An introduction to CFTK, the co-fractionation toolkit

–

CFTK, the co-fractionation toolkit, is an R package designed for the analysis of co-fractionation mass spectrometry (CF-MS) data. CFTK provides convenient and flexible implementations of many of the most common analysis tasks that investigators working with CF-MS data seek to perform. These include loading CF-MS data and known protein complexes into R, preprocessing the data, filtering low-quality chromatograms, evaluating data quality, scoring proteins with similar patterns of fractionation, and integrating data across multiple replicates using a machine-learning approach.

This vignette demonstrates the major functionality of the CFTK package through application to an example dataset. The CF-MS data was obtained from Kirkwood *et al.*, 2013 (1), who profiled 40 size exclusion chromatography (SEC) fractions from human osteosarcoma (U2OS) cells, in three biological replicates. The raw data was obtained from the ProteomeXchange, under accession PXD001220, and reprocessed with MaxQuant (version 1.6.5.0), using default parameters. All of the data analyzed in this vignette is available from the `data-raw` folder on the GitHub repository, at https://github.com/fosterlab/CFTK (https://github.com/fosterlab/CFTK).

The code presented in this vignette assumes that you have set your working directory to the location of the CFTK package. This can be accomplished using the `setwd` function. For example, if you are using a Linux or macOS machine and have downloaded the CFTK package to a directory named `git` inside your home directory, run the following command:

```
setwd("~/git/CFTK")
```

Adjust the path to the CFTK package directory as needed.

The code in this vignette also makes use of functions from the `magrittr`, `dplyr`, and `purrr` packages, which are installed along with the CFTK package. Load these packages before running this code, and set `stringsAsFactors` to `FALSE`, using the following command:

```
library(magrittr)
library(dplyr)
library(purrr)
options(stringsAsFactors = FALSE)
```

Loading CF-MS data from MaxQuant

CFTK is agnostic to the specific pipeline used to search and quantify raw mass spectrometric data. However, it does provide a convenience function to import CF-MS chromatograms searched using MaxQuant (2), by parsing MaxQuant `proteinGroups.txt` files. This functionality is implemented in the `read_maxquant()` function, which allows the user to import chromatograms using multiple different quantitation modes (providing, of course, that the information is present in the file), including iBAQ, MaxLFQ, MS1 intensity, spectral counts, and isotopologue ratios. The function also provides a method to map protein groups to genes, using information contained in the `Gene names` column of the `proteinGroups.txt` file. To read the first replicate from PXD001220, using the label-free quantitation implemented in MaxLFQ and mapping protein groups to gene names, run the following function:

```
filepath = "data-raw/PXD001220/rep1/proteinGroups.txt.gz"
mat = read_maxquant(filepath, quant_mode = "MaxLFQ", identifiers = 'genes')
dim(mat)
```

```
## [1] 5229   40
```

```
str(mat)
```

```
##  num [1:5229, 1:40] 0 2021400 0 0 0 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:5229] "A2ML1" "AAAS" "AACS" "AAGAB" ...
##   ..$ : chr [1:40] "LFQ.intensity.PT1541S1F00" "LFQ.intensity.PT1541S1F01"
"LFQ.intensity.PT1541S1F02" "LFQ.intensity.PT1541S1F03" ...
##  - attr(*, "gene_map")='data.frame': 5229 obs. of  2 variables:
##   ..$ gene         : chr [1:5229] "A2ML1" "AAAS" "AACS" "AAGAB" ...
##   ..$ protein_group: chr [1:5229] "A8K2U0" "Q9NRG9;F8VZ44;H3BU82;Q9NRG9-2"
"Q86V21;Q86V21-2" "Q6PD74-2;Q6PD74;H0YL49" ...
```

Because the `read_maxquant` function wraps the base R function `read.delim`, the filepath can also be a remote URL. For instance, if we wished to read a `proteinGroups.txt` file hosted on the server “my-website.com”, we could simply pass in the full URL of the file instead:

```
filepath = "http://my-website.com/maxquant/PXD001220/rep1/proteinGroups.txt.gz"
mat = read_maxquant(filepath, quant_mode = "MaxLFQ", identifiers = 'genes')
```

As mentioned, CFTK is not limited to MaxQuant outputs, but can work with chromatograms obtained from any database search program. For alternative search programs, CFTK simply expects a numeric matrix, with proteins in rows and fractions in columns. Please note that this matrix must have

rownames set in order to link calculated features or predicted interactions to the proteins identified within the CF-MS experiment.

During the development of the CFTK package, we systematically re-analyzed 206 published CF-MS experiments using MaxQuant. The processed chromatograms obtained from this re-analysis are available from Zenodo, at https://zenodo.org/record/4106578#.X–haulKh24 (https://zenodo.org/record/4106578#.X--haulKh24). In this upload, separate chromatogram matrices are provided for each protein quantification method implemented in MaxQuant (e.g., iBAQ vs. MaxLFQ). A sample of the chromatograms available from Zenodo is also available in the `data-raw` directory. To read one of these chromatograms into R as a numeric matrix for use with the CFTK, enter the following command:

```
mat = read.delim("data-raw/zenodo/PXD001220/rep1/iBAQ.tsv.gz") %>%
  as.matrix()
```

To keep only the first UniProt accession associated with each protein group, enter the following command:

```
rownames(mat) %<>% gsub(";.*$", "", .)
```

Preprocessing CF-MS data

Having read one or more CF-MS datasets into an R session, CFTK now provides several functions to pre-process the data. Low-quality chromatograms can be removed with the `filter_matrix()` function, which removes proteins not quantified in a certain minimum number of fractions. These typically represent a minority of identified proteins but in some cases, can account for a substantial fraction:

```
dim(mat)
```

```
## [1] 5229   40
```

```
mat %<>% filter_matrix(min_fractions = 4)
dim(mat)
```

```
## [1] 3641   40
```

Missing values are ubiquitous in CF-MS data, occurring for example in SEC data because proteins do not participate in any macromolecular complexes at the approximate molecular weight of a particular fraction, and can be handled in several different ways. CFTK provides the `sanitize_missing_values()` and `replace_missing_values()` functions to deal with missing values. The first, `sanitize_missing_values()`, identifies `NA`s, `NaN`s, infinite values, and zeroes, and replaces them all with `NA`s. The second, `replace_missing_values()`, both enforces this consistent representation of missing data and then optionally replaces them either with zeroes, or random near-zero noise. To treat all missing values as zeroes, run the following command:

```
mat %<>% replace_missing_values(missing = 'zero')
```

Finally, CFTK also provides the `transform_matrix()` function to normalize the matrix, although we suggest that this function should be used sparingly and only when appropriate. This can be achieved either by log-transformation (because CF-MS data is approximately log-normally distributed), or quantile normalization. Users must be cautious about the order in which these operations are performed. For instance, replacing missing values with zeroes must be done *after* log-transformation, or this transformation may introduce a large number of `-Inf` values. Similarly, users wishing to treat missing values as zeroes should perform quantile normalization only *after* this step is complete. We suggest the following order of operations:

i. filter the matrix
ii. log-transform chromatograms (optional)
iii. handle missing values
iv. quantile normalization (optional)

Computing features for protien pairs

One of the central tasks in CF-MS data analysis is to identify pairs (or groups) of proteins with similar patterns of elution across a chromatographic separation. This is commonly achieved using ubiquitous measures of association such as the Pearson correlation or Euclidean distance. In addition to these, CFTK implements a total of 24 measures of association that have been proposed for CF-MS or more general '-omics' data analysis. These can be applied to obtain a square matrix, scoring the similarity of every protein pair in the experiment, using the `score_pairs()` function. For example, to score all protein pairs using the Bray-Curtis distance, enter the following command:

```
pairs = score_pairs(mat, metric = 'bray_curtis')
str(pairs)
```

```
##  num [1:3641, 1:3641] 0 -0.962 -1 -0.91 -0.999 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:3641] "AAAS" "AAK1" "AAMDC" "AAR2" ...
##   ..$ : chr [1:3641] "AAAS" "AAK1" "AAMDC" "AAR2" ...
```

The complete list of all 24 metrics can be accessed using the `metrics()` function.

```
metrics()
```

```
##              Bayesian correlation          Biweight midcorrelation
##                        "bayes_cor"                           "bicor"
##               Co-dependency index             Bray-Curtis distance
##                         "binomial"                    "bray_curtis"
##                Canberra distance                  Cosine distance
##                         "canberra"                          "cosine"
##                  Dice coefficient             Distance correlation
##                             "dice"                   "distance_cor"
##                Euclidean distance                           GENIE3
##                        "euclidean"                         "GENIE3"
##                  Hamming distance                   Jaccard index
##                         "hamming"                       "jaccard"
##               Kendall correlation               Manhattan distance
##                         "kendall"                       "manhattan"
##                Mutual information               Pearson correlation
##                               "MI"                         "pearson"
##            Proportionality (phi)               Profile correlation
##                           "phi_s"                     "profile_cor"
##            Proportionality (rho)             Spearman correlation
##                           "rho_p"                        "spearman"
##                         treeClust       Weighted cross-correlation
##                        "treeClust"                           "wccor"
##        Weighted rank correlation Zero-inflated Kendall correlation
##                   "weighted_rank"                       "zi_kendall"
```

The matrix of protein pairs can itself be filtered to replace pairs of proteins that were not *jointly* detected in some minimum number of fractions with `NA`s, using the `filter_pairs()` function. For instance, to mask protein pairs that did not co-occur in at least one fraction, enter the following command:

```
pairs0 = filter_pairs(pairs, mat, min_pairs = 1)
```

Combining information across replicates

While the `score_pairs()` function implements a number of different approaches to scoring co-eluting protein pairs within a single CF-MS experiment, investigators generally collect multiple CF-MS replicates from the same biological system in order to enable more robust network inference. The standard workflow for combining information across multiple replicates involves the application of a machine-learning workflow to score interacting protein pairs, by learning from a dataset of known protein complexes. (The same workflow can be applied to integrate evidence from multiple features within a single CF-MS replicate). Briefly, after calculating one or more measures of association (or 'features') for each protein pair in each replicate, these features are merged across replicates to create a single, consolidate data frame scoring every possible protein pair. Then, a 'gold standard' set of protein-protein interactions is loaded—for example, intra-complex interactions from the CORUM database. The 'gold standard' is split into a number of folds, and a series of machine-learning classifiers are trained on the features using the gold standard, with each fold excluded one at a time. The trained classifiers are used to compute the likelihood that all protein pairs *not* overlapping with the training set are interacting, based on the same CF-MS features. Protein pairs are then sorted by their mean classifier score across all folds, in descending order, to obtain a ranked list of all candidate protein-protein interactions.

Implementing the workflow described above using CFTK proceeds as follows. First, features for every protein pair must be calculated and merged into a single data frame. This can be done by repeatedly applying the `score_pairs()` function to caluclate various features of interest, then merging them using the `merge_features()` function. The use of this function is demonstrated below to concatenate the Pearson correlation and Euclidean distance in a single replicate:

```
pairs1 = score_pairs(mat, metric = 'pearson')
pairs2 = score_pairs(mat, metric = 'euclidean')
feature_df = merge_features(list(pairs1, pairs2))
head(feature_df)
```

```
##   protein_A protein_B   feature.x feature.y
## 1      AAAS      AAK1 -0.09411139 -23260111
## 2      AAAS     AAMDC -0.08536994 -87324269
## 3      AAK1     AAMDC -0.11300250 -90066078
## 4      AAAS      AAR2 -0.07592153  -7575923
## 5      AAK1      AAR2 -0.11634107 -22942817
## 6     AAMDC      AAR2 -0.16347969 -87333061
```

Alternatively, if only a single measure of association is of interest, this can be calculated for a list of CF-MS matrices using the `calculate_features()` function. For instance, to calculate the Pearson correlation for three matrices simultaneously, run the following command:

```
replicates = c('rep1', 'rep2', 'rep3')
filepaths = file.path("data-raw/PXD001220", replicates, "proteinGroups.txt.gz")
mats = map(filepaths, read_maxquant, quant_mode = "MaxLFQ", identifiers = 'gene
s')
feature_df = calculate_features(mats, metric = 'pearson')
head(feature_df)
```

```
##   protein_A protein_B   pearson.x   pearson.y     pearson
## 1     A2ML1      AAAS -0.04712832          NA -0.02564103
## 2     A2ML1      AACS -0.04246607          NA -0.04556039
## 3      AAAS      AACS -0.07805282 -0.06936836 -0.04556039
## 4     A2ML1     AAGAB -0.04072171          NA -0.07851446
## 5      AAAS     AAGAB -0.07484668 -0.04305747 -0.07851446
## 6      AACS     AAGAB -0.06744234 -0.04130934  0.42458579
```

These two functions can be mixed and matched: for example, to calculate *two* measures of association per replicate, we could run the `calculate_features()` function again, then run `merge_features()`, like so:

```
feature_df1 = feature_df
feature_df2 = calculate_features(mats, metric = 'euclidean')
feature_df = merge_features(list(feature_df1, feature_df2))
head(feature_df)
```

```
##   protein_A protein_B   pearson.x   pearson.y     pearson euclidean.x
## 1     A2ML1      AAAS -0.04712832          NA -0.02564103    -5862973
## 2     A2ML1      AACS -0.04246607          NA -0.04556039   -14988257
## 3      AAAS      AACS -0.07805282 -0.06936836 -0.04556039   -15794796
## 4     A2ML1     AAGAB -0.04072171          NA -0.07851446    -3897018
## 5      AAAS     AAGAB -0.07484668 -0.04305747 -0.07851446    -6325701
## 6      AACS     AAGAB -0.06744234 -0.04130934  0.42458579   -15175240
##   euclidean.y euclidean
## 1          NA -22801490
## 2          NA -31431462
## 3   -33866995 -21714889
## 4          NA -27568896
## 5    -4420671 -15609088
## 6   -33682918 -19576405
```

When features are obtained from multiple replicates, there will typically be missing values, because not all proteins are quantified in all replicates. Some classifiers, such as the naive Bayes, can naturally handle these missing values. However, most will throw an error if missing values are included in the input. The `impute_missing_features()` function can be used to replace these missing values with the median value for that feature, plus or minus some random noise:

```
feature_df %<>% impute_missing_features()
```

Loading known protein complexes from CORUM

At this point, we have loaded a collection of CF-MS datasets, preprocessed them, computed one or more features for each dataset, and combined multiple features into a single data frame. The next step is to distinguish protein-protein interactions from non-interacting pairs, based on these features. To achieve this, CFTK uses a supervised machine-learning framework. Conceptually, CFTK takes a set of known protein-protein interactions as input, and learns what values the features we have computed tend to take on for known interacting pairs. To do so, CFTK requires a set of known protein-protein interactions to learn from. CFTK provides a convenience function to read protein complex data distributed through the CORUM database (3), using the `read_corum()` function, which allows the user to map complex subunits to either UniProt accessions, Entrez gene IDs, or gene names. However, CFTK is agnostic to the source of known protein complexes (or binary protein-protein interactions) used for CF-MS data analysis, and can easily learn from an alternative resource of known protein-protein interactions. CFTK also provides a variety of functions to convert protein complex data between various R data structures, including tidy data frames, lists, adjacency matrices, and pairwise data frames enumerating all intra-complex interactions.

To read the ‘core’ set of CORUM protein complexes, mapping subunits to their gene symbols, run the following function:

```
corum = read_corum("data-raw/CORUM/coreComplexes.txt.gz", identifiers = 'gene_n
ame')
head(corum)
```

```
## # A tibble: 6 x 2
##   complex            subunit
##   <chr>              <chr>
## 1 BCL6-HDAC4 complex BCL6
## 2 BCL6-HDAC4 complex HDAC4
## 3 BCL6-HDAC5 complex BCL6
## 4 BCL6-HDAC5 complex HDAC5
## 5 BCL6-HDAC7 complex BCL6
## 6 BCL6-HDAC7 complex HDAC7
```

The `read_corum()` function returns a data frame in which the first column contains complex names, and the second column contains complex subunits. Often, however, we need to convert this data into a different format. For example, we might wish to convert this data frame into a list, where each entry represents a protein complex and contains a vector of its subunits. This can be achieved using the function `as_annotation_list()`:

```
complex_list = as_annotation_list(corum, 'complex', 'subunit')
str(complex_list[1:5])
```

```
## List of 5
##  $ (E.F.G) complex                                                    : chr [1:
3] "SNRPE" "SNRPF" "SNRPG"
##  $ (ER)-localized multiprotein complex, Ig heavy chains associated: chr [1:1
0] "Pdia4" "Hsp90b1" "P4hb" "Hspa5" ...
##  $ 12S U11 snRNP                                                      : chr [1:1
5] "SNRPB" "SNRPE" "SNRPF" "SNRPG" ...
##  $ 14-3-3 gamma-CXCR2 complex, unstimulated                           : chr [1:
2] "CXCR2" "YWHAG"
##  $ 17S U2 snRNP                                                       : chr [1:3
3] "U2SURP" "DHX15" "SF3B1" "DNAJC8" ...
```

Alternatively, we might wish to enunmerate all possible pairs of proteins that are members of the same protein complex. From a list of protein complexes, we can construct a pairwise data frame of intra-complex pairs using the function `to_pairwise_df()`:

```
complex_pairs = to_pairwise_df(complex_list)
head(complex_pairs)
```

```
## # A tibble: 6 x 2
##   protein_A protein_B
##   <chr>     <chr>
## 1 SNRPE     SNRPF
## 2 SNRPE     SNRPG
## 3 SNRPF     SNRPG
## 4 Cabp1     Dnajb11
## 5 Cabp1     Hsp90b1
## 6 Cabp1     Hspa5
```

Finally, we might wish to create an adjacency matrix, in which each cell represents a potential protein-protein interaction and intra-complex pairs are marked as ones. This can be done from the pairwise data frame using the function `to_adjacency_matrix()`:

```
complex_adj = to_adjacency_matrix(complex_pairs)
str(complex_adj)
```

```
##  num [1:5163, 1:5163] 0 1 0 0 0 0 0 0 0 0 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:5163] "SNRPE" "SNRPF" "Cabp1" "Dnajb11" ...
##   ..$ : chr [1:5163] "SNRPE" "SNRPF" "Cabp1" "Dnajb11" ...
```

These different representations of protein complexes are used by various functions throughout the CFTK package.

Loading functional annotations from the Gene Ontology

CFTK also provides a utility function to read protein function annotations from the Gene Ontology in GAF format, implemented in the function `read_gaf()`. This function also allows users to filter the GO: for instance, by removing annotations with less reliable or potentially recursive evidence codes, such as 'IEA' (inferred by electronic annotation) or 'IPI' (inferred from physical interaction). To read Gene Ontology annotations for human, removing interactions with the code 'IEA', run the following function:

```
goa = read_gaf("data-raw/GO/goa_human.gaf.gz", filter_evidence = "IEA")
head(goa)
```

```
## # A tibble: 6 x 17
##   DB    `DB Object ID` `DB Object Symb… Qualifier `GO ID` `DB:Reference`
##   <chr> <chr>          <chr>            <chr>     <chr>   <chr>
## 1 UniP… A0A024RBG1     NUDT4B           <NA>      GO:000… GO_REF:0000052
## 2 UniP… A0A075B6P5     IGKV2-28         <NA>      GO:000… Reactome:R-HS…
## 3 UniP… A0A075B6P5     IGKV2-28         <NA>      GO:000… Reactome:R-HS…
## 4 UniP… A0A075B6P5     IGKV2-28         <NA>      GO:000… Reactome:R-HS…
## 5 UniP… A0A075B6P5     IGKV2-28         <NA>      GO:000… Reactome:R-HS…
## 6 UniP… A0A075B6P5     IGKV2-28         <NA>      GO:000… Reactome:R-HS…
## # … with 11 more variables: `Evidence Code` <chr>, `With (or) From` <chr>,
## #   Aspect <chr>, `DB Object Name` <chr>, `DB Object Synonym` <chr>, `DB Obj
ect
## #   Type` <chr>, Taxon <chr>, Date <dbl>, `Assigned By` <chr>, `Annotation
## #   Extension` <lgl>, `Gene Product Form ID` <lgl>
```

CFTK also provides a convenience function, `go_roots()`, which returns a vector containing the Gene Ontology codes for 'biological process', 'molecular function', and 'cellular compartment'. This function can be used to filter these terms, which are usually considered uninteresting, from the set of annotations:

```
goa %<>% filter(!`GO ID` %in% go_roots())
```

To create a list in which each entry is a GO term, and contains a vector of all proteins annotated to that GO term, we can again make use of the function `as_annotation_list()`:

```
go_ann = as_annotation_list(goa, 'GO ID', 'DB Object Symbol')
str(go_ann[1:5])
```

```
## List of 5
##  $ GO:0000002: chr [1:11] "OPA1" "SLC25A4" "TYMP" "LONP1" ...
##  $ GO:0000003: chr [1:4] "MMP23B" "GNRH1" "LIN9" "GNRH2"
##  $ GO:0000010: chr [1:2] "PDSS1" "PDSS2"
##  $ GO:0000012: chr [1:8] "TNP1" "LIG4" "ERCC6" "ERCC8" ...
##  $ GO:0000014: chr [1:10] "ERCC1" "MRE11" "SETMAR" "RAD50" ...
```

Identifying co-eluting protein pairs

Having calculated a series of features for this experiment, we are now ready to provide it as input to the cross-validation procedure, along with the complexes from the CORUM database that we loaded in above. This procedure is implemented in the `score_interactions()` function, which allows the user to specify the number of cross-validation folds to perform (the default is 10) and select a classifier (one of 'NB' for naive Bayes, 'RF' for random forest, 'SVM' for support vector machine, or 'LR' for logistic regression). The `score_interactions()` function also allows the user to control the way in which cross-validation is performed. Specifically, the 'gold standard' complexes can be divided into folds that are disjoint either with respect to individual protein subunits (argument `split_by = 'proteins'`), or with respect to pairwise interactions (`split_by = 'pairs'`). Although splitting complexes by interacting pairs provides more data to the classifier, it can lead to overfitting. These cross-validation procedures are implemented in the `split_by()` function.

To run the `score_interactions()` function, enter the following command. (Please note that this can take quite a while to run, depending on the size of the dataset, classifier selection, and number of cross-validation folds).

```
interactions = score_interactions(feature_df, complex_pairs, n_folds = 3, class
ifier = 'NB')
```

```
## working on split 1 of 3 ...
```

```
## working on split 2 of 3 ...
```

```
## working on split 3 of 3 ...
```

```
head(interactions)
```

```
##   protein_A protein_B     score label
## 1      CCT3      CCT8 0.9999998     1
## 2      MCM3      MCM5 0.9999998     1
## 3      CCT4      CCT7 0.9999998     1
## 4      CCT2      CCT4 0.9999998     1
## 5      CCT2      TCP1 0.9999998     1
## 6     DCTN1     DCTN2 0.9999998     1
```

Next steps

The output of the `score_interactions` function is a ranked list of candidate protein-protein interactions. Each possible protein pair is associated with a score, assigned by the machine-learning classifier, that reflects the likelihood of a physical interaction occurring. A subset of protein pairs that were among the known protein-protein interactions used to train the classifier may also have an entry in the fourth column `label`. The value of this column is `1` for known protein-protein interactions, `0` for non-interacting proteins known to participate in at least one protein complex ("inter-complex interactions"), and `NA` for proteins not found in the training dataset.

Many different analyses are possible from this point. One common workflow involves keeping only the largest subset of interactions above some user-defined precision threshold (where precision is defined as the proportion of true positives among the labelled pairs). The `calculate_precision` function provides a means to compute the precision for each row within the ranked list:

```
interactions %<>% mutate(precision = calculate_precision(label))
head(interactions)
```

```
##   protein_A protein_B     score label precision
## 1      CCT3      CCT8 0.9999998     1         1
## 2      MCM3      MCM5 0.9999998     1         1
## 3      CCT4      CCT7 0.9999998     1         1
## 4      CCT2      CCT4 0.9999998     1         1
## 5      CCT2      TCP1 0.9999998     1         1
## 6     DCTN1     DCTN2 0.9999998     1         1
```

Once the precision has been calculated, the `threshold_precision` function can be applied to select interactions above a given precision. For instance, to retain only the network of interactions that surpass a 50% precision threshold, enter the following command:

```
net50 = threshold_precision(interactions, 0.5)
dim(net50)
```

```
## [1] 1390    5
```

Further analyses of the resulting high-confidence interaction network will generally require the use of other R packages. One especially common R package for network analysis is the `igraph` R package. Although it is outside the scope of this vignette to provide a complete guide to the functionality of the `igraph` package, we note that the filtered list of interactions can be converted to an unweighted, undirected graph using the following command:

```
## not run
g = net50 %>%
  dplyr::select(protein_A, protein_B) %>%
  igraph::graph_from_data_frame(directed = FALSE)
```

For further details about installing and using `igraph`, please see the package documentation at https://igraph.org/r/ (https://igraph.org/r/).

An alternative set of analyses operates on the entire ranked list of protein pairs, without first filtering these to a high-confidence subset. For instance, we can compute the area under the receiver operating characteristic curve (AUROC, or AUC) for the *entire* ranked list of protein pairs using the `calculate_complex_auc` function:

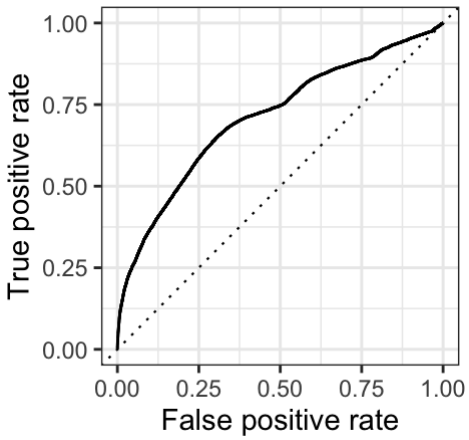
```
auc = calculate_complex_auc(interactions, complex_adj, score_column = 'score')
format(auc, digits = 3)
```

```
## [1] "0.719"
```

An analogous function, `calculate_go_auc`, is provided to perform AUROC analysis using annotations from the Gene Ontology as the ground truth, instead of membership in known protein-protein interactions. The major difference between the two functions is that `calculate_go_auc` returns a data frame, instead of a single value, containing results for each GO term that was provided as input. Enter `?calculate_go_auc` into the R command prompt for further details.

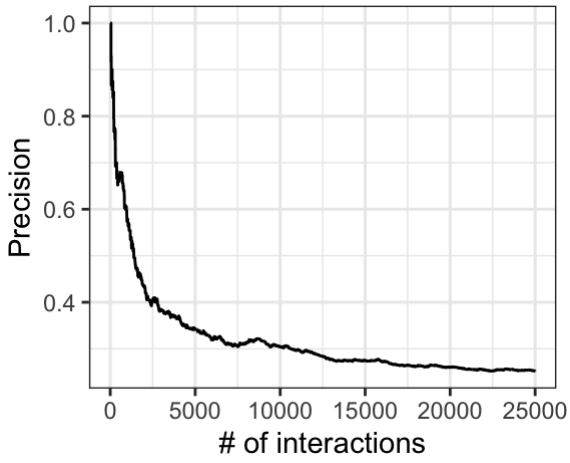
Finally, users may simply wish to visualize the quality of the resulting networks. To this end, CFTK provides two plotting functions, `plot_roc` and `plot_pr`, which take the data frame output by `score_interactions` as input (that is, a data frame of protein pairs, each associated with a score and a label), and use the information therein to plot receiver operating characteristic (ROC) or precision-recall (PR) curves, respectively. To plot a ROC curve for the inferred network, for instance, enter the following command:

```
plot_roc(interactions)
```



Alternatively, to visualize the precision-recall curve, use the `plot_pr` function. The following command plots the precision over the first 25,000 protein pairs:

```
plot_pr(interactions, max_n = 25e3)
```



This concludes the demonstration of the main functionality in the CFTK package. For more information, explore the R documentation, and please do not hesitate to contact us with any error reports at http://github.com/fosterlab/CFTK/issues (http://github.com/fosterlab/CFTK/issues).

References

1. Kirkwood KJ, Ahmad Y, Larance M, Lamond AI (2013) Characterization of native protein complexes and protein isoform variation using size-fractionation-based quantitative proteomics. *Mol Cell Proteomics* 12(12):3851–3873.

2. Tyanova S, Temu T, Cox J (2016) The MaxQuant computational platform for mass spectrometry-based shotgun proteomics. *Nat Protoc* 11(12):2301–2319.

3. Giurgiu M, et al. (2019) CORUM: The comprehensive resource of mammalian protein complexes—2019. *Nucleic Acids Res* 47(D1):D559–D563.