Tutorial to Evaluation of linear models and missing value imputation for the analysis of peptide-centric proteomics

Philip Berg
May 22, 2018

Contents

Loading scripts 1
Loading data and its required structure 2
Running the pipeline 2
Creating ROC curves 5
  The pipeline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
  t-test . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
Plotting the curves . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
  Random forest . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

Loading scripts

The pipeline currently offers the possibility to receive an email when a run is finished. For this to function, the Mailer.R scripts needs to be properly configured. Opening the script will give instruction for how this is done. No error will occur from skipping this step. The easiest way to load the scripts is to use the R.utils package. First we will load another package, pacman which is used in the pipeline for managing packages. It can be loaded and/or installed with the following code:

```r
if(!("pacman" %in% .packages(all.available = T))){
  install.packages("pacman")
  library("pacman")
}else if(!("pacman" %in% .packages())){
  library("pacman")
}
```

we can then install and/or load R.utils with p_load:

```r
p_load(R.utils)
```

From R.utils we can read in the pipeline with the sourceDirectory function.

```r
sourceDirectory("Scripts/")
```

Do note that the user has to specify the correct directory to the repository of functions used in the pipeline. Here, we assume that the Scripts folder is in the working directory.
Loading data and its required structure

Here we start by using the UPS1 dataset as an example. We start by reading in the data using `read.csv` and store it in a variable.

```r
Data <- read.csv(file = "UPS1.csv", stringsAsFactors = F)
```

We can look at the needed structure of the data with `str`.

```r
str(Data)
```

```
'data.frame': 10962 obs. of 13 variables:
$ Identifier: chr "1::Cre01.g000350.t1.1|PACid:30788481|--AVLLFATGSGISPLR" ...
$ X25.1        : num 695.2 736 680.9 491.3 7.9 ...  
$ X25.2        : num 673.7 636.24 726.96 447.56 1.19 ...  
$ X25.3        : num 689 657 638 463 0 ...  
$ X25.4        : num 694.1 607.6 646.6 443.7 26.9 ...  
$ X50.1        : num 741 793 732 372 0 ...  
$ X50.2        : num 738.8 782.7 725.6 408.3 33.4 ...  
$ X50.3        : num 733 694 654 408 0 ...  
$ X50.4        : num 796 734 661 475 0 ...  
$ X100.1       : num 644 654 628 413 0 ...  
$ X100.2       : num 675 642 694 389 0 ...  
$ X100.3       : num 715 619 607 476 0 ...  
$ X100.4       : num 735 613 684 433 0 ...  

There has to be a column called Identifier which holds an unique ID for each peptide, and then each condition has to have the same name and end with a unique number for each replicate.

Running the pipeline

First, `variance-stabilizing` transformation is done by converting the raw data to log2 scale. This can be done with the `Log_Transformer` function in the pipeline.

```r
log_data <- Log_Transformer(Data)
```

After this, there is a need to define which comparisons to be made. There is three different conditions, X25, X50, and X100. The comparisons are defined by making a matrix with two rows, where each number in each column is compared. Each number corresponds to the j:th condition in the order they occur in the dataframe. If we want to do all possible comparisons we would define the following matrix:

```r
comp <- matrix(c(2,1, #Comparing X50 with X25  
                 3,2, #Comparing X100 with X50  
                 3,1), #Comparing X100 with X25  
               nrow = 2)

#The output is as follows:
comp
```

```
[,1] [,2] [,3]  
[1,] 2 3 3  
[2,] 1 2 1
```

We are then ready to run the pipeline. This can be done with the `multiple_imputation` function.
Sig_hits <- multiple_imputation(Data = log_data, #adding the transformed data
    as the dataset to run the analysis on
    Pre_cleaned = F, #We did not remove outliers before running
        #this function (F is the standard setting)
    n = 100, #Running 20 multiple imputations
    Comparisons = comp, #Adding which comparisons to run.
        #As defined above in this case.
    FDR_Criteria = 0.05, #Using an alpha level of 0.05
        #(0.05 is the standard setting)
    LFC_limits = c(-Inf, 1), #all true positives has an increase
        #in log fold change, hence we set
        #the lower cut-off to -Inf
    Binom_Prob = 1/2, #The probability hypothesis for the
        #binomial test
    Robust = F, #Running limma with normal (least squares)
        #regression
    verbose = T) #Print progress of the pipeline

## Generating 100 imputed datasets.
## Multiple data imputation: 79.02 sec elapsed
## Running LIMMA
## LIMMA: 13.67 sec elapsed
## Extracting FDR corrected p-value from LIMMA and calculating LFC.
## Limma FDR extraction and LFC calculation: 108.39 sec elapsed
## Significance: 122.06 sec elapsed
## Calculating 5:th and 95:th percentile, median, and mean for FDR and LFC.
## Descriptive statistics: 28.14 sec elapsed
## Running binomial test.
## Calculate binomial p-value: 7.06 sec elapsed
## Plotting results
## Saving 13 x 9 in image
## Saving 13 x 9 in image
## Saving 13 x 9 in image
## Plot results: 4.44 sec elapsed
## Saving results
## Save results: 0.03 sec elapsed
## Total: 241.37 sec elapsed
## Sending email
## NULL
## No internet or Mailer not setup; skipping email

#The output from multiple_imputation() contains a summary dataset with the 95:th and
#5:th percentile, median, and mean of the p-value from limma and the LFC for the different
#imputations.
#It also contains the number of missing values each feature contained in each condition
#The categorize can be seen with head of the dataset:
str(Sig_hits[['Data_Summary']])

```r
## 'data.frame': 10959 obs. of 40 variables:
## $ Identifier : chr "1::Cre01.g000350.t1.1|PACid:30788481|--AVLLFATGSISPLR" "1::Cre01.g000350.t1.1|PACid:30788481|--VVIDVGAPLAAGYTVPGQFVQVK" "1::Cre01.g000350.t1.1|PACid:30788481|--VVSVYSESK" ... 
## $ X25.1 : num 9.44 9.52 9.41 8.94 2.98 ... 
## $ X25.2 : num 9.396 9.313 9.506 8.806 0.255 ... 
## $ X25.3 : num 9.43 9.36 9.32 8.86 2.57 ... 
## $ X25.4 : num 9.44 9.25 9.34 8.79 4.75 ... 
## $ X50.1 : num 9.53 9.63 9.52 8.54 4.93 ... 
## $ X50.2 : num 9.53 9.61 9.5 8.67 5.06 ... 
## $ X50.3 : num 9.52 9.44 9.35 8.67 5.04 ... 
## $ X50.4 : num 9.64 9.52 9.37 8.89 5.18 ... 
## $ X100.1 : num 9.33 9.35 9.3 8.69 3.03 ... 
## $ X100.2 : num 9.4 9.33 9.44 8.6 2.96 ... 
## $ X100.3 : num 9.48 9.27 9.25 8.9 3 ... 
## $ X100.4 : num 9.52 9.26 9.42 8.76 3.01 ... 
## $ X25._Missing : num 0 0 0 0 1 0 0 0 0 0 ... 
## $ X50._Missing : num 0 0 0 0 3 0 0 0 0 0 ... 
## $ X100._Missing : num 0 0 0 0 4 0 0 0 0 0 ... 
## $ X50._Vs_X25._P_val_5Perc : num 0.336 0.247 0.827 0.533 0.115 ... 
## $ X50._Vs_X25._P_val_median : num 0.337 0.247 0.827 0.534 0.136 ... 
## $ X50._ Vs _X25. _P_val_median : num 0.338 0.248 0.827 0.534 0.169 ... 
## $ X50._Vs_X25._P_val_mean : num 0.337 0.248 0.827 0.534 0.14 ... 
## $ X50. _Vs _X25. _LFC_5Perc : num 0.1279 0.19 0.0425 -0.1541 2.2966 ... 
## $ X50. _Vs _X25. _LFC_median : num 0.1279 0.19 0.0425 -0.1541 2.413 ... 
## $ X50. _Vs _X25. _LFC_95Perc : num 0.1279 0.19 0.0425 -0.1541 2.5005 ... 
## $ X50. _Vs _X25. _LFC_mean : num 0.1279 0.19 0.0425 -0.1541 2.4036 ... 
## $ X50._Vs_X50._P_val_5Perc : num 0.4183 0.0809 0.8576 0.9861 0.1915 ... 
## $ X50._Vs_X50._P_val_median : num 0.4201 0.0814 0.8591 0.9867 0.242 ... 
## $ X50._Vs_X50._P_val_mean : num 0.4201 0.0814 0.859 0.9868 0.2409 ... 
## $ X50._Vs_X50._LFC_5Perc : num -0.1207 -0.2477 -0.0857 0.0421 -2.2077 ... 
## $ X50._Vs_X50._LFC_median : num -0.1207 -0.2477 -0.0857 0.0421 -2.0848 ... 
## $ X50._Vs_X50._LFC_95Perc : num -0.1207 -0.2477 -0.0857 0.0421 -1.9652 ... 
## $ X50._Vs_X50._LFC_mean : num -0.1207 -0.2477 -0.0857 0.0421 -2.0832 ... 
## $ X50._Vs_X25._P_val_5Perc : num 0.965 0.722 0.795 0.552 0.819 ... 
## $ X50._Vs_X25._P_val_median : num 0.966 0.723 0.796 0.553 0.871 ... 
## $ X50._Vs_X25._P_val_mean : num 0.966 0.723 0.796 0.553 0.898 ... 
## $ X50._Vs_X25._LFC_5Perc : num 0.00718 -0.05764 -0.04324 -0.11197 0.23077 ... 
## $ X50._Vs_X25._LFC_median : num 0.00718 -0.05764 -0.04324 -0.11197 0.31485 ... 
## $ X50._Vs_X25._LFC_95Perc : num 0.00718 -0.05764 -0.04324 -0.11197 0.42931 ... 
## $ X50._Vs_X25._LFC_mean : num 0.00718 -0.05764 -0.04324 -0.11197 0.32044 ... 
```

The pipeline saves one figure and one tsv file for each comparison. The tsv files are saved in the working directory and the figures are made in the sub-directory `figures/Scatter significance`. They look as follows:
Creating ROC curves

The pipeline

To generate ROC curves there is a need to tweak the parameters. Here we exemplify by tweaking both the p-value and the log\(_2\) fold change (LFC) at the same time. We set up the parameters as follows:

```r
# Sample 10 parameters between from 0.05 to 0.001
p_val <- seq(from = 0.05, to = 0.001, length.out = 10)
# Set lower limit to -Inf and then sample 10 parameters for upper limit between 0 and 2
LFC <- matrix(c(rep(-Inf, length.out = 10), seq(from = 0, to = 2, length.out = 10)), ncol = 2)
```

to get two evenly inter-spaced parameter sets. We can then loop over the different parameters and generate one result for each set.

```r
Pipe_both <- NULL
for (i in seq_along(p_val)) {
  Pipe_both[[i]] <- multiple_imputation(Data = log_data,
                                        n = 100,
                                        Comparisons = comp,
                                        FDR_Criteria = p_val[i],
                                        LFC_limits = LFC[i,],
                                        verbose = F) # Silencing redundant messages of the progress of each run to not flood the tutorial.
}
```

To calculate the true positive rate (TPR) and the false positive rate (FPR) of each parameter set, we can use the TPR_FDR function in the pipeline.

```r
ROC_pipe <- TPR_FDR(Data = log_data,# Note that we use the transformed data as it contains all ID's. In case a TP were to be filtered out that will penalize the pipeline.
                      Results = Pipe_both,# The results from tweaking the parameters
                      ID_TP = '_HUMAN')# Lastly a part of the ID for the TPs
# The output is then given as
lapply(ROC_pipe, head, 3)
## $FPR
## X50._Vs_X25._FPR X100._Vs_X50._FPR X100._Vs_X25._FPR
## 1 0.3788238 0.4545885 1.174354
## 2 0.3598826 0.3882943 1.079648
## 3 0.1420589 0.2651766 0.653471
```
t-test

For the t-test, we implemented a function called `My_ttest`, it runs on cleaned data. Data can be cleaned using the `Cleaner` function:

```r
clean_data <- Cleaner(log_data, F)
```

```r
## Replicate filter: 0.08 sec elapsed
## Missing annotation: 0.22 sec elapsed
## No id variables; using all as measure variables
## Imputation: 0.86 sec elapsed
## Data imputation: 0.93 sec elapsed
```

The cleaner function also returns the number of missing values there was for a feature in a category.

```r
head(clean_data, 3)
```

```r
## Identifier X25.1
## 1 1::Cre01.g000350.t1.1|PACid:30788481|--AVLLFATGSIPLR 9.441353
## 2 1::Cre01.g000350.t1.1|PACid:30788481|--GFALDRLPASTTR 9.523582
## 3 1::Cre01.g000350.t1.1|PACid:30788481|--VVIDVGAPLAVGPQFVVK 9.411349
```

. Then running the t-test with the same parameters as for the pipeline:

```r
ttest_both <- NULL
for (i in seq_along(p_val)) {
  ttest_both[[i]] <- My_ttest(Data = clean_data,
                               Pval = p_val[i],
                               LFC = LFC[i,2],
                               Comparisons = comp)
}
```

. We get TPR and FPR just like for the pipeline:

```r
ROC_ttest <- TPR_FPR(Data = log_data,
                      Results = ttest_both,
                      ID_TP = 'HUMAN')
lapply(ROC_ttest, head, 3)
```
## $TPR$

### X50.VsX25._TPR X100.VsX50._TPR X100.VsX25._TPR

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>47.39454</td>
<td>62.77916</td>
<td>82.13400</td>
</tr>
<tr>
<td>2</td>
<td>46.65012</td>
<td>62.28288</td>
<td>81.14144</td>
</tr>
<tr>
<td>3</td>
<td>45.65757</td>
<td>60.29777</td>
<td>79.40447</td>
</tr>
</tbody>
</table>

Plotting the curves

Each plot can then be made with the ROC_Plotter function.

```r
ROC_Plotter(ROC_pipe[[1]], ROC_ttest[[1]],
            ROC_pipe[[2]], ROC_ttest[[2]],
            # Add first TPR then FPR in the same order
            main = "Example figure")
legend("bottomright", inset = 0.01,
       legend = paste0(
         rep("Both", c = 6),
         rep("_Lim_", c = 3),
         rep("_ttest_", c = 3),
         times = 2),
       col = rainbow(6), lty = 1, cex = 1,
       box.lty = 0, x.intersp = .25, y.intersp = 1,
       lwd = 1, pt.cex = 2, xpd = T, seg.len = 1, bg = "transparent",
       pch = 1:6)
```

![Example figure](attachment:image.png)
Random forest

For random forest imputation, we used the `rforimp` function. Here this will be exemplified with the Yeast dataset. Which first has to be loaded and log₂ transformed.

```r
# Load data
Data <- read.csv(file = 'Yeast.csv', stringsAsFactors = F)

# And log transform
log_data <- Log_Transformer(Data)
```

Then we can do imputation with `rforimp`

```r
RF_imp <- rforimp(log_data)
```

# missForest iteration 1 in progress...done!
# missForest iteration 2 in progress...done!
# missForest iteration 3 in progress...done!
# missForest iteration 4 in progress...done!
# missForest iteration 5 in progress...done!
# missForest iteration 6 in progress...done!
# missForest iteration 7 in progress...done!
# missForest iteration 8 in progress...done!
# missForest iteration 9 in progress...done!
# missForest iteration 10 in progress...done!
# missForest iteration 11 in progress...done!

`rforimp` also returns the number of missing values in a condition and feature.

We then have to run the data through limma using `My_limma`. And to get ROC curves we have to run it using different parameters.

```r
# Load needed packages
p_load('magrittr', 'tibble', 'dplyr', 'purrr')

# Setting up comparisons
comp <- matrix(c(2,1), nrow = 2)

# Generate datasets
RF_lim <- lapply(rep(list(RF_imp), length(p_val)),
  MyLimma,
  Comparisons = comp,
  Robust = F)

# Extract p-value from limma and calculate LFC
for (i in seq_along(RF_lim)) {
  RF_lim[[i]][[1]] <- RF_lim[[i]][[1]][5] > p_val
  rownames_to_column(var = 'Identifier')
  mutate(LFC = apply(RF_imp %>% add.LFC
                      Data_Mean() %>% get the mean of each condition
                      select(contains('mean'),
                             1, function(x) x[2] - x[1])
                      Calculation the LFC
                      )
  }

# Take the significant hits in each dataset
Subs <- function(x, p_val, LFC){
}
for (i in seq_along(RF_lim)) {
  ```
Flag <- apply(RF_lim[[i]][[1]], 1, 
  function(x) Subs(x, p_val = p_val[i], LFC = LFC[,2]))
RF_lim[[i]][[1]] <- subset(RF_lim[[i]][[1]],Flag)

And to compare with our method we run it like before:
Pipe_both <- NULL
for (i in seq_along(p_val)) {
  Pipe_both[[i]] <- multiple_imputation(Data = log_data,
    n = 100, 
    Comparisons = comp, 
    FDR_Criteria = p_val[i], 
    LFC_limits = LFC[,], 
    verbose = F)
}

Finally, we get the TPR and FPR and make a plot:

ROC_RF <- TPR_FPR(Data = log_data, 
  Results = RF_lim, 
  ID_TP = "YEAST")

ROC_pipe <- TPR_FPR(Data = log_data, 
  Results = Pipe_both, 
  ID_TP = "YEAST")

ROC_Plotter(ROC_pipe[[1]], ROC_RF[[1]], 
  ROC_pipe[[2]], ROC_RF[[2]], 
  main = "Example figure")

legend("bottomright", inset = 0.01, 
  legend = paste0( 
    rep(c("Both"), times = 2), 
    rep(c("_Norm_"), "_RF_"), each = 1), 
    "50_vs_25"), 
  col = rainbow(2), lty = 1, cex = 1, 
  box.lty = 0, x.intersp = .25, y.intersp = 1, 
  lwd = 1, pt.cex = 2, xpd = T, seg.len = 1, bg = "transparent", 
  pch = 1:2)