Analysis of Label-free OOPS data

Manasa Ramakrishna

17 April, 2019

Contents

Introduction

This dataset is the first quick-LOPIT experiment exploring RBPs following the OOPS protocol. Given the small amounts of protein in some LOPIT fractions, the aim is to work out if a Label-free quantitation is a feasible way of analysing these proteins. Labelling with TMT requires that all fractions be scaled down to the lowest one and hence can compromise a full exploration of proteins.

There are some caveats to this experiment (1) It was the first of its kind (2) only 2 trizol-wash steps were performed rather than the standard three making the results a little noisy (3) RNAse wasn't added to the last step so there is a chance of some floating contaminating RNA (4) Input volume was greater than ideal so first wash wasn't as efficient.

However, it is worth looking at the data as it stands to see if it can provide any insights that TMT-labelled data cannot.

In the following lines of code, we will:
1. Look at the peptide-level raw data
2. Filter data to remove those mapping to multiple proteins
3. Imputing missing values
4. Aggregate the data into protein-level quantification
5. Filter the data to remove contaminants
6. Transform data to MSnSets and use those to analyse data 7. Try and classify unknown proteins based on known profiles 8. Look for functionally unifying themes across the 20 fractions 9. Validate findings using an independent dataset (LORNA)

Startup

We start by installing and loading the libraries required for our analysis. Additionally, tell R where you are running your program by setting your working directory as shown below using the variable 'wd'. We will use this later on. Also make your input and output directories (indir/outdir) as shown below.

```
suppressMessages(library(reshape2))
suppressMessages(library(ggplot2))
suppressMessages(library(ggsci))
suppressMessages(library(dplyr))
suppressMessages(library(MSnbase))
suppressMessages(library(ggbiplot))
suppressMessages(library(pRoloc))
suppressWarnings(library(mygene))
suppressWarnings(library(data.table))
suppressWarnings(library(patchwork))

# Setting working directories Note: Change the next
# line of card to point to your working directory
wd = "~/Documents/Work/TTT/02_Proteomics/15_OOPS-Label-Free-RBPs/OOPS-label-free/"
setwd(wd)
# getwd()

# Declaring input and output directories
indir = paste(wd, "Input", sep = "/")
outdir = paste(wd, "Output", sep = "/")
plotdir = paste(wd, "Plots", sep = "/")

# If output and plots directory exist, clear them
# out and start afresh
if (exists(outdir)) {
    system(paste0("rm -r", outdir))
}
if (exists(plotdir)) {
    system(paste0("rm -r", plotdir))
}

dir.create(outdir)
dir.create(plotdir)
```

01. Read in the peptide-level quantification (Raw data)

We'll start by reading the peptide-level quantification data into a dataframe. If we take a look at the colnames of the `peptides` dataframe, we can see we have 56 columns. We'll filter these to only keep the ones that are potantially useful to us. The first 12 columns describe the sequence of the peptide, the modifications which were detected and the protein which the peptide has been assigned to. Columns 13-32 indicate whether a given protein was found in the fraction or not.Columns 33-52 provide the quantification values for the 20 fractions of samples that have gone through qLOPIT followed by OOPS.The last 4 columns indicate the confidence of protein assignment across different algorithms used for peak identification.

Of these columns, we keep 4 information columns (Sequence, Modifications, Number.of.Proteins, Master.Protein.Accessions) and all area-under-the-curve value columns (33-52).

```
peptides <- read.table(paste(indir, "OOPS_qLOPIT_LabelFree_PeptideGroups.txt",
    sep = "/"), sep = "\t", header = T, stringsAsFactors = F)
# colnames(peptides)
# head(peptides[,c(3:4,8,10,33:52)])

# Keeping only those that are of use for downstream
# analysis
peptides_quant = peptides[, c(3:4, 8, 10, 33:52)]
colnames(peptides_quant) = gsub(".Sample", "", gsub("Area.",
    "", colnames(peptides_quant)))
dim(peptides_quant)
```

```
## [1] 29067    24
```

```
# View(peptides_quant)

# How many peptides in each fraction
missing = colSums(is.na(peptides_quant[, 5:24]))
peptide.nums = nrow(peptides_quant) - missing

# Plot missing/value fractions
t = cbind(Peptides = 100 * peptide.nums/nrow(peptides_quant),
    Missing = 100 * missing/nrow(peptides_quant))
tmelt = reshape2::melt(t)
colnames(tmelt) = c("Fraction", "Type", "Percentage")

gmiss = ggplot(tmelt, aes(Fraction, Percentage))
gmiss = gmiss + geom_bar(stat = "identity", aes(fill = Type)) +
    scale_fill_jco() + geom_hline(yintercept = mean(t[,
    2]), colour = "#CD534CFF", size = 1) + labs(title = "Missing-values-raw-data") +
    theme(plot.title = element_text(hjust = 0.5), axis.text.x = element_text(angle = 90,
        hjust = 1), legend.position = "none")
```

02. Removing multi-map proteins

The "Number.of.Proteins" column in `peptides_quant` tells us how many proteins each peptide has mapped too. We can see that 22598 (78%) out of 29067 peptides map uniquely to one protein while the rest don't. We will filter to remove these multi-mappings as it makes the data less reliable. In addition, we remove proteins that don't have a mapping to a Uniprot ID (column = Master.Protein.Accessions) as we cannot do much downstream analysis without the IDs.

```
table(peptides_quant$Number.of.Proteins)
```

```
##
##     1     2     3     4     5     6     7     8     9    10    11    12
## 23677  3824   807   302   143    66    44    51    41    18    15    12
##    13    14    15    16    17    18    19    20    21    22    23    24
##    10     5     2     5     2     2     2     4     2     8     3     2
##    25    27    29    31    35    37    43    44    46    58    61
##     4     2     1     2     2     1     3     1     2     1     1
```

```
# Remove non-unique peptide mappingsand those
# missing Master Protein assignments
pep.uniq = peptides_quant %>% filter(Number.of.Proteins ==
    1 & Master.Protein.Accessions != "")

pep.uniq = data.frame(pep.uniq)

dim(peptides_quant)
```

```
## [1] 29067    24
```

```
dim(pep.uniq)
```

```
## [1] 22598    24
```

```
# Loss
non.uniq.perc = 100 * (nrow(peptides_quant) - nrow(pep.uniq))/nrow(peptides_quant)  # 22.3%

# Re-draw the missing value plots for the filtered
# data
miss.uniq = colSums(is.na(pep.uniq[, 5:24]))
miss.rows = rowSums(is.na(pep.uniq[, 5:24]))
uniq.nums = nrow(pep.uniq) - miss.uniq

# Plot missing/value fractions
t.uniq = cbind(Peptides = 100 * uniq.nums/nrow(pep.uniq),
    Missing = 100 * miss.uniq/nrow(pep.uniq))
t.uniq.melt = melt(t.uniq)
colnames(t.uniq.melt) = c("Fraction", "Type", "Percentage")

guniq = ggplot(t.uniq.melt, aes(Fraction, Percentage))
guniq = guniq + geom_bar(stat = "identity", aes(fill = Type)) +
    scale_fill_jco() + geom_hline(yintercept = mean(t.uniq[,
    2]), colour = "#CD534CFF", size = 1) + labs(title = "Missing-values-after-multimap-removal") +
    theme(plot.title = element_text(hjust = 0.5), axis.text.x = element_text(angle = 90,
        hjust = 1))

# Print to file
pdf(paste(plotdir, "Fraction-of-missing-values.pdf",
    sep = "/"), paper = "a4r", width = 12, height = 8)
print(gmiss + guniq)
dev.off()
```

```
## pdf
##   2
```

The left-hand plot above shows what percentage of peptides have values (blue) and what percentage are missing (yellow) for each of the 20 fractions. The red dotted lines shows the average %Missing which is ~85%. This means that on average, each fraction has values for 15% of total peptides captured across all fractons. After removing multi-mapped peptides, the average missing fraction remains the same and we haven't lost a large number of proteins (visible changes in F2, F6).

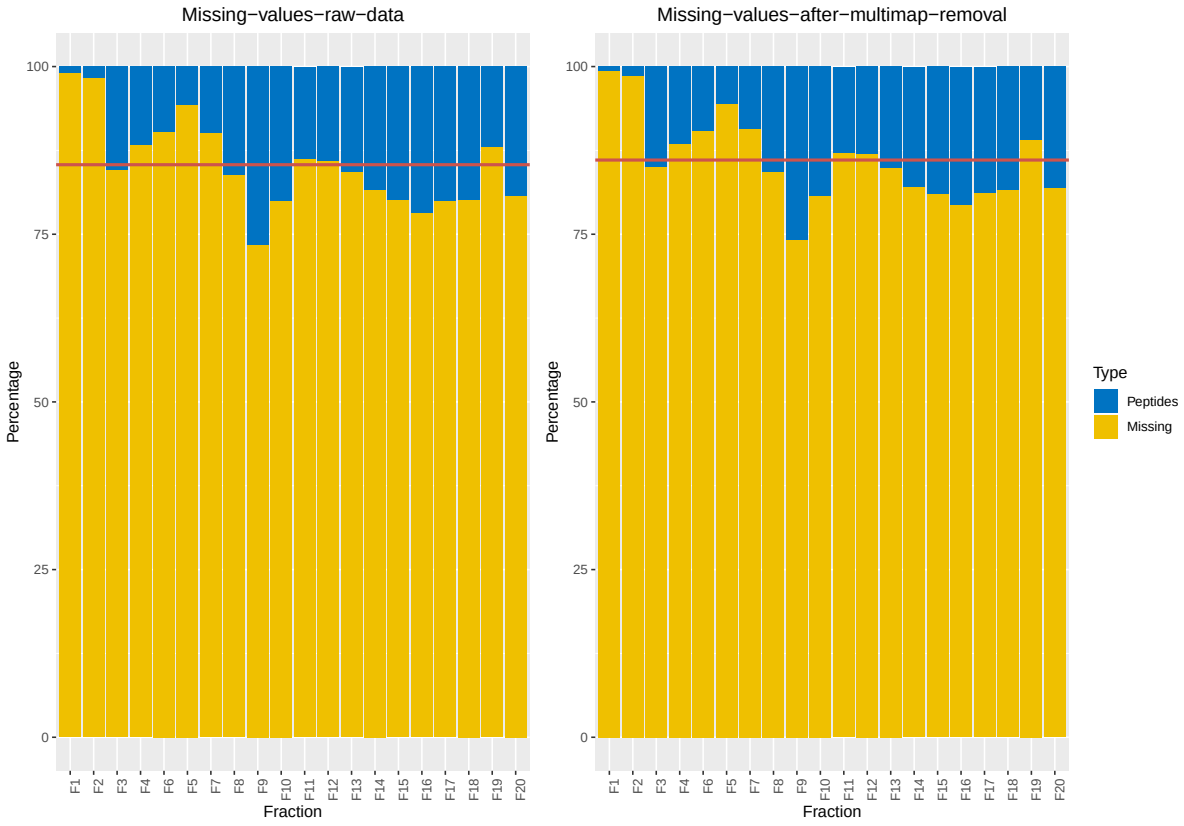

Figure 1: Barplot of missing values across fractions before and after multi-mapper protein removal

03. Imputing missing values

As you can see in the plots above, there are a lot of missing values per sample and if we do want to impute (extrapolate) some of them, we have to remove those that are missing across all fraction (at least majority of them).The paper https://pubs.acs.org/doi/pdf/10.1021/acs.jproteome.5b00981 indicates that it is always better to impute at the peptide level and then aggregate into proteins. Having gone through the steps of imputation (for both random and non-random missing values) it became apparent that we retain more proteins without imputation. This is because we have to remove huge chunks of data to be able to impute missing values in the first place.

An alternative to imputing values given how "lossy" the process is, is to keep all peptides and proceed with aggregating them into proteins in the hope of recovering more proteins than we did before. This also enables us to capture proteins across all fractions. Given this is a a label-free experiment, each fraction is a separate experiment so it is useful to see what we get by aggregating first without imputing.

04. Aggregating non-imputed data

We will merge AUC values for multiple peptides belonging to the same protein into one value per sample. We use the function 'summarize_all' in dplyr to do this. We add abundances across all peptides for a given protein.

```
suppressMessages(library(dplyr))

prot.dat = pep.uniq %>% dplyr::group_by(Master.Protein.Accessions) %>%
    dplyr::select(F1:F20) %>% dplyr::summarise_all(sum,
    na.rm = T) %>% data.frame()

dim(prot.dat)  # n = 2276
```

```
## [1] 2276   21
```

```
prot.dat = prot.dat[, c(1:5, 7, 6, 8:21)]
```

After aggregating, we do want to remove all proteins where despite the aggregation, there is no abundance value for any of the fractions. We will count up how many such proteins exist by counting up the zeroes in each row. We will remove all rows where there are only zeroes for all 20 fractions. Consider this a simple filtering step post aggregation.

```
# Counting zero abundances and removing all zero
# rows 82 such rows/proteins exist and are removed
table(rowSums(prot.dat[, 2:21] == 0))
```

```
##
##   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  11  23  39  29  37  32  49  50  64  73  96 111 109 125 141 169 231 233
##  18  19  20
## 289 283  82
```

```
prot.dat = prot.dat[-which(rowSums(prot.dat[, 2:21] ==
    0) == 20), ]
dim(prot.dat)  # n = 2194
```

```
## [1] 2194   21
```

```
# How many proteins are retrieved in each fraction?
prot.counts = nrow(prot.dat) - colSums(prot.dat ==
    0)[-1]
```

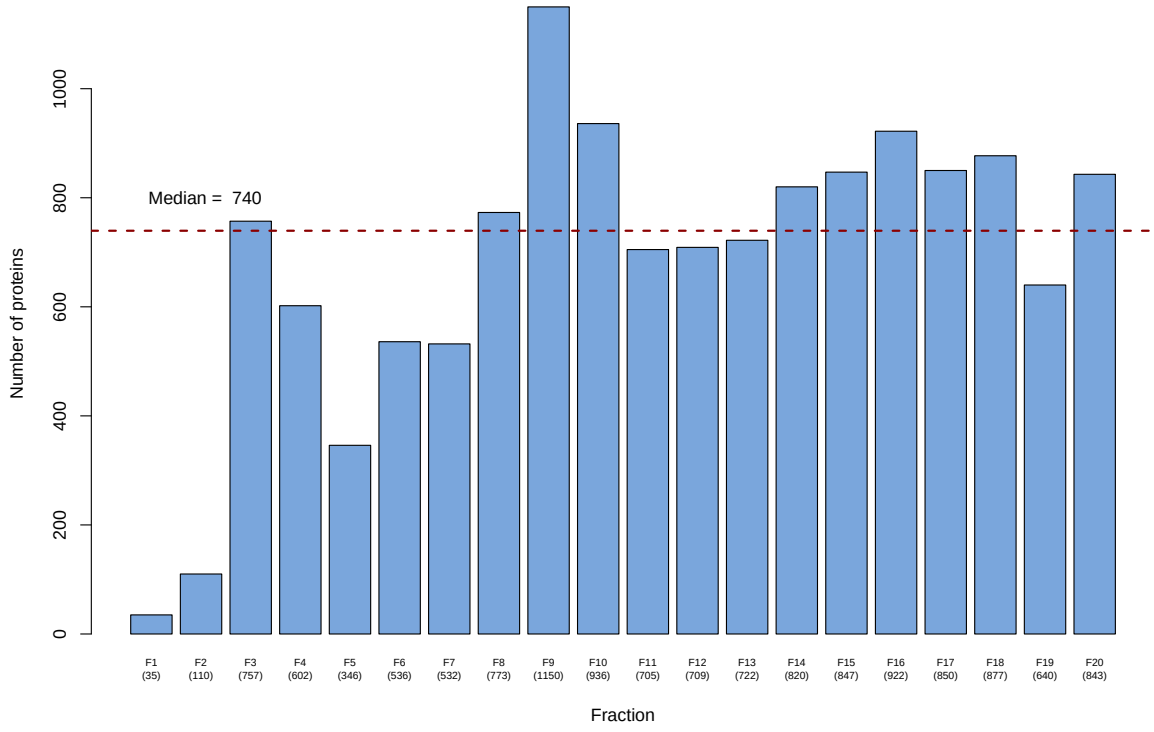

Figure 2: Barplot showing number of proteins in each fraction after aggegation of peptides

```
bp.names = paste(names(prot.counts), paste("(", prot.counts,
    ")", sep = ""), sep = "\n")

pdf(paste(plotdir, "Barplot-of-protein-counts-across-fractions.pdf",
    sep = "/"), paper = "a4r", width = 12, height = 8)
barplot(prot.counts, names.arg = bp.names, cex.names = 0.6,
    col = "#7AA6DCFF", ylab = "Number of proteins",
    xlab = "Fraction")
abline(h = median(prot.counts), col = "darkred", lty = 2,
    lwd = 2)
text(2, 800, paste("Median = ", round(median(prot.counts)[1],
    0)))
dev.off()
```

```
## pdf
##   2
```

As you can see from the plot above, there is a wide range of protein numbers across the 20 fractions with F1 and F2 being the least protein rich and F9, F10, F15 being highest in protein numbers. The red dotted line shows the median number of proteins across the fractions which is 740 proteins. The mean is 686 proteins if you are interested to know.

05. Further filtering

We haven't yet filtered for common proteomics "contaminants" such as proteins from hair, nail etc. There are a few ways of removing these.
a. Some are annotated by the prefix "cRAP"
b. Some map to non-human proteins. Given these are U2OS cells, we expect all to be human.
c. Some overlap with our `contam.txt` file
d. Finally, there are glycoproteins that appear in our list of RNA-binding proteins. We want to flag them to give us the option of removing them from downstream analyses.

```
# Make a copy of prot.dat in case you over-write it
store.dat = prot.dat
dim(store.dat)  # n = 2194
```

```
## [1] 2194   21
```

```
# 5a: Remove 'cRAP' proteins
rownames(prot.dat) = prot.dat$Master.Protein.Accessions
prot.dat = prot.dat[grep("cRAP", prot.dat$Master.Protein.Accessions,
    invert = T), ]  # Removing proteins annotate as cRAP in the list
dim(prot.dat)  # n = 2187
```

```
## [1] 2187   21
```

```
# Obtaining more protein-level information from
# Uniprot This list is uploaded to Uniprot and the
# additional annotations are downloaded from
# Uniprot. https://www.uniprot.org/uploadlists/ Of
# 2187 proteins, 2149 are human; rest to various
# other species.
write.table(prot.dat, paste(indir, "Aggregated-proteins-2187-for-Uniprot.txt",
    sep = "/"), sep = "\t", row.names = F, quote = F)

# Reading in additional annotations from Uniprot
uniprot.info = read.delim(paste(indir, "Aggregated-proteins-2187-with-uniprot.tab",
    sep = "/"), sep = "\t", header = T, stringsAsFactors = F)
rownames(uniprot.info) = uniprot.info$Entry
colnames(uniprot.info)[1] = "Query"
# sort(table(uniprot.info$Organism))

# 5b. Remove non-human proteins
non.human = uniprot.info[grep("Human", uniprot.info$Organism,
    invert = T), "Query"]
non.human.prots = prot.dat[non.human, ]
prot.dat = prot.dat[-which(prot.dat$Master.Protein.Accessions %in%
    non.human), ]
dim(prot.dat)  # n = 2221
```

```
## [1] 2149   21
```

```
# Merge prot.dat with uniprot information
human.rbps = merge(uniprot.info, prot.dat, by.x = "Query",
    by.y = "Master.Protein.Accessions", all.x = F,
    all.y = T)

# 5c. Remove any additonal contaminants based on
# the contamination file
```

```
contam = read.table(paste(indir, "contam.txt", sep = "/"),
    sep = "\t", header = F)

final.rbps = human.rbps[-which(human.rbps$Entry.name %in%
    contam$V1), ]
rownames(final.rbps) = final.rbps$Entry
dim(final.rbps)  # n = 2140
```

```
## [1] 2140   28
```

```
# How many proteins in each fraction ?
colSums(final.rbps[9:28] != 0)
```

```
##   F1   F2   F3   F4   F5   F6   F7   F8   F9  F10  F11  F12  F13  F14  F15
##   29  104  738  586  339  524  518  759 1125  916  686  690  702  803  828
##  F16  F17  F18  F19  F20
##  897  826  852  623  826
```

```
# 5d. Add glycosylation information
final.rbps$is.glyco = FALSE
final.rbps$is.glyco[which(final.rbps$Glycosylation !=
    "")] = TRUE
glycomelt = melt(final.rbps[, c(1, 9:29)]) %>% filter(value !=
    0)  # Only include those proteins that have an abundance value

# Count of proteins that are glycoproteins
glycocount = as.data.frame.table(table(glycomelt$variable,
    glycomelt$is.glyco))
colnames(glycocount) = c("Fraction", "is.glyco", "Count")

# Percentage of proteins that are glycoproteins
glycoperc = as.data.frame.table(100 * table(glycomelt$variabl,
    glycomelt$is.glyco)/rowSums(table(glycomelt$variabl,
    glycomelt$is.glyco)))
colnames(glycoperc) = c("Fraction", "is.glyco", "Percentage")
glycoperc$is.glyco = factor(glycoperc$is.glyco, levels = c("TRUE",
    "FALSE"))

# Plot barplot
suppressMessages(library(scales))
glycobar = ggplot(glycoperc, aes(fill = is.glyco, y = Percentage,
    x = Fraction)) + geom_col(position = "fill") +
    scale_y_continuous(labels = percent_format())

# How many RBPs do we have in total
dim(final.rbps)
```

```
## [1] 2140   29
```

```
pdf(paste(plotdir, "Glycoproteins-in-fractions.pdf",
    sep = "/"), paper = "a4r", width = 12, height = 8)
glycobar + labs(title = "Percentage of RBPs that are glycoproteins in each fraction") +
    theme(plot.title = element_text(hjust = 0.5))
dev.off()
```

```
## pdf
```

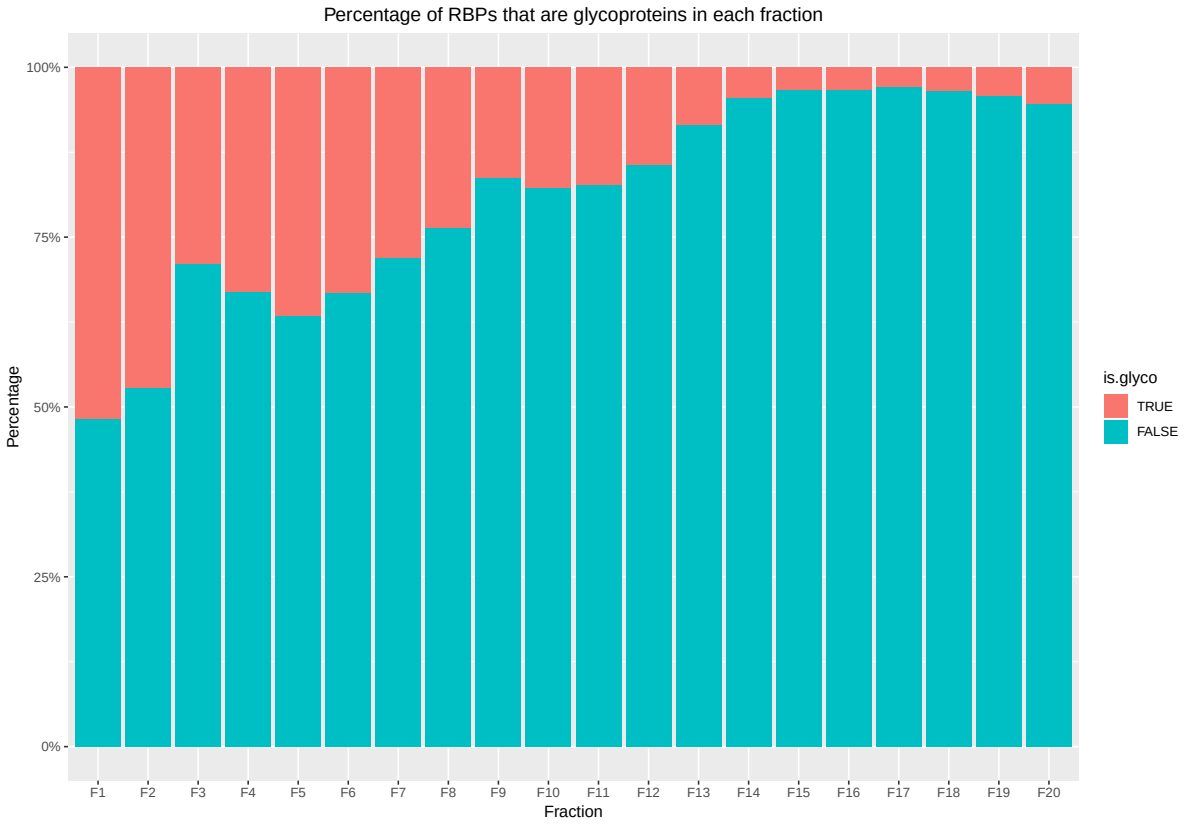

Figure 3: Percentage of RBPs that are glycoproteins in each fraction

```
##    2
```

There seem to be more glycoprotein RBPs in the earlier than in the latter fractions which corresponds to glycoproteins being more present in the Lysosomic/ER/Mitochondrial fraction. From a starting number of 2276, we now have 2140 proteins after filtering out contaminants. We are now interested in what these proteins are and if they show any association with the subcellular localisation that each fraction is linked to. We have performed Western Blot experiments using known localisation markers and we hope to see proteins in the fractions correlate to this blot.

06. Working with MSnSets

An MsnSet is an object that is used as part of the MSnbase package. It helps explore protein localisation data which is what we have. It is based largely on the ExpressionSet which was an object created to analyse microarray gene expression datWestern-blot.jpega.

6a. Creating an MSnSet

An MSnset contains 3 main slots - 'exprs' which contains protein abundance values, 'fData' which contains feature data or information about each protein the the dataset and 'pData' or phenotypic data which contains information about the samples (fractions) being studied using mass spectrometry.

```
head(final.rbps)
```

```
##              Query  Entry.name   Status
## UBA6_HUMAN  A0AVT1  UBA6_HUMAN reviewed
## MEX3A_HUMAN A1L020 MEX3A_HUMAN reviewed
## ILVBL_HUMAN A1L0T0 ILVBL_HUMAN reviewed
## PKHG3_HUMAN A1L390 PKHG3_HUMAN reviewed
## SPD2B_HUMAN A1X283 SPD2B_HUMAN reviewed
## NBAS_HUMAN  A2RRP1  NBAS_HUMAN reviewed
##
## UBA6_HUMAN  Ubiquitin-like modifier-activating enzyme 6 (Ubiquitin-activating enzyme 6) (EC 6.2.1.45)
## MEX3A_HUMAN
## ILVBL_HUMAN
## PKHG3_HUMAN                                                                                  Pleckstrin
## SPD2B_HUMAN                      SH3 and PX domain-containing protein 2B (Adapter protein HOFI) (Facto
## NBAS_HUMAN
##                          Organism Length Gene.names...primary..
## UBA6_HUMAN  Homo sapiens (Human)   1052                   UBA6
## MEX3A_HUMAN Homo sapiens (Human)    520                  MEX3A
## ILVBL_HUMAN Homo sapiens (Human)    632                  ILVBL
## PKHG3_HUMAN Homo sapiens (Human)   1219                PLEKHG3
## SPD2B_HUMAN Homo sapiens (Human)    911                SH3PXD2B
## NBAS_HUMAN  Homo sapiens (Human)   2371                   NBAS
##             Glycosylation F1 F2      F3      F4      F5 F6     F7       F8
## UBA6_HUMAN                  0  0       0       0       0  0      0        0
## MEX3A_HUMAN                 0  0       0       0       0  0      0        0
## ILVBL_HUMAN                 0  0       0 1900000       0  0 670000        0
## PKHG3_HUMAN                 0  0       0       0       0  0      0  6800000
## SPD2B_HUMAN                 0  0       0       0 2200000  0      0 12600000
## NBAS_HUMAN                  0  0 5300000       0       0  0      0  3170000
##                    F9      F10    F11     F12 F13 F14   F15 F16     F17
## UBA6_HUMAN          0        0      0       0   0   0 0e+00   0 8200000
## MEX3A_HUMAN         0        0      0       0   0   0 9e+06   0       0
## ILVBL_HUMAN   9200000  3500000 820000       0   0   0 0e+00   0       0
## PKHG3_HUMAN  50000000  6300000 900000       0   0   0 0e+00   0       0
## SPD2B_HUMAN  16800000 20700000      0       0   0   0 0e+00   0       0
## NBAS_HUMAN   15800000  8300000      0 5900000   0   0 0e+00   0       0
##                   F18     F19      F20 is.glyco
## UBA6_HUMAN   42400000 5200000 16400000    FALSE
## MEX3A_HUMAN   3300000       0        0    FALSE
## ILVBL_HUMAN         0       0        0    FALSE
## PKHG3_HUMAN         0       0        0    FALSE
## SPD2B_HUMAN         0       0  2900000    FALSE
## NBAS_HUMAN          0       0        0    FALSE
```

```
rownames(final.rbps) = final.rbps$Query

# Filling in the three slots
dat = final.rbps %>% dplyr::select(F1:F20) %>% as.matrix()
fd = final.rbps %>% dplyr::select(Query:is.glyco)
pd = data.frame(samples = colnames(dat))
rownames(pd) = pd$samples
```

Figure 4: Western blot for U2OS cells after qLOPIT and OOPS on 20 fractions

```
# Based on Western blot above
pd$Organelle = c("unknown", "Lysosome", "Lysosome+someER",
    "Lysosome+someER", "Lysosome+someER", "Lysosome+someER",
    "Lysosome+ER", "ER", "ER+Mitochondria", "ER+Mitochondria",
    "ER+someMitochondira", "Cytoplasm", "Nucleus+Histone+Cytoplasm",
    "Nucleus+Histone+Cytoplasm", "someNucleus+Cytoplasm",
    "someNucleus+Cytoplasm", "Cytoplasm", "Cytoplasm",
    "Cytoplasm", "Cytoplasm")

# We will add a bit more information to the fData
# file 1. List of known RBPs across cell lines in
# the XRNAX paper (Table S2)
xrnax = read.delim(paste(indir, "xrnax-genelist.txt",
    sep = "/"), sep = "\t", header = T)
xrnax.rbps = xrnax %>% dplyr::filter(!is.na(MCF7.RBP) |
    !is.na(HEK293.RBP) | !is.na(HeLa.RBP)) %>% dplyr::select(Uniprot.ID:Protein.name)
rownames(xrnax.rbps) = xrnax.rbps$Uniprot.ID

# Check how many are common to the cell lines in
# the XRNAX paper
xrnax %>% dplyr::select(MCF7.RBP:ihRBP) %>% apply(2,
    table, useNA = "always")
```

```
##             MCF7.RBP HEK293.RBP HeLa.RBP ihRBP
## non-poly(A)      617        698      565   775
## poly(A)          590        659      674   978
## <NA>            1276       1126     1244   730
```

```
# 2. List of RBPs from SILAC experiments using OOPS
# after wash step2 (Table S1)
oops = read.delim(paste(indir, "oops-genelist.txt",
    sep = "/"), sep = "\t", header = T)
oops.rbps = oops %>% dplyr::filter(step == 2 & CL_NC_Ratio >=
    1) %>% dplyr::select(master_protein, RBP_glyco)
rownames(oops.rbps) = oops.rbps$master_protein

# fData file augmentation head(fd)
fd$xrnax = FALSE
```

```
fd$xrnax[which(fd$Query %in% xrnax.rbps$Uniprot.ID)] = TRUE
fd$oops = FALSE
fd$oops[which(fd$Query %in% oops.rbps$master_protein)] = TRUE

# MsnSet creation with and without glycoproteins
# and filtering for high confidence RBPs
rbps.res <- MSnSet(exprs = dat, fData = fd, pData = pd)  # With glycoproteins, n = 2140
rbps.noglyc = rbps.res[which(fData(rbps.res)$is.glyco ==
    FALSE)]  # Without glycoproteins, n = 1851
rbps.highconf = rbps.noglyc[which(fData(rbps.noglyc)$xrnax ==
    TRUE | fData(rbps.noglyc)$oops == TRUE)]  # Verified as RBPS by XRNAX or OOPS and no glycoproteins,

# How many glycoproteins in each fraction?
# glycocount %>% tidyr::spread(key =
# is.glyco,value=Count) %>% dplyr::collect()
```

We have created three MSnSets here - 'rbps.res' which contains all potential RBPs across 20 fractions and 'rbps.noglyc' where glycoproteins have been removed in each fraction. Numbers show that there are nearly 300 glycoproteins, majority of them in the early fractions. Remember that there is redundancy in the proteins in each fraction so the non-redundant count is 297. Finally, we have rbps.highconf which only contain those rbps found in the XRNAX study (HeLa,HEK293,MCF7) or OOPS study (U2OS) and are not known glycoproteins.

6b. Using MSnSets for Plotting and Normalisation

The plotting functons below currently only show maps for the high-confindence dataset. This can be changed to show the raw data by swapping the definition of "sets" and "setnames" to those that are commented out. The imputation step has been left out as there are too many missing values to add any value post-imputation. The data is sum-normalised which means that each row (each protein) is divided by the sum of the expression of that protein across all 20 fractions. This is helpful while viewing the data, particularly using Profile Plots as one can easily see where a given protein peaks in expression.

```
library(RColorBrewer)
my_palette <- colorRampPalette(c("yellow", "purple"))(n = 20)

# Look at each MSnSet with various tools sets =
# list(rbps.res,rbps.noglyc,rbps.highconf) setnames
# =
# c('All.RBPs','RBPs.No.Glycoproteins','RBPs.Highconf.No.Glycoproteins')

sets = list(rbps.highconf)
setnames = c("RBPs.Highconf.No.Glycoproteins")
highconf.msnset = NULL

for (i in 1:length(sets)) {

    # The dataset
    ds = sets[[i]]
    dim(ds)

    # Heatmap of raw data Yellow = Missing; purple =
    # Non-missing)
    pdf(paste(plotdir, "Heatmap-of-OOPS-label-free-data.pdf",
        sep = "/"), paper = "a4r", width = 12, height = 8)
```

```
# heatmap(exprs(ds),Colv=NA,Rowv = NA,col =
# my_palette,labRow = NA, main =
# paste(setnames[i],'All data',sep = ' : '))
heatmap(exprs(ds), Colv = NA, Rowv = NA, col = my_palette,
    labRow = NA)
dev.off()

# Impute missing values if possible.  Using NBAVG
# as it is from samples collected along a density
# gradient ds = impute(ds,'nbavg')

# Re-draw heatmap after imputation
# heatmap(exprs(ds),Colv=NA,Rowv = NA,col =
# my_palette,labRow = NA, main =
# paste(setnames[i],'After NBAVG imputation',sep =
# ' : '))

# Add markers
fData(ds)$markers <- NULL
ds = addMarkers(ds, paste(indir, "FullHumanMarker.csv",
    sep = "/"), mcol = "markers")

# Normalise data
ds2 = normalise(ds, "sum")

# Profile plots with proLoc markers
orgs = unique(sort(fData(ds2)$markers))[-1]
cols = colorRampPalette(brewer.pal(9, "Set1"))(12)
pdf(paste(plotdir, "Profile-plots-of-OOPS-label-free-data.pdf",
    sep = "/"))
# jpeg(paste(plotdir,'Profile-plots-of-OOPS-label-free-data.jpg',sep='/'))
par(mfrow = c(4, 3))
for (k in 1:length(orgs)) {
    z = ds2[fData(ds2)$markers == orgs[k], ]
    plotDist(z, ylim = range(exprs(z), na.rm = T),
        pcol = cols[k], lty = 2, las = 2)
    title(main = paste(orgs[k], " (n = ", nrow(z),
        ")", sep = ""), cex = 0.8)
}
dev.off()

# Necessary to plot 2D maps
pdf(paste(plotdir, "PCA-plots-of-OOPS-label-free-data.pdf",
    sep = "/"), paper = "a4r", width = 12, height = 8)
par(mfrow = c(2, 2))
plot2D(ds2, main = paste(setnames[i], " Dims 1:2",
    sep = ""), fcol = "markers", dims = c(1, 2))
plot2D(ds2, main = paste(setnames[i], " Dims 1:3",
    sep = ""), fcol = "markers", dims = c(1, 3))
plot2D(ds2, main = paste(setnames[i], " Dims 2:3",
    sep = ""), fcol = "markers", dims = c(2, 3))
plot2D(ds2, fcol = "markers", dims = c(3, 4), col = "white")
addLegend(ds2, fcol = "markers", where = "center",
```

```
        cex = 0.75)
    par(mfrow = c(1, 1))
    dev.off()

    # Mark glycoproteins in the data if it hasn't been
    # filtered out
    if (setnames[i] == "All.RBPs") {
        # Highlighting glycoproteins on plot fro all
        # proteins only
        foi1 <- FeaturesOfInterest(description = "Glycoproteins",
            fnames = fData(ds)$Query[which(fData(ds)$is.glyco ==
                TRUE)])

        foi2 <- FeaturesOfInterest(description = "High-conf-RBPs",
            fnames = fData(ds)$Query[which(fData(ds)$xrnax ==
                TRUE | fData(ds)$oops == TRUE)])

        par(mfrow = c(1, 1))
        plot2D(ds2, fcol = "markers", dims = c(1, 2),
            main = "All RBPs with Glycoproteins marked")
        addLegend(ds2, fcol = "markers", where = "bottomright",
            cex = 0.7)
        highlightOnPlot(ds2, foi1, col = "black", lwd = 1,
            pch = 4, cex = 0.5)
        # highlightOnPlot(ds2, foi2,col = 'purple',lwd = 1)
        # legend('topright', c('Glycoproteins',
        # 'High-conf-RBPs'),bty = 'n', col = c('black',
        # 'purple'),pch = 1)
        legend("topright", c("Glycoproteins"), bty = "n",
            col = c("black"), pch = 4)
    }
    highconf.msnset = ds2
}
```

```
## organelleMarkers
##           CYTOSOL                ER            GOLGI      MITOCHONDRIA
##                10                14                 2                17
##           NUCLEUS NUCLEUS-CHROMATIN                PM        PROTEASOME
##                29                13                 3                 2
##      RIBOSOME 40S      RIBOSOME 60S           unknown
##                20                32               794
```

From the the PCA, we can see the separation of proteins along three axes : ER+Mitochondria, Nucleus+Nucleus-Chromatin,Cytosol+Ribosomes. From the profile plots, we have clear resolution for ER, Mitochondria, Nucleus, Nucleus-Chromatin, Ribosome 40S and 60S. Golgi and Lysosome to a lesser extent.

From the highlighted Glycoproteins plot, it appears that glycoproteins are not present in the nuclear/cytosolic fractions but in the membranous fractions as well as the ER and Mitochondria.

07. Predicting classification for unknown markers

Knowing some markers, it can be useful to extrapolate to those proteins who don't have a cellular localisation. That's what we hope to achieve in this segment. However, we have a very small fraction of known markers so the prediction of the unknown might be a bit ropey.

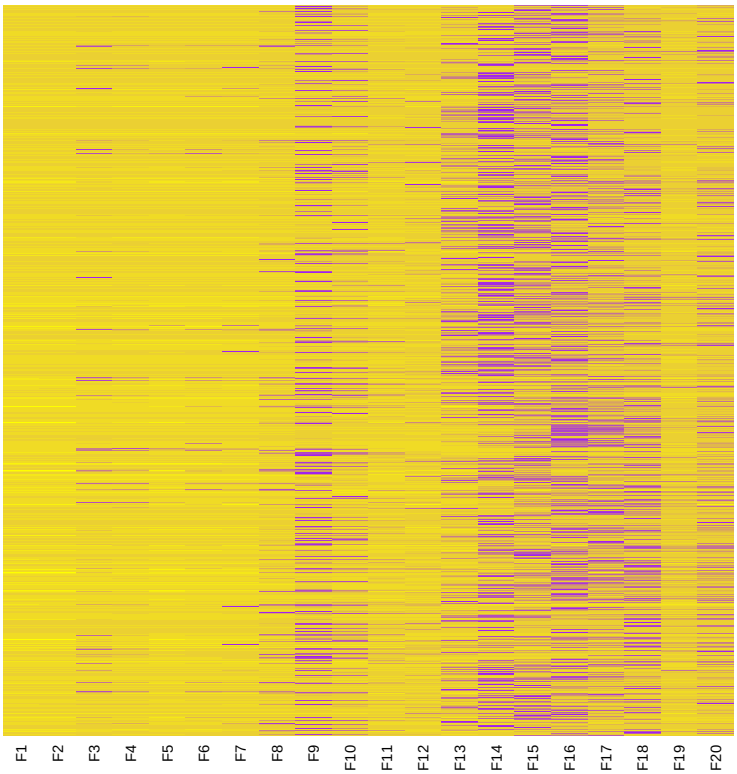

Figure 5: Heatmap of protein abundance from label-free OOPS on 20 fractions

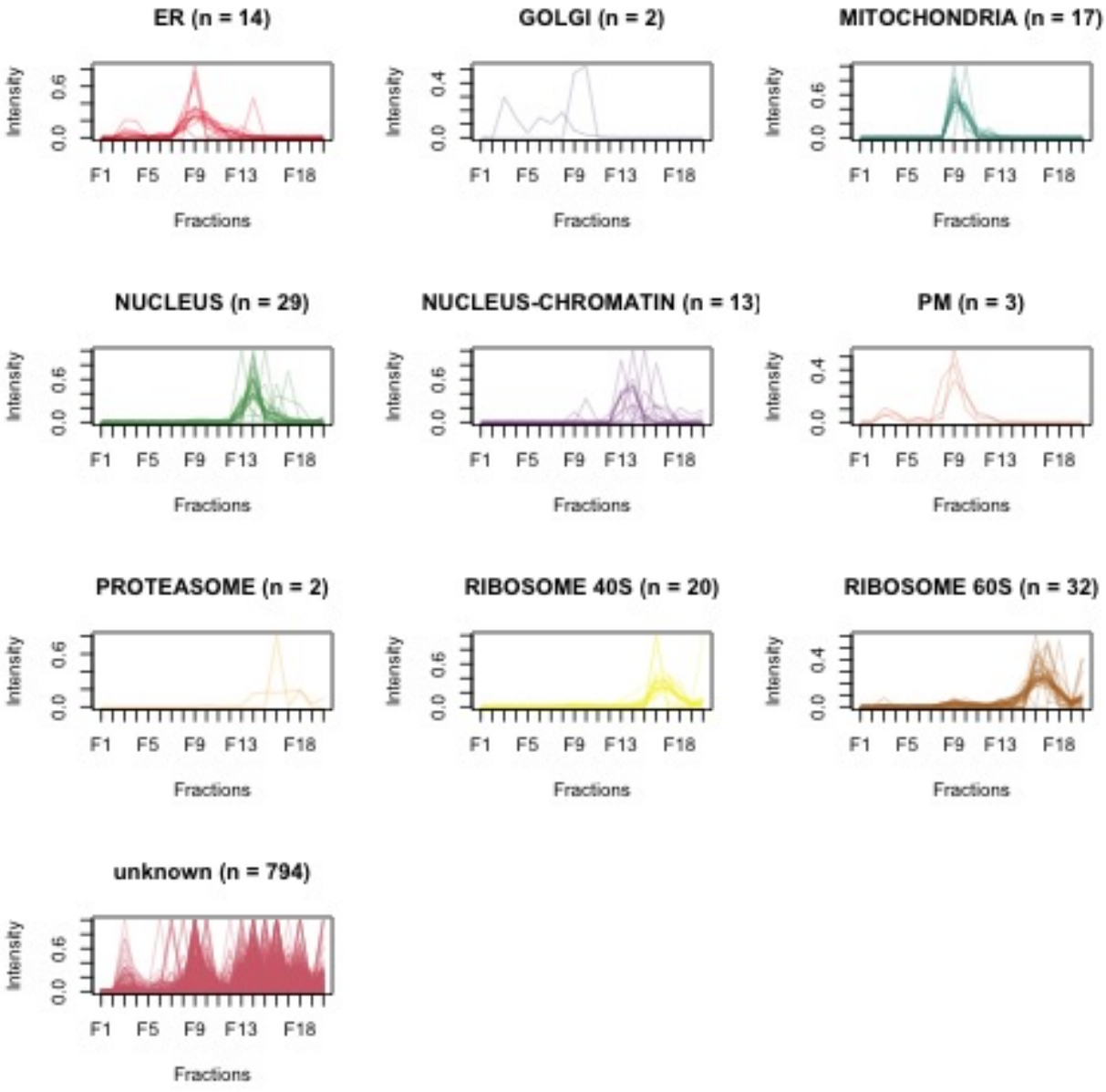

Figure 6: Profile plots of protein abundance from label-free OOPS on 20 fractions

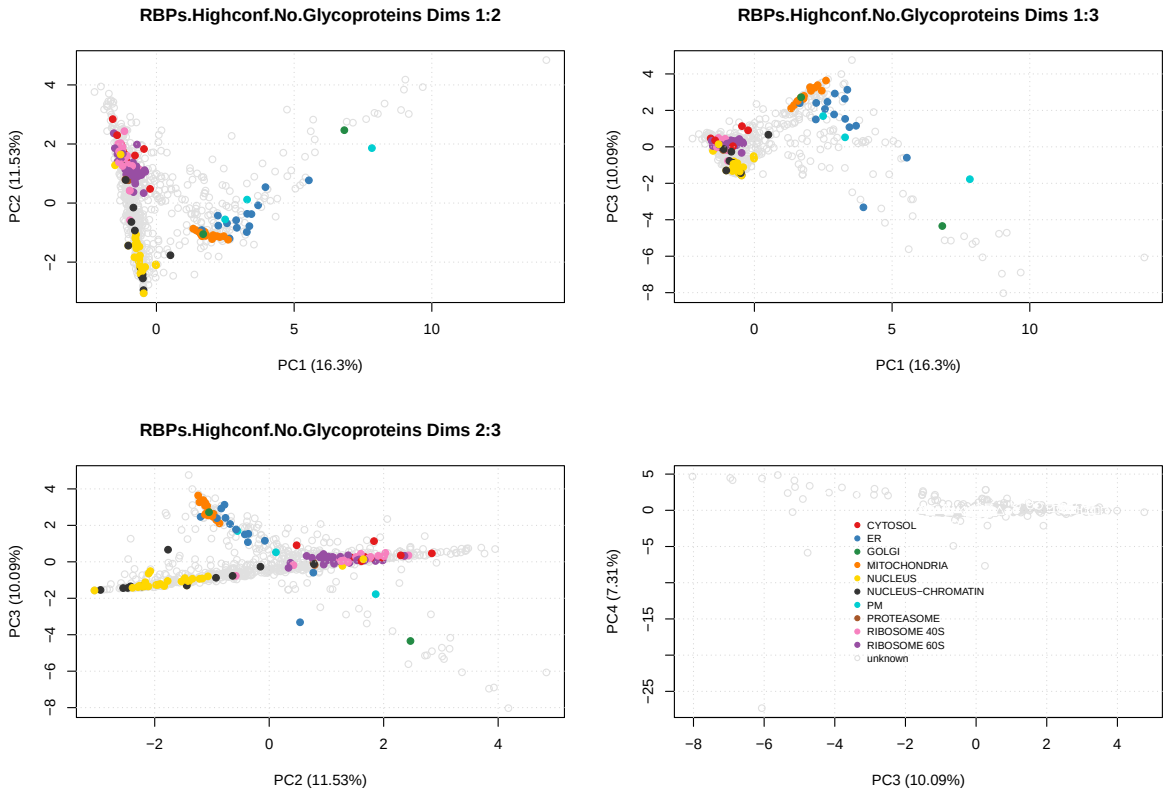

Figure 7: PCA plots of protein abundance from label-free OOPS on 20 fractions

```
ds = addMarkers(rbps.res, paste(indir, "FullHumanMarker.csv",
    sep = "/"), mcol = "markers")
ds = normalise(ds, "sum")

w <- table(getMarkers(ds))
w <- 1/w[names(w) != "unknown"]

## 100 rounds of optimisation with five-fold
## cross-validation params <-
## svmOptimisation(sets[[i]], fcol = 'markers',times
## = 100, xval = 5,class.weights =
## w,fun=mean,na.rm=T) best = getParams(params) #
## sigma = 1; cost = 16

# Classify unknown markers
ds3 <- svmClassification(ds, cost = 16, sigma = 1,
    class.weights = w, fcol = "markers")
(ts <- orgQuants(ds3, fcol = "svm", scol = "svm.scores",
    mcol = "markers", t = 0.5))

## set point size of each protein to be inversely
## proportional to the
ptsze <- exp(fData(ds3)$svm.scores) - 1

## plot new predictions
par(mfrow = c(1, 2))
plot2D(ds, fcol = "markers", cex = 0.7, dims = c(1,
    3))
plot2D(ds3, fcol = "svm", cex = ptsze, dims = c(1,
    3))
addLegend(ds3, fcol = "svm", where = "bottomright",
    bty = "n", cex = 0.7, ncol = 2)

# get predicted values
ds3 <- getPredictions(ds3, fcol = "svm", scol = "svm.scores",
    mcol = "markers", t = ts)
plot2D(ds3, fcol = "svm.pred", cex = ptsze, dims = c(1,
    3))
addLegend(ds3, fcol = "svm", where = "bottomright",
    bty = "n", cex = 0.7, ncol = 2)

## First remove the markers
preds <- unknownMSnSet(ds3)

## Plot a boxplot of the scores of each organelle
par(oma = c(10.5, 0, 0, 0))  ## sets outer margins
boxplot(svm.scores ~ svm, data = fData(preds), ylab = "SVM scores",
    las = 2)

plot2D(preds, fcol = "svm.pred", cex = ptsze, dims = c(1,
    3))
```

08. Functional enrichment analysis

Once we have identified proteins in each fraction with or without imputation, we'd like to see if each fraction is enriched for any functional groups.

8a. Setting up the background for enrichment

For this, we use a list of ~ 7500 bonafide RNA-binding proteins defined in the U2OS cell line (the cell line in our study) from the Geiger et al paper. We then query the Interpro database to annotate this list with GO terms, interpro terms and KEGG pathways. Following this, we use the 'goseq' package and function to test enrichment using a Wallenius' hypergeometric test which where items in the hypergeometric distribution are sampled with bias. The bias term used here is the maximum abundance of each protein obtain by averaging the logged abundance values from 3 replicates in the Geiger et al paper.

```
# Source functions
source("mcf10aFunctions.R")

# Background list
u2os.bg = read.delim(paste(indir, "U2OS_Background-list-of-proteins.txt",
    sep = "/"), header = T, sep = "\t", stringsAsFactors = F)
rownames(u2os.bg) = u2os.bg$master_protein

# Source functionsu
source("mcf10aFunctions.R")

# Building background of U2OS proteins from Geiger
# et al.
univ = u2os.bg$master_protein
univ.ann = myProtMapper(univ, out.fields = c("interpro.short_desc",
    "ensembl.gene", "go.MF.id", "go.CC.id", "go.BP.id",
    "pathway.kegg"))
univ.ann$kegg.id = sapply(univ.ann$pathway.kegg, function(x) paste0(unique(unlist(x[[1]])),
    collapse = ";"))
univ.ann$kegg.name = sapply(univ.ann$pathway.kegg,
    function(x) paste0(unique(unlist(x[[2]])), collapse = ";"))

# Make mapping for goseq analysis
univ.cat.go = makeGene2Cat(univ.ann, "query", "go.all",
    ";")
univ.cat.doms = makeGene2Cat(univ.ann, "query", "domains",
    ";")
univ.cat.kegg = makeGene2Cat(univ.ann, "query", "kegg.name",
    ";")

saveRDS(univ.cat.go, paste(indir, "UnivGO.rds", sep = "/"))
saveRDS(univ.cat.doms, paste(indir, "UnivDOMS.rds",
    sep = "/"))
saveRDS(univ.cat.kegg, paste(indir, "UnivKEGG.rds",
    sep = "/"))
```

8b. Function for enrichment analysis of fraction data

This is the function we will be using repeatedly to analyse functional themes in our label-free data, if these exist. Ideally, we use all our data but can also perform analysis on glyco-protein free or high-confidence RBP data.

```
#-----------------------------------------------------------------------------------------------------
# Input : channels = Samples/Fractions over which
# functional enrichment is required data = The
# normalised protein abundance data to inform on
# which proteins to use for enrichment suf = Suffix
# used for naming output files
#-----------------------------------------------------------------------------------------------------

runEnrich <- function(channels, data, suf) {

    source("mcf10aFunctions.R")

    # Read in Go/Interpro/KEGG information
    univ.cat.go = readRDS(paste(indir, "UnivGO.rds",
        sep = "/"))
    univ.cat.doms = readRDS(paste(indir, "UnivDOMS.rds",
        sep = "/"))
    univ.cat.kegg = readRDS(paste(indir, "UnivKEGG.rds",
        sep = "/"))

    u2os.bg = read.delim(paste(indir, "U2OS_Background-list-of-proteins.txt",
        sep = "/"), header = T, sep = "\t", stringsAsFactors = F)
    rownames(u2os.bg) = u2os.bg$master_protein

    univ = u2os.bg$master_protein

    all.go = NULL
    all.pro = NULL
    all.kegg = NULL

    for (l in 1:length(channels)) {
        prots = names(which(exprs(ds)[, l] != 0))
        go = rungoseq(prots, univ.cat.go, univ, b = u2os.bg$max,
            0.05)
        pro = rungoseq(prots, univ.cat.doms, univ,
            b = u2os.bg$max, 0.05)
        kegg = rungoseq(prots, univ.cat.kegg, univ,
            b = u2os.bg$max, 0.05)

        # Save results
        if (nrow(go[[2]]) > 0) {
            all.go = rbind(all.go, cbind(Cluster = channels[l],
                go[[2]]))
        }
        if (nrow(pro[[2]]) > 0) {
            all.pro = rbind(all.pro, cbind(Cluster = channels[l],
                pro[[2]]))
        }
```

```
    if (nrow(kegg[[2]]) > 0) {
        all.kegg = rbind(all.kegg, cbind(Cluster = channels[l],
            kegg[[2]]))
    }
}

# Prepare data for plotting
all.go$Description = paste("(", all.go$ontology,
    ") ", all.go$term, sep = "")
all.pro$Description = all.pro$category
all.kegg$Description = all.kegg$category

# More preparation
colnames(all.kegg)[3] = colnames(all.pro)[3] = colnames(all.go)[3] = "pvalue"
all.kegg$Description = gsub(" \\- Homo sapiens \\(human\\)",
    "", all.kegg$Description)
all.go$Cluster = factor(all.go$Cluster, levels = as.character(channels))

# Plots
ego = enricherPlot(all.go, paste(suf, "All-GO-U2OS-Label-free",
    sep = "_"), N = 3, colorBy = "neg.log10.BH",
    sizeBy = "foldEnrich", low.col = "#E69F00",
    high.col = "#999999", trunc.len = 40, all.size = 10,
    y.size = 8, x.size = 8)
epro = enricherPlot(all.pro, paste(suf, "All-Interpro-U2OS-Label-free",
    sep = "_"), N = 3, colorBy = "neg.log10.BH",
    sizeBy = "foldEnrich", low.col = "#E69F00",
    high.col = "#999999", trunc.len = 40, all.size = 10,
    y.size = 8, x.size = 8)
ego.cc = enricherPlot(all.go[which(all.go$ontology ==
    "CC"), ], paste(suf, "GO-Cellular-component-U2OS-Label-free",
    sep = "_"), N = 4, colorBy = "neg.log10.BH",
    sizeBy = "foldEnrich", low.col = "#E69F00",
    high.col = "#999999", trunc.len = 40, all.size = 10,
    y.size = 8, x.size = 8)
ego.mf = enricherPlot(all.go[which(all.go$ontology ==
    "MF"), ], paste(suf, "GO-Molecular-Function-U2OS-Label-free",
    sep = "_"), N = 4, colorBy = "neg.log10.BH",
    sizeBy = "foldEnrich", low.col = "#E69F00",
    high.col = "#999999", trunc.len = 40, all.size = 10,
    y.size = 8, x.size = 8)
ego.bp = enricherPlot(all.go[which(all.go$ontology ==
    "BP"), ], paste(suf, "GO-Biological-Process-U2OS-Label-free",
    sep = "_"), N = 4, colorBy = "neg.log10.BH",
    sizeBy = "foldEnrich", low.col = "#E69F00",
    high.col = "#999999", trunc.len = 40, all.size = 10,
    y.size = 8, x.size = 8)
ego.bp = enricherPlot(all.go[which(all.go$ontology ==
    "BP"), ], paste(suf, "GO-Biological-Process-U2OS-Label-free",
    sep = "_"), N = 3, colorBy = "neg.log10.BH",
    sizeBy = "foldEnrich", low.col = "#E69F00",
    high.col = "#999999", trunc.len = 40, all.size = 10,
    y.size = 8, x.size = 8)
```

```
    ekegg = enricherPlot(all.kegg, paste(suf, "All-KEGG-U2OS-Label-free",
        sep = "_"), N = 15, colorBy = "neg.log10.BH",
        sizeBy = "foldEnrich", low.col = "#E69F00",
        high.col = "#999999", trunc.len = 40, all.size = 10,
        y.size = 8, x.size = 8)

    pdf(paste(plotdir, paste(format(Sys.time(), "%Y-%m-%d_%H.%M.%S"),
        paste(suf, "GO-Interpro-enrichment-plots-for-U2OS-label-free-data.pdf",
            sep = "_"), sep = "_"), sep = "/"), paper = "a4r",
        width = 12, height = 8)
    print(ego)
    print(ego.cc)
    print(ego.mf)
    print(ego.bp)
    print(epro)
    print(ekegg)
    dev.off()

    return(list(all.go, all.pro, all.kegg))
}
```

8c. Testing functional enrichment

We will have to test a few different versions of RBPs across fractions 1. RBPs across the entire dataset 2. RBPs with glycoproteins removed 3. High-confidence RBPs with glycoproteins removed

```
# Running functional enrichment
library(goseq)

# Running enrichment for non-imputed,
# glycoprotein-containing samples channels =
# pData(rbps.res)$samples res.rbps.all =
# runEnrich(channels,rbps.res,'All-RBPs')

# Running enrichment for non-imputed,
# glycoprotein-free samples channels =
# pData(rbps.noglyc)$samples res.noglyc =
# runEnrich(channels,rbps.noglyc,'RBPs-Without-Glycoproteins')

# Running enrichment for high-confidence,
# glycoprotein-depleted samples
channels = pData(rbps.highconf)$samples
# res.rbps.highconf =
# runEnrich(channels,rbps.highconf,'RBPs-Highconf-Without-Glycoproteins')

# Checking how many proteins are in each fraction
# This is for verifying against the GO/Interpro
# runs.
for (l in 1:length(channels)) {
    prots = names(which(exprs(rbps.highconf)[, l] !=
        0))
    print(paste(l, length(prots), sep = " : "))
}
```

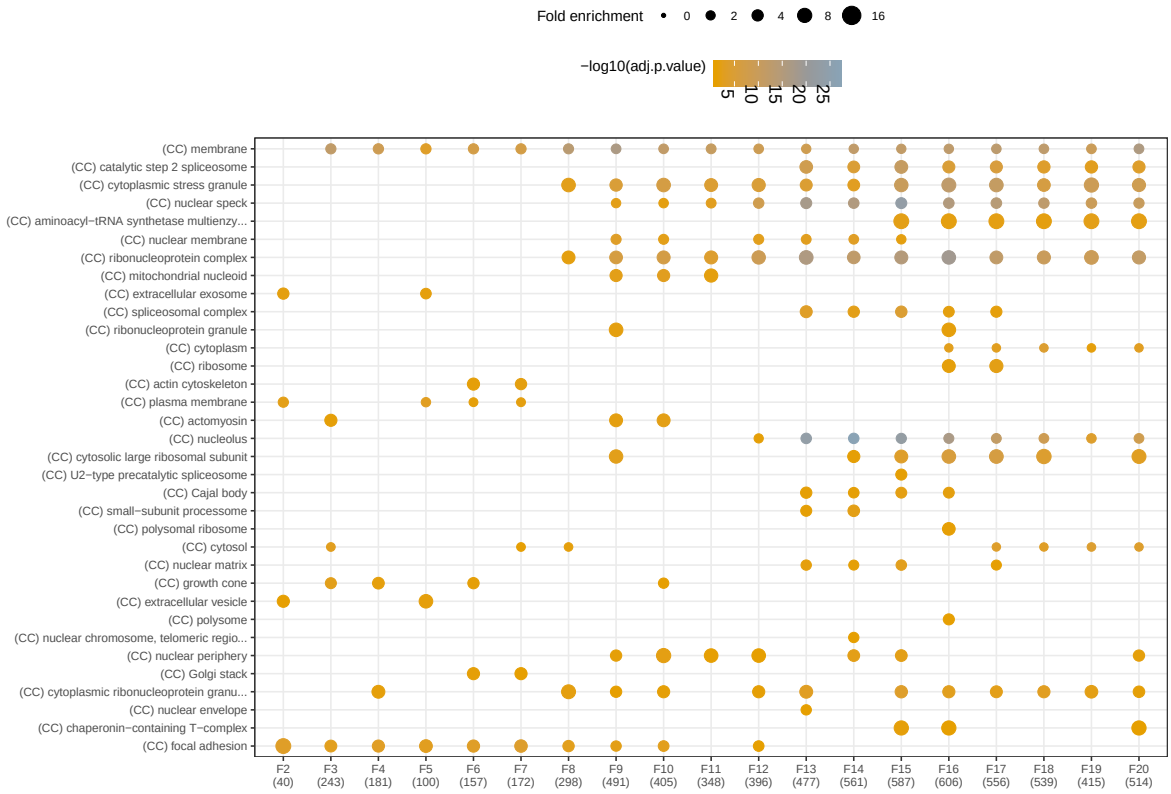

Figure 8: GO Cellular fraction enrichment data for label-free OOPS on 20 fractions

```
## [1] "1 : 7"
## [1] "2 : 40"
## [1] "3 : 247"
## [1] "4 : 182"
## [1] "5 : 101"
## [1] "6 : 160"
## [1] "7 : 175"
## [1] "8 : 302"
## [1] "9 : 498"
## [1] "10 : 411"
## [1] "11 : 352"
## [1] "12 : 400"
## [1] "13 : 479"
## [1] "14 : 565"
## [1] "15 : 590"
## [1] "16 : 612"
## [1] "17 : 561"
## [1] "18 : 545"
## [1] "19 : 419"
## [1] "20 : 519"
```

While the enrichment analysis works well, we aren't able to resolve each fraction into a particular cellular organelle based on GO/Interpro terms. This is relative to what we expect to see based on protein profiles

and PCA plots.

GO terms are a huge collection of annotations made using published scientific data. While it contains a lot of information, some GO terms can be biased as they are based purely on a handful (<5) studies. Furthermore, given the nature of our label-free experiments, the proteins in each fraction don't just comprise those at the peak of a given fraction but also those that are in the long tails of surrounding fractions.

Perhaps an idea could be to narrow in on those proteins that have maximum abundance at a given fraction and re-run enrichment analyses on the smaller list.

8d. Printing results to file

The functional enrichment plots show just the top 3/4 terms for each fraction for each category - GO, KEGG, Interpro. To see the full list and to access identifiers from the list, we will print them to tab-delimited text files. The reordering is partly to enable us to see the data better in R and partly to print to file and avoid any Excel quirks when opened.

```
all.go = res.rbps.highconf[[1]]
all.pro = res.rbps.highconf[[2]]
all.kegg = res.rbps.highconf[[3]]

# Re-order all.go and all.pro
# all.go[1:5,c(1:2,12,15,13,16,3,8:9,20,5:6,10:11,17)]
all.go.1 = all.go[, c(1:2, 12, 15, 13, 16, 3, 8:9,
    20, 5:6, 10:11, 17)]
all.go.1$geneNames = sapply(all.go$geneID, function(x) paste(queryMany(strsplit(x,
    "/")[[1]], scopes = "uniprot", fields = c("symbol"))$symbol,
    collapse = "/"))

# all.pro[1:5,c(1:2,10,13,11,14,3,5:6,8:9,15)]
all.pro.1 = all.pro[, c(1:2, 10, 13, 11, 14, 3, 5:6,
    8:9, 15)]
all.pro.1$geneNames = sapply(all.pro$geneID, function(x) paste(queryMany(strsplit(x,
    "/")[[1]], scopes = "uniprot", fields = c("symbol"))$symbol,
    collapse = "/"))

# all.kegg[1:5,c(1:2,10,13,11,14,3,5:6,8:9,15)]
all.kegg.1 = all.kegg[, c(1:2, 10, 13, 11, 14, 3, 5:6,
    8:9, 15)]
all.kegg.1$geneNames = sapply(all.kegg$geneID, function(x) paste(queryMany(strsplit(x,
    "/")[[1]], scopes = "uniprot", fields = c("symbol"))$symbol,
    collapse = "/"))
all.kegg.1$category = gsub(" \\- Homo sapiens \\(human\\)",
    "", all.kegg.1$category)

# Write to table
write.table(all.go.1, paste(outdir, "All-Enriched-GO-terms-U2OS-label-free.txt",
    sep = "/"), sep = "\t", row.names = F, quote = F)
write.table(all.pro.1, paste(outdir, "All-Enriched-Interpro-U2OS-label-free.txt",
    sep = "/"), sep = "\t", row.names = F, quote = F)
write.table(all.kegg.1, paste(outdir, "All-Enriched-KEGG-pathways-U2OS-label-free.txt",
    sep = "/"), sep = "\t", row.names = F, quote = F)
```

09. Validating OOPS-label free data with labelled LORNA/LOPIT data

Having looked at this data with Tom, Eneko and Rayner, we want to confirm that what we see isn't artefactual. We stick to just the rbps.highconf for this part of the exercise and will overlap a subset of proteins (cytoskeletal,paraspeckle) with LORNA data which represents the total proteome to see if there is a difference in their abundance profiles.

We expect the paraspeckle proteins in the RBP experiment to be picked up the nuclear fractions.

For the LORNA data, the columns names are the density of the gradient that Rayner/Enekp ran to separate the fractions. Note: Density 0.0948 is actually 0.9480. I've taken the column names, sorted with indexes and used the indices to re-roder the columns from F1-F20. Following this, I've renamed the columns F1-F20 to make them comparable to the label-free OOPS data. I've also normalised both datasets using "sum-normalisation" to make them comparable. No imputation was done on label-free OOPS due to vast swathes of missing data.

```
highconf.msnset = addMarkers(rbps.highconf, paste(indir,
    "FullHumanMarker.csv", sep = "/"), mcol = "markers")
highconf.msnset = normalise(highconf.msnset, "sum")

# Read in list of proteins with GO annotations
go.prots = read.delim(paste(outdir, "All-Enriched-GO-terms-U2OS-label-free.txt",
    sep = "/"), sep = "\t", header = T, stringsAsFactors = F)

# Cytoskeletal genes/proteins
cyto.prot = go.prots[grep("cytoskeleton", go.prots$Description),
    "geneID"]  # F6 and F7
cyto.prot = unique(c(strsplit(cyto.prot[1], "/")[[1]],
    strsplit(cyto.prot[2], "/")[[1]]))

# Paraspeckle proteins
paraspeck = read.delim(paste(indir, "paraspeckle-prots.txt",
    sep = "/"), header = F, stringsAsFactors = F)
colnames(paraspeck) = c("Uniprot.ID", "Gene.Description",
    "Misc", "GO.term", "Taxonomy.ID", "Evidence", "Ensembl.ID")
ps.prot = unique(paraspeck$Uniprot.ID)

# Nucleoplasm proteins (from Human Protein Atlas)
if (!requireNamespace("BiocManager", quietly = TRUE)) install.packages("BiocManager")
BiocManager::install("hpar", version = "3.8")
library(hpar)

data(hpaSubcellularLoc)

sub = hpaSubcellularLoc %>% filter(grepl("Nucleoplasm",
    GO.id), Reliability != "Supported", Reliability !=
    "Uncertain") %>% dplyr::filter(grepl("Nucleoplasm",
    Approved) | grepl("Nucleoplasm", Enhanced)) %>%
    dplyr::select(Gene:Reliability, Enhanced, Approved,
        GO.id) %>% dplyr::mutate(GO.count = sapply(GO.id,
    function(x) length(strsplit(as.character(x), split = ";",
        fixed = T)[[1]]))) %>% dplyr::filter(GO.count ==
    1)

nucleop.prot = unique(unlist(queryMany(as.character(sub$Gene),
```

```
    scopes = "ensembl.gene", fields = "uniprot")$uniprot.Swiss.Prot))
length(nucleop.prot)

# LORNA cytoskeletal proteins The headers are the
# density indices. Sort by this and re-order
# columns
total.prot = readRDS(paste(indir, "prot_res_20_fractions_imputed_markers.rds",
    sep = "/"))
total.prot = normalise(total.prot, "sum")
hnames = sapply(strsplit(colnames(exprs(total.prot)),
    "CV."), "[[", 1)
hnames[which(hnames == "0.0948")] = 0.948
hnames
ind = sort(hnames, index.return = T)$ix
ind

# Melt the total.prot dataset
total.exp = data.frame(exprs(total.prot))
head(total.exp)
head(total.exp[, ind])
total.exp = total.exp[, ind]
head(total.exp)
colnames(total.exp) = paste("F", seq(1:20), sep = "")
total.exp$Uniprot = rownames(total.exp)
total.exp.gat <- melt(total.exp, id.vars = 21, measure.vars = 1:20,
    variable.name = "Fraction", value.name = "Abundance",
    na.rm = T)
head(total.exp.gat)

# Melt the rbp dataset
rbp.exp = data.frame(exprs(highconf.msnset))
rbp.exp$Uniprot = rownames(rbp.exp)
rbp.exp.gat = melt(rbp.exp, id.vars = 21, measure.vars = 1:20,
    variable.name = "Fraction", value.name = "Abundance",
    na.rm = T)
rbp.exp.gat$Abundance = log10(rbp.exp.gat$Abundance +
    1)
head(rbp.exp.gat)

# Filter paraspeckle proteins from both datasets
# and bind
paraspeck.all = rbind(cbind(rbp.exp.gat[which(rbp.exp.gat$Uniprot %in%
    ps.prot), ], class = "RBP"), cbind(total.exp.gat[which(total.exp.gat$Uniprot %in%
    ps.prot), ], class = "Total.protein"))

# Paraspeckle ggplot
ps.gg = ggplot(data = paraspeck.all, aes(x = Fraction,
    y = Abundance, colour = class, lty = Uniprot, group = Uniprot)) +
    geom_line() + facet_wrap(~class, scales = "free_y",
    nrow = 2) + labs(colour = "Paraspeckle.prots")

pdf(paste(plotdir, "Paraspeckle.pdf", sep = "/"))
print(ps.gg)
```

```
dev.off()

# Filter cytoskeletal proteins
cyto.all = rbind(cbind(rbp.exp.gat[which(rbp.exp.gat$Uniprot %in%
    cyto.prot), ], class = "RBP"), cbind(total.exp.gat[which(total.exp.gat$Uniprot %in%
    cyto.prot), ], class = "Total.protein"))

# Cytoskeleton ggplot
cyto.gg = ggplot(data = cyto.all, aes(x = Fraction,
    y = Abundance, colour = class, group = Uniprot)) +
    geom_line() + facet_wrap(~class, scales = "free_y",
    nrow = 2) + labs(colour = "Actin.cytoskeleton.prots")

pdf(paste(plotdir, "Actin-cytoskeleton.pdf", sep = "/"))
print(cyto.gg)
dev.off()

# Filter nucleoplasm proteins from both datasets
# and bind
nucleop.rbp = unique(rbp.exp.gat[which(rbp.exp.gat$Uniprot %in%
    nucleop.prot), "Uniprot"])
nucleop.all = rbind(cbind(rbp.exp.gat[which(rbp.exp.gat$Uniprot %in%
    nucleop.prot), ], class = "RBP"), cbind(total.exp.gat[which(total.exp.gat$Uniprot %in%
    nucleop.rbp), ], class = "Total.protein"))

# Nucleoplasm proteins ggplot
np.gg = ggplot(data = nucleop.all, aes(x = Fraction,
    y = Abundance, colour = class, group = Uniprot)) +
    geom_line() + facet_wrap(~class, scales = "free_y",
    nrow = 2) + labs(colour = "Nucleoplasm.prots")

pdf(paste(plotdir, "Nucleoplasm.pdf", sep = "/"))
print(np.gg)
dev.off()

# Extracting Golgi, PM and Proteasome proteins for
# Rayner
rayner.prot = fData(highconf.msnset)[grep("GOLGI|PM",
    fData(highconf.msnset)$markers), ]
write.table(rayner.prot, paste(outdir, "Golgi-PM_ForRayner.txt",
    sep = "/"), sep = "\t", row.names = F, quote = F)
```

Paraspeckle proteins

Of the paraspeckle proteins, O43809 was not found in the RBP experiment.It is a known RBP but the label-free experiment didn't pick it up, however, the total proteome did. Q8WXF1 (paraspeckle component 1 PSPC1) which peaks in the cytoplasm in total protein LOPIT also peaks in the cytoplasm in the RBP experiment. The remaining 4 proteins P23246 (SFPQ),P52272(HNRNPM), Q15233(NONO) and Q16630(CPSF6) peak in the nuclear fractions F13-F15 in the RBP experiment but in the cytosolic fractions in the Total protein experiment.

Cytoskeletal proteins

There were 19 proteins with the GO term "actin cytoskeleton (GO:0015629)" and all were found in the RBP and total protein datasets. In the RBPs these proteins have a very distinct peak at F9 which is the end of the ER and start of the mitochondrial fractions based on the Western blot. In the total proteome, these

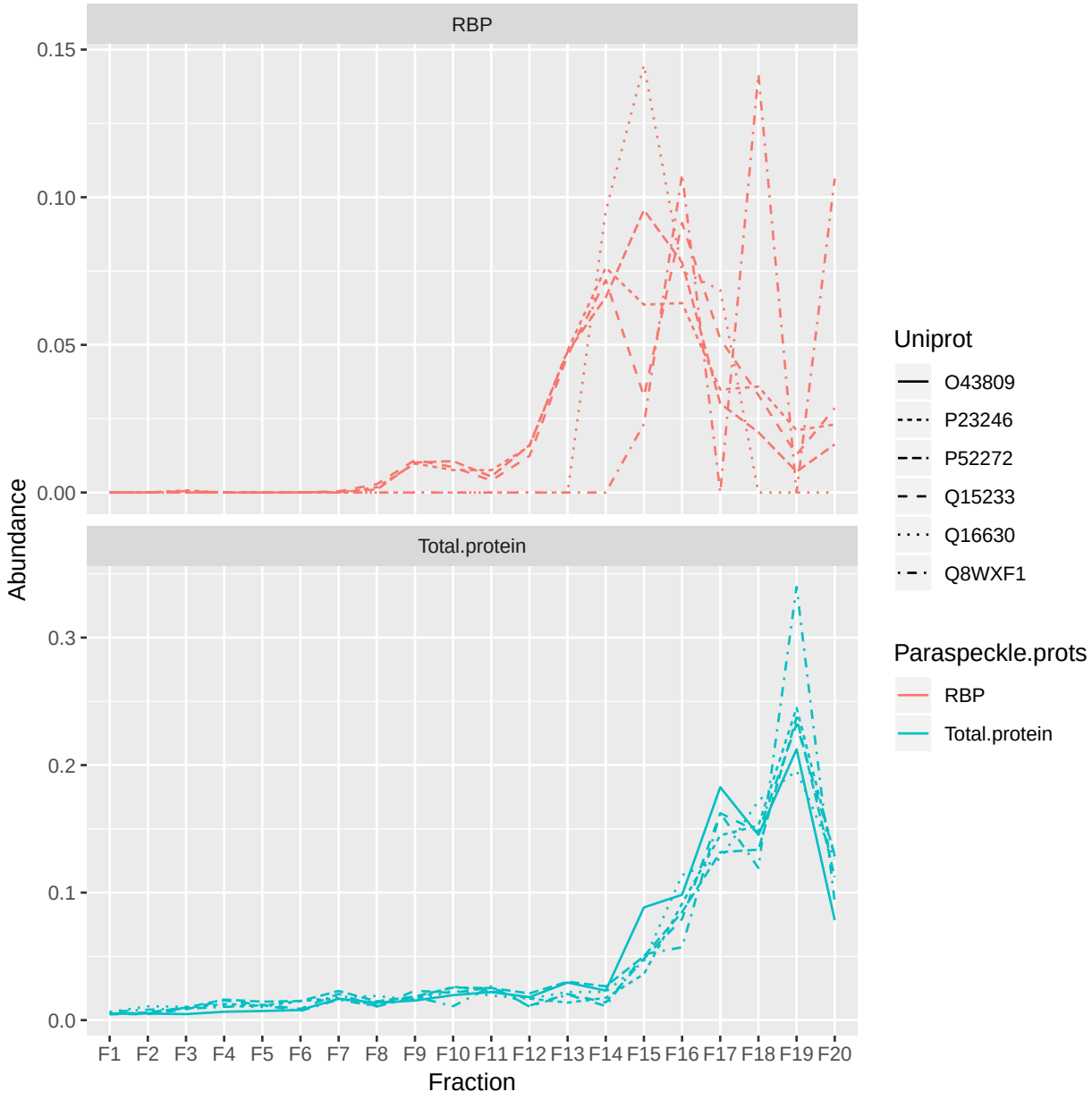

Figure 9: Paraspeckle proteins in RBP and Total protein capture

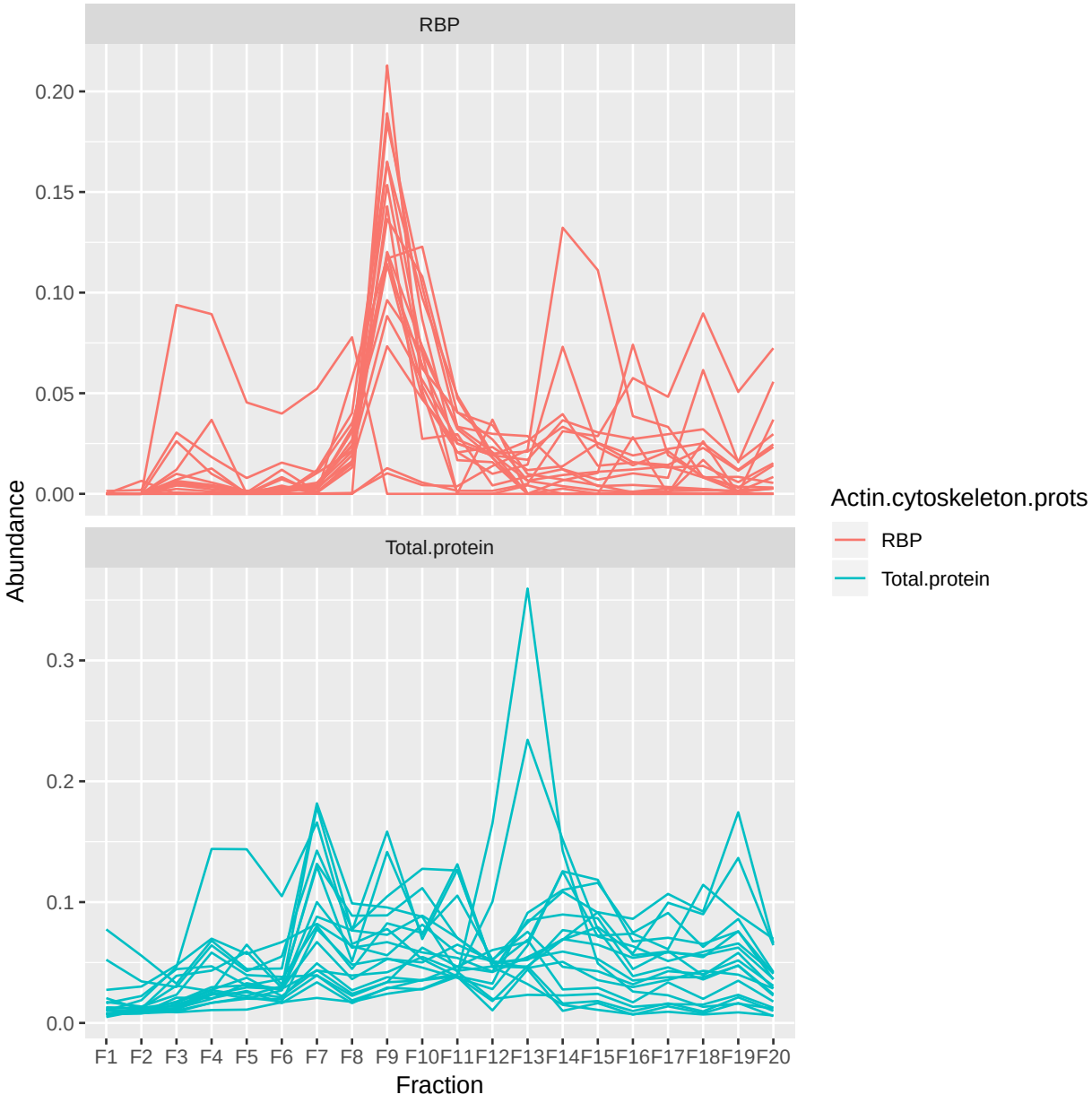

Figure 10: Actin cytoskeleton proteins in RBP and Total protein capture

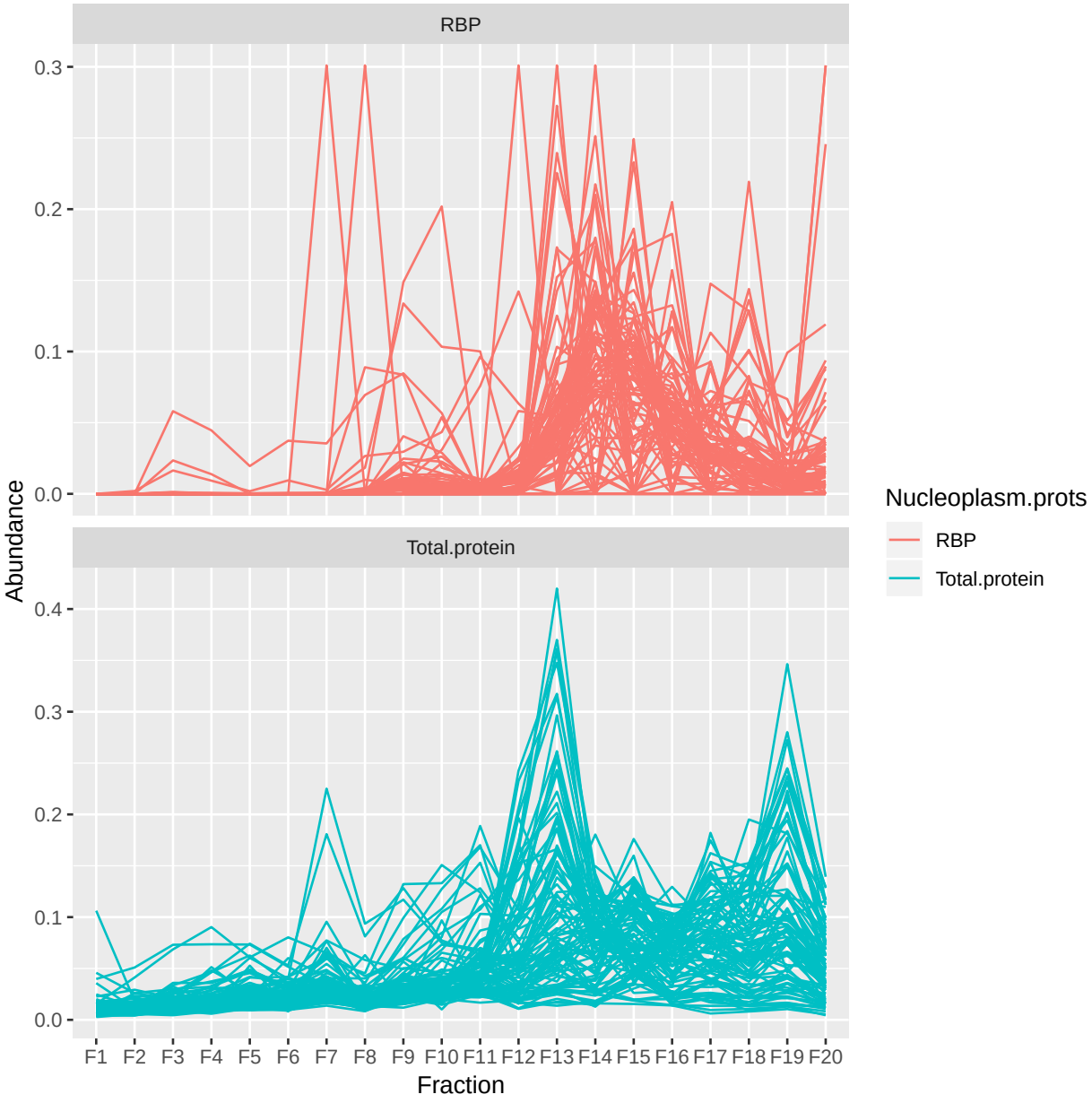

Figure 11: Nucleoplasm proteins in RBP and Total protein capture

proteins have no distinct peaks other than a couple at F13.

Nucleoplasm proteins
There were 906 proteins uniquely defined as nucleoplasm loacted in the Human Protein Atlas. Of these,there were 95 proteins present in the label-free OOPS RBP dataset. A lot more (350) were found in the Total protein dataset but we only kept the 95 to make the comparison more readable. In the RBPs these proteins have a broad peak encompassing F13-F16 which is where we expect the nuclear fraction to be. In the total protein fraction, the peak is distinct at F13 and then at F20 - very different to the RBP peak.

10. Session Information

Finally, we print the information necessary to run this session so users can see the setup of R and R-Studio used for this analysis. It should provide the reader with information about the environment that enabled this analysis.

```
sessionInfo()
```

```
## R version 3.5.1 (2018-07-02)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Sierra 10.12.6
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8
##
## attached base packages:
##  [1] stats4    grid      parallel  stats     graphics  grDevices utils
##  [8] datasets  methods   base
##
## other attached packages:
##  [1] goseq_1.32.0           geneLenDataBase_1.16.0 BiasedUrn_1.07
##  [4] RColorBrewer_1.1-2     bindrcpp_0.2.2         patchwork_0.0.1
##  [7] data.table_1.11.4      mygene_1.16.2          GenomicFeatures_1.32.2
## [10] GenomicRanges_1.32.7   GenomeInfoDb_1.16.0    pRoloc_1.20.1
## [13] MLInterfaces_1.60.1    cluster_2.0.7-1        annotate_1.58.0
## [16] XML_3.98-1.16          AnnotationDbi_1.42.1   IRanges_2.14.12
## [19] S4Vectors_0.18.3       ggbiplot_0.55          scales_1.0.0
## [22] plyr_1.8.4             MSnbase_2.6.4          ProtGenerics_1.12.0
## [25] BiocParallel_1.14.2    mzR_2.14.0             Rcpp_0.12.18
## [28] Biobase_2.40.0         BiocGenerics_0.26.0    dplyr_0.7.6
## [31] ggsci_2.9              ggplot2_3.1.1          reshape2_1.4.3
##
## loaded via a namespace (and not attached):
##   [1] proto_1.0.0                tidyselect_0.2.4
##   [3] RSQLite_2.1.1              htmlwidgets_1.2
##   [5] trimcluster_0.1-2.1        lpSolve_5.6.13
##   [7] rda_1.0.2-2.1              munsell_0.5.0
##   [9] codetools_0.2-15           preprocessCore_1.42.0
##  [11] chron_2.3-53               withr_2.1.2
##  [13] colorspace_1.3-2           BiocInstaller_1.30.0
##  [15] knitr_1.20                 rstudioapi_0.7
```

```
##  [17] geometry_0.3-6              robustbase_0.93-2
##  [19] dimRed_0.1.0                mzID_1.18.0
##  [21] labeling_0.3                GenomeInfoDbData_1.1.0
##  [23] hwriter_1.3.2               bit64_0.9-7
##  [25] ggvis_0.4.3                 rprojroot_1.3-2
##  [27] ipred_0.9-7                 randomForest_4.6-14
##  [29] diptest_0.75-7              R6_2.2.2
##  [31] doParallel_1.0.14           flexmix_2.3-14
##  [33] DRR_0.0.3                   bitops_1.0-6
##  [35] DelayedArray_0.6.6          assertthat_0.2.1
##  [37] promises_1.0.1              nnet_7.3-12
##  [39] gtable_0.3.0                affy_1.58.0
##  [41] ddalpha_1.3.4               timeDate_3043.102
##  [43] rlang_0.3.4                 CVST_0.2-2
##  [45] genefilter_1.62.0           RcppRoll_0.3.0
##  [47] splines_3.5.1               rtracklayer_1.40.6
##  [49] lazyeval_0.2.2              acepack_1.4.1
##  [51] ModelMetrics_1.2.0          impute_1.54.0
##  [53] hexbin_1.27.2               checkmate_1.8.5
##  [55] broom_0.5.0                 yaml_2.2.0
##  [57] abind_1.4-5                 threejs_0.3.1
##  [59] crosstalk_1.0.0             backports_1.1.2
##  [61] httpuv_1.4.5                Hmisc_4.1-1
##  [63] caret_6.0-80                tools_3.5.1
##  [65] lava_1.6.3                  affyio_1.50.0
##  [67] proxy_0.4-22                gsubfn_0.7
##  [69] base64enc_0.1-3             progress_1.2.0
##  [71] zlibbioc_1.26.0             purrr_0.2.5
##  [73] RCurl_1.95-4.11             prettyunits_1.0.2
##  [75] sqldf_0.4-11                rpart_4.1-13
##  [77] viridis_0.5.1               sampling_2.8
##  [79] sfsmisc_1.1-2               SummarizedExperiment_1.10.1
##  [81] magrittr_1.5                pcaMethods_1.72.0
##  [83] mvtnorm_1.0-8               whisker_0.3-2
##  [85] matrixStats_0.54.0          hms_0.4.2
##  [87] mime_0.5                    evaluate_0.11
##  [89] xtable_1.8-3                mclust_5.4.1
##  [91] gridExtra_2.3               compiler_3.5.1
##  [93] biomaRt_2.36.1              tibble_2.1.1
##  [95] crayon_1.3.4                htmltools_0.3.6
##  [97] mgcv_1.8-24                 later_0.7.5
##  [99] Formula_1.2-3               tidyr_0.8.1
## [101] lubridate_1.7.4             DBI_1.0.0
## [103] formatR_1.5                 magic_1.5-9
## [105] MASS_7.3-50                 fpc_2.1-11.1
## [107] Matrix_1.2-14               vsn_3.48.1
## [109] gdata_2.18.0                mlbench_2.1-1
## [111] bindr_0.1.1                 gower_0.1.2
## [113] igraph_1.2.2                pkgconfig_2.0.2
## [115] GenomicAlignments_1.16.0    foreign_0.8-71
## [117] recipes_0.1.3               MALDIquant_1.18
## [119] foreach_1.4.4               XVector_0.20.0
## [121] prodlim_2018.04.18          stringr_1.3.1
## [123] digest_0.6.18               pls_2.7-0
```

```
## [125] Biostrings_2.48.0          rmarkdown_1.10
## [127] htmlTable_1.12             dendextend_1.8.0
## [129] kernlab_0.9-27             shiny_1.1.0
## [131] Rsamtools_1.32.3           gtools_3.8.1
## [133] modeltools_0.2-22          jsonlite_1.5
## [135] nlme_3.1-137               viridisLite_0.3.0
## [137] limma_3.36.5               pillar_1.3.1
## [139] lattice_0.20-35            GO.db_3.6.0
## [141] httr_1.3.1                 DEoptimR_1.0-8
## [143] survival_2.42-6            glue_1.3.0
## [145] FNN_1.1.2.1                gbm_2.1.4
## [147] prabclus_2.2-6             iterators_1.0.10
## [149] bit_1.1-14                 class_7.3-14
## [151] stringi_1.2.4              blob_1.1.1
## [153] latticeExtra_0.6-28        memoise_1.1.0
## [155] e1071_1.7-0
```