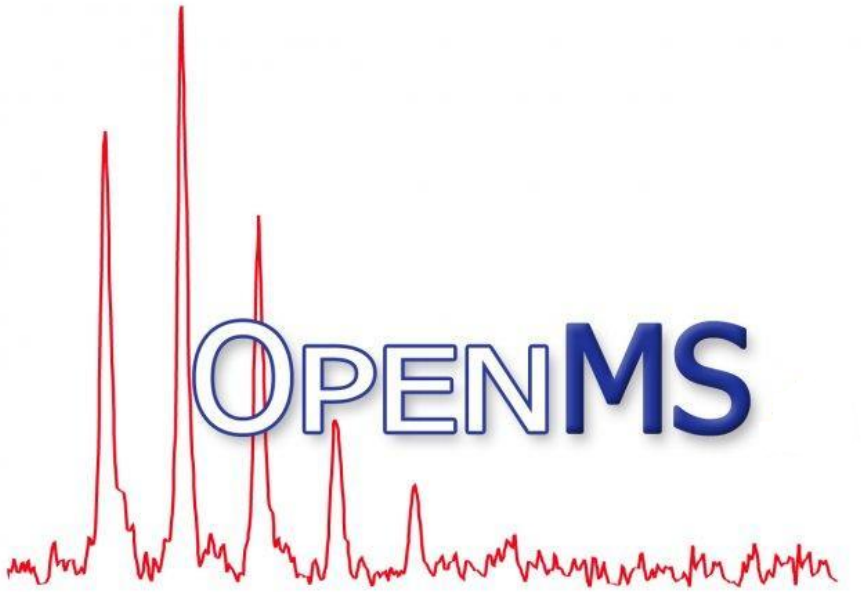

OpenMS Tutorial

The OpenMS Developers

Mathias Walzer, Timo Sachsenberg, Fabian Aicheler,
Marc Rurik, Johannes Veit,
Bludau Isabell, Patrick Pedrioli,
Julianus Pfeuffer, Xiao Liang,
Knut Reinert, and Oliver Kohlbacher

Contents

1 General remarks

- This handout will guide you through an introductory tutorial for the OpenMS/TOPP software package [1].
- OpenMS [2] is a versatile open-source library for mass spectrometry data analysis. Based on this library, we offer a collection of command-line tools ready to be used by end users. These so-called TOPP tools (short for "The OpenMS Proteomics Pipeline") [3] can be understood as small building blocks of arbitrary complex data analysis workflows.
- In order to facilitate workflow construction, OpenMS was integrated into KNIME [4], the Konstanz Information Miner, an open-source integration platform providing a powerful and flexible workflow system combined with advanced data analytics, visualization, and report capabilities. Raw MS data as well as the results of data processing using TOPP can be visualized using TOPPView [5].
- In this hands-on tutorial session, you will become familiar with some of the basic functionalities of OpenMS/TOPP, TOPPView, and KNIME and learn how to use a selection of TOPP tools used in the tutorial workflows.
- All data referenced in this tutorial can be found in the 🗀 **Example_Data** folder that came with this tutorial.

2 Getting started

Before we get started we will install OpenMS and KNIME using the installers provided on the USB stick. Please choose the directory that matches your operating system and execute the installer. Note that these steps are not necessary if you use one of our laptops.

For example for Windows you call

- the OpenMS installer: 🗀 **Windows ▸ OpenMS-2.0_Win64_setup.exe**
- the KNIME installer: 🗀 **Windows ▸ OpenMS-2.0-prerequisites-installer.exe** and 🗀 **Windows ▸ KNIME 2.12.0 Installer (64bit).exe**

on Mac you call

- the OpenMS installer: 🗀 **Mac ▸ OpenMS-2.0.0_setup.dmg**
- the KNIME installer: 🗀 **Mac ▸ knime_2.12.0.macosx.cocoa.x86_64.dmg**

and follow the instructions.

2.1 Data conversion

Each MS instrument vendor has one or more formats for storing the acquired data. Converting these data into an open format (preferably mzML) is the very first step when you want to work with open-source mass spectrometry software. A freely available conversion tool is ProteoWizard. The OpenMS installation package for Windows automatically installs ProteoWizard, so you do not need to download and install it separately.

Please note that due to restrictions from the instrument vendors, file format conversion for most formats is only possible on Windows systems, so exporting from the acquisition PC connected to the instrument is usually the most convenient option. All files used in this tutorial have already been converted to mzML by us, so you do not need to do it yourself.

2.2 Data visualization using TOPPView

Visualizing the data is the first step in quality control, an essential tool in understanding the data, and of course an essential step in pipeline development. OpenMS provides a

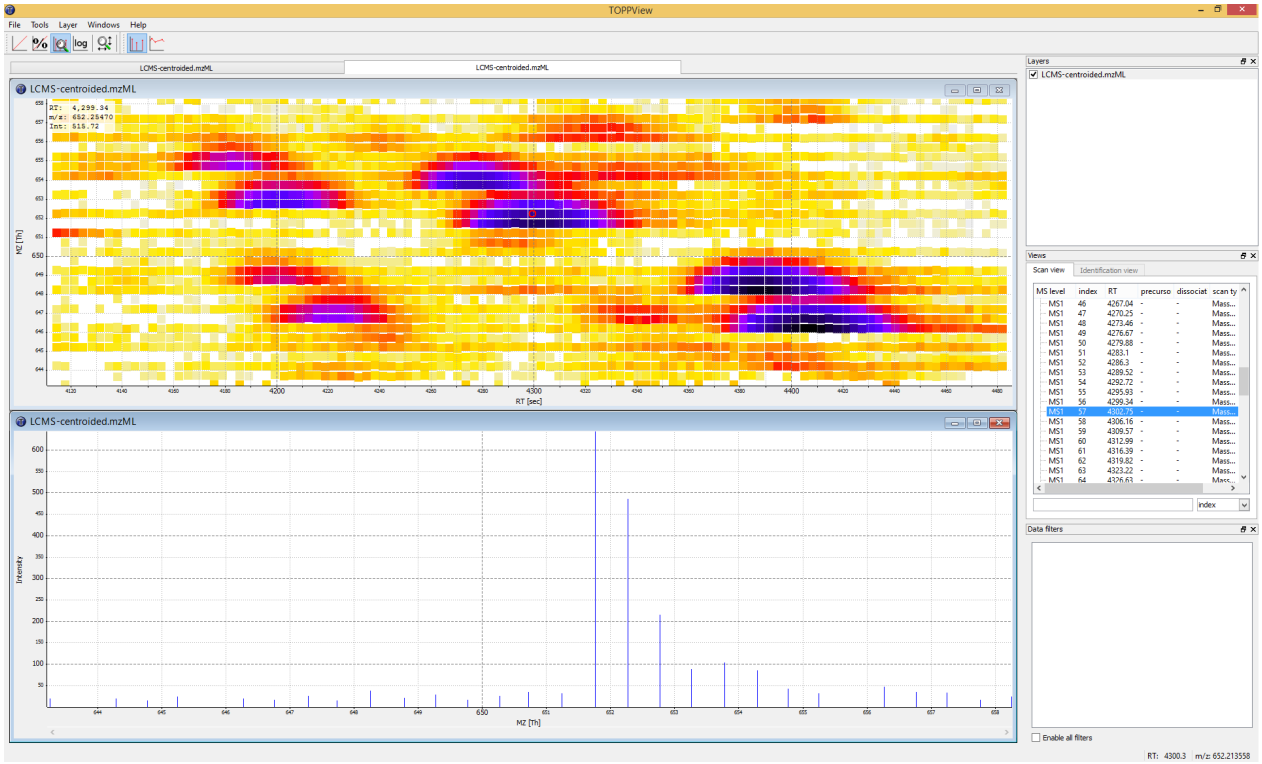

Figure 1: TOPPView, the graphical application for viewing mass spectra and analysis results. Top window shows a small region of a peak map. In this 2D representation of the measured spectra, signals of eluting peptides are colored according to the raw peak intensities. The lower window displays an extracted spectrum (=scan) from the peak map. On the right side, the list of spectra can be browsed.

convenient viewer for some of the data: **TOPPView**.

We will guide you through some of the basic features of **TOPPView**. Please familiarize yourself with the key controls and visualization methods. We will make use of these later throughout the tutorial. Let's start with a first look at one of the files of our tutorial data set:

- Start **TOPPView** (see Start-Menu or Applications on MacOS)
- Go to File > Open File, navigate to the directory where you copied the contents of the USB stick to, and select **Example_Data ▸ Introduction ▸ datasets ▸ small ▸ velos005614.mzM**. This file contains a reduced LC-MS map (only a selected RT and m/z range was extracted using the TOPP tool **FileFilter**) of a label-free measurement of the human

platelet proteome recorded on an Orbitrap velos. The other two mzML files contain technical replicates of this experiment. First, we want to obtain a global view on the whole LC-MS map - the default option Map view 2D is the correct one and we can click the Ok button.

- Play around.
- Three basic modes allow you to interact with the displayed data: scrolling, zooming and measuring:
  - Scroll mode
    * Is activated by default (though each loaded spectra file is displayed zoomed out first, so you do not need to scroll).
    * Allows you to browse your data by moving around in RT and m/z range.
    * When zoomed in, to scroll the spectra map, click-drag on the current view.
    * Arrow keys can be used to scroll the view as well.
  - Zoom mode
    * Zooming into the data: either mark an area in the current view with your mouse while holding the left mouse button plus the ctrl key to zoom to this area or use your mouse wheel to zoom in and out.
    * All previous zoom levels are stored in a zoom history. The zoom history can be traversed using ctrl + + or ctrl + - or the mouse wheel (scroll up and down).
    * Pressing the Backspace key zooms out to show the full LC-MS map (and also resets the zoom history).
  - Measure mode
    * It is activated using the ⇧ key.
    * Press the left mouse button down while a peak is selected and drag the mouse to another peak to measure the distance between peaks.
    * This mode is implemented in the 1D and 2D mode only.
- Right click on your 2D map and select Switch to 3D view and examine your data in 3D mode

- Go back to the 2D view. In 2D mode, visualize your data in different normalization modes, use linear, percentage and log-view (icons on the upper left tool bar).

  > Note: On Apple OS X, due to a bug in one of the external libraries used by OpenMS, you will see a small window of the 3D mode when switching to 2D. Close the 3D tab in order to get rid of it.

- In **TOPPView** you can also execute TOPP tools. Go to Tools » Apply tool (whole layer) and choose a TOPP tool (e.g., FileInfo) and inspect the results.

2.3 Introduction to KNIME / OpenMS

Using OpenMS in combination with KNIME you can create, edit, open, save, and run workflows combining TOPP tools with the powerful data analysis capabilities of KNIME. Workflows can be created conveniently in a graphical user interface. The parameters of all involved tools can be edited within the application and are also saved as part of the workflow. Furthermore, KNIME interactively performs validity checks during the workflow editing process, in order to make it more difficult to create an invalid workflow.

Throughout most of the parts of this tutorial you will use KNIME to create and execute workflows. This first step is to make yourself familiar with KNIME.

2.3.1 Install OpenMS using KNIME

Before we can start with the tutorial we need to install all the required extensions for KNIME.

First, we install some additional extensions that are required by our OpenMS nodes or used in the Tutorials e.g. for visualization.

1. Click on Help » Install New Software...
2. From the Work with: drop down list select http://update.knime.org/analytics-platform/2.12
3. Now select the following plugins from the KNIME & Extensions category
   - KNIME Base Chemistry Types & Nodes

- KNIME Chemistry Add-Ons
- KNIME File Handling Nodes
- KNIME Interactive R Statistics Integration
- KNIME Math Expression (JEP)
- KNIME R Statistics Integration (Windows Binaries)
- KNIME Report Designer
- KNIME SVG Support
- KNIME XLS Support
- KNIME XML-Processing

4. From the Work with: drop down list select http://tech.knime.org/update/community-contributions/trusted/2.12
5. Now select the following plugin from the "KNIME Community Contributions - Cheminformatics" category
   - RDKit KNIME integration
6. Follow the instructions and after a restart of KNIME the dependencies will be installed.

You are now ready to install the OpenMS nodes.

1. Open KNIME.
2. Click on Help > Install New Software...
3. From the Work with: drop down list select the http://tech.knime.org/update/community-contributions/trusted/2.12
4. Select the OpenMS nodes in the category: "KNIME Community Contributions - Bioinformatics & NGS" and click Next.
5. Follow the instructions and after a restart of KNIME the OpenMS nodes will be available under "Community Nodes".

2.3.2 KNIME concepts

A workflow is a sequence of computational steps applied to a single or multiple input data sets to process and analyze the data. In KNIME such workflows are implemented graphically by combining so-called nodes. A node represents a single analysis step in a workflow. Nodes have input and output ports where the data enters the node or the results are provided for other nodes after processing, respectively. KNIME distinguishes between different port types, representing different types of data. The most common representation of data in KNIME are tables (similar to an excel sheet). Ports that accept tables are marked with a small triangle. For OpenMS we use a different port type, so called file ports, representing complete files. Those ports are marked by a small grey box. Dark grey boxes represent mandatory inputs and light grey boxes optional inputs.

A typical OpenMS workflow in KNIME can be divided in two conceptually different parts:

- Nodes for signal and data processing, filtering and data reduction. Here, files are passed between nodes. Execution times of the individual steps are longer as the main computational steps are performed.
- Downstream statistical analysis and visualization. Here, tables are passed between nodes.

Between file-based processing and table-based analysis a conversion node typically performs the conversion from OpenMS results into KNIME tables.

Nodes can have three different states, indicated by the small traffic light below the node.

- Inactive, failed, and not yet fully configured nodes are marked red.
- Configured but not yet executed nodes are marked yellow.
- Successfully executed nodes are marked green.

If the node execution failed the node will switch to the red state.

Most nodes will be configured as soon as all input ports are connected. For some nodes additional parameters have to be provided that cannot be either guessed from the

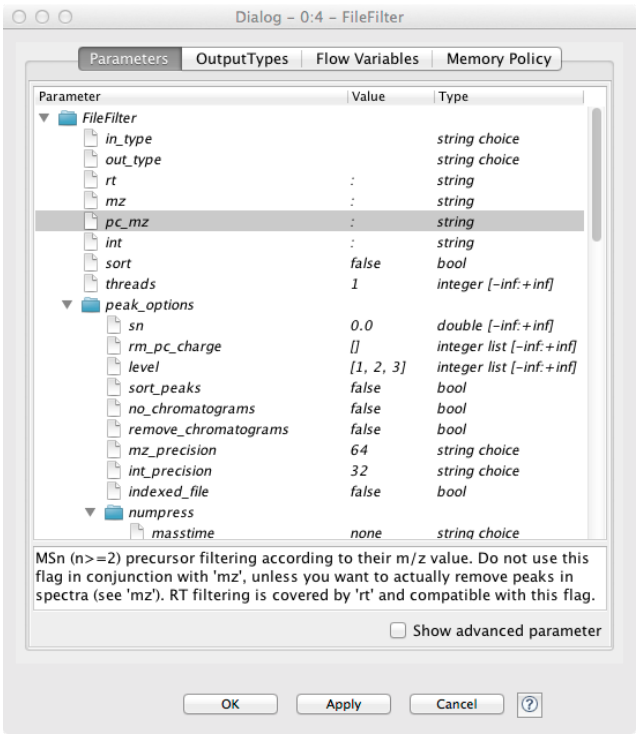

Figure 2: Node configuration dialog of an OpenMS node.

data or filled with sensible defaults. In this case, of if you want to customize the default configuration, you can open the configuration dialog of a node with a double-click on the node. For OpenMS you will see a configuration dialog like the one shown in Figure 2.

> Note: OpenMS distinguishes between normal parameters and advanced parameters. Advanced parameters are by default hidden from the users since they should only rarely be customized. In case you want to have a look at the parameters or need to customize them in one of the tutorials you can show them by clicking on the checkbox Show advanced parameter in the lower part of the dialog.

The dialog shows the individual parameters, their current value and type, and, in the lower part of the dialog, the documentation for the currently selected parameter.

2.3.3 Overview of the graphical user interface

The graphical user interface (GUI) of KNIME consists of different components or so called panels that are shown in Figure 3. We will shortly introduce the individual panels and their

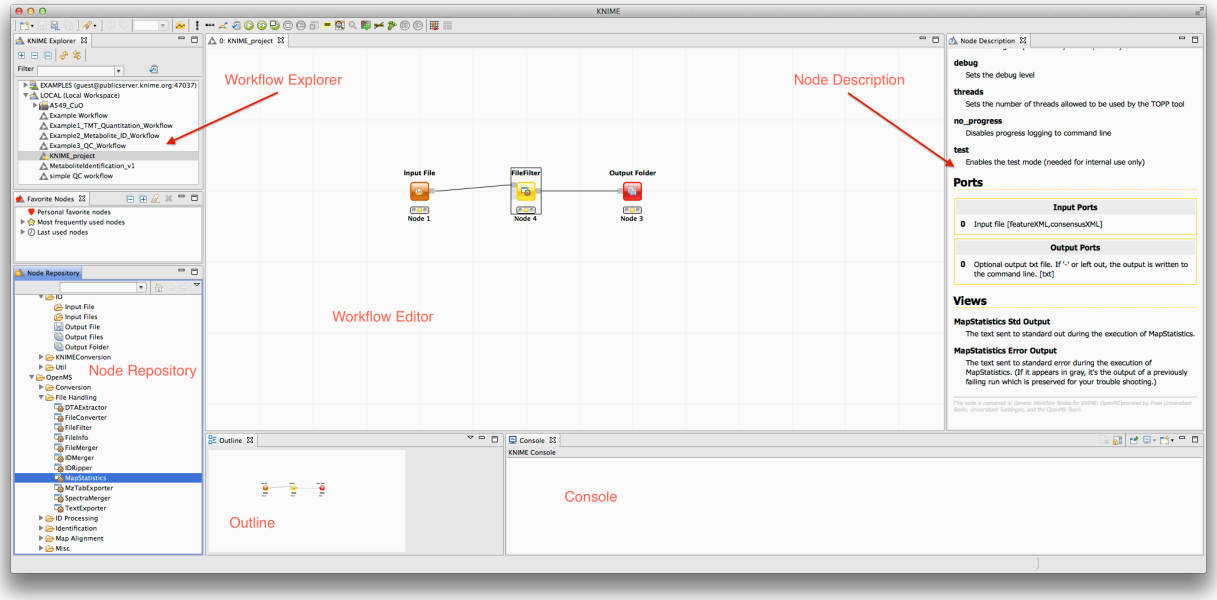

Figure 3: The KNIME workbench.

purposes below.

Workflow Editor: The workflow editor is the central part of the KNIME GUI. Here you assemble the workflow by adding nodes from the Node Repository via "drag & drop". Nodes can be connected by clicking on the output port of one node and releasing the mouse at the desired input port of the next node.

Workflow Explorer: Shows a list of available workflows (also called workflow projects). You can open a workflow by double clicking it. A new workflow can be created with a right-click in the Workflow Explorer followed by selecting New KNIME Workflow....

Node Repository: Shows all nodes that are available in your KNIME installation. Every plugin you install will provide new nodes that can be found here. The OpenMS nodes can be found in Community Nodes > OpenMS. Nodes for managing files (e.g., Input Files or Output Folders) can be found in Community Nodes > GenericKnimeNodes. You can search the node repository by typing the node name into the small text box in the upper part of the node repository.

Outline: The Outline panel contains a small overview of the complete workflow. While

of limited use when working on a small workflow, this feature is very helpful as soon as the workflows get bigger.

Console: In the console panel warning and error messages are shown. This panel will provide helpful information if one of the nodes failed or shows a warning sign.

Node Description: As soon as a node is selected, the Node Description window will show the documentation of the node including documentation for all its parameters. For OpenMS nodes you will also find a link to the tool page in the online documentation.

2.3.4 Creating workflows

Workflows can easily be created by a right click in the Workflow Explorer followed by clicking on New KNIME Workflow....

2.3.5 Sharing workflows

To be able to share a workflow with others, KNIME supports the import and export of complete workflows. To export a workflow, select it in the Workflow Explorer and select File > Export KNIME Workflow.... KNIME will export workflows as a zip file containing all the information on nodes, their connections, and their configuration. Those zip files can again be imported by selecting File > Import KNIME Workflow....

> Note: For your convenience we added all workflows discussed in this tutorial to the **Workflows** folder. If you want to check your own workflow by comparing it to the solution or got stuck, simply import the full workflow from the corresponding zip file.

2.3.6 Duplicating workflows

During the tutorial a lot of the workflows will be created based on the workflow from a previous task. To keep the intermediate workflows we suggest you create copies of your workflows so you can see the progress. To create a copy of your workflow follow the next steps.

- Right click on the workflow you want to create a copy of in the Workflow Explorer and select Copy.
- Right click again somewhere on the workflow explorer and select Paste.
- This will create a workflow with same name as the one you copied with a (2) appended.
- To distinguish them later on you can easily rename the workflows in the Workflow Explorer by right clicking on the workflow and selecting Rename.

> Note: To rename a workflow it has to be closed.

2.3.7 A minimal workflow

Let us now start with the creation of our very first, very simple workflow. As a first step, we will gather some basic information about the data set before starting the actual development of a data analysis workflow.

- Create a new workflow.
- Add an **Input File** node and an **Output Folder** node (to be found in Community Nodes > GenericKnimeNodes > IO and a **FileInfo** node (to be found in the category Community Nodes > OpenMS > File Handling) to the workflow.
- Connect the **Input File** node to the **FileInfo** node, and the first output port of the **FileInfo** node to the **Output Folder** node.

> Note: In case you are unsure about which node port to use, hovering the cursor over the port in question will display the port name and what kind of input it expects.

The complete workflow is shown in Figure 4. FileInfo can produce two different kinds of output files.

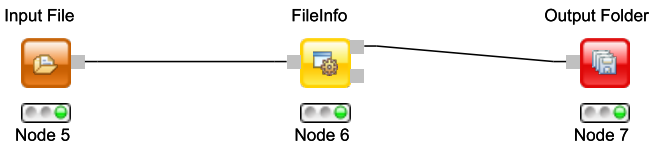

Figure 4: A minimal workflow calling FileInfo on a single file.

- All nodes are still marked red, since we are missing an actual input file. Double-click the Input File node and select Browse. In the file system browser select **Example_Data ▸ Introduction ▸ datasets ▸ tiny ▸ velos005614.mzML** and click Open. Afterwards close the dialog by clicking Ok.

  > Note: Make sure to use the "tiny" version this time, not "small", for the sake of faster workflow execution.

- The **Input File** node and the **FileInfo** node should now have switched to yellow, but the **Output Folder** node is still red. Double-click on the **Output Folder** node and click on Browse to select an output directory for the generated data.

- Great! Your first workflow is now ready to be run. Press ⇧ + F7 to execute the complete workflow. You can also right click on any node of your workflow and select Execute from the context menu.

- The traffic lights tell you about the current status of all nodes in your workflow. Currently running tools show either a progress in percent or a moving blue bar, nodes waiting for data show the small word "queued", and successfully executed ones become green. If something goes wrong (e.g., a tool crashes), the light will become red.

- In order to inspect the results, you can just right-click the **Output Folder** node and select View: Open the output folder. You can then open the text file and inspect its contents. You will find some basic information of the data contained in the mzML file, e.g., the total number of spectra and peaks, the RT and m/z range, and how many MS1 and MS2 spectra the file contains.

Workflows are typically constructed to process a large number of files automatically. As a simple example, consider you would like to gather this information for more than

one file. We will now modify the workflow to compute the same information on three different files and then write the output files to a folder.

- We start from the previous workflow.
- First we need to replace our single input file with multiple files. Therefore we add the **Input Files** node from the category Community Nodes > GenericKnimeNodes > IO.
- To select the files we double-click on the **Input Files** node and click on Add. In the filesystem browser we select all three files from the directory **Example_Data** ▸ **Introduction** ▸ **datasets** ▸ **tiny**. And close the dialog with Ok.
- We now add two more nodes: the **ZipLoopStart** and the **ZipLoopEnd** node from the category Community Nodes > GenericKnimeNodes > Flow.
- Afterwards we connect the **Input Files** node to the first port of the **ZipLoopStart** node, the first port of the **ZipLoopStart** node to the **FileInfo** node, the first output port of the **FileInfo** node to the first input port of the **ZipLoopEnd** node, and the first output port of the **ZipLoopEnd** node to the **Output Folder** node (NOT to the **Output File**). The complete workflow is shown in Figure 5
- The workflow is already complete. Simply execute the workflow and inspect the output as before.

In case you had trouble to understand what ZipLoopStart and ZipLoopEnd do - here is a brief explanation:

- The **Input Files** node passes a list of files to the **ZipLoopStart** node.
- The **ZipLoopStart** node takes the files as input, but passes the single files sequentially (that is: one after the other) to the next node.
- The **ZipLoopEnd** collects the single files that arrive at its input port. After all files have been processed, the collected files are passed again as file list to the next node that follows.

Figure 5: A minimal workflow calling FileInfo on multiple files in a loop.

2.3.8 Advanced topic: Meta nodes

Workflows can get rather complex and may contain dozens or even hundreds of nodes. KNIME provides a simple way to improve handling and clarity of large workflows:

Meta Nodes allow to bundle several nodes into a single **Meta Node**.

Task

Select multiple nodes (e.g. all nodes of the ZipLoop including the start and end node). To select a set of nodes, draw a rectangle around them with the left mouse button or hold Ctrl to add/remove single nodes from the selection. Open the context menu (right-click on a node in the selection) and select Collapse into Meta Node. Enter a caption for the **Meta Node**. The previously selected nodes are now contained in the **Meta Node**. Double clicking on the **Meta Node** will display the contained nodes in a new tab window.

Task

Undo the packaging. First select the **Meta Node**, open the context menu (right-click) and select Expand Meta Node.

2.3.9 Advanced topic: R integration

KNIME provides a large number of nodes for a wide range of statistical analysis, machine learning, data processing and visualization. Still, more recent statistical analysis methods, specialized visualizations or cutting edge algorithms may not be covered in KNIME. In order to expand its capabilities beyond the readily available nodes, external scripting

languages can be integrated. In this tutorial, we primarily use scripts of the powerful statistical computing language R. Note that this part is considered advanced and might be difficult to follow if you are not familiar with R. In this case you might skip this part.

R View (Table) allows to seamlessly include R scripts into KNIME. We will demonstrate on a minimal example how such a script is integrated.

Task

First we need some example data in KNIME, which we will generate using the **Data Generator** node. You can keep the default settings and execute the node. The table contains 4 columns, each containing random coordinates and one column containing a cluster number (Cluster_0 to Cluster_3). Now place a **R View (Table)** node into the workflow and connect the upper output port of the **Data Generator** node to the input of the **R View (Table)** node. Right-click and configure the node.

If you get an error message like "Execute failed: R_HOME does not contain a folder with name 'bin'.": please change the R settings in the preferences. To do so open File > Preferences > KNIME > R and enter the path to your R installation (the folder that contains the bin directory).

If R is correctly recognized we can start writing an R script. Consider that we are interested in plotting the first and second coordinates and color them according to their cluster number. In R this can be done in a single line.

In the **R View (Table)** text editor, enter the following code:

```
plot(x=knime.in$Universe_0_0, y=knime.in$Universe_0_1, main="Plotting column ↩
    Universe_0_0 vs. Universe_0_1", col=knime.in$"Cluster Membership")
```

Explanation: The table provided as input to the **R View (Table)** node is available as R **data.frame** with name **knime.in**. Columns (also listed on the left side of the R View window) can be accessed in the usual R way by first specifying the **data.frame** name and then the column name (e.g. **knime.in$Universe_0_0**). **plot** is the plotting function we use to generate the image. We tell it to use the data in column **Universe_0_0** of the

dataframe object **knime.in** (denoted as **knime.in$Universe_0_1**) as x-coordinate and the other column **knime.in$Universe_0_1** as y-coordinate in the plot. **main** is simply the main title of the plot and **col** the column that is used to determine the color (in this case it is the **Cluster Membership** column).

Now press the Eval script and Show plot buttons.

> Note: Note that we needed to put some extra quotes around **Cluster Membership**. If we omit those, R would interpret the column name only up to the first space (**knime.in$Cluster**) which is not present in the table and leads to an error. Quotes are regularly needed if column names contain spaces, tabs or other special characters like $ itself.

3 Label-free quantification

3.1 Introduction

In this chapter, we will build a workflow with OpenMS / KNIME to quantify a label-free experiment. Label-free quantification is a method aiming to compare the relative amounts of proteins or peptides in two or more samples. We will start from the minimal workflow of the last chapter and, step-by-step, build a label-free quantitation workflow.

3.2 Peptide Identification

As a start, we will extend the minimal workflow so that it performs a peptide identification using the OMSSA [6] search engine. Since OpenMS version 1.10, OMSSA is included in the OpenMS installation, so you do not need to download and install it yourself.

- Let's start by replacing the input files in our **Input Files** node by the three mzML files in **Example_Data ▸ Labelfree ▸ datasets ▸ lfq_spikein_dilution_1-3.mzML**. This is a reduced toy dataset where each of the three runs contains a constant background of S. pyogenes peptides as well as human spike-in peptides in different concentrations. [7]
- Instead of FileInfo, we want to perform OMSSA identification, so we simply replace the **FileInfo** node with the **OMSSAAdapter** node Community Nodes › OpenMS › Identification, and we are almost done. Just make sure you have connected the **ZipLoopStart** node with the **in** port of the **OMSSAAdapter** node.
- OMSSA, like most mass spectrometry identification engines, relies on searching the input spectra against sequence databases. Thus, we need to introduce a search database input. As we want to use the same search database for all of our input files, we can just add a single **Input File** node to the workflow and connect it directly with the **OMSSAAdapter database** port. KNIME will automatically reuse this Input node each time a new ZipLoop iteration is started. In order to specify the database, select **Example_Data ▸ Labelfree ▸ databases ▸ s_pyo_sf370_potato_human_target_decoy_with_contaminants.fasta**, and we have a very basic peptide identification workflow.

> Note: You might also want to save your new identification workflow under a different name. Have a look at Section 2.3.6 for information on how to create copies of workflows.

- The result of a single OMSSA run is basically a number of peptide-spectrum-matches (PSM) with a score each, and these will be stored in an idXML file. Now we can run the pipeline and after execution is finished, we can have a first look at the results: just open the input files folder with a file browser and from there open an mzML file in **TOPPView**.
- Here, you can annotate this spectrum data file with the peptide identification results. Choose Tools > Annotate with identification from the menu and select the idXML file that **OMSSAAdapter** generated (it is located within the output directory that you specified when starting the pipeline).
- On the right, select the tab Identification view. Using this view, you can see all identified peptides and browse the corresponding MS2 spectra.

> Note: Opening the output file of **OMSSAAdapter** (the idXML file) directly is also possible, but the direct visualization of an idXML file is less useful.

- The search results stored in the idXML file can also be read back into a KNIME table for inspection and subsequent analyses: Add a **TextExporter** Community Nodes > OpenMS > File Handling node to your workflow and connect the output port of your **OMSSAAdapter** (the same port your **ZipLoopEnd** is connected to) to its input port. This tool will convert the idXML file to a more human-readable text file which can also be read into a KNIME table using the **IDTextReader** node. Add an **IDTextReader** node Community Nodes > OpenMS > Conversion after **TextExporter** and execute it. Now you can right-click **IDTextReader** and select ID Table to browse your peptide identifications.
- From here, you can use all the tools KNIME offers for analyzing the data in this table. As a simple example, you could add a **Histogram** Data Views node after **IDTextReader**, double-click it, select peptide_charge as binning column, hit OK, and execute it.

Right-clicking and selecting View: Histogram view will open a plot showing the charge state distribution of your identifications.

In the next step, we will tweak the parameters of OMSSA to better reflect the instrument's accuracy. Also, we will extend our pipeline with a false discovery rate (FDR) filter to retain only those identifications that will yield an FDR of < 1 %.

- Open the configuration dialog of **OMSSAAdapter**. The dataset was recorded using an LTQ Orbitrap XL mass spectrometer, so we can set the precursor mass tolerance to a smaller value, say 10 ppm. Set precursor_mass_tolerance to 10 and precursor_mass_tolerance_unit_ppm to true.

  > Note: Whenever you change the configuration of a node, the node as well as all its successors will be reset to the Configured state.

- Set max_precursor_charge to 5, in order to also search for peptides with charges up to 5.

- Add Carbamidomethyl (C) as fixed modification and Oxidation (M) as variable modification.

  > Note: To add a modification click on the empty value field in the configuration dialog to open the list editor dialog. In the new dialog click Add. Then select the newly added modification to open the drop down list where you can select the correct modification.

- A common step in analyis is to search not only against a regular protein database, but to also search against a decoy database for FDR estimation. The fasta file we used before already contains such a decoy database. For OpenMS to know which OMSSA PSM came from which part of the file (i.e. target versus decoy), we have to index the results. Therefore extend the workflow with a **PeptideIndexer** node Community Nodes » OpenMS » ID Processing. This node needs the idXML as input as well as the database file.

> Note: You can direct the files of an **Input File** node to more than just one destination port.

- The decoys in the database are prefixed with "REV_", so we have to set decoy_string to REV_ and prefix to true in the configuration dialog of **PeptideIndexer**.
- Now we can go for the FDR estimation, which the **FalseDiscoveryRate** node will calculate for us Community Nodes › OpenMS › ID Processing. As we have a combined search database and thus only one idXML per mzML we will only use the in port of the **FalseDiscoveryRate** node.
- In order to set the FDR level to 1%, we need an **IDFilter** node from Community Nodes › OpenMS › ID Processing. Configuring its parameter score $\rightarrow$ pep to 0.01 will do the trick. The FDR calculations (embedded in the idXML) from the **FalseDiscoveryRate** node will go into the in port of the **IDFilter** node.
- Execute your workflow and inspect the results using **IDTextReader** like you did before. How many peptides did you identify at this FDR threshold?

> Note: The finished identification workflow is now sufficiently complex that we might want to encapsulate it in a Meta node. For this, select all nodes inside the ZipLoop (including the **Input File** node) and right-click to select Collapse into Meta node and name it ID. Meta nodes are useful when you construct even larger workflows and want to keep an overview.

3.2.1 Bonus task: identification using several search engines

> Note: If you are ahead of the tutorial or later on, you can further improve your FDR identification workflow by a so-called consensus identification using several search engines. Otherwise, just continue with section 3.3.

It has become widely accepted that the parallel usage of different search engines can increase peptide identification rates in shotgun proteomics experiments. The ConsensusID algorithm is based on the calculation of posterior error probabilities (PEP) and a combination of the normalized scores by considering missing peptide sequences.

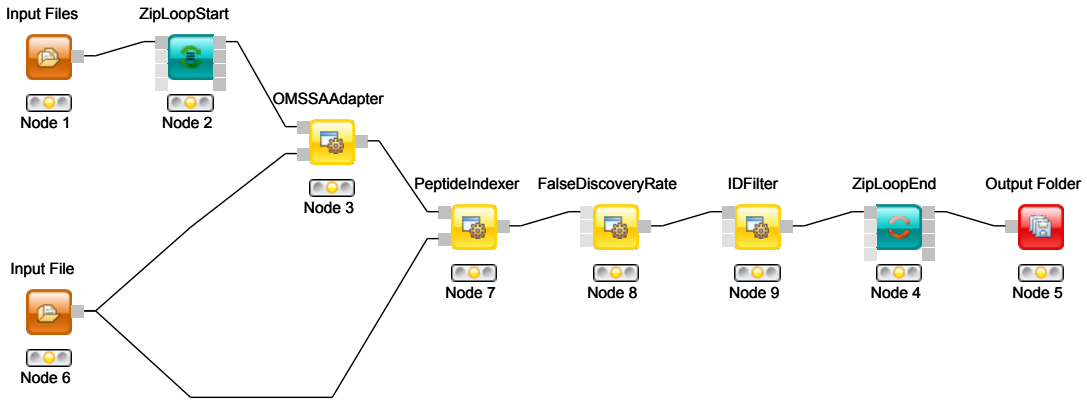

Figure 6: OMSSA ID pipeline including FDR filtering.

- Next to the **OMSSAAdapter** add a **XTandemAdapter** Community Nodes > OpenMS > Identification node and set its parameters and ports analogously to the **OMSSAAdapter**.

- To calculate the PEP, introduce each a **IDPosteriorErrorProbability** Community Nodes > OpenMS > ID Processing node to the output of each ID engine adapter node. This will calculate the PEP to each hit and output an updated idXML.

- To create a consensus, we must first merge these two files with a **FileMerger** node Community Nodes > GenericKnimeNodes > Flow so we can then merge the corresponding IDs with a **IDMerger** Community Nodes > OpenMS > File Handling.

- Now we can create a consensus identification with the **ConsensusID** Community Nodes > OpenMS > ID Processing node. We can connect this to the **PeptideIndexer** and go along with our existing FDR filtering.

> Note: By default, X!Tandem takes additional enzyme cutting rules into consideration (besides the specified tryptic digest). Thus you have to set PeptideIndexer's enzyme → specificity parameter to **semi** to accept X!Tandems semi tryptic identifications as well.

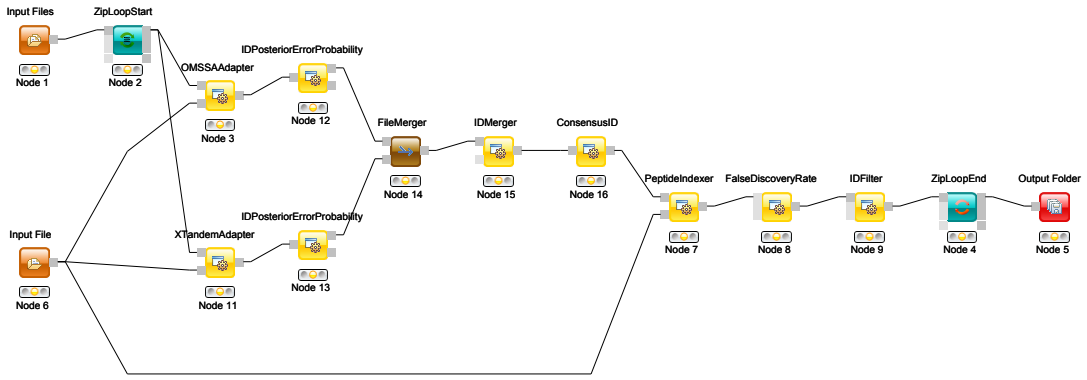

Figure 7: Complete consensus identification workflow.

3.3 Quantification

Now that we have successfully constructed a peptide identification pipeline, we can add quantification capabilities to our workflow.

- Add a **FeatureFinderCentroided** node Community Nodes > OpenMS > Quantitation which gets input from the first output port of the **ZipLoopStart** node. Also, add an **IDMapper** node Community Nodes > OpenMS > ID Processing which gets input from the **FeatureFinderCentroid** node and the ID Meta node (or **IDFilter** node if you haven't used the Meta node). The output of the **IDMapper** is then connected to the **ZipLoopEnd** node.

- **FeatureFinderCentroided** finds and quantifies peptide ion signals contained in the MS1 data. It reduces the entire signal, i.e., all peaks explained by one and the same peptide ion signal, to a single peak at the maximum of the chromatographic elution profile of the monoisotopic mass trace of this peptide ion and assigns an overall intensity.

- **FeatureFinderCentroided** produces a featureXML file as output, containing only quantitative information of so-far unidentified peptide signals. In order to annotate these with the corresponding ID information, we need the **IDMapper** node.

- Run your pipeline and inspect the results of the **IDMapper** node in TOPPView.

- In order to assess how well the feature finding worked, you can project the features contained in the featureXML file on the raw data contained in the mzML file. In

TOPPView choose File > Open file and select the mzML file corresponding to your featureXML file in 🗀 **Example_Data** ▸ **Labelfree** ▸ **datasets**. In the dialog that pops up, select Open in > New layer. Zoom in until you see boxes (found features) around the peptide signals in the raw data.

> Note: The RT range is very narrow. Thus, select the full RT range and zoom only into the m/z dimension by holding down CTRL (CMD on Mac) and repeatedly dragging a narrow box from the very left to the very right.

- You can see which features were annotated with a peptide identification by first selecting the featureXML file in the **Layers** window on the upper right side and then clicking on the icon with the letters A, B and C on the upper icon bar. Now, click on the small triangle next to that icon and select **Peptide identification**.

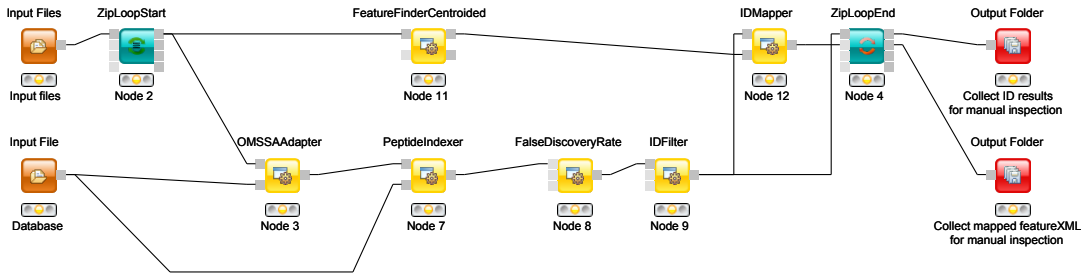

Figure 8: Extended workflow featuring peptide identification and quantification.

3.4 Combining quantitative information across several label-free experiments

So far, we successfully performed peptide identification as well as quantification on individual LC-MS runs. For differential label-free analyses, however, we need to identify and quantify corresponding signals in different experiments and link them together to compare their intensities. Thus, we will now run our pipeline on all three available input files and extend it a bit further, so that it is able to find and link features across several runs.

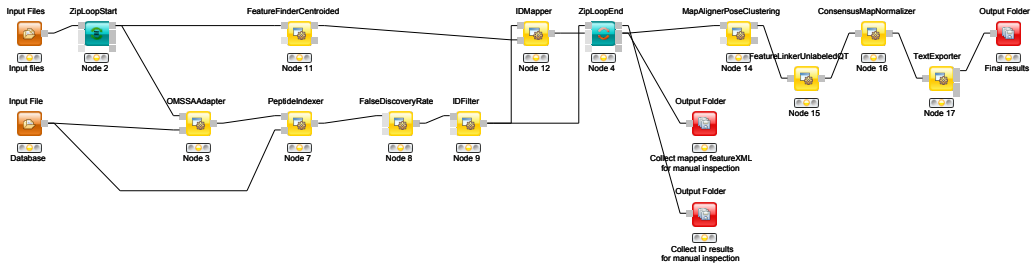

Figure 9: Complete identification and label-free quantification workflow.

- To find features across several maps, we first have to align them to correct for retention time shifts between the different label-free measurements. With the **MapAlignerPoseClustering** Community Nodes » OpenMS » Map Alignment, we can align corresponding peptide signals to each other as closely as possible by applying a transformation in the RT dimension.

  > Note: **MapAlignerPoseClustering** consumes several featureXML files and its output should still be several featureXML files containing the same features, but with the transformed RT values. In its configuration dialog, make sure that OutputTypes is set to featureXML.

- With the **FeatureLinkerUnlabeledQT** node Community Nodes » OpenMS » Map Alignment, we can then perform the actual linking of corresponding features. Its output is a consensusXML file containing linked groups of corresponding features across the different experiments.

- Since the overall intensities can vary a lot between different measurements (for example, because the amount of injected analytes was different), we apply the **ConsensusMapNormalizer** Community Nodes » OpenMS » Map Alignment as a last processing step. Configure its parameters with setting algorithm_type to **median**. It will then normalize the maps in such a way that the median intensity of all input maps is equal.

- Finally, we export the resulting normalized consensusXML file to a csv format using **TextExporter**. Connect its out port to a new **Output Folder** node.

> Note: You can specify the desired column separation character in the parameter settings (by default, it is set to " " (a space)). The output file of **TextExporter** can also be opened with external tools, e.g., Microsoft Excel, for downstream statistical analyses.

3.4.1 Basic data analysis in KNIME

For downstream analysis of the quantification results within the KNIME environment, you can use the **ConsensusTextReader** node Community Nodes > OpenMS > Conversion instead of the **Output Folder** node to convert the output into a KNIME table (indicated by a triangle as output port). After running the node you can view the KNIME table by right clicking on the **ConsensusTextReader** and selecting Consensus Table. Every row in this table corresponds to a so-called consensus feature, i.e., a peptide signal quantified across several runs. The first couple of columns describe the consensus feature as a whole (average RT and m/z across the maps, charge, etc.). The remaining columns describe the exact positions and intensities of the quantified features separately for all input samples (e.g., intensity_0 is the intensity of the feature in the first input file). The last 11 columns contain information on peptide identification.

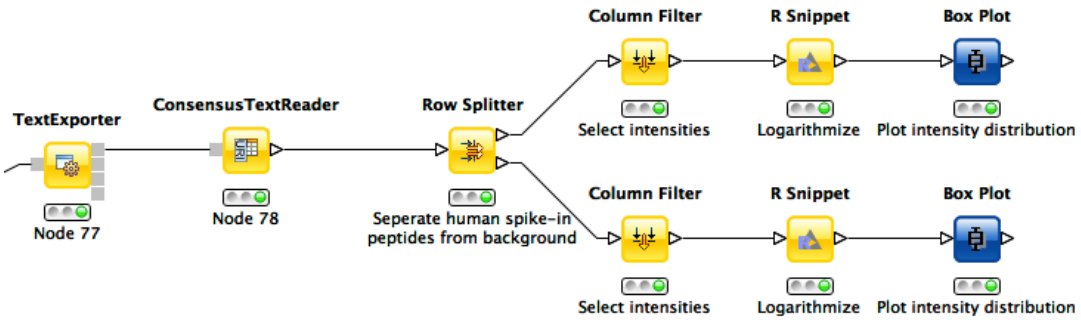

Figure 10: Simple KNIME data analysis example for LFQ.

- Now, let's say we want to plot the log intensity distributions of the human spike-in peptides for all input files. In addition, we will plot the intensity distributions of the background peptides.

- As shown in Fig. 10, add a **Row Splitter** node Data Manipulation > Row > Filter after **ConsensusTextReader**. Double-click it to configure. The human spike-in peptides have accessions starting with "hum". Thus, set the column to test to accessions, select pattern matching as matching criterion, enter hum* into the corresponding text field, and check the contains wild cards box. Press OK and execute the node.

- **Row Splitter** produces two output tables: the first one contains all rows from the input table matching the filter criterion, and the second table contains all other rows. You can inspect the tables by right-clicking and selecting Filtered and Filtered Out. The former table should now contain only peptides with a human accession, whereas the latter should contain all remaining peptides (including unidentified ones).

- Now, since we only want to plot intensities, we can add a **Column Filter** node Data Manipulation > Column > Filter, connect its input port to the Filtered output port of the **Row Filter**, and open its configuration dialog. We could either manually select the columns we want to keep, or, more elegantly, select Wildcard/Regex Selection and enter intensity_? as the pattern. KNIME will interactively show you which columns your pattern applies to while you're typing.

- Since we want to plot log intensities, we will now compute the log of all intensity values in our table. The easiest way to do this in KNIME is a small piece of R code. Add an **R Snippet** node R after **Column Filter** and double-click to configure. In the R Script text editor, enter the following code:

```
x <- knime.in       # store copy of input table in x
x[x == 0] <- NA     # replace all zeros by NA (= missing value)
x <- log10(x)       # compute log of all values
knime.out <- x      # write result to output table
```

- Now we are ready to plot! Add a **Box Plot** node Data Views after the **R Snippet** node, execute it, and open its view. If everything went well, you should see a significant fold change of your human peptide intensities across the three runs.

- In order to verify that the concentration of background peptides is constant in all

three runs, you can just copy and paste the three nodes after **Row Splitter** and connect the duplicated **Column Filter** to the second output port (Filtered Out) of **Row Splitter**, as shown in Fig. 10. Execute and open the view of your second **Box Plot**.

- That's it! You have constructed an entire identification and label-free quantification workflow including a simple data analysis using KNIME!

> Note: For further inspiration you might want to take a look at the more advanced KNIME data analysis examples in the metabolomics tutorial.

4 Protein Inference

In the last chapter, we have successfully quantified peptides in a label-free experiment. As a next step, we will further extend this label-free quantification workflow by protein inference and protein quantification capabilities. This workflow uses some of the more advanced concepts of KNIME, as well as a few more nodes containing R code. For these reasons, you will not have to build it yourself. Instead, we have already prepared and copied this workflow to the USB sticks. Just import 🗀 **Workflows > Protein Inference > protein_inference.zip** into KNIME via File > Import KNIME workflow > Select archive file and double-click the imported workflow in order to open it.

Before you can execute the workflow, you again have to correct the locations of the files in the **Input Files** nodes (don't forget the one for the FASTA database inside the "ID" meta node). Try and run your workflow.

4.1 Extending the LFQ workflow by protein inference and quantification

We have made the following changes compared to the original label-free quantification workflow from the last chapter:

- First, we have added a **ProteinQuantifier** node and connected its input port to the output port of **ConsensusMapNormalizer**.
- This already enables protein quantification. ProteinQuantifier quantifies peptides by summarizing over all observed charge states and proteins by summarizing over their quantified peptides. It stores two output files, one for the quantified peptides and one for the proteins.
- In this example, we consider only the protein quantification output file, which is written to the first output port of **ProteinQuantifier**
- Because there is no dedicated node in KNIME to read back the ProteinQuantifier output file format into a KNIME table, we have to use a workaround. Here, we have added an additional **URI Port to Variable** node which converts the name of the output file to a so-called "flow variable" in KNIME. This variable is passed on to the next node **CSV Reader**, where it is used to specify the name of the input file to be read.

If you double-click on **CSV Reader**, you will see that the text field, where you usually enter the location of the CSV file to be read, is greyed out. Instead, the flow variable is used to specify the location, as indicated by the small green button with the "v=?" label on the right.

- The table containing the **ProteinQuantifier** results is filtered one more time in order to remove decoy proteins. You can have a look at the final list of quantified protein groups by right-clicking the **Row Filter** and selecting Filtered.
- By default, i.e., when the second input port protein_groups is not used, ProteinQuantifier quantifies proteins using only the unique peptides, which usually results in rather low numbers of quantified proteins.
- In this example, however, we have performed protein inference using Fido and used the resulting protein grouping information to also quantify indistinguishable proteins
- As a prerequisite for using FidoAdapter, we have added an **IDPosteriorErrorProbability** node within the **ID** meta node, between OMSSAAdapter and PeptideIndexer. We have set its parameter prob_correct to true, so it computes posterior probabilities instead of posterior error probabilities (1 - PEP). These are stored in the resulting idXML file and later on used by the Fido algorithm.
- Next, we have added a third outgoing connection to our **ID** meta node and connected it to the second input port of **ZipLoopEnd**. Thus, KNIME will wait until all input files have been processed by the loop and then pass on the resulting list of idXML files to the subsequent **IDMerger** node, which merges all identifications from all idXML files into a single idXML file.
- Instead of the meta node **Protein inference with FidoAdapter**, we could have just used a **FidoAdapter** node Community Nodes » OpenMS » ID Processing. However, the meta node contains an additional subworkflow which, besides calling **FidoAdapter**, performs a statistical validation of the protein inference results using some of the more advanced KNIME nodes.

4.2 Statistical validation of protein inference results

In the following, we will explain the subworkflow contained in the **Protein inference with FidoAdapter** meta node.

4.2.1 Data preparation

For downstream analysis on the protein ID level in KNIME, it is again necessary to convert the idXML-file-format result generated from **FidoAdapter** into a KNIME table.

- By setting proteins_only to true in **TextExporter**, only the protein IDs are exported.
- As the built-in table file reader **IDTextReader** only reads peptide hits, we have to use **URI Port to Variable** which collects the URIs from a URI port object and puts them into variables.
- However, doing this will cause missing column information in the converted KNIME table. Open the **File Viewer** and check what each column stands for, filter away unused columns, such as sequence, coverage and rank in the **Column Filter** node. Add the column names manually in **Column Rename**.

4.2.2 ROC curve of protein ID

ROC Curves (Receiver Operating Characteristic curves) are graphical plots that visualize sensitivity (true-positive rate) against fall-out (false positive rate). They are often used to judge the quality of a discrimination method like e.g., peptide or protein identification engines. **ROC Curve** already provides the functionality of drawing ROC curves for binary classification problems. Before applying this node, an extra column with the class values (target and decoy proteins) has to be appended in the **Rule engine** node.

In protein or peptide identification, the ground-truth (i.e., which target identifications are true, which are false) is usually not known. Instead, so called pseudo-ROC Curves are regularly used to plot the number of target proteins against the false discovery rate (FDR). The FDR is approximated by using the target-decoy estimate in order to distinguish true IDs from false IDs by separating target IDs from decoy IDs.

4.2.3 Posterior probability and FDR of protein IDs

ROC curves illustrate the discriminative capability of the scores of IDs. In the case of protein identifications, Fido produces the posterior probability of each protein as the output score. However, a perfect score should not only be highly discriminative (distinguishing true from false IDs), it should also be "calibrated" (for probability indicating that all IDs with reported posterior probability scores of 95% should roughly of 5% probability be false. This implies that the estimated number of false positives can be computed as the sum of posterior error probability (= 1 - posterior probability), further an posterior probability estimated FDR is also possible to be computed. Therefore, we can plot calibration curves to help us visualize the quality of the score (when the score is interpreted as a probability as Fido does), by comparing how similar the target-decoy estimated FDR and the posterior probability estimated FDR are. Good results should show a close correspondence between these two measurements.

The calculation is done by using a simple R script in **R snippet**. First, the target decoy protein FDR is computed as the proportion of decoy proteins among all significant protein IDs. Then posterior probabilistic-driven FDR is estimated by the average of the posterior error probability of all significant protein IDs. Since FDR is the property for a group of protein IDs, we can also calculate a local property: the q-value of a certain protein ID by the minimum value of FDRs of any groups of protein IDs that contain this protein ID. We plot the protein ID results versus two different kinds of FDR estimates in **R View(Table)** (see Figure 12).

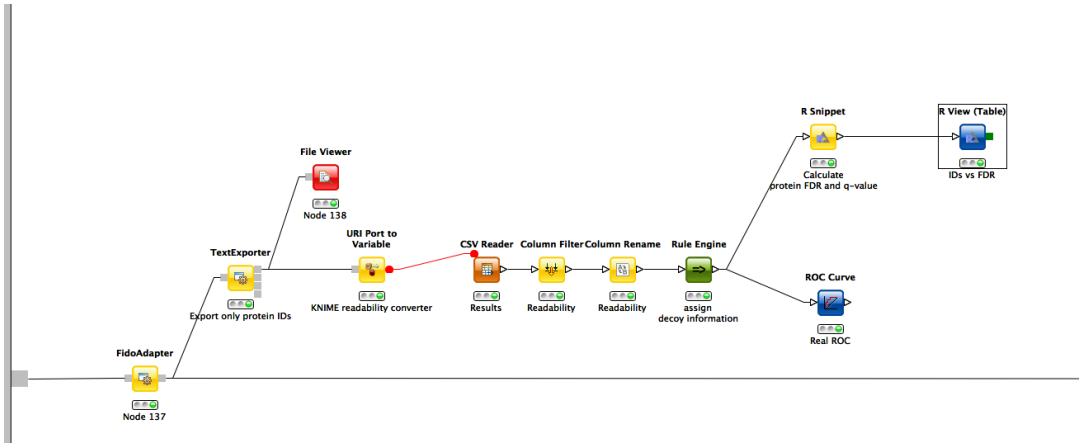

Figure 11: The workflow of statistical analysis of protein inference results

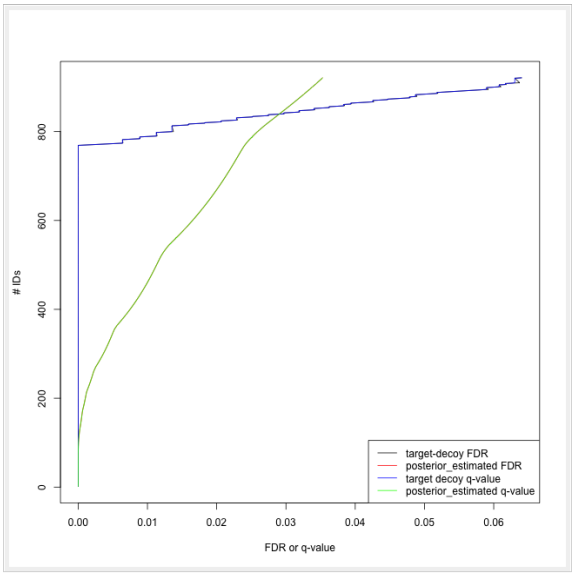

Figure 12: the pseudo-ROC Curve of protein IDs. The accumulated number of protein IDs is plotted on two kinds of scales: target-decoy protein FDR and Fido posterior probability estimated FDR. The largest value of posterior probability estimated FDR is already smaller than 0.04, this is because the posterior probability output from Fido is generally very high.

5 Metabolomics

5.1 Introduction

Quantitation and identification of chemical compounds are basic tasks in metabolomic studies. In this tutorial session we construct a UPLC-MS based, label-free quantitation and identification workflow. Following quantitation and identification we then perform statistical downstream analysis to detect quantitation values that differ significantly between two conditions. This approach can, for example, be used to detect biomarkers. Here, we use two spike-in conditions of a dilution series (0.5 mg/l and 10.0 mg/l, male blood background, measured in triplicates) comprising seven isotopically labeled compounds. The goal of this tutorial is to detect and quantify these differential spike-in compounds against the complex background.

5.2 Quantifying metabolites across several experiments

For the metabolite quantification we choose an approach similar to the one used for peptides, but this time based on the OpenMS **FeatureFinderMetabo** method. This feature finder again collects peak picked data into individual mass traces. The reason why we need a different feature finder for metabolites lies in the step after trace detection: the aggregation of isotopic traces belonging to the same compound ion into the same feature. Compared to peptides with their averagine model, small molecules have very different isotopic trace distributions. To group small molecule mass traces correctly, an aggregation model tailored to small molecules is thus needed.

- Create a new workflow called for instance ”Metabolomics”.
- Add a **Input Files** node and configure it with all mzML files from 🗀 **Example_Data ▸ Metabolomics ▸ datasets**.
- Add a **ZipLoopStart** node and connect the **Input Files** node to the first port of the **ZipLoopStart** node.
- Add a **FeatureFinderMetabo** node (from Community Nodes » OpenMS » Quantitation and connect the first output port of the **ZipLoopStart** to the **FeatureFinderMetabo**.

- For an optimal result adjust the following settings. Please note that some of these are advanced parameters.

parameter	value
algorithm → common → chrom_fwhm	8.0
algorithm → mtd → trace_termination_criterion	sample_rate
algorithm → mtd → min_trace_length	3.0
algorithm → mtd → max_trace_length	600.0
algorithm → epd → width_filtering	off

- Add a **ZipLoopEnd** node and connect the output of the **FeatureFinderMetabo** to the first port of the **ZipLoopEnd** node.

To facilitate the collection of features corresponding to the same compound ion across different samples, an alignment of the samples' feature maps along retention time is often helpful. In addition to local, small-scale elution differences, one can often see constant retention time shifts across large sections between samples. We can use linear transformations to correct for these large scale retention differences. This brings the majority of corresponding compound ions close to each other. Finding the correct corresponding ions is then faster and easier, as we don't have to search as far around individual features.

- After the **ZipLoopEnd** node add a **MapAlignerPoseClustering** node (Community Nodes > OpenMS > Map Alignment), set its Output Type to featureXML, and adjust the following settings

parameter	value
algorithm → max_num_peaks_considered	-1
algorithm → superimposer → mz_pair_max_distance	0.005
algorithm → superimposer → num_used_points	10000
algorithm → pairfinder → distance_RT → max_difference	20.0
algorithm → pairfinder → distance_MZ → max_difference	20.0
algorithm → pairfinder → distance_MZ → unit	ppm

The next step after retention time correction is the grouping of corresponding features in multiple samples. In contrast to the previous alignment, we assume no linear relations of features across samples. The used method is tolerant against local swaps in elution order.

- After the **MapAlignerPoseClustering** add a **FeatureLinkerUnlabeledQT** (Community Nodes > OpenMS > Map Alignment) and adjust the following settings

parameter	value
algorithm → distance_RT → max_difference	40.0
algorithm → distance_MZ → max_difference	20.0
algorithm → distance_MZ → unit	ppm

- After the **FeatureLinkerUnlabeledQT** add a **TextExporter** node (Community Nodes > OpenMS > File Handling).
- Add an **Output Folder** node and configure it with an output directory where you want to store the resulting files.
- Run the pipeline and inspect the output.

You should find a single, tab-separated file containing the information on where metabolites were found and with which intensities. You can also add **Output Folder** nodes at different stages of the workflow and inspect the intermediate results (e.g., identified metabolite features for each input map). The complete workflow can be seen in Figure 13. In the following section we will try to identify those metabolites.

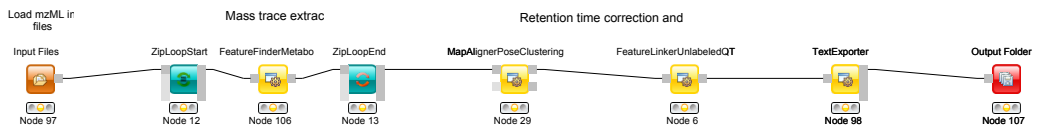

Figure 13: Label-free quantification workflow for metabolites

5.3 Identifying metabolites in LC-MS/MS samples

At the current state we found several metabolites in the individual maps but so far don't know what they are. To identify metabolites OpenMS provides multiple tools, including search by mass: the **AccurateMassSearch** node searches observed masses against the Human Metabolome Database (HMDB)[8, 9, 10]. We start with the workflow from the previous section (see Figure 13).

- Add a **FileConverter** node and connect the output of the **FeatureLinkerUnlabeledQT** to the incoming port.
- Open the Configure dialog of the **FileConverter** and select the tab "OutputTypes". In the drop down list for FileConverter.1.out select "featureXML".
- Add an **AccurateMassSearch** node and connect the output of the **FileConverter** to the first port of the **AccurateMassSearch**.
- Add four **Input File** nodes and configure them with the following files
  - 🗀 **Example_Data ▸ Metabolomics ▸ databases ▸ PositiveAdducts.tsv**
    This file specifies the list of adducts that are considered in the positive mode. Each line contains the formula and charge of an adduct separated by a semicolon (e.g. M+H;1+). The mass of the adduct is calculated automatically.
  - 🗀 **Example_Data ▸ Metabolomics ▸ databases ▸ NegativeAdducts.tsv**
    This file specifies the list of adducts that are considered in the negative mode analogous to the positive mode.
  - 🗀 **Example_Data ▸ Metabolomics ▸ databases ▸ HMDBMappingFile.tsv**
    This file contains information from a metabolite database in this case from HMDB. It has three (or more) tab-separated columns: mass, formula, and identifier(s). This allows for an efficient search by mass.
  - 🗀 **Example_Data ▸ Metabolomics ▸ databases ▸ HMDB2StructMapping.tsv**
    This file contains additional information about the identifiers in the mapping file. It has four tab-separated columns that contain the identifier, name, SMILES, and INCHI. These will be included in the result file. The identifiers in this file must match the identifiers in the HMDBMappingFile.tsv.

- In the same order as they are given above connect them to the remaining input ports of the **AccurateMassSearch** node.
- Add an **Output Folder** node and connect the first output port of the **AccurateMassSearch** node to the **Output Folder**.

The result of the **AccurateMassSearch** node is in the mzTab format [11] so you can easily open it in a text editor or import it into Excel or KNIME, which we will do in the next section. The complete workflow from this section is shown in Figure 14.

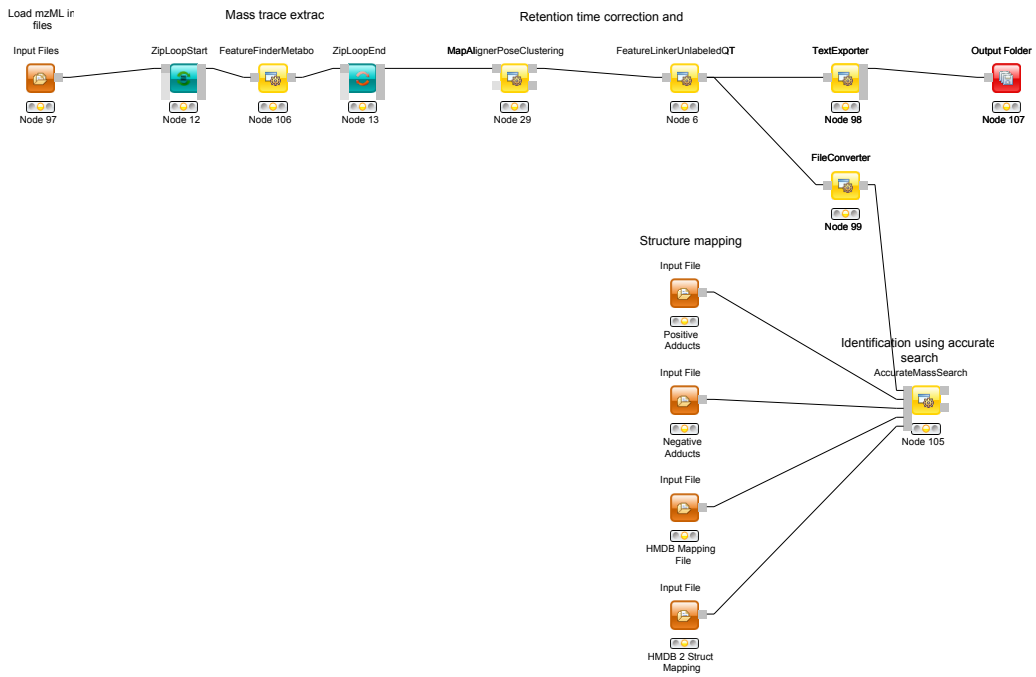

Figure 14: Label-free quantification and identification workflow for metabolites

5.4 Convert your data into a KNIME table

The result from the **TextExporter** node as well as the result from the **AccurateMassSearch** node are files while standard KNIME nodes display and processes only KNIME tables. To convert these files into KNIME tables we need two different nodes. For the **AccurateMassSearch** results we use the **MzTabReader** node (Community Nodes > OpenMS > Conversion > mzTab), for

the result of the **TextExporter** we use the **ConsensusTextReader** (Community Nodes › OpenMS › Conversion).

When executed, both nodes will import the OpenMS files and provide access to the data as KNIME tables. You can now easily combine both tables using the **Joiner** node (Data Manipulation › Column › Split & Combine) and configuring it to match the m/z and retention time values of the respective tables. The full workflow is shown in Figure 15.

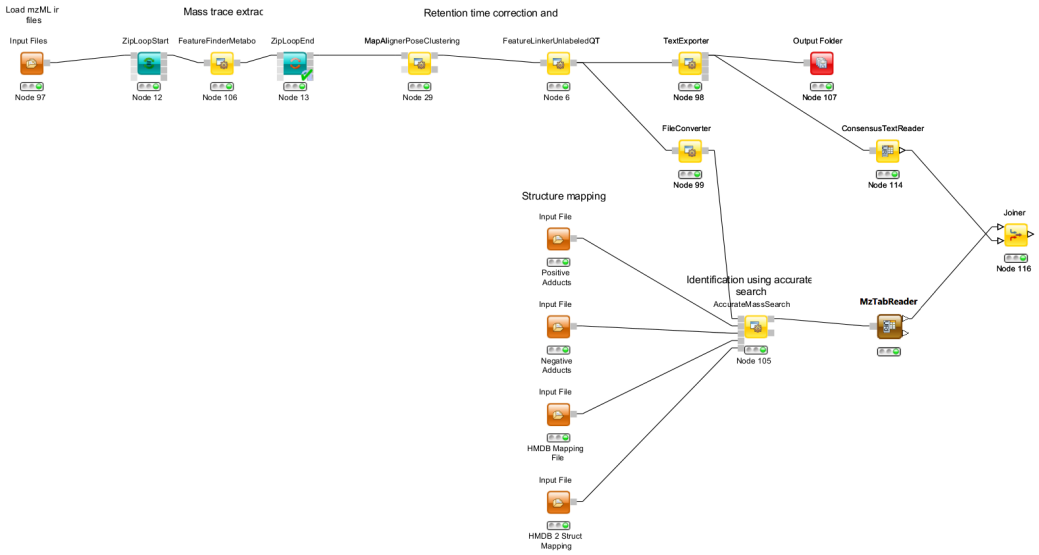

Figure 15: Label-free quantification and identification workflow for metabolites that loads the results into KNIME and joins the tables.

5.4.1 Bonus task: Visualizing data

Now that you have your data in KNIME you should try to get a feeling for the capabilities of KNIME.

Task

- [x] Check out the **Molecule Type Cast** node (Chemistry › Translators) together with subsequent cheminformatics nodes (e.g. **RDKit From Molecule** (Community Nodes › RDKit › Converters)) to render the structural formula contained in the result table.

Task

☑ Have a look at the **Column Filter** node to reduce the table to the interesting columns, e.g., only the Ids, chemical formula, and intensities.

Task

☑ Try to compute and visualize the m/z and retention time error of the different elements of the consensus features.

5.5 Downstream data analysis and reporting

In this part of the metabolomics session we take a look at more advanced downstream analysis and the use of the statistical programming language R. As laid out in the introduction we try to detect a set of spike-in compounds against a complex blood background. As there are many ways to perform this type of analysis we provide a complete workflow.

Task

☑ Import the workflow from 🗀 **Workflows ▸ metabolite_analysis.zip** in KNIME: File 〉 Import KNIME Workflow...

The section below will guide you in your understanding of the different parts of the workflow. Once you understood the workflow you should play around and be creative. Maybe create a novel visualization in KNIME or R? Do some more elaborate statistical analysis? Feel free to experiment and show us your results if you like. Note that some basic R knowledge is required to fully understand the processing in **R Snippet** nodes.

5.5.1 Data preparation ID

This part is analogous to what you did for the simple metabolomics pipeline.

5.5.2 Data preparation Quant

The first part is identical to what you did for the simple metabolomics pipeline. Additionally, we convert zero intensities into NA values and remove all rows that contain at least

one NA value from the analysis. We do this using a very simple **R Snippet** and subsequent **Missing Value filter** node.

Task

Inspect the **R Snippet** by double-clicking on it. The KNIME table that is passed to an **R Snippet** node is available in R as a data.frame named knime.in. The result of this node will be read from the data.frame knime.out after the script finishes. Try to understand and evaluate parts of the script (Eval Selection). In this dialog you can also print intermediary results using for example the R command head() or cat() to the Console pane.

5.5.3 Statistical analysis

After we linked features across all maps, we want to identify features that are significantly deregulated between the two conditions. We will first scale and normalize the data, then perform a t-test, and finally correct the obtained p-values for multiple testing using Benjamini-Hochberg. All of these steps will be carried out in individual **R Snippet** nodes.

- Double-click on the first **R Snippet** node labeled "log scaling" to open the **R Snippet** dialog. In the middle you will see a short R script that performs the log scaling. To perform the log scaling we use a so-called regular expression (grepl) to select all columns containing the intensities in the six maps and take the log_2 logarithm.
- The output of the log scaling node is also used to draw a boxplot that can be used to examine the structure of the data. Since we only want to plot the intensities in the different maps (and not m/z or rt) we first use a **Column Filter** node to keep only the columns that contain the intensities. We connect the resulting table to a **Box Plot** node which draws one box for every column in the input table. Right-click and select View: Box Plot.
- The median normalization is performed in a similar way to the log scaling. First we calculate the median intensity for each intensity column, then we subtract the median from every intensity.

- Open the **Box Plot** connected to the normalization node and compare it to the box plot connected to the log scaling node to examine the effect of the median normalization.
- To perform the t-test we defined the two groups we want to compare. Then we call the t-test for every consensus feature unless it has missing values. Finally we save the p-values and fold-changes in two new columns named p-value and FC.
- The **Numeric Row Splitter** is used to filter less interesting parts of the data. In this case we only keep columns where the fold-change is ≥ 2.
- We adjust the p-values for multiple testing using Benjamini-Hochberg and keep all consensus features with a q-value ≤ 0.01 (i.e. we target a false-discovery rate of 1%).

5.5.4 Interactive visualization

KNIME supports multiple nodes for interactive visualization with interrelated output. The nodes used in this part of the workflow exemplify this concept. They further demonstrate how figures with data dependent customization can be easily realized using basic KNIME nodes. Several simple operations are concatenated in order to enable an interactive volcano plot.

- We first log-transform fold changes and p-values in the **R Snippet** node. We then append columns noting interesting features (concerning fold change and p-value).
- With this information, we can use various Manager nodes (Data Views › Property) to emphasize interesting data points. The configuration dialogs allow us to select columns to change color, shape or size of data points dependent on the column values.
- The **Scatter Plot** node (Data Views) enables interactive visualization of the logarithmized values as a volcano plot: the log-transformed values can be chosen in the 'Column Selection' tab of the plot view. Data points can be selected in the plot and HiLited via the menu option. HiLiteing transfers to all other interactive nodes connected to the same data table. In our case, selection and HiLiteing will also occur in the **Interactive Table** node (Data Views).

- Output of the interactive table can then be filtered via the HiLite menu tab. For example, we could restrict shown rows to points HiLited in the volcano plot.

Task

Inspect the nodes of this section. Customize your visualization and possibly try to visualize other aspects of your data.

5.5.5 Advanced visualization

R Dependencies: This section requires that the R packages ggplot2 and ggbiplot are both installed. ggplot2 is part of the KNIME R Statistics Integration (Windows Binaries) installed during the tutorial setup section, ggbiplot however is not. In case that you use an R installation where one or both of them are not yet installed, add an **R Snippet** node and double-click to configure. In the R Script text editor, enter the following code:

```
#Include the next line if you also have to install ggplot2:
install.packages("ggplot2")
#Include the following lines to install ggbiplot:
install.packages("devtools")
library(devtools)
install_github("vqv/ggbiplot")
```

Press Eval script to execute the script.

Even though the basic capabilities for (interactive) plots in KNIME are valuable for initial data exploration, professional looking depiction of analysis results often relies on dedicated plotting libraries. The statistics language R supports the addition of a large variety of packages, including packages providing extensive plotting capabilities. This part of the workflow shows how to use R nodes in KNIME to visualize more advanced figures. Specifically, we make use of different plotting packages to realize heatmaps and a PCA plot.

- The used **RView (Table)** nodes combine the possibility to write R snippet code with visualization capabilities inside KNIME. Resulting images can be looked at in the output RView, or saved via the **Image Port Writer** node.

- The heatmap nodes make use of the gplots libary, which is by default part of the R Windows binaries for the KNIME 2.12 full installation. We again use regular expressions to extract all measured intensity columns for plotting. For clarity, feature names are only shown in the heatmap after filtering by fold changes.
- The remaining node performs principal component analysis and plots the first two principal components. Data points are colored by group membership to treated or control samples. The R commands to install the necessary plotting package are part of the node snippet. Removal of the comment character '#' and execution of the individual lines in the snippet node allows for package installation from inside KNIME.

5.5.6 Data preparation for Reporting

Following the identification, quantification and statistical analysis our data is merged and formatted for reporting. First we want to discard our normalized and logarithmized intensity values in favor of the original ones. To this end we first remove the intensity columns (**Column Filter**) and add the original intensities back (**Joiner**). Note that we use an Inner Join [1]. Combining ID and Quantification table into a single table is again achieved using a **Joiner** node.

Question

What happens if we use an Left Outer Join, Right Outer Join or Full Outer Join instead of the Inner Join?

Task

Inspect the output of the join operation after the Molecule Type Cast and RDKit molecular structure generation.

While all relevant information is now contained in our table the presentation could be improved. Currently, we have several rows corresponding to a single consensus feature (=linked feature) but with different, alternative identifications. It would be more

[1] Inner Join is a technical term that describes how database tables are merged.

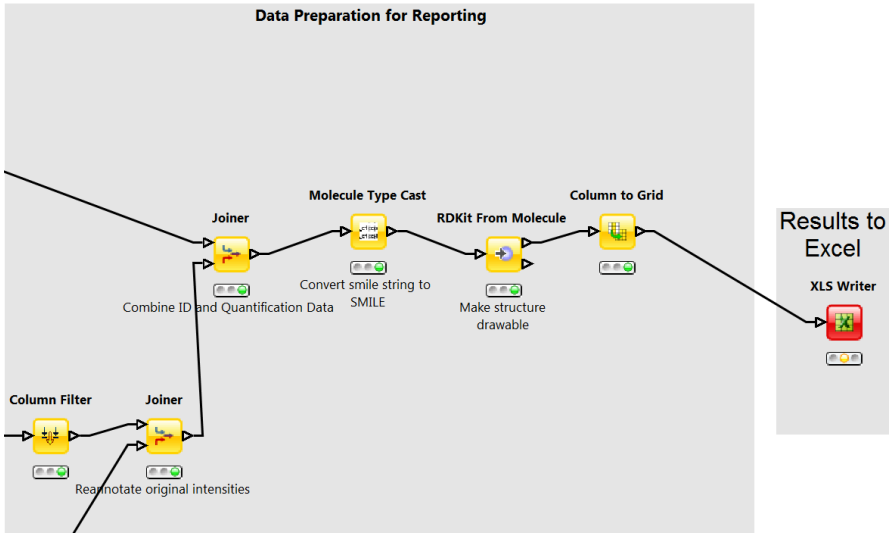

Figure 16: Data preparation for reporting

convenient to have only one row for each consensus feature with all accurate mass identifications added as additional columns. To achieve we use the **Column to Grid** node that flattens several rows with the same consensus number into a single one. Note that we have to specify the maximum number of columns in the grid so we set this to a large value (e.g. 100). We finally export the data to an Excel file (**XLS Writer**).

6 OpenSWATH

6.1 Introduction

OpenSWATH [12] is a module of OpenMS that allows analysis of LC-MS/MS DIA (data independent acquisition) data using the approach described by Gillet et al. [13]. The DIA approach described there uses 32 cycles to iterate through precursor ion windows from 400-426 Da to 1175-1201 Da and at each step acquires a complete, multiplexed fragment ion spectrum of all precursors present in that window. After 32 fragmentations (or 3.2 seconds), the cycle is restarted and the first window (400-426 Da) is fragmented again, thus delivering complete "snapshots" of all fragments of a specific window every 3.2 seconds.

The analysis approach described by Gillet et al. extracts ion traces of specific fragment ions from all MS2 spectra that have the same precursor isolation window, thus generating data that is very similar to SRM traces.

6.2 Installation of OpenSWATH

OpenSWATH has been fully integrated since OpenMS 1.10 (**http://open-ms.sourceforge.net** [3, 2, 14]).

6.3 Installation of mProphet

mProphet (**http://www.mprophet.org/**) [15] is available as standalone script in 🗀 **External_Tools ▸ mProphet**. R (**http://www.r-project.org/**) and the package MASS (**http://cran.r-project.org/web/packages/MASS/**) are further required to execute mProphet. Please obtain a version for either Windows, Mac or Linux directly from CRAN.

pyprophet, a much faster reimplementation of the mProphet algorithm is available from PyPI (**https://pypi.python.org/pypi/pyprophet/**). The usage of pyprophet instead of mProphet is suggested for large-scale applications, but the installation requires more dependencies and therefore, for this tutorial the application of mProphet is described.

6.4 Generating the Assay Library

6.4.1 Generating TraML from transition lists

OpenSWATH requires the assay libraries to be supplied in the TraML format [16]. To enable manual editing of transition lists, the TOPP tool **ConvertTSVToTraML** is available that uses tab separated files as input. Example datasets are provided in 🗀 **OpenSWATH ▸ assay**. Please note that the transition lists need to be named **.csv** or **.tsv**.

The header of the transition list contains the following variables (with example values in brackets):

PrecursorMz
: The mass-to-charge (m/z) of the precursor ion. (728.88)

ProductMz
: The mass-to-charge (m/z) of the product or fragment ion. (924.539)

Tr_recalibrated
: The normalized retention time (or iRT) [17] of the peptide. (26.5)

transition_name
: A unique identifier for the transition.
(AQUA4SWATH_HMLangeA_ADSTGTLVITDPTR(UniMod:267)/2_y8)

CE
: The collision energy that should be used for the acquisition. (27)
Optional (not used by OpenSWATH)

LibraryIntensity
: The relative intensity of the transition. (3305.3)

transition_group_id
: A unique identifier for the transition group.
(AQUA4SWATH_HMLangeA_ADSTGTLVITDPTR(UniMod:267)/2)

decoy
: A binary value whether the transition is target or decoy (target:0, decoy:1). (0)

PeptideSequence

The unmodified peptide sequence. (ADSTGTLVITDPTR)

ProteinName

A unique identifier for the protein. (AQUA4SWATH_HMLangeA)

Annotation

The fragment ion annotation. (y8)

Optional (not used by OpenSWATH)

FullUniModPeptideName

The peptide sequence with UniMod modifications. (ADSTGTLVITDPTR(UniMod:267))

MissedCleavages

The number of missed cleavages during enzymatic digestion. (0)

Optional (not used by OpenSWATH)

Replicates

The number of replicates. (0)

Optional (not used by OpenSWATH)

NrModifications

The number of modifications. (0)

Optional (not used by OpenSWATH)

PrecursorCharge

The precursor ion charge. (2)

GroupLabel

The stable isotope label. (light)

Optional (not used by OpenSWATH)

UniprotID

The Uniprot ID of the protein. ()

Optional (not used by OpenSWATH)

To convert transitions lists to TraML, use **ConvertTSVToTraML**:

Linux or Mac

On the Terminal:

```
ConvertTSVToTraML -in OpenSWATH_SGS_AssayLibrary.csv -out OpenSWATH_SGS_AssayLibrary.↩
    TraML
```

Windows

On the TOPP command line:

```
ConvertTSVToTraML.exe -in OpenSWATH_SGS_AssayLibrary.csv -out OpenSWATH_SGS_AssayLibrary↩
    .TraML
```

6.4.2 Appending decoys to a TraML

In addition to the target assays, OpenSWATH further requires decoy assays in the library which are later used for classification and error rate estimation. For the decoy generation it is crucial that the decoys represent the targets in a realistic but unnatural manner without interfering with the targets. The methods for decoy generation implemented in OpenSWATH include 'shuffle', 'pseudo-reverse', 'reverse' and 'shift'. To append decoys to a TraML, the TOPP tool **OpenSwathDecoyGenerator** can be used:

Linux or Mac

On the Terminal:

```
OpenSwathDecoyGenerator -in OpenSWATH_SGS_AssayLibrary.TraML -out ↩
    OpenSWATH_SGS_AssayLibrary_with_Decoys.TraML -method shuffle -append -↩
    exclude_similar -remove_unannotated
```

Windows

On the TOPP command line:

```
OpenSwathDecoyGenerator.exe -in OpenSWATH_SGS_AssayLibrary.TraML -out ↩
    OpenSWATH_SGS_AssayLibrary_with_Decoys.TraML -method shuffle -append -↩
    exclude_similar -remove_unannotated
```

The flag **-append** generates an output TraML with the complete set of decoy and target assays. The flag **-exclude_similar** is used to exclude decoys which are very similar to the target assays.

6.5 OpenSWATH KNIME

An example KNIME workflow for OpenSWATH is supplied in 🗀 **Workflows** (Figure ??). The example dataset can be used for this workflow (filenames in brackets):

1. Open 🗀 **Workflows ▸ OpenSWATH.zip** in KNIME: File 〉 Import KNIME Workflow...
2. Select the normalized retention time (iRT) assay library in TraML format by double-clicking on node Input File 〉 iRT Assay Library.
   (🗀 **OpenSWATH ▸ assay ▸ OpenSWATH_iRT_AssayLibrary.TraML**)
3. Select the SWATH MS data in mzML format as input by double-clicking on node Input File 〉 SWATH-MS files.
   (🗀 **OpenSWATH ▸ data ▸ split_napedro_L120420_010_SW-*.nf.pp.mzML**)
4. Select the target peptide assay library in TraML format as input by double-clicking on node Input Files 〉 Assay Library.
   (🗀 **OpenSWATH ▸ assay ▸ OpenSWATH_SGS_AssayLibrary.TraML**)
5. Set the output destination by double-clicking on node Output File.
6. Run the workflow.

The resulting output can be found at your selected path, which will be used as input for mProphet. Execute the script on the Terminal (Linux or Mac) or cmd.exe (Windows) in 🗀 **OpenSWATH ▸ result**:

```
R --slave --args bin_dir=../../../External_Tools/mProphet/ mquest=OpenSWATH_output.csv workflow=↩
    LABEL_FREE num_xval=5 run_log=FALSE write_classifier=1 write_all_pg=1 < ../../../↩
    External_Tools/mProphet/mProphet.R
```

The main output will be called
OpenSWATH ▸ result ▸ mProphet_all_peakgroups.xls
with statistical information available in
OpenSWATH ▸ result ▸ mProphet.pdf.

Please note that due to the semi-supervised machine learning approach of mProphet the results differ slightly when mProphet is executed several times.

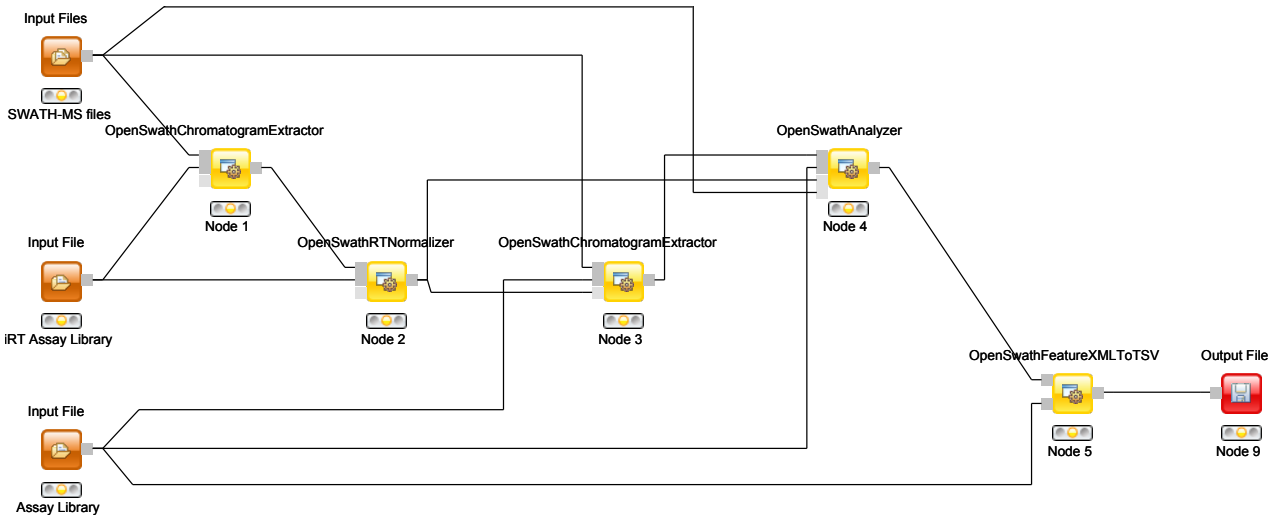

Figure 17: OpenSWATH KNIME Workflow.

6.6 From the example dataset to real-life applications

The sample dataset used in this tutorial is part of the larger SWATH MS Gold Standard (SGS) dataset which is described in the publication of Roest et al. [12]. It contains one of 90 SWATH-MS runs with significant data reduction (peak picking of the raw, profile data) to make file transfer and working with it easier. Usually SWATH-MS datasets are huge with several gigabyte per run. Especially when complex samples in combination with large assay libraries are analyzed, the TOPP tool based workflow requires a lot of computational resources. For this reason, an integrated tool (**OpenSwathWorkflow**) has been developed, combining all the steps shown in the KNIME-Workflow into a single executable. It is shipped with OpenMS 2.0.0. Instructions on how to use this tool can be found on **http://www.openswath.org**.

7 An introduction to pyOpenMS

7.1 Introduction

pyOpenMS provides Python bindings for a large part of the OpenMS library for mass spectrometry based proteomics. It thus provides access to a feature-rich, open-source algorithm library for mass-spectrometry based proteomics analysis. These Python bindings allow raw access to the data-structures and algorithms implemented in OpenMS, specifically those for file access (mzXML, mzML, TraML, mzIdentML among others), basic signal processing (smoothing, filtering, de-isotoping and peak-picking) and complex data analysis (including label-free, SILAC, iTRAQ and SWATH analysis tools).

pyOpenMS is integrated into OpenMS starting from version 1.11. This tutorial is addressed to people already familiar with Python. If you are new to Python, we suggest to start with a Python tutorial (**http://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for_Python_2.6**).

7.2 Installation

7.2.1 Windows

1. Install Python 2.7 (**http://www.python.org/download/**)
2. Install NumPy (**http://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy**)
3. Install setuptools (**https://pypi.python.org/pypi/setuptools**)
4. On the command line:

```
easy_install pyopenms
```

7.2.2 Mac OS X 10.10

1. On the Terminal:

```
sudo easy_install pyopenms
```

7.2.3 Linux

1. Install Python 2.6 or 2.7 (Debian: python-dev, RedHat: python-devel)
2. Install NumPy (Debian / RedHat: python-numpy)
3. Install setuptools (Debian / RedHat: python-setuptools)
4. On the Terminal:

```
easy_install pyopenms
```

7.3 Build instructions

Instructions on how to build pyOpenMS can be found online (**http://ftp.mi.fu-berlin.de/OpenMS/documentation/html/pyOpenMS.html**).

7.4 Your first pyOpenMS tool: pyOpenSwathFeatureXMLToTSV

The first tool that you are going to re-implement is a TOPP tool called OpenSwathFeatureXMLToTSV. Take a look at the help of the tool:

```
OpenSwathFeatureXMLToTSV --help

OpenSwathFeatureXMLToTSV -- Converts a featureXML to a mProphet tsv.
Version: 2.0.0 Apr 11 2015, 02:02:58, Revision: 66a7739

Usage:
  OpenSwathFeatureXMLToTSV <options>

Options (mandatory options marked with '*'):
  -in <files>*                    Input files separated by blank (valid formats: 'featureXML')
  -tr <file>*                     TraML transition file (valid formats: 'traML')
  -out <file>*                    Tsv output file (mProphet compatible) (valid formats: 'csv')
  -short_format                   Whether to write short (one peptide per line) or long format (↩
      one transition per line).
```

```
  -best_scoring_peptide <varname>  If only the best scoring feature per peptide should be printed↩
      , give the variable name

Common TOPP options:
  -ini <file>                      Use the given TOPP INI file
  -threads <n>                     Sets the number of threads allowed to be used by the TOPP tool↩
       (default: '1')
  -write_ini <file>                Writes the default configuration file
  --help                           Shows options
  --helphelp                       Shows all options (including advanced)
```

OpenSwathFeatureXMLToTSV converts a featureXML file to a tab-separated value text file. This example will teach you how to use pyOpenMS in combination with Python to implement such a tool very quickly.

7.4.1 Basics

The first task that your tool needs to be able to do is to read the parameters from the command line and provide a main routine. This is all standard Python and no pyOpenMS is needed so far:

```
#!/usr/bin/env python
import sys

def main(options):

    # test parameter handling
    print options.infile, options.traml_in, options.outfile

def handle_args():
    import argparse

    usage = ""
    usage += "\nOpenSwathFeatureXMLToTSV -- Converts a featureXML to a mProphet tsv."

    parser = argparse.ArgumentParser(description = usage )
    parser.add_argument('-in', dest='infile', help = 'An input file containing features ↩
        featureXML]')
    parser.add_argument('-tr', dest='traml_in', help='An input file containing the transitions ↩
        TraML]')
    parser.add_argument('-out', dest='outfile', help='Output mProphet TSV file [tsv]')

    args = parser.parse_args(sys.argv[1:])
    return args
```

```
if __name__ == '__main__':
    options = handle_args()
    main(options)
```

Execute this code in the example script

./pyOpenMS/OpenSwathFeatureXMLToTSV_basics.py

```
python OpenSwathFeatureXMLToTSV_basics.py --help
usage: OpenSwathFeatureXMLToTSV_basics.py [-h] [-in INFILE] [-tr TRAML_IN]
                                          [-out OUTFILE]

OpenSwathFeatureXMLToTSV -- Converts a featureXML to a mProphet tsv.

optional arguments:
  -h, --help    show this help message and exit
  -in INFILE    An input file containing features [featureXML]
  -tr TRAML_IN  An input file containing the transitions [TraML]
  -out OUTFILE  Output mProphet TSV file [tsv]
```

```
python OpenSwathFeatureXMLToTSV_basics.py -in data/example.featureXML -tr assay/↩
    OpenSWATH_SGS_AssayLibrary.TraML -out example.tsv
data/example.featureXML assay/OpenSWATH_SGS_AssayLibrary.TraML example.tsv
```

The parameters are being read from the command line by the function handle_args() and given to the main() function of the script, which prints the different variables.

7.4.2 Loading data structures with pyOpenMS

Now we're going to import the pyOpenMS module with **import pyopenms** in the header of the script and load the featureXML:

```
#!/usr/bin/env python
import pyopenms
import sys

def main(options):
    #  load featureXML
    features = pyopenms.FeatureMap()
    fh = pyopenms.FileHandler()
```

```
    fh.loadFeatures(options.infile, features)
    keys = []
    features[0].getKeys(keys)
    print keys

def handle_args():
    import argparse

    usage = ""
    usage += "\nOpenSwathFeatureXMLToTSV -- Converts a featureXML to a mProphet tsv."

    parser = argparse.ArgumentParser(description = usage )
    parser.add_argument('-in', dest='infile', help = 'An input file containing features [↩
        featureXML]')
    parser.add_argument('-tr', dest='traml_in', help='An input file containing the transitions [↩
        TraML]')
    parser.add_argument('-out', dest='outfile', help='Output mProphet TSV file [tsv]')

    args = parser.parse_args(sys.argv[1:])
    return args

if __name__ == '__main__':
    options = handle_args()
    main(options)
```

The function pyopenms.FeatureMap() initializes an OpenMS FeatureMap data structure. The function pyopenms.FileHandler() prepares a filehandler with the variable name fh and fh.loadFeatures(options.infile, features) takes the filename and imports the featureXML into the FeatureMap data structure.

In the next step, we're accessing the keys using the function getKeys() and printing them to stdout:

```
python OpenSwathFeatureXMLToTSV_datastructures1.py -in data/example.featureXML -tr assay/↩
    OpenSWATH_SGS_AssayLibrary.TraML -out example.tsv
['PeptideRef', 'leftWidth', 'rightWidth', 'total_xic', 'peak_apices_sum', 'var_xcorr_coelution', ↩
    'var_xcorr_coelution_weighted ', 'var_xcorr_shape', 'var_xcorr_shape_weighted', '↩
    var_library_corr', 'var_library_rmsd', 'var_library_manhattan', 'var_library_dotprod', '↩
    delta_rt', 'assay_rt', 'norm_RT', 'rt_score', 'var_norm_rt_score', 'var_intensity_score', '↩
    nr_peaks', 'sn_ratio', 'var_log_sn_score', 'var_elution_model_fit_score', '↩
    xx_lda_prelim_score', 'var_isotope_correlation_score', 'var_isotope_overlap_score', '↩
    var_massdev_score', 'var_massdev_score_weighted', 'var_bseries_score', 'var_yseries_score', ↩
    'var_dotprod_score', 'var_manhatt_score', 'main_var_xx_swath_prelim_score', 'PrecursorMZ', '↩
    xx_swath_prelim_score']
```

In the next task, please load the TraML into an OpenMS TargetedExperiment data

structure, analogously to the featureXML. You might want to consult the pyOpenMS manual (**http://proteomics.ethz.ch/pyOpenMS_Manual.pdf**), which provides an overview of all functionality. If you have trouble reading the TraML, search for TraMLFile(). If you can't solve the task, take a look at OpenSwathFeatureXMLToTSV_datastructures2.py

7.4.3 Converting data in the featureXML to a TSV

Now that all data structures are populated, we need to access the data using the provided API and store it in something that is directly accessible from Python. We prepared two functions for you: get_header() & convert_to_row():

```
def get_header(features):
    keys = []
    features[0].getKeys(keys)
    header = [
        "transition_group_id",
        "run_id",
        "filename",
        "RT",
        "id",
        "Sequence" ,
        "FullPeptideName",
        "Charge",
        "m/z",
        "Intensity",
        "ProteinName",
        "decoy"]
    header.extend(keys)
    return header
```

get_header() takes as input a FeatureMap and uses the getKeys() function that you have seen before to extend a predefined header list based on the contents of the FeatureMap. The return variable is a native Python list.

```
def convert_to_row(first, targ, run_id, keys, filename):
    peptide_ref = first.getMetaValue("PeptideRef")
    pep = targ.getPeptideByRef(peptide_ref)
    full_peptide_name = "NA"
    if (pep.metaValueExists("full_peptide_name")):
        full_peptide_name = pep.getMetaValue("full_peptide_name")

    decoy = "0"
```

```
    peptidetransitions = [t for t in targ.getTransitions() if t.getPeptideRef() == peptide_ref]
    if len(peptidetransitions) > 0:
        if peptidetransitions[0].getDecoyTransitionType() == pyopenms.DecoyTransitionType().DECOY↩
            :
            decoy = "1"
        elif peptidetransitions[0].getDecoyTransitionType() == pyopenms.DecoyTransitionType().↩
            TARGET:
            decoy = "0"

    protein_name = "NA"
    if len(pep.protein_refs) > 0:
        protein_name = pep.protein_refs[0]

    row = [
        first.getMetaValue("PeptideRef"),
        run_id,
        filename,
        first.getRT(),
        first.getUniqueId(),
        pep.sequence,
        full_peptide_name,
        pep.getChargeState(),
        first.getMetaValue("PrecursorMZ"),
        first.getIntensity(),
        protein_name,
        decoy
    ]

    for k in keys:
        row.append(first.getMetaValue(k))

    return row
```

convert_to_row() is a bit more complicated and takes as first input a Feature OpenMS class. From this, we access stored values using the provided functions (getRT(), getUniqueId(), etc). It further takes a TargetedExperiment to access information from the TraML with the provided routines. This data is then stored in a standard Python list with the variable name row and returned.

7.4.4 Putting things together

Now put these two functions into the header of

OpenSwathFeatureXMLToTSV_datastructures2.py.

Your final goal is to implement the conversion functionality into the main function using get_header() & convert_to_row() and to write a TSV using the standard csv module from Python **http://docs.python.org/2/library/csv.html**. Compare the results with **./result/example.tsv**. Are the results identical? Congratulations to your first pyOpenMS tool!

Hint: If you struggle at any point, take a look at OpenSwathFeatureXMLToTSV_solution.py.

7.4.5 Bonus task

Task

Implement all other 184 TOPP tools using pyOpenMS.

8 Quality control

8.1 Introduction

In this chapter, we will build on an existing workflow with OpenMS / KNIME to add some quality control (QC). We will utilize the qcML tools in OpenMS to create a file with which we can collect different measures of quality to the mass spectrometry runs themselves and the applied analysis. The file also serves the means of visually reporting on the collected quality measures and later storage along the other analysis result files. We will, step-by-step, extend the label-free quantitation workflow from section 3 with QC functions and thereby enrich each time the report given by the qcML file. But first, to make sure you get the most of this tutorial section, a little primer on how we handle QC on the technical level.

QC metrics and qcML

To assert the quality of a measurement or analysis we use quality metrics. Metrics are describing a certain aspect of the measurement or analysis and can be anything from a single value, over a range of values to a image plot or other summary. Thus, qcML metric representation is divided into QC parameters (QP) and QC attachments (QA) to be able to represent all sorts of metrics on a technical level.
A QP may (or may not) have a value which would equal a metric describable with a single value. If the metric is more complex and needs more than just a single value, the QP does not require the single value but rather depends on an attachment of values (QA) for full meaning. Such a QA holds the plot or the range of values in a table-like form. Like this, we can describe any metric by a QP and an optional QA.
To assure a consensual meaning of the quality parameters and attachments, we created a controlled vocabulary (CV). Each entry in the CV describes a metric or part/extension thereof. We embed each parameter or attachment with one of these and by doing so, connect a meaning to the QP/QA. Like this, we later know exactly what we collected and the programs can find and connect the right dots for rendering the report or calculating new metrics automatically. You can find the constantly growing controlled vocabulary here:

https://github.com/qcML/qcML-development/blob/master/cv/qc-cv.obo.
Finally, in a qcml file, we split the metrics on a per mass-spectrometry-run base or a set of mass-spectrometry-runs respectively. Each run or set will contain its QP/QA we calculate for it, describing their quality.

8.2 Building a qcML file per run

> Note: For this part of the tutorial, you need the latest version of OpenMS in KNIME (bleeding edge development version). Therefore you have to add the software site http://tech.knime.org/update/community-contributions/trunk/ in KNIME's Help > Install New Software menu and update OpenMS.

As a start, we will build a basic qcML file for each mzML file in the label-free analysis. We are already creating the two necessary analysis files to build a basic qcML file upon each mzML file, a feature file and an identification file. We use the **QCCreator** node from Community Nodes > OpenMS > Utilities where also all other **QC*** nodes will be found. The **QCCreator** will create a very basic qcML file in which it will store collected and calculated quality data.

- Copy your label-fee quantitation workflow into a new lfq-qc workflow and open it.
- Place the **QCCreator** node after the **IDMapper** node. Being inside the **ZipLoop**, it will execute for each of the three mzML files the **Input** node.
- Connect the first **QCCreator** port to the first **ZipLoopStart** outlet port, which will carry the individual mzML files.
- Connect the last's **ID** outlet port (**IDFilter** or the **ID** metanode) to the second **QCCreator** port for the identification file.
- Finally, connect the **IDMapper** outlet to the third **QCCreator** port for the feature file.

The created qcML files will not have much to show for, basic as they are. So we will extend them with some basic plots.

- First, we will add an 2D overview image of the given mass spectrometry run as you may know it from **TOPPView**. Add a the **ImageCreator** node from Community Nodes › OpenMS › Utilities. Change the width and heigth parameters to 640x640 as we don't want it to be too big. Connect it the first **ZipLoopStart** outlet port, so it will create an image file of the mzML's contained run.

- Now we have to embed this file into the qcML file, attach it to the right QualityParameter. For this, place a **QCEmbedder** node behind the **ImageCreator** and connect that to its third inlet port. Its first inlet port connect to the outlet of the **QCCalculator** node to pass on the qcML file. Now change the parameter qp_att_acc to QC:0000055 which designates the attached image to be of type **QC:0000055 - MS experiment heatmap**. Finally, change the parameter cv_acc to QC:0000004, to attach the image to the QualityParameter **QC:0000004 - MS acquisition result details**.

- For a reference of which CVs are already defined for qcML, have a look at https://github.com/qcML/qcML-development/blob/master/cv/qc-cv.obo.

There are two other basic plots which we almost always might want to look at before judging the quality of a mass spectrometry run and its identifications: the total ion current (TIC) and the PSM mass error (Mass accuracy), which we have available as prepackaged QC metanodes.

Task

Import the workflow from **Workflows ▸ Quality Control ▸ QC Metanodes.zip** in KNIME: File › Import KNIME Workflow...

- Copy the **Mass accuracy** metanode into the workflow behind the **QCEmbedder** node and connect it. The qcML will be passed on and the Mass accuracy plots added. The information needed was already collected by the **QCCalculator**.

- Do the same with the **TIC** metanode so that your qcML file will get passed on and enriched on each step.

R Dependencies: This section requires that the R packages ggplot2 and scales are both installed. This is the same procedure as in section 5.5.5. In case that you use an R

installation where one or both of them are not yet installed, open the **R Snippet** nodes inside the metanodes you just used (double-click). Edit the script in the R Script text editor from:

```
#install.packages("ggplot2")
#install.packages("scales")
```

to

```
install.packages("ggplot2")
install.packages("scales")
```

Press Eval script to execute the script.

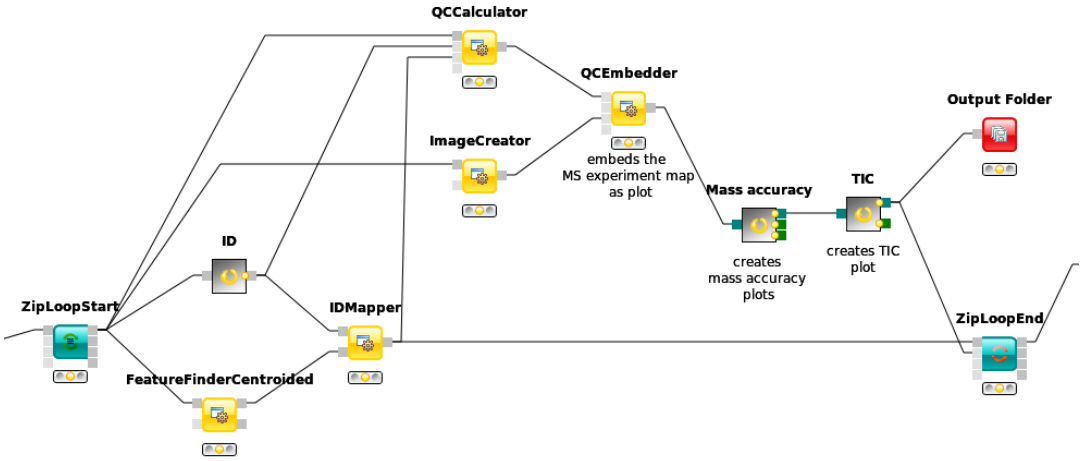

Figure 18: Basic QC setup within a LFQ workflow

> Note: To have a peek into what our qcML now looks like for one of the **ZipLoop** iterations, we can add an **Output Folder** node from Community Nodes › GenericKnimeNodes › IO and set its destination parameter to somewhere we want to find our intermediate qcML files in, for example tmp › qc_lfq. If we now connect the last

metanode with the **Output Folder** and restart the workflow, we can start inspecting the qcML files.

Task

Find your first created qcML file and open it with the browser (not IE), and the contained QC parameters will be rendered for you.

8.3 Adding brand new QC metrics

We can also add brand new QC metrics to our qcML files. Remember the **Histogram** you added inside the **ZipLoop** during the label-free quantitation section? Let's imagine for a moment this was a brand new and utterly important metric and plot for the assessment of your analyses quality. There is an easy way to pick up such new finds along the workflow into your qcMLs. Though the **Histogram** node cannot pass its plot to an image, we will do with a **R View (table)**.

- Add an **R View (table)** next to the **IDTextReader** node and connect them.
- Edit the **R View (table)** by adding the R Script according to this:

```
#install.packages("ggplot2")
library("ggplot2")
ggplot(knime.in, aes(x=peptide_charge)) +
 geom_histogram(binwidth=1, origin =-0.5) +
 scale_x_discrete() +
 ggtitle("Identified peptides charge histogram") +
 ylab("Count")
```

- This will create a plot like the **Histogram** node on peptide_charge and pass it on as an image.
- Now add and connect a **Image2FilePort** node from Community Nodes > GenericKnimeNodes > Flow to the **R View (table)**.

- We can now use a **QCEmbedder** node like before to add our new metric plot into the qcML.
- After looking for an appropriate target in `https://github.com/qcML/qcML-development/blob/master/cv/qc-cv.obo`, we found that we can attach our plot to the MS identification result details by setting the parameter qp_att_acc to QC:0000025, as we are plotting the charge histogram of our identified peptides.
- To have the plot later displayed properly, we assign it the parameter cv_acc of QC:0000051, a generic plot. Also we made sure in the R Script, that our plot carries a caption so that we know which is which, if we had more than one new plot.
- Now we redirect the **QCEmbedder**s output to the **Output Folder** from before and can have a look at how our qcML is coming along after restarting the workflow.

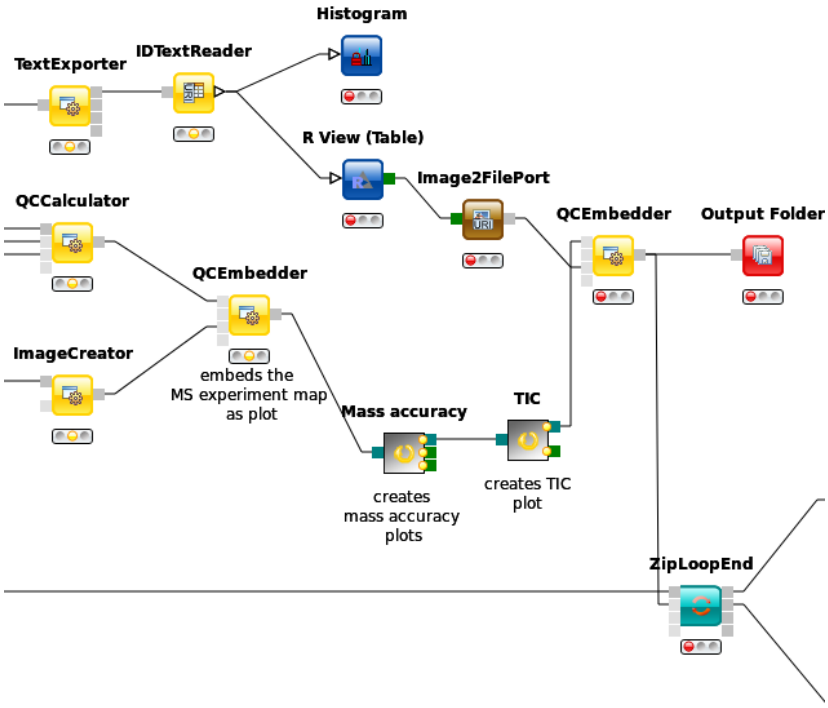

Figure 19: QC with new metric

8.4 Set QC metrics

Besides monitoring the quality of each individual mass spectrometry run analysis, another capability of QC with OpenMS and qcML is to monitor the complete set. The easiest control is to compare mass spectrometry runs which should be similar, e.g. technical replicates, to spot any aberrations in the set.
For this, we will first collect all created qcML files, merge them together and use the qcML onboard set QC properties to detect any outliers.

- connect the **QCEmbedder**s output from last section to the **ZipLoopEnd**s second input port.
- The corresponding output port will collect all qcML files from each **ZipLoop** iteration and pass them on as a list of files.
- Now we add a **QCMerger** node after the **ZipLoopEnd** and feed it that list of qcML files. In addition, we set its parameter setname to give our newly created set a name - say spikein_replicates.
- To inspect all the QCs next to each other in that created qcML file, we have to add a new **Output Folder** to which we can connect the **QCMerger** output.

When inspecting the set-qcML file in a browser, we will be presented another overview. After the set content listing, the basic QC parameters (like number of identifications) are each displayed in a graph. Each set member (or run) has its own section on the x-axis and each run is connected with that graph via a link in the mouseover on one of the QC parameter values.

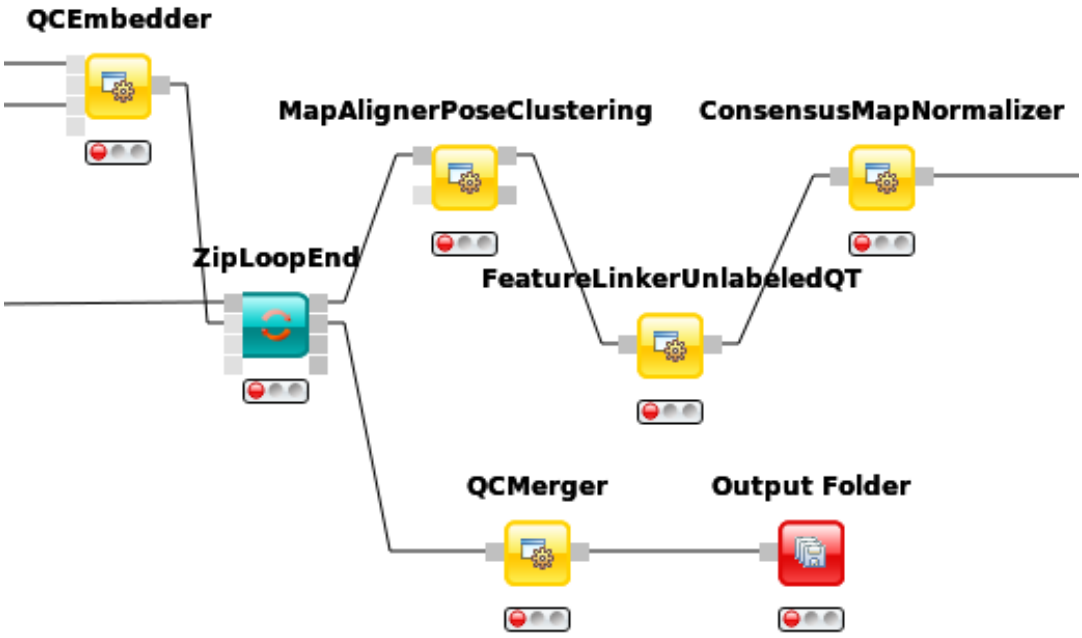

Figure 20: QC set creation from ZipLoop

Task

- [x] For ideas on new QC metrics and parameters -as you add them in your qcML files as generic parameters, feel free to contact us, so we can include them in the CV.

References

[1] OpenMS, OpenMS home page [online]. 6

[2] M. Sturm, A. Bertsch, C. Gröpl, A. Hildebrandt, R. Hussong, E. Lange, N. Pfeifer, O. Schulz-Trieglaff, A. Zerck, K. Reinert, and O. Kohlbacher, OpenMS - an open-source software framework for mass spectrometry., BMC bioinformatics 9(1) (2008), **doi:10.1186/1471-2105-9-163**. 6, 50

[3] O. Kohlbacher, K. Reinert, C. Gröpl, E. Lange, N. Pfeifer, O. Schulz-Trieglaff, and M. Sturm, TOPP–the OpenMS proteomics pipeline., Bioinformatics 23(2) (Jan. 2007). 6, 50

[4] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel, KNIME: The Konstanz Information Miner, in Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007), Springer, 2007. 6

[5] M. Sturm and O. Kohlbacher, TOPPView: An Open-Source Viewer for Mass Spectrometry Data, Journal of proteome research 8(7), 3760–3763 (July 2009), **doi:10.1021/pr900171m**. 6

[6] L. Y. Geer, S. P. Markey, J. A. Kowalak, L. Wagner, M. Xu, D. M. Maynard, X. Yang, W. Shi, and S. H. Bryant, Open mass spectrometry search algorithm, Journal of Proteome Research 3(5), 958–964 (2004). 22

[7] A. Chawade, M. Sandin, J. Teleman, J. Malmström, and F. Levander, Data Processing Has Major Impact on the Outcome of Quantitative Label-Free LC-MS Analysis, Journal of Proteome Research 14(2), 676–687 (2015), PMID: 25407311, **arXiv:http://dx.doi.org/10.1021/pr500665j**, **doi:10.1021/pr500665j**. 22

[8] D. S. Wishart, D. Tzur, C. Knox, et al., HMDB: the Human Metabolome Database, Nucleic Acids Res 35(Database issue), D521–6 (Jan 2007), **doi:10.1093/nar/gkl923**. 41

[9] D. S. Wishart, C. Knox, A. C. Guo, et al., HMDB: a knowledgebase for the human metabolome, Nucleic Acids Res 37(Database issue), D603–10 (Jan 2009), **doi:10.1093/nar/gkn810**. 41

[10] D. S. Wishart, T. Jewison, A. C. Guo, M. Wilson, C. Knox, et al., HMDB 3.0–The Human Metabolome Database in 2013, Nucleic Acids Res 41(Database issue), D801–7 (Jan 2013), **doi:10.1093/nar/gks1065**. 41

[11] J. Griss, A. R. Jones, T. Sachsenberg, M. Walzer, L. Gatto, J. Hartler, G. G. Thallinger, R. M. Salek, C. Steinbeck, N. Neuhauser, J. Cox, S. Neumann, J. Fan, F. Reisinger, Q.-W. Xu, N. Del Toro, Y. Perez-Riverol, F. Ghali, N. Bandeira, I. Xenarios, O. Kohlbacher, J. A. Vizcaino, and H. Hermjakob, The mzTab Data Exchange Format: communicating MS-based proteomics and metabolomics experimental results to a wider audience, Mol Cell Proteomics (Jun 2014), **doi:10.1074/mcp.O113.036681**. 42

[12] H. L. Röst, G. Rosenberger, P. Navarro, L. Gillet, S. M. Miladinovic, O. T. Schubert, W. Wolski, B. C. Collins, J. Malmstrom, L. Malmström, and R. Aebersold, OpenSWATH enables automated, targeted analysis of data-independent acquisition MS data, Nature Biotechnology 32(3), 219–223 (Mar. 2014). 50, 55

[13] L. C. Gillet, P. Navarro, S. Tate, H. Röst, N. Selevsek, L. Reiter, R. Bonner, and R. Aebersold, Targeted Data Extraction of the MS/MS Spectra Generated by Data-independent Acquisition: A New Concept for Consistent and Accurate Proteome Analysis., Molecular & Cellular Proteomics 11(6) (June 2012), **doi:10.1074/mcp.O111.016717**. 50

[14] A. Bertsch, C. Gröpl, K. Reinert, and O. Kohlbacher, OpenMS and TOPP: open source software for LC-MS data analysis., Methods in molecular biology (Clifton, N.J.) 696, 353–367 (2011), **doi:10.1007/978-1-60761-987-1_23**. 50

[15] L. Reiter, O. Rinner, P. Picotti, R. Huttenhain, M. Beck, M.-Y. Brusniak, M. O. Hengartner, and R. Aebersold, mProphet: automated data processing and statistical validation for large-scale SRM experiments, Nature Methods 8(5), 430–435 (May 2011), **doi:10.1038/nmeth.1584**. 50

[16] E. W. Deutsch, M. Chambers, S. Neumann, F. Levander, P.-A. Binz, J. Shofstahl, D. S. Campbell, L. Mendoza, D. Ovelleiro, K. Helsens, L. Martens, R. Aebersold, R. L. Moritz,

and M.-Y. Brusniak, TraML—A Standard Format for Exchange of Selected Reaction Monitoring Transition Lists, Molecular & Cellular Proteomics 11(4) (Apr. 2012), **doi: 10.1074/mcp.R111.015040**. 51

[17] C. Escher, L. Reiter, B. MacLean, R. Ossola, F. Herzog, J. Chilton, M. J. MacCoss, and O. Rinner, Using iRT, a normalized retention time for more targeted measurement of peptides., Proteomics 12(8), 1111–1121 (Apr. 2012), **doi:10.1002/pmic.201100463**. 51