmres, fcol = "svm", t = .5)
```

```
##           ER         Golgi           PM
##    0.8308459    0.6103146    0.7644641
## mitochondrion
##    0.7503573
```

Using these scores equates to choosing the 50% predictions with highest scores for organelle.

```
getPredictions(svmres, fcol = "svm", t = ts)
```

```
## ans
##  Cytoskeleton            ER         Golgi
##             7           132            26
##      Lysosome       Nucleus            PM
##             8            21           160
##    Peroxisome    Proteasome  Ribosome 40S
##             4            15            20
##  Ribosome 60S mitochondrion       unknown
##            32           125           338
## MSnSet (storageMode: lockedEnvironment)
## assayData: 888 features, 4 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: X114 X115 X116 X117
##   varLabels: Fractions
##   varMetadata: labelDescription
## featureData
##   featureNames: P20353 P53501 ... P07909 (888
##     total)
##   fvarLabels: FBgn Protein.ID ... svm.pred (19
##     total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
##   pubMedIds: 19317464
## Annotation:
## - - - Processing information - - -
## Added markers from  'mrk' marker vector. Thu Jul 16 22:53:44 2015
## Performed svm prediction (sigma=0.1 cost=0.5) Fri Apr  7 18:03:28 2017
## Added svm predictions according to thresholds: ER = 0.83, Golgi = 0.61, PM = 0.76, mi
##  MSnbase version: 1.17.12
```

We can now visualise these results using the plotting functions presented in section 5.1, as shown on figure 16. We clearly see that besides the organelle marker clusters that have been assigned high confidence members, many other proteins have substantially lower prediction scores.

```
ptsze <- exp(fData(svmres)$svm.scores) - 1
plot2D(svmres, fcol = "svm", fpch = "markers.orig",
       cex = ptsze)
addLegend(svmres, fcol = "svm",
          where = "bottomright",
          cex = .5)
```

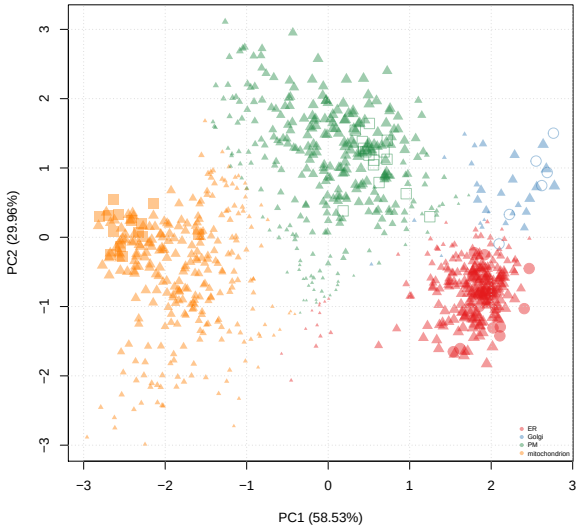


**Figure 16: Representation of the svm prediction on the 888 data set.** The svm scores have been used to set the point size (`cex` argument; the scores have been transformed to emphasise the extremes). Different symbols (`fpch`) are used to differentiate markers and new assignments.

## 5.3 Semi-supervised ML

It is obvious that the original set of markers initially used (ER, Golgi, mitochondrion, PM) is not a biologically realistic representation or the organelle diversity. Manually finding markers is however time consuming, as it requires careful verification of the annotation, and possibly critical for the subsequent analysis, as markers are directly used in the training phase of the supervised ML approach.

As can be seen in the PCA plots above, there is inherent structure in the data that can be made use of to automate the detection of new clusters. The *phenoDisco* algorithm [11] is an iterative method, that combines classification of proteins to known groups and detection of new clusters. It is available in *pRoloc* though the `phenoDisco` function[6].

```
pdres <- phenoDisco(tan2009r1, GS = 10,
                    times = 100, fcol = "PLSDA")
```

The results are also appended to the `featureData` slot.

```
processingData(pdres)
```

```
## - - - Processing information - - -
## Combined [888,4] and [1,4] MSnSets Wed Feb 13 17:28:54 2013
## Run phenoDisco using 'PLSDA': Wed Feb 13 17:28:54 2013
##   with parameters times=100, GS=10, p=0.05, r=1.
##  MSnbase version: 1.5.13
```

```
tail(fvarLabels(pdres), 3)
```

```
## [1] "PLSDA"   "markers" "pd"
```

The `plot2D` function, can, as previously, be utilised to visualise the results, as shown on figure 17.

## 5.4 Following up on novelty discovery

The newly discovered phenotypes need to be carefully validated prior to further analysis. Indeed, as the structure of the data is made use of in the discovery algorithm, some might represent peculiar structure in the data and not match with biologically relevant groups. The

[6] In the interest of time, `phenoDisco` is not executed when the vignette is dynamically built. The data object can be located with `dir(system.file("extdata", package="pRoloc"),full.names=TRUE,pattern="pdres.rda")` and loaded with `load`.

```
plot2D(pdres, fcol = "pd")
addLegend(pdres, fcol = "pd", ncol = 2,
          where = "bottomright",
          cex = .5)
```

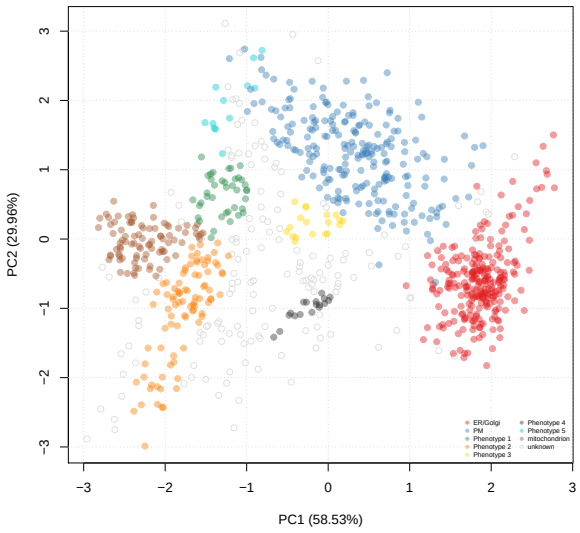


**Figure 17: Representation of the phenoDisco prediction and cluster discovery results.**

`tan2009r1` data has been submitted to a careful `phenodisco` analysis and validation in [11]. The results of this new, augmented marker set is available in the `pd.markers` feature data. These markers represent a combined set of the original markers and validated proteins from the new phenotypes.

```
getMarkers(tan2009r1, fcol = "pd.markers")
```

```
## organelleMarkers
##  Cytoskeleton            ER         Golgi
##             7            20             6
##      Lysosome       Nucleus            PM
##             8            20            15
##    Peroxisome    Proteasome  Ribosome 40S
##             4            11            14
##  Ribosome 60S mitochondrion       unknown
```

```
##           25           14          744
```

The augmented set of markers is now employed to repeat the classification using the support vector machine classifier. We apply a slightly different analysis than described in section 5.2.2 (page 35). In the code chunks below[7], we use class specific weights when creating the svm model; the weights are set to be inversely proportional to class frequencies.

[7] As previously, the results are pre-computed and available in the `extdata` package directory.

```
w <- classWeights(tan2009r1, fcol = "pd.markers")
w
```

```
##
##  Cytoskeleton            ER         Golgi
##    0.14285714    0.05000000    0.16666667
##      Lysosome       Nucleus            PM
##    0.12500000    0.05000000    0.06666667
##    Peroxisome    Proteasome  Ribosome 40S
##    0.25000000    0.09090909    0.07142857
##  Ribosome 60S mitochondrion
##    0.04000000    0.07142857
```

```
params2 <- svmOptimisation(tan2009r1, fcol = "pd.markers",
                           times = 10, xval = 5,
                           class.weights = w,
                           verbose = FALSE)
```

```
tan2009r1 <- svmClassification(tan2009r1, params2,
                               class.weights = w,
                               fcol = "pd.markers")
```

The data is visualised as described previously, were we use the svm classification a-posteriori probability to set the point size.

```
ptsze <- exp(fData(tan2009r1)$svm.scores) - 1
plot2D(tan2009r1, fcol = "svm", cex = ptsze)
addLegend(tan2009r1, fcol = "svm",
          where = "bottomright",
          ncol = 2, cex = .5)
```

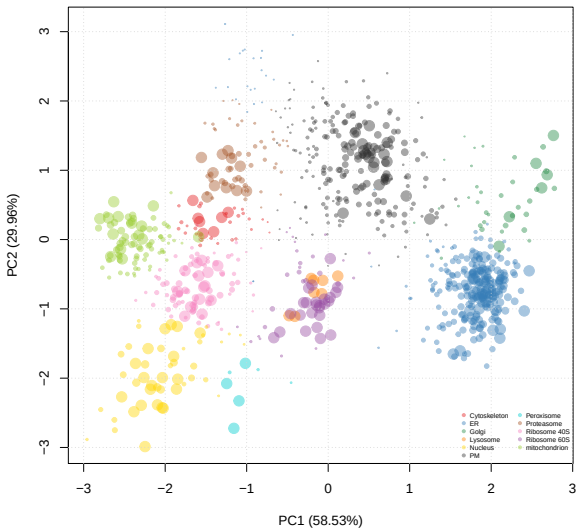


**Figure 18: Classification results.** The results of the second round of classification, using the augmented set of markers obtained using `phenoDisco` as detailed in [11] and a weighted svm classifier.

# 6 Conclusions

This tutorial focuses on practical aspects of organelles proteomics data analysis using *pRoloc*. Two important aspects have been illustrates: (1) data generation, manipulation and visualisation and (2) application of contemporary and novel machine learning techniques. Other crucial parts of a full analysis pipeline that were not covered here are raw mass-spectrometry quality control, quantitation, post-analysis and data validation.

Data analysis is not a trivial task, and in general, one can not assume that any off-the-shelf algorithm will perform well. As such, one of the emphasis of the software presented in this document is allowing users to track data processing and critically evaluate the results.

# Acknowledgement

We would like to thank Dr Daniel J.H. Nightingale, Dr Arnoud J. Groen, Dr Claire M. Mulvey and Dr Andy Christoforou for their organelle marker contributions.

# Session information

All software and respective versions used to produce this document are listed below.

- R version 3.3.3 (2017-03-06), `x86_64-apple-darwin13.4.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.36.2, Biobase 2.34.0, BiocGenerics 0.20.0, BiocParallel 1.8.2, BiocStyle 2.2.1, GO.db 3.4.0, IRanges 2.8.2, MLInterfaces 1.54.0, MSnbase 2.0.2, ProtGenerics 1.6.0, Rcpp 0.12.10, S4Vectors 0.12.2, XML 3.98-1.6, annotate 1.52.1, biomaRt 2.30.0, class 7.3-14, cluster 2.0.6, dplyr 0.5.0, hpar 1.16.0, knitr 1.15.1, mzR 2.8.1, pRoloc 1.14.6, pRolocdata 1.12.0, xtable 1.8-2
- Loaded via a namespace (and not attached): BiocInstaller 1.24.0, DBI 0.6-1, DEoptimR 1.0-8, FNN 1.1, MALDIquant 1.16.2, MASS 7.3-45, Matrix 1.2-8, MatrixModels 0.4-1, ModelMetrics 1.1.0, R6 2.2.0, RColorBrewer 1.1-2, RCurl 1.95-4.8, RSQLite 1.1-2, SparseM 1.76, affy 1.52.0, affyio 1.44.0, assertthat 0.1, backports 1.0.5, base64enc 0.1-3, bitops 1.0-6, car 2.1-4, caret 6.0-73, codetools 0.2-15, colorspace 1.3-2, dendextend 1.5.2, digest 0.6.12, diptest 0.75-7, doParallel 1.0.10, e1071 1.6-8, evaluate 0.10, flexmix 2.3-13, foreach 1.4.3, fpc 2.1-10, gbm 2.1.3, gdata 2.17.0, genefilter 1.56.0, ggplot2 2.2.1, ggvis 0.4.3, grid 3.3.3,

gridExtra 2.2.1, gtable 0.2.0, gtools 3.5.0, highr 0.6, htmltools 0.3.5, htmlwidgets 0.8, httpuv 1.3.3, hwriter 1.3.2, impute 1.48.0, iterators 1.0.8, jsonlite 1.3, kernlab 0.9-25, lattice 0.20-35, lazyeval 0.2.0, limma 3.30.13, lme4 1.1-12, lpSolve 5.6.13, magrittr 1.5, mclust 5.2.3, memoise 1.0.0, mgcv 1.8-17, mime 0.5, minqa 1.2.4, mlbench 2.1-1, modeltools 0.2-21, munsell 0.4.3, mvtnorm 1.0-6, mzID 1.12.0, nlme 3.1-131, nloptr 1.0.4, nnet 7.3-12, pbkrtest 0.4-7, pcaMethods 1.66.0, pls 2.6-0, plyr 1.8.4, prabclus 2.2-6, preprocessCore 1.36.0, proxy 0.4-17, quantreg 5.29, randomForest 4.6-12, rda 1.0.2-2, reshape2 1.4.2, rmarkdown 1.4, robustbase 0.92-7, rpart 4.1-10, rprojroot 1.2, sampling 2.8, scales 0.4.1, sfsmisc 1.1-0, shiny 1.0.1, splines 3.3.3, stringi 1.1.3, stringr 1.2.0, survival 2.41-3, threejs 0.2.2, tibble 1.3.0, tools 3.3.3, trimcluster 0.1-2, viridis 0.4.0, viridisLite 0.2.0, vsn 3.42.3, whisker 0.3-2, yaml 2.1.14, zlibbioc 1.20.0

# References

[1] Tom P. J. Dunkley, Svenja Hester, Ian P. Shadforth, John Runions, Thilo Weimar, Sally L. Hanton, Julian L. Griffin, Conrad Bessant, Federica Brandizzi, Chris Hawes, Rod B. Watson, Paul Dupree, and Kathryn S. Lilley. Mapping the arabidopsis organelle proteome. *Proc Natl Acad Sci USA*, 103(17):6518–6523, Apr 2006. URL: http://dx.doi.org/10.1073/pnas.0506958103, doi:10.1073/pnas.0506958103.

[2] Leonard J. Foster, Carmen L. de Hoog, Yanling Zhang, Yong Zhang, Xiaohui Xie, Vamsi K. Mootha, and Matthias Mann. A mammalian organelle map by protein correlation profiling. *Cell*, 125(1):187–199, Apr 2006. URL: http://dx.doi.org/10.1016/j.cell.2006.03.022, doi:10.1016/j.cell.2006.03.022.

[3] Laurent Gatto, Juan Antonio Vizcaíno, Henning Hermjakob, Wolfgang Huber, and Kathryn S Lilley. Organelle proteomics experimental designs and analysis. *Proteomics*, 2010. doi:10.1002/pmic.201000244.

[4] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0. URL: http://www.R-project.org/.

[5] Robert C. Gentleman, Vincent J. Carey, Douglas M. Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarry, Friedrich Leisch, Cheng Li, Martin Maechler, Anthony J. Rossini, Gunther Sawitzki, Colin Smith, Gordon Smyth, Luke Tierney, Jean Y. H. Yang, and Jianhua Zhang. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biol*, 5(10):–80, 2004. URL: http://dx.doi.org/10.1186/gb-2004-5-10-r80, doi:10.1186/gb-2004-5-10-r80.

[6] Laurent Gatto and Kathryn S Lilley. MSnbase – an R/Bioconductor package for isobaric tagged mass spectrometry data visualization, processing and quantitation. *Bioinformatics*, 28(2):288–9, Jan 2012. doi:10.1093/bioinformatics/btr645.

[7] Denise J Tan, Heidi Dvinge, Andy Christoforou, Paul Bertone, Alfonso A Martinez, and Kathryn S Lilley. Mapping organelle proteins and protein complexes in drosophila melanogaster. *J Proteome Res*, 8(6):2667–78, Jun 2009. doi:10.1021/pr800866n.

[8] Philip L. Ross, Yulin N. Huang, Jason N. Marchese, Brian Williamson, Kenneth Parker, Stephen Hattan, Nikita Khainovski, Sasi Pillai, Subhakar Dey, Scott Daniels, Subhasish Purkayastha, Peter Juhasz, Stephen Martin, Michael Bartlet-Jones, Feng He, Allan Jacobson, and Darryl J. Pappin. Multiplexed protein quantitation in saccharomyces cerevisiae using amine-reactive isobaric tagging reagents. *Mol Cell Proteomics*, 3(12):1154–1169, Dec 2004. URL: http://dx.doi.org/10.1074/mcp.M400129-MCP200, doi:10.1074/mcp.M400129-MCP200.

[9] RStudio and Inc. *shiny: Web Application Framework for R*, 2014. R package version 0.10.1. URL: http://CRAN.R-project.org/package=shiny.

[10] A Christoforou, C M Mulvey, L M Breckels, A Geladaki, T Hurrell, P C Hayward, T Naake, L Gatto, R Viner, A Martinez Arias, and K S Lilley. A draft map of the mouse pluripotent stem cell spatial proteome. *Nat Commun*, 7:8992, 2016. doi:10.1038/ncomms9992.

[11] Lisa M Breckels, Laurent Gatto, Andy Christoforou, Arnoud J Groen, Kathryn S Lilley, and Matthew W B Trotter. The effect of organelle discovery upon sub-cellular protein localisation. *J Proteomics*, Mar 2013. doi:10.1016/j.jprot.2013.02.019.