cobia Vignette – Peptide-specific scores for cofragmentation risk

J. Scott P. McCain[1] and Erin M. Bertrand[1]

[1]Department of Biology, Dalhousie University, Halifax, Nova Scotia, Canada

Contents

1 Motivation

The ability to observe a peptide changes across samples. This is because the ‘background’ of any given peptide is also changing. If examining peptides is like examining a needle in a haystack, when we examine peptides across samples, we are changing the haystack.

We sought out to address this problem by predicting peptide-specific scores that relate to the ‘risk’ of being biased by this changing haystack, aka cofragmentation. If you want to know more about the details of our validation, rationale, and testing of these scores, please see the actual publication (and if you don’t have access, email us!). This vignette works through how you install and use *cobia*. All of the code is open source and has an MIT license.

2 What are the steps?

There are three key steps: 1) converting your protein database to digested, modified peptides, 2) retention time prediction and 3) cofragmentation score calculation. In the paper, we used multiple retention time predictors to demonstrate extensibility (RTPredict and RTModel from OpenMS, and BioLCCC). *cobia* can be used with any retention time predictors, as long as the retention time prediction output is formatted as required for input to *cobia*.

3 Installation

The installation instructions assume a linux environment, although they can be adapted to work on a Windows or MacOS system as needed. *cobia* is written in Python (2.7).

To install *cobia*, download the source code and in this directory run the *setup.py* file. Running the following command will build and install:

```
python setup.py install
```

We recommend installing on a conda environment:

```
python setup.py install --user
```

Here is a conda environment command that would setup your environment:

```
conda create --name pyteo_27 python=2.7 pandas
conda activate pyteo_27
pip install pyteomics.biolccc
pip install pyteomics
pip install pyopenms
```

4 Protein Database to Modified Peptides and Retention Time Prediction

The first input is your protein database. This should be everything that could *potentially* comprise of the actual protein going into the mass spectrometer. The more this database represents the material being injected to the mass spectometer, the more these scores will represent true cofragmentation risk.

A protein database *fasta* file is required as the first input. So far, we connect peptide modifications with two retention time predictors (RTPredict and BioLCCC). Peptides are modified using pyteomics. The following modifications are fixed and applied to all peptides:

1. Peptides digested with trypsin, with no missed cleavages.
2. Peptides with unknown amino acids ('' or 'X') or containing selenocysteine ('U') are removed.
3. Peptides have fixed chemical modifications. The following modifications are currently in place: oxidation of methionine and carbamidomethylation of cysteine.
4. Charge states are assigned deterministically. If a peptide has an internal histidine or arginine/lysine followed by a proline, it is assigned a charge state of 3. Otherwise it is assigned a charge state of 2.

If you are predicting retention times with BioLCCC, this is done with one command, which predicts the retention time and modifies peptides (see below for further description of these input files):

```
cobia peptide_mod_biolccc_rt_prediction
             -f potential_proteome.fasta
             -l lc_parameter_file.csv
             -g custom_gradient_file.csv
             -n output_name
```

If you are predicting retention times with RTPredict, there are two steps. The first is to fit a retention time model from observed peptides from an MS experiment, and then to predict retention times from all tryptic peptides in your input database.

```
RTModel -in observed_peptides_ms_experiment.txt
             -out rt_model.txt
             -kernel_type 'OLIGO'
             -total_gradient_time gradient_retention_time
```

Once you have your RT model trained, you can digest your database with cobia, then predict retention times with RTPredict, and then modify peptides with cobia:

```
cobia database_trypsin -f protein_database.fasta -n protein_database_trypsin.txt
RTPredict -in_text protein_database_trypsin.txt
             -out_text:file tryptic_peptide_rt.csv
             -svm_model rt_model.txt
             -total_gradient_time gradient_retention_time
cobia openms_modelled_rt -f tryptic_peptide_rt.csv -n tryptic_peptide_rt
```

4.1 BioLCCC Retention Time Prediction

Retention times can be predicted with BioLCCC using the following command:

```
cobia peptide_mod_biolccc_rt_prediction
             -f potential_proteome.fasta
             -l lc_parameter_file.csv
             -g custom_gradient_file.csv
             -n output_name
```

The lc_parameter_file.csv input is a file of liquid chromatography column characteristics. See here for an example file for the formatting requirements. These are the necessary parameters: column_diameter (mm), column_pore_size (A), second_solvent_concentration_a (%), second_solvent_concentration_b (%), gradient_0 (%), gradient_1 (%), gradient_2 (%), flow_rate (ml/min), code_format (should always be set to 'aas', for amino acids), linear (TRUE or FALSE), model (FA or TFA).

5 Cofragmentation Score Prediction

After predicting retention times for all potential peptides, you now score peptides based on their cofragmentation risk (‘cofragmentation score’), using *cobia*.

We use a relatively simple approach, essentially counting the number of isobaric (+/- precursor selection window / 2) and co-eluting peptides from the potential proteome (ie. the *fasta* file). One key parameter is the ion peak width - this is the length of time that a given peptide is eluting from a column into your mass spectrometer. In the paper, we’ve estimated the ion peak width with a simple linear model derived from Hsieh et al (2013) mean ion peak width as a function of gradient and column length (peak width 0.01978 - 0.0005563*(Column Length) + 0.0065488*(Gradient Length). The parameter input file structure is similar to the LC prediction for BioLCCC input file, see example-data-files/ for the structure.

There are two approaches to cofragmentation score prediction – global and targeted. Global determines cofragmentation scores for all peptides supplied, while targeted determines just a subset.

5.1 Global Cofragmentation Approach

In a global approach, the number of cofragmenting ions for all peptides in the potential proteome (via the fasta file) will be determined. This approach is computationally intensive (depending on the size of your fasta file), and so two approaches are used to increase speed: sparse sampling and parallelization. Sparse sampling approximates the number of cofragmenting ions by subsampling every nth injection bin. The number of cores can be set as well. Here is an example command:

```
cobia cofrag_prediction -f lc_predicted_output.csv
                                  -l dda_parameters.csv
                                  -n output_name
                                  --global global
```

The dda_paramters.csv file describes the required input parameters, and the formatting is shown in this example file folder. It includes the following parameters: max_injection_time which is the binning parameter described in the paper, precursor_selection_window which is the MS1 ion selection width, ion_peak_width describes the average length of peptide elution (described above), number_of_parallel is the number of cores to use when using the global approach, and every_nth which describes how sparse to sample injection ‘bins’.

See the case study below for choosing these values based on your dataset.

5.2 Targeted Cofragmentation Prediction

In a targeted approach, the number of cofragmenting ions for a subset of peptides in the potential proteome will be determined. This approach is much faster than the global approach, so we do not use sparse sampling or parallelization. Importantly, LC prediction as described above must still be run on the entire potential proteome first. Peptides (for example, in the file below called ‘target_peps.csv’) must be supplied as a .csv file with the column header ’pep_seq’ (for an example of this file format, look here). Example data use:

```
cobia cofrag_prediction -f lc_predicted_output.csv
                                  -l dda_parameters.csv
                                  -n output_name --global targeted
                                  -t target_peps.csv
```

6 What do you do with the scores?

Each peptide-specific score represents the *risk* of cofragmentation. This does not necessarily mean that peptides with high scores were not observed, it means that peptides with high scores are less likely to be observed, and their observability may shift across samples. Below we describe several use-cases of these scores.

6.1 Biomarker Choice

If you are chosing biomarkers, and want to examine them across a range of samples, this score could guide choice – you would chose peptides with low scores.

6.2 Imputation of Missing Values

A key challenge in proteomics is when to impute values for unobserved peptides. If a peptide has not been observed, it could be because it is very low abundance (hence imputing a low value would make sense), or it could be because there were cofragmenting peptides. We suggest that our score could guide imputation of missing values. Above some threshold, certain peptides should not be used for imputing.

What's the threshold? This is challenging to define – for every study, the range of cofragmentation scores observed will be different, and different users will have different tolerance for risk. A simple threshold is to determine the maximum cofragmentation score that you observed in your MS experiment. Peptides above that threshold may not have been observed because of cofragmentation.

7 Which type of mass spectrometry can this be used for?

Our approach is aimed for discovery-based proteomics. The peptide-specific scores are most relevant to this approach, including (but not limited to) label-free, labelled, data-depedendent acquisition, or data-independent acquisition. Theoretically, the scores would also be relevant for targeted proteomics, however this approach is likely more suitable for that scenario.

8 Case Study: Prostate Cancer Biomarker

Part of our validation strategy was to determine if the scores *cobia* produces are associated with whether or not we identify a peptide. So we used previously published datasets, calculated cofragmentation scores for all peptides in a database, and then compared cofragmentation scores with those peptides that were observed versus those that were not observed.

The rationale is really: do these scores manifest themselves in a meaningful way in mass spectrometry? In doing this validation, we also demonstrate how a researcher might use *cobia*. Here, I detail how to analyze a dataset of prostate cancer biomarkers from Davalieva *et al* (2017) Comparative Proteomics Analysis of Urine Reveals Down-Regulation of Acute Phase Response Signaling and LXR/RXR Activation Pathways in Prostate Cancer. Despite using this dataset to validate our model, *we are not necessarily recommending this approach for this application. Targeted mass spectrometry approaches may be more appropriate, and SRMCollider is another, similar tool, that is directed to that type of mass spectrometry.* We do still continue with this example as an illustrative example.

8.1 Data Munging

First we need to train a retention time model on the observed peptides. Peptides, and retention times, are reported from PRIDE, 'Peptide list 16012017.csv'. We take that file, and format it appropriately, using R:

```
pro_cancer <- read.csv("data/prostate_cancer_data/Peptide list 16012017.csv", skip = 2)

pro_cancer_seq <- pro_cancer$Sequence
pro_cancer_sec <- pro_cancer$Retention.time..min.*60

df1 <- data.frame(as.factor(pro_cancer_seq), pro_cancer_sec)

write.table(df1, file = 'data/prostate_cancer_data/prostate_cancer_peptides_formatted.txt',
            col.names = FALSE, sep = "\t",
            row.names = FALSE, quote = FALSE)
```

Note that for datasets with many peptides, it would be wise to subsample all observed peptides because it might take a very long time to train an SVM.

8.2 Predicting Retention Time

We then digest the database using cobia and train a retention time model with RTModel and RTPredict (a support vector machine). We also use the *fasta* database of human protein coding genes supplied here, the file 'Human_uniprot_09082016_all.fasta'.

```
# Declare a common directory
DIR='../data/prostate_cancer_data/'

# take protein coding genome (MS search database) and digest it with trypsin

cobia database_trypsin -f "$DIR"Human_uniprot_09082016_all.fasta
-n "$DIR"Human_uniprot_09082016_all_trypsin.fasta
-c no-write

# fit retention time predictor model with RTModel. The total gradient time is in seconds.

RTModel -in "$DIR"prostate_cancer_peptides_formatted.txt
-out "$DIR"prostate_cancer_model_oligo
-kernel_type 'OLIGO'
-total_gradient_time 7200

# predict retention times with model

RTPredict -in_text "$DIR"Human_uniprot_09082016_all_trypsin.fasta
-out_text:file "$DIR"Human_uniprot_09082016_rt_oligo.csv
-svm_model "$DIR"prostate_cancer_model_oligo.txt
-total_gradient_time 7200
```

8.3 Calculating Cofragmentation Scores

Now that we have retention times calculated, we need to process the peptide/retention time file. This next step formats the peptide sequences for use by pyteomics, and it also modifies the peptide sequences with chemical modifications as described in the paper (methionine is oxidized and cysteine is carbamidomethylated), which alters peptide masses.

```
cobia openms_modelled_rt -f "$DIR"Human_uniprot_09082016_rt_oligo.csv
-n Run4_C4_2000ng_linear
```

The last step (and the main innovation of *cobia*!) is producing cofragmentation scores. A parameter file is required that contains the 'maximum_injection_time' (minutes), 'precursor_selection_window' (m z), 'ion_peak_width' (minutes), 'number_of_parallel' (number of cores to use), and 'every_nth' (how sparse to sample). The maximum injection time is the length of time that peptides are injected into an Orbitrap. We recommend using a time of 500ms (based on MS/MS ion injection times and settings from an Orbitrap Elite, Kalli et al 2013) which corresponds to 0.00833333 minutes. You can use your precursor ion selection window size settings for a given mass spectrometry method, typically 3 around m z.

The ion peak width represents the amount of time a given peptide is eluting off the column into the mass spectrometer. This varies with the abundance of a given peptide. But since we cannot include abundance in our approach (discussed in the paper as to why), we assume a constant ion peak width. From a previous study (Hsieh *et al* 2013), we used a simple linear model to calculate the peak width as a function of column length (mm) and gradient length (minutes):

```
peak width ~ 0.01978 - 0.0005563*(Column Length) + 0.0065488*(Gradient Length)
```

The last input parameter is the 'every_nth' parameter. This represents out of every nth injection bin, how many are you actually sampling to determine the cofragmentation score? If this is set to 1, it means there is exhaustive sampling. We recommend having a ratio of 'every_nth' to 'ion_peak_width' of around 14, which from our testing still gave highly correlated cofragmentation scores with exhaustive sampling but also improved computational run-time.

An example of the parameter input file required is shown here in this folder.

```
cobia cofrag_prediction -l "$DIR"dda_params_prostate_canncer.csv
-f "$DIR"Human_uniprot_09082016_rt_oligo_lc-retention-times.csv
-n "$DIR"Human_uniprot_09082016_rt_oligo_cofrag
--global global
```

You are then left with cofragmentation scores per peptide. From here, it's a bit more open-ended. If, for example, you have very low tolerance for cofragmentation risk, then your cutoff would be lower. If you have a higher tolerance, your cutoff would be higher. As discussed above, one heuristic would be to take all your observed peptides, calculate cofragmentation scores, and assume that those above your cutoff are susceptible or not detected because of cofragmentation.