SWATH2stats

Peter Blattmann, Moritz Heusel, and Ruedi Aebersold

Institute of Molecular Systems Biology, Department of Biology,
ETH Zurich, Switzerland

June 15, 2018

This vignette describes how the different functions from the SWATH2stats package can be applied. The functions from the SWATH2stats package are intended to be used on SWATH data that has been generated by the OpenSWATH pipeline. The SWATH2stats package provides functions to annotate such SWATH data with experimental meta-data, perform initial data analysis, perform a false-discovery rate (FDR) estimation, perform filtering, and to convert the SWATH data into a format readable by downstream statistical and quantification software tools such as MSstats, aLFQ, mapDIA or imsbInfer. The SWATH2stats package thus represents a link between the OpenSWATH pipeline and the downstream analysis packages MSstats, aLFQ, mapDIA, or imsbInfer. The SWATH2stats package was programmed and intended for use by researchers in proteomics working with SWATH data without extensive programming skills, but with basic R knowledge.

Contents

1 Introduction

1.1 SWATH-MS data analysis via open source tools

SWATH-MS as an implementation of data-independent acquisition (DIA) mass spectrometry is an emerging proteomic approach that allows systematic quantification of peptides in complex samples (Gillet et al. 2012, Venable et al. 2004). The acquired mass spectra can be queried for the presence and quantity of peptide analytes using the open-source OpenSWATH pipeline. The OpenSWATH pipeline consists of the OpenSWATH software (Roest et al. 2014) coupled to statistical validation using the mProphet algorithm (Reiter et al. 2011), or its re-implementation pyProphet (Teleman et al. 2015). OpenSWATH extracts ion chromatograms of both the peptide precursor and the fragment ions and quantifies peak groups. It then generates scores for how well a given candidate peak group corresponds to an analyte from a spectral or assay library (Roest et al. 2014). mProphet uses a machine learning algorithm to identify an optimal linear combination of these scores (d-score) to discriminate targets from decoys. In addition, it fits a function to the distribution of the d-scores for the decoy peptides, that is used as the null distribution. This null distribution is then used to calculate a q-value/m-score of each peakgroup (Storrey et al. 2003, Reiter et al. 2011). Hence, filtering the results with an m-score of 0.01 results in an FDR of 1% of the target assays within this run.

1.2 The usage of SWATH2stats in the open-source SWATH-MS data analysis workflow

This package creates a link between the OpenSWATH/mProphet .tsv output table and popular downstream tools for statistical and advanced data analysis. With very large assay libraries (Rosenberger et al. 2014) the SWATH results can become too large to analyse and process via tools such as Microsoft Excel. Therefore this package offers functionality to annotate the data with the study design (such as condition, biological and unique MS run id), perform initial data analysis, and offers substantial filtering capabilities. The data can be filtered based on frequency of observation among the samples, number of sibling peptides of a protein entry or directly on the FDR as estimated by the mProphet model (m-score, equivalent to q-value; for details see previous section or Reiter et al. 2011). Furthermore the package features estimation of global false discovery rates according to the target-decoy rationale (Elias and Gygi 2008, Kaell et al. 2008). The last step is the conversion to formats that can be read directly by the downstream analysis tools MSstats (Choi et al. 2014), mapDIA (Teo et al. unpublished), aLFQ (Rosenberger et al. 2014), and imsbInfer (Wolski et al. unpublished).

1.3 Implementation of a target-decoy strategy to estimate false target discovery rates (FDR)

Mass-spectrometry-based proteomic experiments produce large amounts of data that require statistical validation. In the SWATH2stats package a target-decoy strategy was implemented to estimate the FDR (Elias and Gygi, 2007). The target-decoy strategy relies on the assumption that the decoys have the same characteristics (distribution of their scores) as the false targets. The FDR among the targets is estimated as the ratio of decoy peptides passing a certain score threshold divided by the total number of targets passing the same score threshold (Choi and Nesvizhskii, 2008). The usage of the target-decoy strategy for SWATH data and to estimate peptide and protein-level FDR has not been extensively tested yet. The target-decoy strategy has been tested to estimate protein-level FDR in DDA data and has been shown to result in more conservative FDR estimates compared to a hypergeometric model-based approach (Reiter et al. 2009). A target-decoy approach was implemented in SWATH2stats because it allows i.) estimation of an FDR over multiple runs and ii.) allows to directly assess the selectivity of a given filter for likely true (target) over false (decoy) data points.
In contrast to the naive target-decoy approach counting the number of decoys, a correction factor can be supplied to many FDR estimation functions in the SWATH2stats package. For example, a correction needs to be applied to correct for the fraction of false targets (FFT). Similar correction factors have been used to adjust FDR estimation in DDA data (PIT: Kaell et al. 2008, p(-): Keller et al. 2002). In the functions, the FFT defaults to 1 to perform a naive target-decoy counting strategy without FFT correction, which will result in an overestimation of the FDR. For a more accurate estimation of the FDR, a FFT correction factor can be provided that corrects for the ratio of false targets to decoys. The number of decoys counted is multiplied with the FFT correction factor. The rationale is that for example if 50% of the samples are true targets,

the number of true negative targets that are modeled by the decoy distribution is around 50% lower than the decoy distribution. Therefore 2 decoy hits passing a certain m-score threshold suggest only one false positive datapoint passing the same threshold. The ratio of true negative (false) targets compared to all targets (FFT) can for example be obtained from the mProphet model statistics (Injection_name]_full_stat.csv (column 1 line 2 corresponding to the maximal q-value).
Alternatively, the FFT can conservatively be approximated by the fraction of assays in the library that do not pass an m-score threshold of e.g. 0.01 (corresponding to 1 % model FDR). For example, acquiring a full cell lysate and searching the data using the combined assay library (200k assays, Rosenberger et al. 2014), 50k assays are typically identified with m-score <= 0.01. Hence a FFT of 0.75 can be estimated. If a full lysate is searched by a sample-specific assay library (e.g. 70k assays) and 40k assays were identified with m-score <=0.01, a FFT of 0.57 can be estimated.

2 Loading and annotating the data

2.1 Installing SWATH2stats

To install the SWATH2stats package the following commands can be executed within R (after package has been accepted to Bioconductor).

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("SWATH2stats")
```

The SWATH2stats package can now be loaded.

```
> library(SWATH2stats)
```

2.2 Loading the data

The example data, that is included in the package, consists of a reduced OpenSWATH output file generated from Hela cells. To avoid making the file of the SWATH2stats package too large, only a fraction of a typical SWATH data table is included as example data. The example data contains data for 9 proteins, 5 decoy-proteins and a set of peptides for retention time calibration (labelled as iRT_protein). In total the data contains 284 peptides for which quantitative data has been extracted from 6 different samples measured on an ABSciex TripleTOF 5600 mass spectrometer and analyzed with the OpenSWATH + pyProphet workflow (Roest et al. 2014, Teleman et al. 2015). These 6 samples consist of biological triplicates of Hela cells grown under control condition and Hela cells that have been perturbed by inhibiting cholesterol synthesis.
The experimental design is described in a table called Study_design that is included in the package. This file that contains the study design information needs to be a table with the following columns: Filename, Condition, BioReplicate, Run (see below). For correct assignment of identifiers into the Run, BioReplicate and Condition column for MSstats, please consult their manual. The values in the column `Filename` have to be unique for every injection file and will be

matched to the OpenSWATH output in the column `align_origfilename` (caution: this matching is case sensitive).
The example SWATH data and the study design table can be loaded from the package with the function `data()`.

```
> data('OpenSWATH_data', package='SWATH2stats')
> data <- OpenSWATH_data
> data('Study_design', package='SWATH2stats')
> head(Study_design)
```

```
                Filename      Condition BioReplicate Run
1     peterb_J131223_043   Hela_Control            1   1
2     peterb_J131223_054 Hela_Treatment            1   2
3 peterb_L150425_003b_SW   Hela_Control            2   3
4  peterb_L150425_011_SW Hela_Treatment            2   4
5  peterb_L150514_001_SW   Hela_Control            3   5
6  peterb_L150514_002_SW Hela_Treatment            3   6
```

Alternatively, the data can be loaded from your working directory. In the code below you see an example how to load the required files from your working directory (code is not executed here but just as an example). As input data for SWATH2stats it is recommended to use the quantitative data matrix after OpenSWATH, pyProphet, and TRIC analysis (see www.openswath.org).

```
> # set working directory
> setwd('~/Documents/MyWorkingDirectory/')
> # Define input data file (e.g. OpenSWATH_output_file.txt)
> file.name <- 'OpenSWATH_output_file.txt'
> # File name for annotation file
> annotation.file <- 'Study_design_file.txt'
> # load data
> data <- data.frame(fread(file.name, sep='\t', header=TRUE))
```

If the file is in a different format the column names have to be renamed accordingly. For this the function `import_data` can perform this. For the requirements for each column please consult the manual page.

```
> # consult the manual page.
> help(import_data)
> # rename the columns
> data <- import_data(data)
```

The function `reduce_openSWATH_output` can be executed to reduce the number of columns from the OpenSWATH result table. This function reduces the number of columns to the ones necessary for MSstats, mapDIA, aLFQ. However for other packages such as imsbInfer all the columns need to be kept and this function should be omitted. The next command shows how specific proteins or peptides can be removed from the data. As an example the iRT peptides (peptides for retention time calibration) were removed.

```
> # reduce number of columns
> data <- reduce_OpenSWATH_output(data)
> # remove the iRT peptides (or other proteins)
> data <- data[grep('iRT', data$ProteinName, invert=TRUE),]
```

2.3 Annotating the data

With the first two commands the number of files in the OpenSWATH data and the names of these files can be printed. This can be helpful to generate the study design table (The script can be executed until here and then the annotation file generated with a text editor). See above for a description of the exact format and column names required for the study design table.

```
> # list number and different Files present
> nlevels(factor(data$filename))
> levels(factor(data$filename))
> # load the study design table from the indicated file
> Study_design <- read.delim2(annotation.file,
+                             dec='.', sep='\t', header=TRUE)
```

With the function `sample_annotation` the data is annotated with the meta-data contained in the study design table. The next commands can be used to shorten the protein names and remove repetitive and non-unique parts of the Protein name as shown by the example removing some parts of the identifier keeping only the unique SwissProt accession identifier (example see below).

```
> # annotate data
> data.annotated <- sample_annotation(data, Study_design)
> head(unique(data$ProteinName))

[1] 1/Protein6  1/Protein1  1/Protein7  1/Protein4  1/Protein8
[6] 10/Protein9
15 Levels: 1/Protein1 1/Protein2 1/Protein3 1/Protein4 ... DECOY_1/Protein6

> # OPTIONAL: for human, shorten Protein Name to remove non-unique information
> #(sp|Q9GZL7|WDR12_HUMAN --> Q9GZL7)
> data$ProteinName <- gsub('sp\\|([[:alnum:]]+)\\|[[:alnum:]]*_HUMAN',
+                          '\\1', data$ProteinName)
> head(unique(data$ProteinName))

[1] "1/Protein6"  "1/Protein1"  "1/Protein7"  "1/Protein4"
[5] "1/Protein8"  "10/Protein9"
```

3 Analyze data

In order to analyze the data we provide different functions to assess the variation or correlation between samples or to calculated the summed signal per peptide and protein. This can be used to quickly assess the overall similarity of the injections or see what the signal for a peptide or protein of interest is.

3.1 Count analytes

With the function `count_analytes` the number of transitions, peptides and proteins can be counted across the different injections. This can be helpful for assessing if a certain injection produced considerably less identifications and what the mean number of identified transitions, peptides or proteins per sample is.

```
> count_analytes(data.annotated)
```

	run_id	transition_group_id	FullPeptideName	ProteinName
1	Hela_Control_1_1	354	250	9
2	Hela_Control_2_3	370	260	9
3	Hela_Control_3_5	373	262	9
4	Hela_Treatment_1_2	354	250	9
5	Hela_Treatment_2_4	369	259	9
6	Hela_Treatment_3_6	373	262	9

3.2 Plot correlation between samples

With the function `plot_correlation_between_samples` the Pearson and Spearman correlation is calculated between the different injections and plotted in a heatmap. This can be used to spot injections that show very different signal or also retention times.

```
> # Plot correlation of intensity
> correlation <- plot_correlation_between_samples(data.annotated, column.values = "Intensity")
> head(correlation)
```

	Var1	Var2	value	method
1	Hela_Control_1	Hela_Control_1	1.0000000	pearson
7	Hela_Control_1	Hela_Control_2	0.9326233	pearson
8	Hela_Control_2	Hela_Control_2	1.0000000	pearson
13	Hela_Control_1	Hela_Control_3	0.9775615	pearson
14	Hela_Control_2	Hela_Control_3	0.9801443	pearson
15	Hela_Control_3	Hela_Control_3	1.0000000	pearson

```
> # Plot correlation of retention times
> correlation <- plot_correlation_between_samples(data.annotated, column.values = "RT")
>
```

3.3 Plot variation

With the function `plot_variation` the coefficient of variation of the signal for the different transitions per condition across replicates is plotted. The coefficient of variation is calculated as the standard deviation divided by the mean of the signal. In order to do different comparisons, the optional parameters can be altered from the default values. For example the coefficient of variation of the summed signal for each peptide can be plotted as shown below. The function uses the cast function from the reshape2 package and the comparison needs to specified accordingly.

```
> # plot variation of transitions for each condition across replicates
> variation <- plot_variation(data.annotated)
> head(variation[[2]])
```

	Condition	mode_cv	mean_cv	median_cv
1	Hela_Control	0.7337451	0.7062701	0.7230614
2	Hela_Treatment	0.8300339	0.7792517	0.7999057

```
> # plot variation of summed signal for Peptides for each condition across replicates
> variation <- plot_variation(data.annotated,
+                Comparison = FullPeptideName + Condition ~ BioReplicate,
+                fun.aggregate = sum)
>
```

3.4 Plot variation within replicates versus total variation

With the function `plot_variation_vs_total` the coefficient of variation of the signal within replicates can be compared to the variation across all samples. This can serve as an assessment if the variation within technical or biological replicates is indeed smaller than the overall variation. Also for this function the signal for which the variation is plotted and the comparison can be changed by altering the default input options.

```
> # plot variation of transitions for each condition within replicates
> # compared to total variation
> variation <- plot_variation_vs_total(data.annotated)
> head(variation[[2]])

      scope   mode_cv   mean_cv median_cv
1 replicate 0.8062472 0.7426103 0.7620005
2     total 0.7599820 0.7040560 0.7233862
```

3.5 Results on protein level

SWATH2stats can write a protein-level summary matrix showing the summed signals of protein (unique `ProteinName` identifiers) over the MS runs (unique `run_id`) using the function `write_matrix_proteins`. It calculates the sum of all transition intensities per assay, all charge states per peptide, and all peptides for the different protein groups. Note that this function does not select consistently quantified peptides, or a certain number of highest intense peptides, and therefore the summed signal should be used with caution as a measure of protein abundance or to compare protein abundance between runs. For other quantitative protein inference strategies, the R package aLFQ can be used (Rosenberger et al. 2014, see below). For testing differential expression we recommend the downstream tools MSstats and mapDIA (Choi et al. 2014, Teo et al. 2015).

Writing the overview matrix of summed intensities per protein entry per MS run:

```
> protein_matrix <- write_matrix_proteins(data,
+                       filename = "SWATH2stats_overview_matrix_proteinlevel",
+                       rm.decoy = FALSE)
```

3.6 Results on peptide level

SWATH2stats can also write a peptide-level summary matrix showing the summed signals of peptide (unique `FullPeptideName` identifiers) over the MS runs (unique `run_id`) using the function `write_matrix_peptides`. It calculates the sum of all transition intensities per assay and all charge states per peptide.

```
> peptide_matrix <- write_matrix_peptides(data,
+                       filename = "SWATH2stats_overview_matrix_peptidelevel",
+                       rm.decoy = FALSE)
```

4 FDR estimation

Mass-spectrometry-based proteomic experiments produce large amounts of data, requiring statistical validation of the obtained results. Large multi-run proteomics studies are prone to the accumulation of false positive identifications and the statistical significance scores must therefore be normalized accordingly(Benjamini and Hochberg, 1995).
This chapter describes first the functionality of SWATH2stats to estimate and visualize the global false discovery rate in OpenSWATH/mProphet result tables and second the functionality to obtain m-score thresholds (peak group level mProphet-estimated FDR quality) to control FDR on a global level.

Assays are identified by unique identifiers in the column `transition_group_id` of the SWATH data table, peptides by unique identifiers in the column `FullPeptideName` and protein(group)s by unique identifiers in the column `ProteinName`. Different MS injections (also termed runs) are identified based on a unique entry in the column run_id.

4.1 FDR: Overview and visualization

SWATH2stats supplies three functions to assess and visualize the false discovery rate in multi-run SWATH data. These functions are useful to get an overview on the relationship between false discovery rate and m-score thresholds. A suitable m-score threshold can subsequently be used to filter the data with the filtering functions described in the next chapter.
The FDR within the results passing a given score cutoff is evaluated as explained in the introduction:

FDR = (number of decoys * FFT)/(number of targets)

Application of the decoy-counting-based FDR assessment functions in interplay with the meta-data filters can help the researcher in selecting an efficient strategy to establish highest possible data quality for downstream analyses. By counting the decoys before and after application of a filter, the selectivity of a given filter for likely true (target) over false (decoy) data can be estimated.
With a first basic function `assess_decoy_rate` the overall number of decoy peptides can be counted in the data:

```
> assess_decoy_rate(data)
```

The function `assess_fdr_overall` creates a global assessment of decoy rates (and estimated FDR) on assay, peptide and protein level. Results are reported by default as .csv table and visualized in a .pdf report. Setting the output option to "Rconsole" reports the results back to R. Included in the pdf report are plots showing the estimated global FDR in relation to the m-score threshold. Because false-positive hits accumulate over different runs, the false discovery rate estimated by this function will be higher than if assessed within each run individually.

```
> # count decoys and targets on assay, peptide and protein level
> # and report FDR at a range of m_score cutoffs
```

```
> assess_fdr_overall(data, FFT = 0.7, output = "pdf_csv", plot = TRUE,
+                    filename='assess_fdr_overall_testrun')
> # The results can be reported back to R for further calculations
> overall_fdr_table <- assess_fdr_overall(data, FFT = 0.7,
+                                         output = "Rconsole")
```

The function `plot.fdr_table` allows to create the report plots from this overall fdr table.

```
> # create plots from fdr_table
> plot(overall_fdr_table, output = "Rconsole",
+                filename = "FDR_report_overall")
```

The function `assess_fdr_byrun` investigates the decoy rate or FDR in individual runs and by default reports the results in a .csv table and .pdf file. Setting the output option to "Rconsole" reports back the results to R. This function is used if the FDR for different injections should be estimated separately.

```
> # count decoys and targets on assay, peptide and protein level per run
> # and report FDR at a range of m_score cutoffs
> assess_fdr_byrun(data, FFT = 0.7, output = "pdf_csv", plot = TRUE,
+                  filename='assess_fdr_byrun_testrun')
> # The results can be reported back to R for further calculations
> byrun_fdr_cube <- assess_fdr_byrun(data, FFT = 0.7,
+                                    output = "Rconsole")
```

The function `plot._fdr_cube` allows to create the report plots from this by-run fdr cube.

```
> # create plots from fdr_table
> plot(byrun_fdr_cube, output = "Rconsole",
+               filename = "FDR_report_overall")
```

4.2 Identification of useful m-score cutoffs to satisfy desired FDR criteria

SWATH2stats supplies three functions for the identification of useful m-score cutoffs to satisfy FDR criteria on assay, peptide and protein level over many different runs. These functions return an m-score value, which can be used to filter the data of these different runs in order to obtain a desired overall FDR. The following functions report an m-score cutoff to achieve a strict global FDR target.
The function `mscore4assayfdr` reports an m-score cutoff to achieve a desired overall (global) assay FDR:

```
> # select and return a useful m_score cutoff in order
> # to achieve the desired FDR quality for the entire table
> mscore4assayfdr(data, FFT = 0.7, fdr_target=0.01)

[1] 0.01
```

The function `mscore4pepfdr` reports an m-score cutoff to achieve a desired overall (global) peptide FDR:

```
> # select and return a useful m_score cutoff
> # in order to achieve the desired FDR quality for the entire table
> mscore4pepfdr(data, FFT = 0.7, fdr_target=0.02)

[1] 0.01
```

The function `mscore4protfdr` reports an m-score cutoff to achieve a desired overall (global) protein FDR. Protein FDR control on peak group quality level is a very strict filter and should be handled with caution. Alternatively, a function `filter_mscore_fdr` is described below applying a two-tiered filtering approach.

```
> # select and return a useful m_score cutoff in order
> # to achieve the desired FDR quality for the entire table
> mscore4protfdr(data, FFT = 0.7, fdr_target=0.02)

[1] 0.0001778279
```

5 Filtering the data

In this chapter the SWATH data is filtered based on the study design or desired global FDR criteria to be achieved. By setting the option rm.decoy=FALSE, the decoy peptides can be kept in the data in order to evaluate the selectivity of a given filter for likely true (target) over false (decoy) data by decoy counting with the functions described in the previous chapter.

Before converting the data for statistical analysis the rm.decoy option is set to 'TRUE' in order to remove any decoy peptides and proteins from the data.

5.1 Filter on m-score

The function `filter_mscore` removes all measured peak groups that are above a certain m-score value. The number of rows removed by the function is indicated.

```
> data.filtered.mscore <- filter_mscore(data.annotated, 0.01)
```

The function `filter_mscore_freqobs` takes into account how many times in the different injection runs a peak group has been confidently (as defined by the m-score threshold) identified. This is useful in large data of many different replicates. For example the data for a certain precursor that has been confidently identified in most of the replicates but does not pass the threshold in one replicate still should be kept for statistical analysis. The function `filter_mscore_freqobs` can be used to filter for precursors that were observed with a certain m-score threshold and frequency across the samples. In the following example, precursors passing an m-score threshold of 0.01 in 80 % of the replicates are selected. The option rm.decoy is set to FALSE to keep the decoys for subsequent FDR assessment.

```
> data.filtered.mscore <- filter_mscore_freqobs(data.annotated, 0.01, 0.8,
+                                               rm.decoy=FALSE)
```

The function `filter_mscore_condition` selects only precursors that have passed a certain m-score threshold in a minimum number of replicates for the same condition (as defined by the study design table). In contrast to the previous function, this selects precursors that are confidently identified a certain number of times within a condition as opposed to being identified a certain number of times across all samples.

```
> data.filtered.mscore <- filter_mscore_condition(data.annotated, 0.01, 3)
```

In order to reach a compromise between a very stringent m-score filter controlling the global protein FDR, and keeping valid peptide quantifications in the data, we introduce here a two-tiered filtering approach with the function `filter_mscore_fdr`. This uses a similar approach as implemented for extracting quantitative data from multi-run DDA data sets (Fermin et al. 2011). In the first step, an m-score cutoff is identified to reach a desired protein-level FDR. All proteins for which one peakgroup passes this strict m-score cutoff criterion are collected in a protein master list. The original data is then filtered i.) for the proteins present in the master list and ii.) filtered for all peptide quantifications passing an m-score cutoff to achieve a desired global peptide-level FDR. Note that the m-score cutoff to filter the protein list will typically be more stringent than the second m-score cutoff to filter the peptides.

```
> data.filtered.fdr <- filter_mscore_fdr(data, FFT=0.7,
+                                        overall_protein_fdr_target = 0.03,
+                                        upper_overall_peptide_fdr_limit = 0.05)
```

5.2 Filter on proteotypic peptides

The function `filter_proteotypic_peptides` selects only data that is based on proteotypic peptides (peptides only contained in one protein and marked by "1/" in the beginning of the protein identifier). These functions also remove the '1' in front of the protein identifier from proteotypic peptides.

```
> data <- filter_proteotypic_peptides(data.filtered.mscore)
> data.all <- filter_all_peptides(data.filtered.mscore)
```

5.3 Filter on number of peptides per protein

With the function `filter_on_max_peptides` the peptides showing the strongest signal over the entire table can be selected (top n approach). Removing the lower intense peptides for a protein can make the statistical analysis faster or result in more accurate quantification of proteins under the assumption that quantification of more intense peptides is more robust.

```
> data.filtered.max <- filter_on_max_peptides(data.filtered.mscore, 5)
```

Conversely maybe only data for proteins with a minimum number of supporting peptides should be selected. With the function `filter_on_min_peptides` only the proteins for which at least a certain number of peptides have been measured are selected. This filter can also be powerful to remove false positive hits from the data as these are enriched in the fraction of single hits. FDR assessment based on decoy counting may still be valid after such filtering (Reiter et al. 2009).

```
> data.filtered.max.min <- filter_on_min_peptides(data.filtered.max, 2)
```

6 Conversion of data for other tools

Dedicated programs or tools exist to asssess the statistical significance of a regulated protein or estimate the absolute quantity thereof. A selection of such tools are listed in Section 8. To facilitate the analyses of SWATH data with these tools, SWATH2stats contains several functions that convert the data into the required format.

6.1 Convert results to transition-level format

6.1.1 Conversion within R

Some tools work based on transition-level data. With the function `disaggregate` the SWATH data is changed from a table where one row corresponds to one quantified peak group to a table where one row corresponds to one measured transition.

```
> data.transition <- disaggregate(data)
> head(data.transition)

  ProteinName PeptideSequence PrecursorCharge      Condition
1    Protein6      AAVDLIIAVK               2   Hela_Control
2    Protein6      AAVDLIIAVK               2 Hela_Treatment
3    Protein6      AAVDLIIAVK               2 Hela_Treatment
4    Protein6      AAVDLIIAVK               2 Hela_Treatment
5    Protein6      AAVDLIIAVK               2   Hela_Control
6    Protein6      AAVDLIIAVK               2   Hela_Control
  BioReplicate Run NakedSequence       RT          FragmentIon
1            2   3    AAVDLIIAVK 3904.151 2914841_AAVDLIIAVK_2
2            3   6    AAVDLIIAVK 3778.492 2914841_AAVDLIIAVK_2
3            1   2    AAVDLIIAVK 4450.159 2914841_AAVDLIIAVK_2
4            2   4    AAVDLIIAVK 3991.374 2914841_AAVDLIIAVK_2
5            3   5    AAVDLIIAVK 3726.064 2914841_AAVDLIIAVK_2
6            1   1    AAVDLIIAVK 4352.417 2914841_AAVDLIIAVK_2
  Intensity
1     17918
2      1863
3      1631
4     24520
5      3665
6     11878

> write.csv(data.transition, file='transition_level_output.csv',
+           row.names=FALSE, quote=FALSE)
```

6.1.2 Conversion using a python script

For very large SWATH data it is faster to use a custom-made python script to transform the data from a peptide-level format to a transition-level format. With the function `convert4pythonscript` the necessary columns are selected and the nomenclature for modified peptides is changed. Subsequently the data is written to disk.

```
> data.python <- convert4pythonscript(data)
```

```
> write.table(data.python, file="input.tsv", sep="\t", row.names=FALSE, quote=FALSE)
```

The .tsv table can be transformed into a transition level table using a python script (as example featurealigner2msstats_withRT.py from msproteomicstools which is available in the scripts folder of the package).

`python ./featurealigner2msstats.py input.csv output.csv`

Afterwards the generated .csv table is loaded again into R.

```
> data.transition <- data.frame(fread('output.csv',
+                                     sep=',', header=TRUE))
```

6.2 MSstats

In order to use the data in the R Bioconductor package `MSstats` (Choi et al. 2014), the transition-level data needs to be converted using the function `convert4MSstats`. Afterwards the data can directly be processed using the `MSstats` package as shown here by application of the function `dataProcess` from the MSstats package.

```
> MSstats.input <- convert4MSstats(data.transition)
> library(MSstats)
> quantData <- dataProcess(MSstats.input)
```

6.3 aLFQ

The package `aLFQ` (Rosenberger et al. 2014) can read the original OpenSWATH output. Alternatively the `aLFQ` package can also be applied to the filtered and annotated data from the `SWATH2stats` package. To convert the data after filtering to the format for aLFQ, the function `convert4aLFQ` is applied to the transition-level data.

```
> aLFQ.input <- convert4aLFQ(data.transition)
> library(aLFQ)
> prots <- ProteinInference(aLFQ.input, peptide_method = 'top',
+                           peptide_topx = 3,
+                           peptide_strictness = 'loose',
+                           peptide_summary = 'mean',
+                           transition_topx = 3,
+                           transition_strictness = 'loose',
+                           transition_summary = 'sum',
+                           fasta = NA, model = NA,
+                           combine_precursors = FALSE)
```

6.4 mapDIA

In order to convert the data into the format for the mapDIA program (Teo et al. 2015), the function `convert4mapDIA` is used. Technical replicates included in the data are not taken into account by mapDIA. Therefore the function `convert4mapDIA` averages the SWATH data from technical replicates if such exist.

```
> mapDIA.input <- convert4mapDIA(data.transition, RT =TRUE)
> head(mapDIA.input)
```

```
  ProteinName PeptideSequence         FragmentIon Hela_Control_1
1    Protein2       TVVVAFLGR 1000686_TVVVAFLGR_2           1930
2    Protein2       TVVVAFLGR 1000688_TVVVAFLGR_2            924
3    Protein2       TVVVAFLGR 1000689_TVVVAFLGR_2           3264
4    Protein2       TVVVAFLGR 1000693_TVVVAFLGR_2           5681
5    Protein2       TVVVAFLGR 1000695_TVVVAFLGR_2           1386
6    Protein2       TVVVAFLGR 1000697_TVVVAFLGR_2            924
  Hela_Control_2 Hela_Control_3 Hela_Treatment_1 Hela_Treatment_2
1           5973           2792              378            14198
2           3192           1785               84             6535
3          10092           4130              483            28929
4          16316           6745             1025            45055
5           4573           1995              294            12174
6           3856           1512              147            11405
  Hela_Treatment_3       RT
1             6494 68.68395
2             3401 68.68395
3            12443 68.68395
4            20683 68.68395
5             4945 68.68395
6             5165 68.68395

> write.table(mapDIA.input, file='mapDIA.txt', quote=FALSE,
+           row.names=FALSE, sep='\t')
```

6.5 PECA

In order to convert the data into the format for the PECA tool (Suomi et al. 2015), the function `convert4PECA` is used. If both biological and technical replicates exist in the data, technical replicates are not taken into account by PECA. Therefore the function `convert4PECA` averages the SWATH data from technical replicates. As the PECA tool does not use transition-level data, the `convert4PECA` function is applied on the peptide-level data.

```
> PECA.input <- convert4PECA(data)
> head(PECA.input)

  ProteinName FullPeptideName_Charge Hela_Control_1 Hela_Control_2
1    Protein1             AEAIKADK_2           3472          10040
2    Protein1 ATITPDEUC(UniMod:4)K_2           6845          20183
3    Protein1          DFVVPGPGTAK_2          64824         132045
4    Protein1        EAEAAHGTVTVTR_2           4292          13571
5    Protein1        EAEAAHGTVTVTR_3          50545          64994
6    Protein1              FEAQQSK_2           5071          21981
  Hela_Control_3 Hela_Treatment_1 Hela_Treatment_2 Hela_Treatment_3
1          11377             2999            11351             13703
2          17263             3587            18426             20815
3         108325            27922            90515            112908
4          12373             1722             9327             11324
5          59386            19307            51076             66352
6          22471             4518            18675             19182
```

For DIA data, the reproducibility-optimized test statistic (ROTS) is the suggested statistic to be used within PECA (Suomi et al. 2017).

```
> library(PECA)
> group1 <- c("Hela_Control_1", "Hela_Control_2", "Hela_Control_3")
> group2 <- c("Hela_Treatment_1", "Hela_Treatment_2", "Hela_Treatment_3")
> # PECA_df
> results <- PECA_df(PECA.input, group1, group2, id="ProteinName", test = "rots")
> head(results)

                slr          t     score   n         p     p.fdr
Protein1  0.56670376  0.49534484 0.5123095  15 0.5386230 0.8597692
Protein2 -1.17220909 -0.90344421 0.4194830   9 0.3085459 0.7612220
Protein3 -1.36195731 -1.16653090 0.3431122   1 0.3431122 0.7612220
Protein4  0.45686451  0.40483100 0.5723418   6 0.6448269 0.8597692
Protein5  0.17214078  0.13969582 0.8649745   4 0.9687296 1.0000000
Protein6  0.05582275  0.04950873 0.9393367 103 1.0000000 1.0000000
```

6.6 imsbInfer

The package imsbInfer needs all the columns from the OpenSWATH output, therefore the function `reduce_OpenSWATH_output` needs to be omitted in the workflow (see above). If the package imsbInfer should be used after SWATH2stats, a decoy column needs to be added as exemplified below if it has been removed by the filtering functions.

```
> data.annotated.full <- sample_annotation(OpenSWATH_data, Study_design)
> data.annotated.full <- filter_mscore(data.annotated.full,
+                                      mscore4assayfdr(data.annotated.full, 0.01))
> data.annotated.full$decoy <- 0 ### imsbInfer needs the decoy column
> library(imsbInfer)
> specLib <- loadTransitonsMSExperiment(data.annotated.full)
```

7 Acknowledgments

We want to acknowledge Koh Ching Chiek, and Dr. Olga Schubert for help in testing the usability of this package and helpful comments to implementation and description. We want to acknowledge George Rosenberger for discussion and advice to bioinformatic questions and the OpenSWATH + pyProphet workflow. We acknowledge Tomi Suomi for assistance in how to best interface SWATH2stats to the PECA package.

8 Software and tools

`OpenSWATH`: http://www.openswath.org
`MSstats`: MSstats is available on Bioconductor (www.bioconductor.org) or on www.mssstats.org
`aLFQ`: aLFQ is available on CRAN (https://cran.r-project.org)
`mapDIA`: mapDIA is available on Sourceforge (http://sourceforge.net/projects/mapdia/)
`PECA`: PECA is available on Bioconductor (www.bioconductor.org)
`imsbInfer`: imsbInfer is available on Github (https://github.com/wolski/imsbInfer)
`msproteomicstools`: https://github.com/msproteomicstools

9 References

Benjamini Y., and Hochberg Y. (1995). Controlling the False Discovery Rate: a Practical and Powerful Approach to Multiple Testing. J. R. Statist. Soc. B, 57(1), 289-300.

Choi H., and Nesvizhskii A.I. (2008). False discovery rates and related statistical concepts in mass spectrometry-based proteomics. Journal of Proteome Research, 7(1), 47-50.

Choi M., et al. (2014). MSstats: an R package for statistical analysis of quantitative mass spectrometry-based proteomic experiments. Bioinformatics 30(17): 2524-2526.

Elias J.E., and Gygi S.P. (2007). Target-decoy search strategy for increased confidence in large-scale protein identifications by mass spectrometry. Nature Methods, 4(3), 207-214.

Fermin D. et al. (2011). Abacus: A computational tool for extracting and pre-processing spectral count data for label-free quantitative proteomic analysis. Proteomics, 11(7), 1340-1345.

Gillet, L., et al. (2012). Targeted data extraction of the MS/MS spectra generated by data-independent acquisition: a new concept for consistent and accurate proteome analysis. Mol Cell Proteomics 11(6).

Kaell L. et al. (2008). Assigning significance to peptides identified by tandem mass spectrometry using decoy databases. Journal of Proteome Research, 7(1), 29-34.

Nesvizhskii, A. I. (2010). A survey of computational methods and error rate estimation procedures for peptide and protein identification in shotgun proteomics. Journal of Proteomics, 73(11), 2092-123.

Reiter L. et al. (2009). Protein identification false discovery rates for very large proteomics data sets generated by tandem mass spectrometry. Molecular and Cellular Proteomics : MCP, 8(11), 2405-17.

Reiter L. et al. (2011). mProphet: automated data processing and statistical validation for large-scale SRM experiments. Nature Methods, 8(5), 430-5.

Rosenberger G., et al. (2014). A repository of assays to quantify 10,000 human proteins by SWATH-MS.' Sci Data 1: 140031.

Rosenberger G., et al. (2014). aLFQ: an R-package for estimating absolute protein quantities from label-free LC-MS/MS proteomics data. Bioinformatics 30(17): 2511-2513.

Rost H.L., et al. (2014). OpenSWATH enables automated, targeted analysis of data-independent acquisition MS data. Nat Biotechnol 32(3): 219-223.

Suomi T., Corthals G., Nevalainen O.S., and Elo L.L. (2015). Using Peptide-Level Proteomics Data for Detecting Differentially Expressed Proteins. J Proteome Res. Nov 6;14(11):4564-70. doi: 10.1021/acs.jproteome.5b00363.

Suomi, T. and Elo L.L. (2017). Enhanced differential expression statistics for data-independent acquisition proteomics" Scientific Reports 7, Article number: 5869.doi:10.1038/s41598-017-05949-y

Teo G., et al. (2015). mapDIA: Preprocessing and statistical analysis of quantitative proteomics data from data independent acquisition mass spectrometry. J Proteomics 129: 108-120.

Teleman J., et al. (2015). DIANA–algorithmic improvements for analysis of data-independent acquisition MS data. Bioinformatics 31(4): 555-562.

Venable J.D., et al. (2004). Automated approach for quantitative analysis of complex peptide mixtures from tandem mass spectra. Nat Methods 1(1): 39-45.