A short tutorial on using pRoloc for spatial proteomics data analysis

Laurent Gatto* and Lisa M. Breckels

Computational Proteomics Unit
University of Cambridge

March 21, 2015

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.

Keywords: Bioinformatics, organelle proteomics, machine learning, visualisation

*lg390@cam.ac.uk

Contents

Foreword

MSnbase and pRoloc are under active developed; current functionality is evolving and new features will be added. 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)

Questions and bugs

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

If you wish to reach a broader audience for general questions about proteomics analysis using `R`, you may want to use the Bioconductor mailing list[2].

[1] https://github.com/lgatto/pRoloc/issues
[2] https://stat.ethz.ch/mailman/listinfo/bioconductor

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 (Dunkley et al., 2006) or PCP (Foster et al., 2006) 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 Gatto et al. (2010) 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 (R Development Core Team, 2011) 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 (Gentleman et al., 2004). Two packages are of particular interest here, namely MSnbase (Gatto and Lilley, 2012) 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[3], power and flexibility of the R language and environment itself and a rich environment for specialised

[3]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.

functionality in many domains of bioinformatics: tools for many omics technologies, including proteomics, bio-statistics, gene ontology and biological pathway analysis, ... Although there exists some specific graphical user interfaces (GUI) – see for example the GUI implemented in the synapter package[4] for the analysis MS^E data, interaction with R is executed through a command line interface. Although 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.

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(paste0(getStockcol(), 70))
```

[4] http://bioconductor.org/packages/devel/bioc/html/synapter.html

2 Data structures

2.1 Test data description

The data used in this tutorial has been published in Tan et al. (2009). The LOPIT technique (Dunkley et al., 2006) 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 (Ross et al., 2004) 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`[5] 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)

[5] http://pubs.acs.org/doi/suppl/10.1021/pr800866n/suppl_file/pr800866n_si_004.xls

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: 1.14.2
```

2.2.3 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 4) 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.

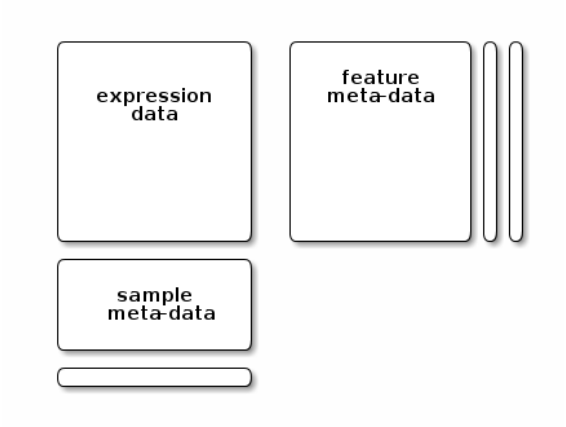

Figure 1: Dimension requirements for the respective expression, feature and sample meta-data slots.

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
- 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 Tan et al. (2009), are distributed as `MSnSet` instances in the `pRolocdata` package. Others include, among others, the *Arabidopsis thaliana* LOPIT data from Dunkley et al. (2006) (`dunkley2006`) and the mouse PCP data from Foster et al. (2006) (`foster2006`). Each data set can be loaded with the `data` function, as show below for the first replicate from Tan et al. (2009).

```
> 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 with `table(fData(tan2009r1)$markers.orig)` and `table(fData(tan2009r1)$PLSDA)` respectively.

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

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

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

organelleMarkers
     ER/Golgi mitochondrion            PM
          235            74           180
      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 proteins when proceeding with data visualisation and clustering (sections 3 and 4.1) and classification analysis (section 4.2.2).

2.3 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()

6 marker lists available:
Arabidopsis thaliana [atha]:
 Ids: TAIR, 543 markers
Drosophila melanogaster [dmel]:
 Ids: Annotation symbol, 144 markers
Gallus gallus [ggal]:
 Ids: IPI, 102 markers
Homo sapiens [hsap]:
 Ids: Uniprot, 205 markers
Mus musculus [mmus]:
 Ids: Uniprot, 1305 markers
Saccharomyces cerevisiae [scer]:
 Ids: SGD, 128 markers

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

```
 FBgn0010638  FBgn0028689  FBgn0032799  FBgn0003732
        "ER" "Proteasome"         "ER"    "Nucleus"
 FBgn0028691  FBgn0017397
"Proteasome"    "Nucleus"

> table(pRolocmarkers("dmel"))


 Cytoskeleton            ER         Golgi
            7            20             6
    Lysosome Mitochondrion       Nucleus
            8            14            20
   Peroxisome            PM    Proteasome
            4            15            11
 Ribosome 40S  Ribosome 60S
           14            25
```

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.

2.4 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[6] and illustrate the effect of `normalise(..., method = "sum")`.

```
> ## create a small illustrative test data
> data(itraqdata)
> itraqdata <- quantify(itraqdata, method = "trap",
+                       reporters = iTRAQ4,
+                       verbose = FALSE, parallel = FALSE)
> ## 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
iTRAQ4 quantification by trapezoidation: Sat Mar 21 20:19:26 2015
Normalised (sum): Sat Mar 21 20:19:27 2015
 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
```

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

```
> 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.

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 sub-setting 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])
```

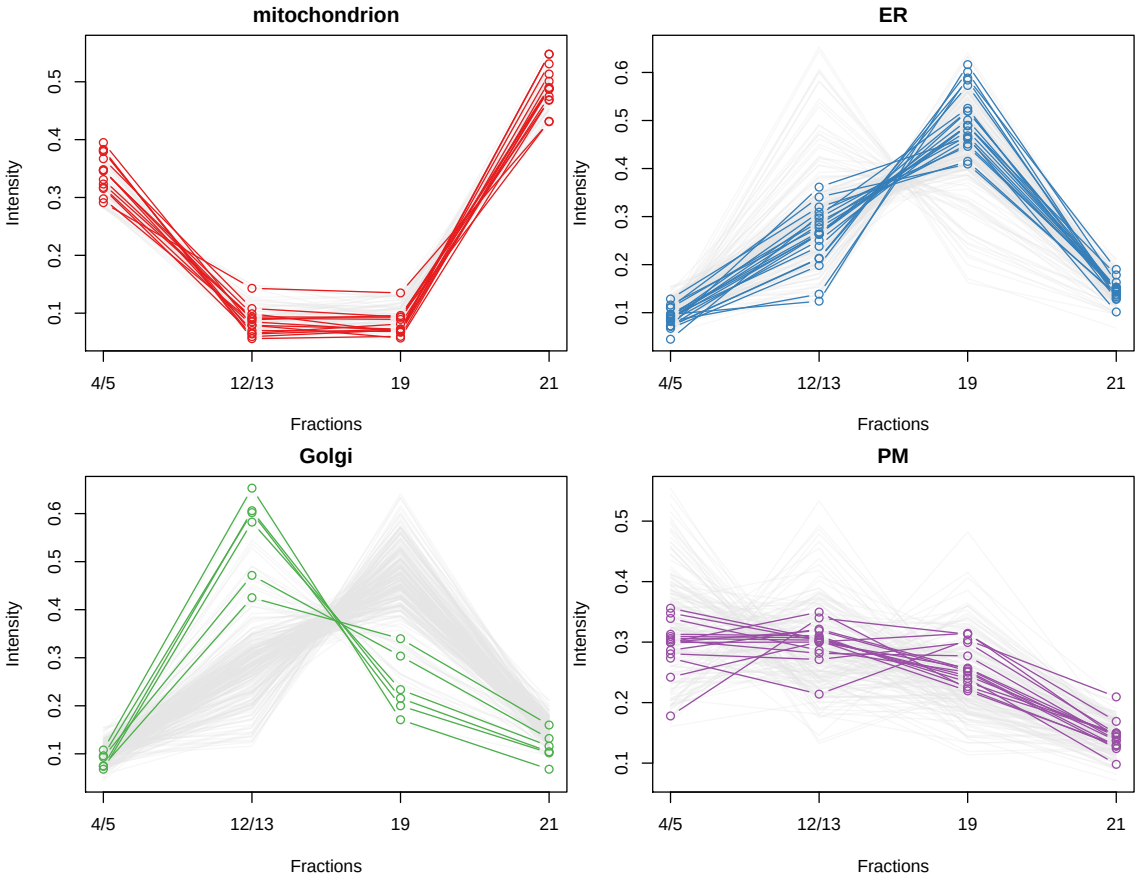

Figure 2: Distribution of protein intensities along the fractions of the separation gradient for 4 organelles: mitochondrion (red), ER/Golgi (blue, ER markers and green, Golgi markers) and plasma membrane (purple).

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 3). 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 (see figure 14 for a case with 11 clusters). These clusters are defined in a feature meta-data column (slot `featureData/fData`) that is declaraed with the `fcol` argument (default is `"markers"` which contains the most current known markers for the experiment under investigation, the original markers published with the data can be found in the slot `"markers.orig"`).

```
> plot2D(tan2009r1, fcol = "PLSDA", fpch = "markers.orig")
> addLegend(tan2009r1, fcol = "PLSDA",
+           where = "bottomright", bty = "n", cex = .7)
```

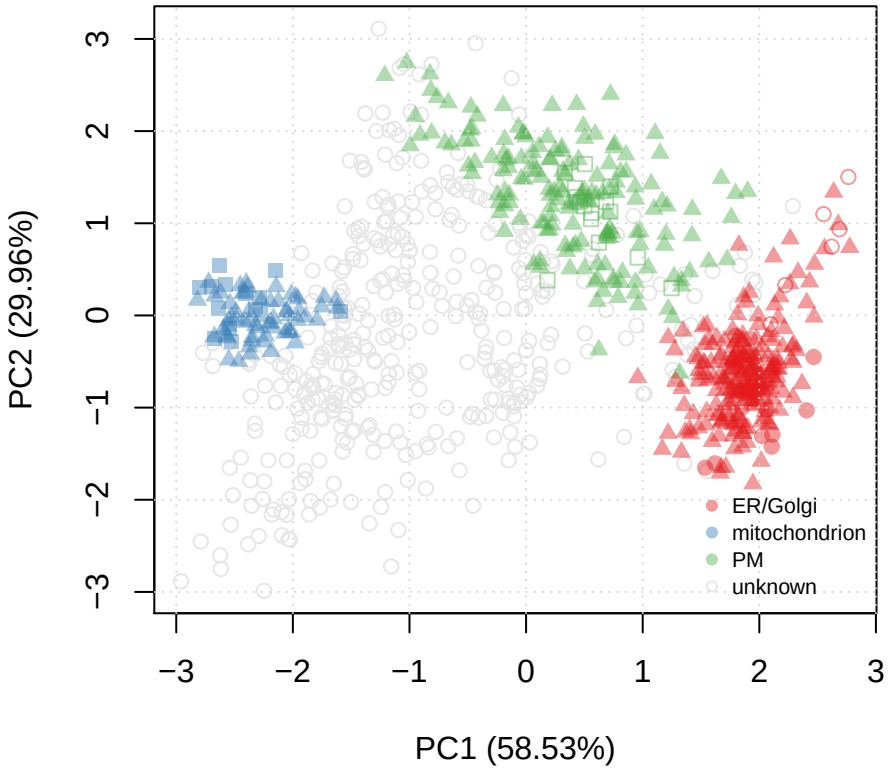

Figure 3: Representation of the 888 protein of `tan2009r1` after reduction of the 4 reporter quantitation data to 2 principal components.

3.1 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):

```
> 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.
```

`FeatureOfInterest` instances can now be overlaid on the PCA plot with the `highlightOnPlot` function.

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

```
> plot2D(tan2009r1, fcol = "PLSDA")
> addLegend(tan2009r1, fcol = "PLSDA",
+           where = "bottomright", bty = "n", 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)
```

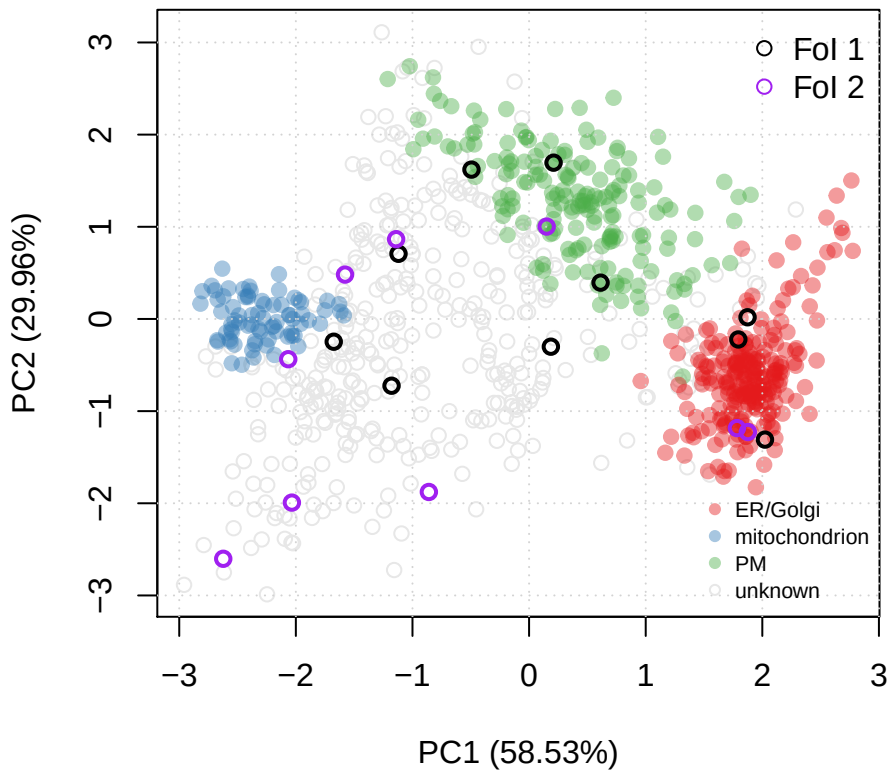

Figure 4: Adding features of interest on a PCA plot.

3.2 Interactive visualisation

The pRolocGUI application allows one to explore the spatial proteomics data using an interactive, web-based shiny interface (RStudio and Inc., 2014). The package is available from Bioconductor and can be installed and started as follows:

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

Figure 5: Screenshot of the pRolocGUI interface.

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[7].

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

4 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 4.2.2). 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 4.1). 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 4.3).

4.1 Unsupervised ML

The `plot2D` can also be used to visualise the data on a PCA plot omitting any marker definitions, as shown on figure 6. 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 7 on page 23), hierarchical (figure 8 on page 24) or partitioning around medoids (figure 9 on page 25), clustering.

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

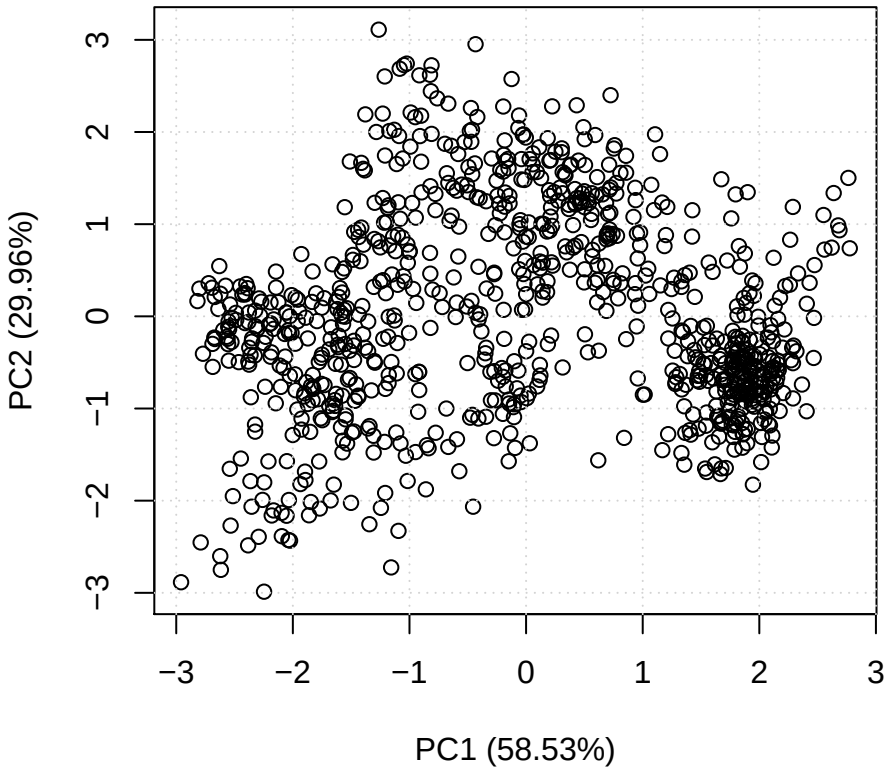

Figure 6: Plain PCA representation of the `tan2009r1` data.

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

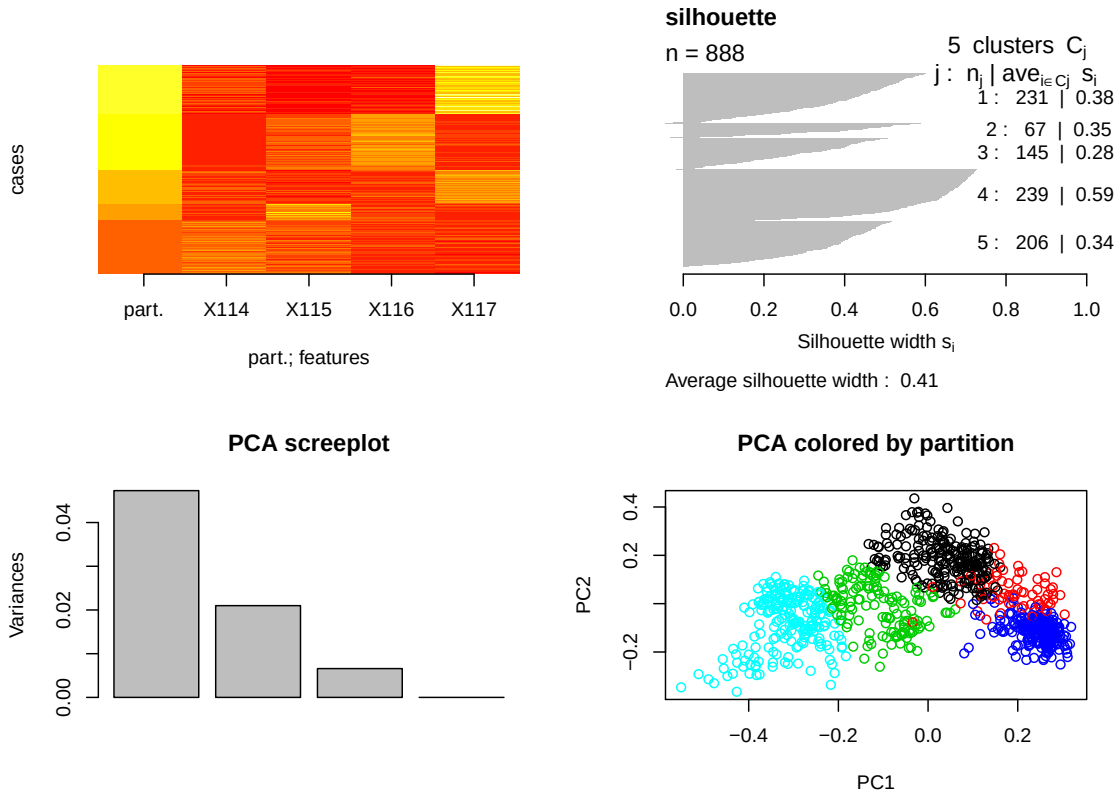

Figure 7: k-means clustering on the `tan2009r1` data.

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

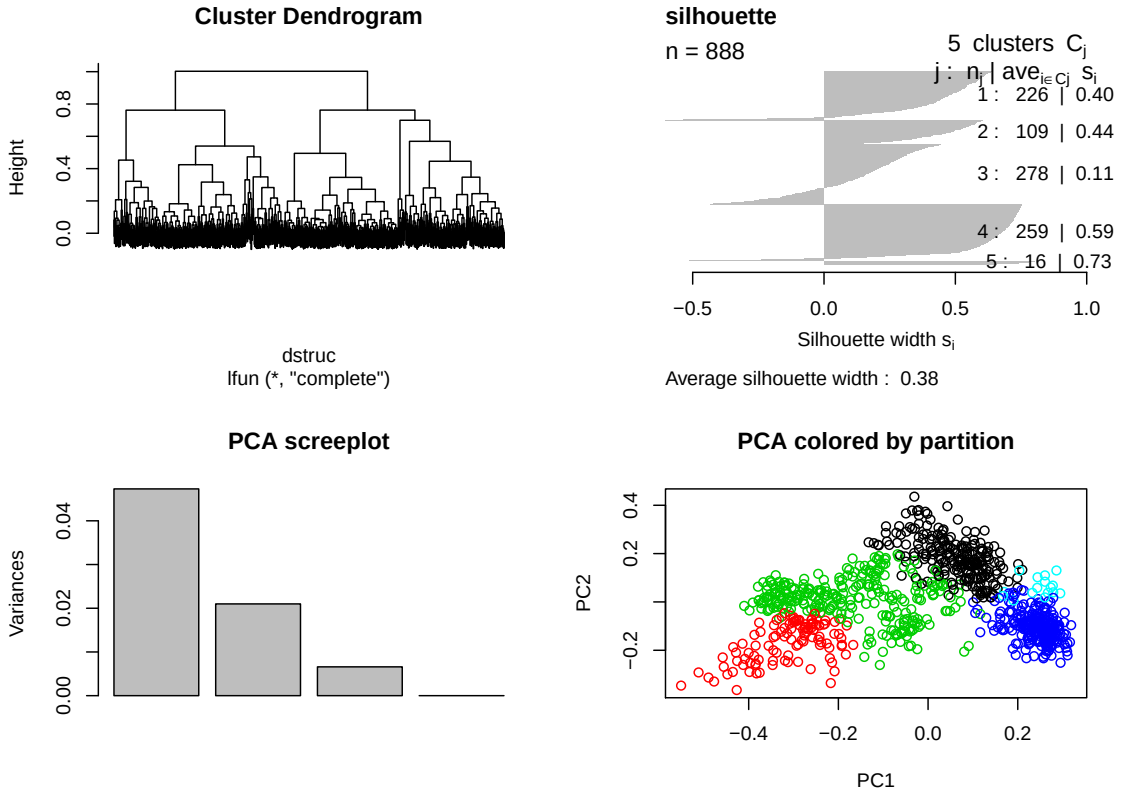

Figure 8: Hierarchical clustering on the `tan2009r1` data.

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

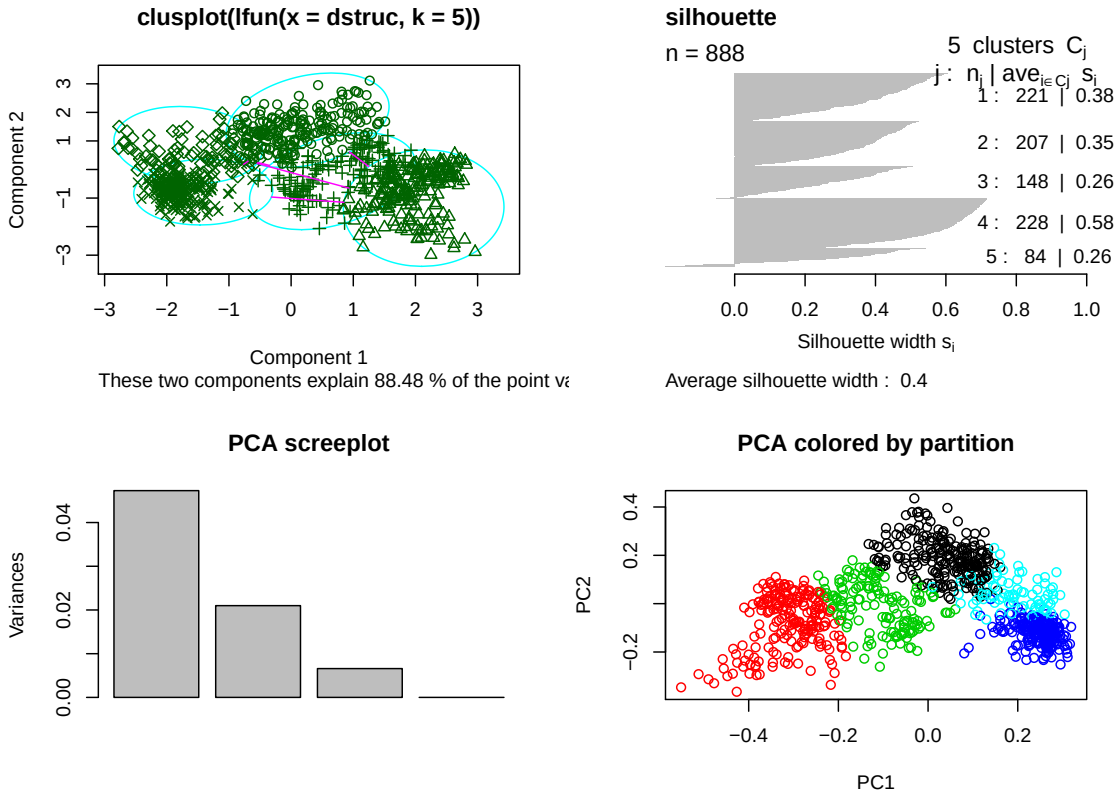

Figure 9: Partitioning around medoids on the `tan2009r1` data.

4.2 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.

4.2.1 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. 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.

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 10). 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[8] (`times = 10`), which is enough for testing, but we recommend 100 (which is the default value) for a more robust analysis.

```
> 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 10, `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
```

[8] 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")`.

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

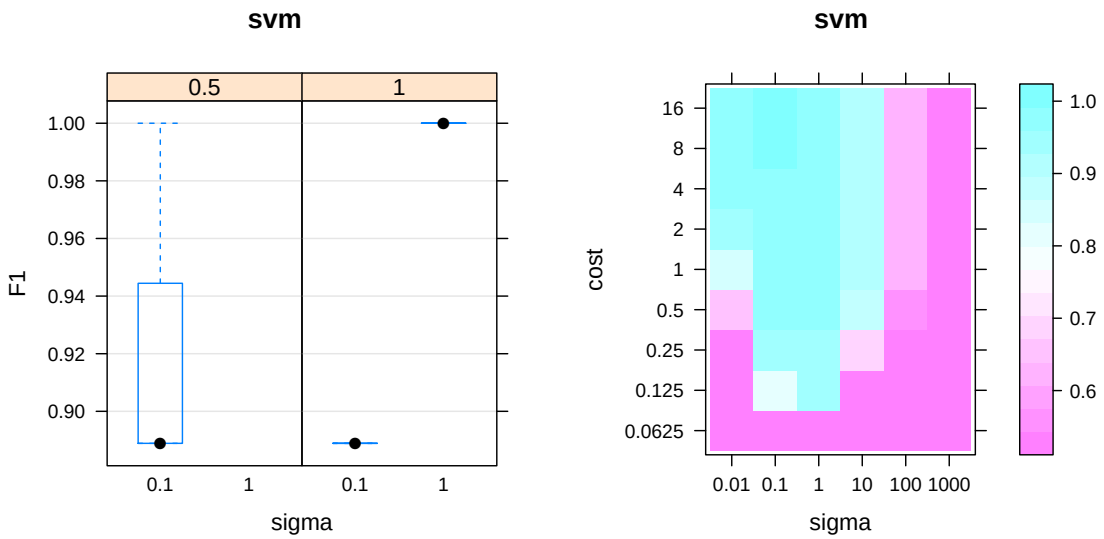

Figure 10: Assessing parameter optimisation. On the left, we see the respective distributions of the 10 macro F1 scores for the best cost/sigma parameter pairs. See also the output of `f1Count` in relation to this plot. On the right, we see the averaged macro F1 scores, for the full range of parameter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

```
> getParams(params)

sigma  cost
  0.1   0.5
```

4.2.2 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 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[9], called scores hereafter, are automatically added to the `featureData` slot; in this case, using the `svm` and `svm.scores` labels.

[9]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 Oct 30 18:21:04 2014
Performed svm prediction (sigma=0.1 cost=0.5) Sat Mar 21 20:19:31 2015
 MSnbase version: 1.13.16

> tail(fvarLabels(svmres), 4)

[1] "markers.orig" "markers"      "svm"
[4] "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
           ER         Golgi mitochondrion
          253            32           288
           PM
          315

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

ans
```

```
           ER          Golgi mitochondrion
          175             15            106
           PM       unknown
          148            444

> table(p1, p2)

               p2
p1               ER Golgi mitochondrion  PM unknown
  ER            175     0             0   0      78
  Golgi           0    15             0   0      17
  mitochondrion   0     0           106   0     182
  PM              0     0             0 148     167
```

To graphically illustrate the organelle-specific score distributions, use

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

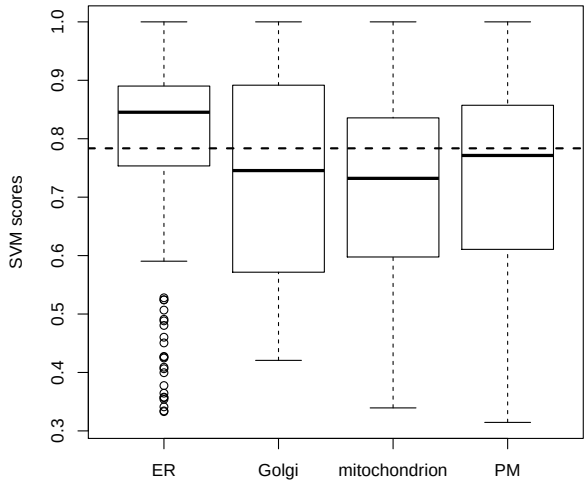

Figure 11: Organelle-specific SVM score distributions.

As can be see on figure 11, different organelles are characterised by different score distributions. Using a unique threshold (`minprob` with value 0.78 above) results in accepting 69 % of the initial ER predictions and only 37 % of the mitochondrion predictions. The `getPredictions` function also accepts organelle-specific score thresholds. Below, we calculate organelle-specific median scores.

```
> ## including marker scores
> (ts1 <- tapply(fData(svmres)$svm.scores, fData(svmres)$svm, median))

           ER         Golgi mitochondrion
    0.8453380     0.7455145     0.7321776
           PM
    0.7713502
```

Above, we include markers proteins (that have scores of 1 by default) when calculating the respective organelle-specific scores. Alternatively, once might choose to ignoring them.

```
> ## ignoring markers scores (i.e. scores == 1)
> (ts2 <- tapply(fData(svmres)$svm.scores, fData(svmres)$svm,
+                function(x) {
+                    ## assuming scores of 1 are markers
+                    scr <- median(x[x != 1])
+                    ## in case no proteins were assigned to an organelle,
+                    ## scr would be NA. Setting these cases to 1.
+                    ifelse(is.na(scr), 1, scr)
+                }))

           ER         Golgi mitochondrion
    0.8340708     0.7162588     0.7246074
           PM
    0.7575094
```

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

```
> getPredictions(svmres, fcol = "svm", t = ts2)

ans
           ER         Golgi mitochondrion
          137            19           151
           PM       unknown
          165           416
```

We can now visualise these results using the plotting functions presented in section 4.1, as shown on figure 12. 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", bty = "n", cex = .5)
```

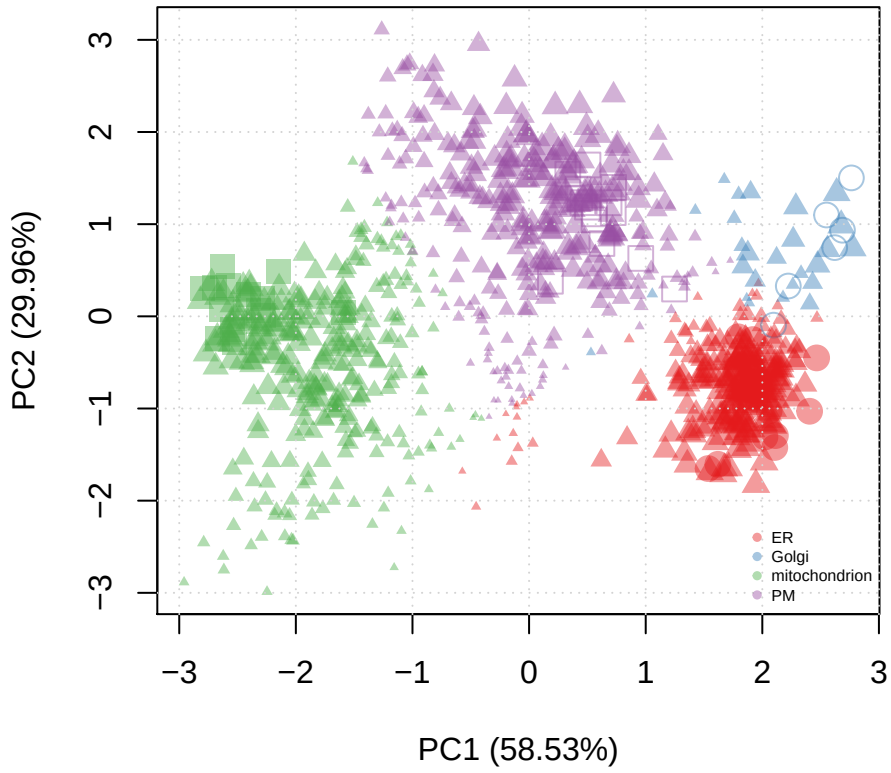

Figure 12: Representation of the svm prediction on the 888 data set. The svm scores have been used to set the point size (`cex` argument; the scores have been transformed to emphasise the extremes). Different symbols (`fpch`) are used to differentiate markers and new assignments.

4.3 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 (Breckels et al., 2013) 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[10].

```
> 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 13.

[10] 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", bty = "n", cex = .5)
```

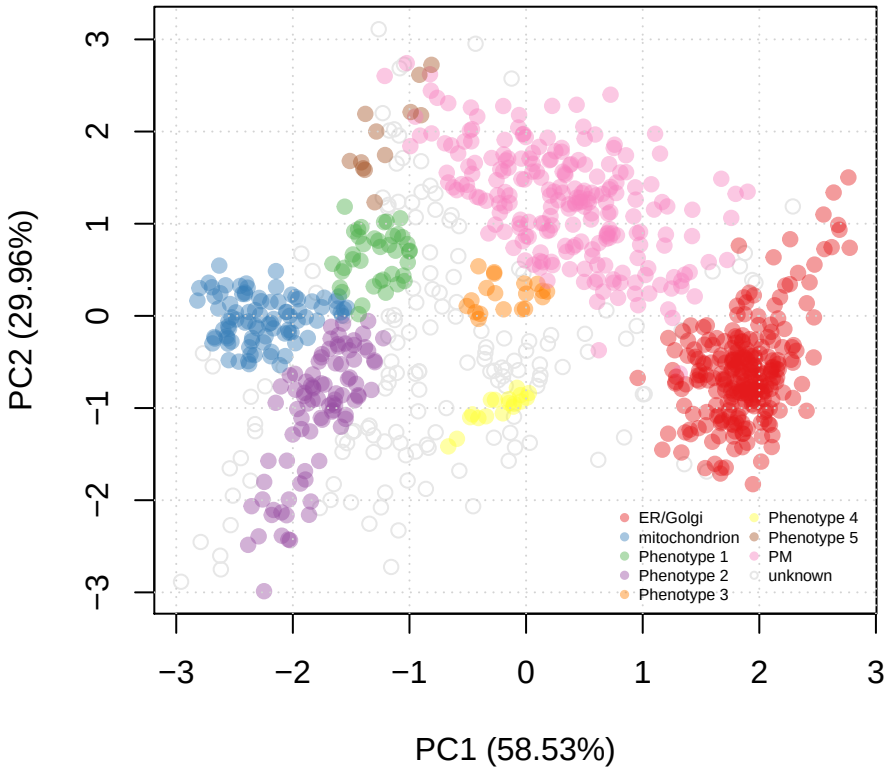

Figure 13: Representation of the phenoDisco prediction and cluster discovery results.

4.4 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 `tan2009r1` data has been submitted to a careful `phenodisco` analysis and validation in Breckels et al. (2013). 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 mitochondrion       Nucleus
            8            14            20
   Peroxisome            PM    Proteasome
            4            15            11
 Ribosome 40S  Ribosome 60S       unknown
           14            25           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 4.2.2 (page 28). In the code chunks below[11], we use class specific weights when creating the svm model; the weights are set to be inversely proportional to class frequencies.

```
> w <- table(fData(tan2009r1)[, "pd.markers"])
> (w <- 1/w[names(w) != "unknown"])


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

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

```
   0.07142857    0.04000000
```

```
> 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, bty = "n", cex = .5)
```

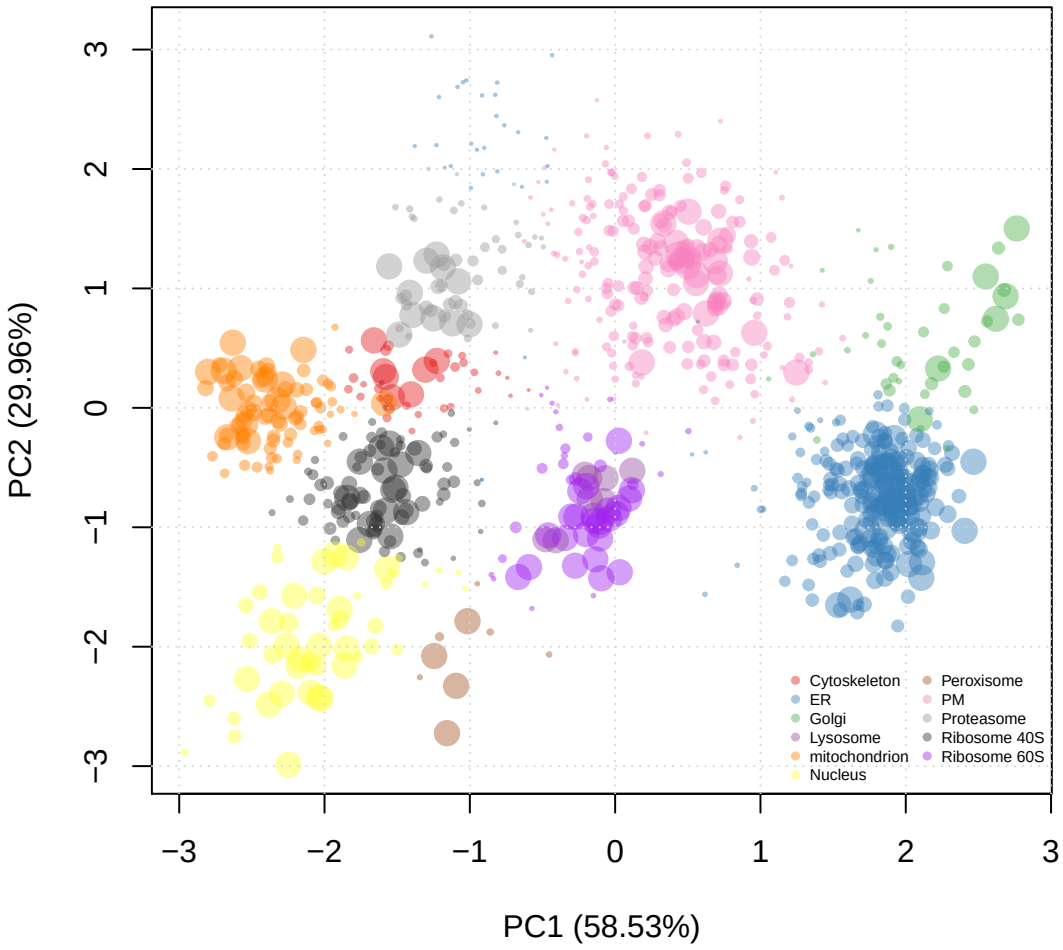

Figure 14: Second round of classification using the augmented set of markers obtained using `phenoDisco` as detailed in Breckels et al. (2013) and a weighted svm classifier.

5 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 Mr 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 3.1.3 (2015-03-09), `x86_64-apple-darwin10.8.0`
- Locale: `en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, stats4, utils
- Other packages: annotate 1.44.0, AnnotationDbi 1.28.2, Biobase 2.26.0, BiocGenerics 0.12.1, BiocParallel 1.0.3, class 7.3-12, cluster 2.0.1, GenomeInfoDb 1.2.4, IRanges 2.0.1, knitr 1.9, MLInterfaces 1.46.0, MSnbase 1.14.2, mzR 2.0.0, pRoloc 1.6.2, pRolocdata 1.4.1, Rcpp 0.11.5, S4Vectors 0.4.0, XML 3.98-1.1, xtable 1.7-4
- Loaded via a namespace (and not attached): affy 1.44.0, affyio 1.34.0, base64enc 0.1-2, BatchJobs 1.6, BBmisc 1.9, BiocInstaller 1.16.2, BradleyTerry2 1.0-6, brew 1.0-6, brglm 0.5-9, car 2.0-25, caret 6.0-41, checkmate 1.5.2, codetools 0.2-11, colorspace 1.2-6, DBI 0.3.1, digest 0.6.8, doParallel 1.0.8, e1071 1.6-4, evaluate 0.5.5, fail 1.2, FNN 1.1, foreach 1.4.2, formatR 1.0, gdata 2.13.3, genefilter 1.48.1, ggplot2 1.0.1, grid 3.1.3, gtable 0.1.2, gtools 3.4.1, highr 0.4, impute 1.40.0, iterators 1.0.7, kernlab 0.9-20, lattice 0.20-30, limma 3.22.7, lme4 1.1-7, lpSolve 5.6.10, MALDIquant 1.11, MASS 7.3-40, Matrix 1.1-5, mclust 4.4, mgcv 1.8-5, minqa 1.2.4, munsell 0.4.2, mvtnorm 1.0-2, mzID 1.4.1, nlme 3.1-120, nloptr 1.0.4, nnet 7.3-9, pbkrtest 0.4-2, pcaMethods 1.56.0, pls 2.4-3, plyr 1.8.1, preprocessCore 1.28.0, proto 0.3-10, proxy 0.4-14, quantreg 5.11, randomForest 4.6-10, RColorBrewer 1.1-2, rda 1.0.2-2, reshape2 1.4.1, rpart 4.1-9, RSQLite 1.0.0, sampling 2.6, scales 0.2.4, sendmailR 1.2-1, sfsmisc 1.0-27, SparseM 1.6, splines 3.1.3, stringr 0.6.2, survival 2.38-1, tools 3.1.3, vsn 3.34.0, zlibbioc 1.12.0

References

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

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. doi: 10.1073/pnas.0506958103. URL http://dx.doi.org/10.1073/pnas.0506958103.

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. doi: 10.1016/j.cell.2006.03.022. URL http://dx.doi.org/10.1016/j.cell.2006.03.022.

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.

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.

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. doi: 10.1186/gb-2004-5-10-r80. URL http://dx.doi.org/10.1186/gb-2004-5-10-r80.

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

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. doi: 10.1074/mcp.M400129-MCP200. URL http://dx.doi.org/10.1074/mcp.M400129-MCP200.

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

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.