The file computer\_scripts.py contains all program codes for the creation of the gold standard, the prediction of interactions as well as evaluation of predicted results. All calculations for the data displayed in the figures is also included in the code.

The code consists of three main parts:

1. Import of used packages
2. Definition of classes used
3. Test execution for the creation of the gold standard and predictions on the four *S.cerevisiae* datasets (starting on line 9334)

Further the classes and their dependencies are described in more detail. Figure 1 displays dependencies of the used classes.

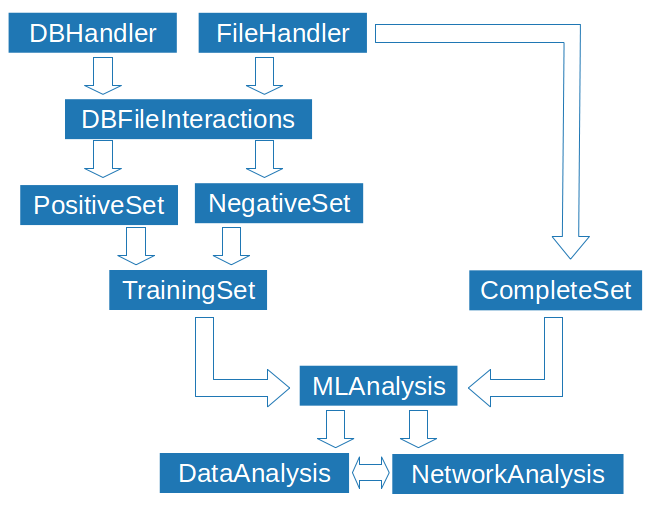


Figure 1: Schematic representation of the object oriented programmatic workflow. Different python classes were created and used analyse the data. Boxes symbolize classes, arrows describe data transfer.

# PrepareData()

Since the supplemental datasets provided include too many tables loading this file into the memory for all further calculations becomes inefficient. The class PrepareData creates a folder structure and extracts the raw datasets for a given organism into separate files.

## create\_folders(self, appendix)

This function prepares the folder structure for all further analyses in the current working directory. A subfolder of the working directory should contain the computer\_scripts.py. The function takes as input an appendix to all folder names describing the chosen experiment (e.g. “\_yeast/“) and creates (if not existent) following folders:

* „databases“ including all databases used across different experiments (per organism)
* „databases+appendix“ including databases trimmed to the proteins/metabolites present in the datasets as well as gold standard subsets used in the current experiment
* „experimental\_data+appendix“ including all datasets for the current experiment
* „analyses+appendix“ for all calculated analyses and predictions

## selectfile(self)

If the excel file „supplemental\_datasets.xlsx“ is not provided in the function prepare\_expdata, this function opens a graphic user interface to select the file.

## prepare\_expdata(self,organism,supplemental\_datasets)

For all following calculations to create the gold standard and set of metabolite-protein pairs to make predictions on the excel files with raw datasets in the folder „experimental\_data+appendix“ are used. This function fills this folder previously created with create\_folders by extracting all datasets of interest (provided by the argument „organism“) form the file supplemental\_datasets (file path provided by the argument „supplemental\_datasets“).

## prepare\_databases(self,supplemental\_datasets)

This function provides descriptions to the user about which raw databases need to be put into the folder „databases“ and extracts supplemental dataset S10 into the folder „databases“ providing manually curated translations between Stitch and PubChem CIDs and metabolite names.

# OverwriteException(Exception):

This class only has an initiator and is used to create exceptions in try-except statements when overwriting of an existing file is requested by the user.

# IDTranslations()

This class contains methods used by every other class such as translations of different protein and metabolite IDs.

## get\_expfiles(self)

collects and returns the names of all excel files in the folder self.experimental used in the current experiments. If none are found it opens a user interface to select manually the folder with experimental datasets.

## ids\_to\_string(self,ids)

for a given set or list of IDs it appends them to one string which is then returned.

## translate\_proteins(self,ids,in\_id\_type,out\_id\_types,save)

Translates a list or set of protein IDs via the UniProt API (see https://www.uniprot.org/help/api\_idmapping) from the in\_id\_type to the IDs specified by the argument out\_id\_types (list of strings) and returns a dataframe with the specified IDs. If save is specified by a string, the dataframe is saved to the file path specified by save.

## padd\_cids(self,cids)

metabolite IDs used in the stitch database have eight digits, with zeros padded in front. For some comparisons of ids this function padds provided cids with zeros in front so that all CIDs consist of at least eight digits and are of type string, which are returned as a list.

## translate\_metabolites\_pubchem(self,metabolites)

This function translates metabolite names to PubChem CIDs with the pubchem REST API. It takes as input a list of metabolite names or if None provided the metabolites from the class method get\_metabolites() and returns a dataframe providing the translations.

## translate\_cids\_pubchem(self,cids)

The counterpart to translate\_metabolites\_pubchem. A list of PubChem CIDs is translated to metabolite names and returned as a dataframe.

## translate\_metabolites\_offline(self,metabolites,online\_translation)

translates a list of metabolite names to Stitch CIDs as CID and PubChem CIDs with the translations provided by the supplemental dataset S10 and returns a dataframe with translations. The commented part is based on a more extented manually curated database with ID translations, where not translated metabolites are translated via the pubchem REST API if online\_translation is selected as True. This part will print not translated metabolites and translate them via PubChem. PubChem CIDs and Stitch CIDs do not always coincide.

## translate\_CIDs\_offline(self,cids)

Uses supplemental dataset S10 to translate given list of Stitch CIDs to metabolite names and returns a dataframe.

## translate\_CIDs\_from\_Stitch\_to\_PubChem(self,cids)

Translates given list of cids (Stitch CIDs) to PubChem CIDs and returns a dataframe

# DBHandler(IDTranslations)

Class that inherits all methods from IDTranslations and provides all communication with the Stitch database to extract interactions used for the positive subset of the gold standard.

## \_\_init\_\_(self,orgstring,score\_cutoff,overwrite,simulation,experimental,databases,analyses)

self.experimental is folder with raw datasets

self.databases is folder with databases specific for the experiment such as gold standard subsets

self.score\_cutoff is confidence threshold used to extract interactions from the Stitch database (default =800, equalling 80% confidence)

self.overwrite: if set to True, it will overwrite existing files

self.simulation: if set to True, will take as input manually curated data to test code.

self.organism corresponds to a excel file including the organism name

self.file refers to the Stitch database filtered to interactions above self.score\_cutoff

Further the init function will collect proteins and metabolites present in the database

## load\_raw\_db(self)

Loads the Stitch database for the organism in self.organism. Prior the database needs to be downloaded manually and put into the folder „databases“.

## compress\_db(self,db)

Takes as input the Stitch database and extracts interactions above self.score\_cutoff and savest he resulting trimmed database to self.file.

## extract\_meta\_and\_prots(self)

Reads all proteins and metabolites from the trimmed Stitch database (from compress\_db) and puts them into the sets self.proteins and self.metabolites, respectively.

## extract\_intersactions(self)

Checks whether the Stitch database trimmed to the self.score\_cutoff exists, and creates it if not by calling first self.load\_raw\_db and self.compress\_db.

## get\_metabolites(self)

Returns self.metabolites.

## get\_proteins(self)

Returns list of self.protein UniProt IDs.

## open\_cog\_db(self):

Opens and returns the STRING database for ID mappings to orthologous groups (COGs) as dataframe. If the file is not present in the “databases” folder, it opens a user interface to select the database.

## get\_orthologs(self,string\_ids,save)

Creates or if already existent loads and returns a dataframe collecting COGs to protein STRING IDs provided by the argument string\_ids or present in the the Stitch database compressed to self.score\_cutoff. This function calls self.open\_cog\_db to lo load the COG mappings. If the argument save is set to True it will save the dataframe together with translated UniProt IDs (translated with the function IDTranslations.translate\_proteins to the folder “databases” with file name self.organism till the first underscore “\_” plus “\_string\_uniprot\_cog.tsv” (e.g. “S.cerevisiae\_string\_uniprot\_cog.tsv”).

## find\_metabolites(self,df\_string\_cogs)

For the dataframe created with self.get\_orthologs provided by the argument df\_string\_cogs this method includes metabolites with interactions in the Stitch database compressed to self.score\_cutoff to the corresponding proteins.

## trim\_db\_to\_YMDB\_containing\_metabolites(self)

This function trims the Stitch database compressed to self.score\_cutoff to only natively in yeast occurring metabolites according to the Yeast Metabolome Data Base (YMDB) by loading the manually downloaded ymdb database to the folder “databases” and translating the InChI keys to PubChem CIDs. Not used in the creation of the gold standard.

## make\_STITCH\_predictions\_file(self)

This function is not finished yet, but intended to compare the predictions from SLIMP with the original Stitch database. It transforms the Stitch database to a format as the SLIMP predictions, by setting interactions above self.score\_cutoff to True and below to False.

## annotate\_STITCH\_predictions(self)

Idea of a function classifying metabolites and proteins in the Stitch database to compare predicted networks and enriched interaction tendencies.

## helpfun()

Prints usage of the program script if run as a separate file directly from the terminal.

# FileHandler(IDTranslations)

Class inheriting methods from IDTranslations that handles communication with the excel files containing experimental data as well as data preparation such as pooling of profiles from different repetitions.

## \_\_init\_\_(self,inputfile,experimental,databases)

self.experimental is folder with raw datasets

self.databases is folder with databases specific for the experiment such as gold standard subsets

self.inputfile refers to current excel file corresponding to one dataset

self.file is the pandas.ExcelFile object for self.inputfile retrieved by calling self.load\_file

self.organism refers to the input file name without path and .xlsx

If no inputfile is provided, the method self.selectfile is called to select an inputfile

## selectfile(self)

Opens a graphic user interface to select an excel file with a raw experimental data set.

## load\_file(self)

Returns a pandas.ExcelFile object of the inputfile

## get\_sheets(self)

Returns the sheet names of self.inputfile

## load\_sheet(self,sheet,\*\*kwargs)

Returns the excel sheet from self.file with the name provided by the argument sheet as pandas dataframe, with \*\*kwargs other options can be passed to pandas ExcelFile.parse(), e.g. specifying datatypes of the sheet columns.

## load\_proteins(self)

Loads all protein UniProt IDs present in the dataset as a set into self.proteins.

## load\_metabolites(self)

Loads all metabolite Stitch CIDs present in the dataset into self.metabolites.

## get\_metabolites(self)

Checks whether the class object already has the attribute self.metabolites, if not creates it by calling self.load\_metabolites and returns self.metabolites.

## get\_proteins(self)

Checks whether the class object already has the attribute self.proteins, if not creates it by calling self.load\_proteins and returns self.proteins.

## load\_prot\_profiles\_and\_split\_into\_reps(self,profiles)

Loads the excel sheet with the protein elution profiles per repetition and returns a dataframe with the protein sheet and a list with dataframes for all repetitions. This function excludes missing and multiple times occurring IDs and returns profiles for every UniProt ID separately, it also counts and prints the number of multiple times occurring IDs and missing IDs. The argument profiles can be adjusted to load deconvoluted profiles from the PROMIS experiment.

## load\_met\_profiles\_and\_split\_into\_reps(self,profiles)

Loads the excel sheet with the metabolite elution profiles per repetition and returns a dataframe with the metabolite sheet and a list with dataframes containing elution profiles for all repetitions. The argument profiles can be adjusted to load deconvoluted profiles from the PROMIS experiment.

## extract\_prot\_profiles(self,xset,xset\_profiles,method,profiles,normalized)

Calls the class method self.load\_prot\_profiles\_and\_split\_into\_reps to load the protein profiles specified by the argument “profiles” per repetition and pools them with the method provided by the argument “method” over repetitions. The argument xset provides the proteins to load and for experiments across organisms which proteins belong to one orthologous group and need to be pooled. xset\_profiles is the resulting and returned dataframe, which is passed as an argument while this function adds profiles from the current experimental dataset. Each elution profile is normalized as specified by the string argument normalized (e.g. to its maximum peak or the sum of the profile); the string must be a method for a pandas dataframe.

## extract\_met\_profiles(self,xset\_profiles,method,profiles,normalized)

Calls the class method self.load\_met\_profiles\_and\_split\_into\_reps to load the metabolite profiles specified by the argument “profiles” per repetition and pools them with the method provided by the argument “method” over repetitions. xset\_profiles is the resulting and returned dataframe, which is passed as an argument while this function adds profiles from the current experimental dataset. Each elution profile is normalized as specified by the string argument normalized (e.g. to its maximum peak or the sum of the profile); the string must be a method for a pandas dataframe. If a metabolite is occurring more than once or includes NaNs its ID is printed to the console to be checked on manually, else an empty set (“set{}”) is printed.

## rv\_coeff(self,repetitionlist)

Calculates the RV-coefficient between the matrices passed in the list argument repetitionlist and returns the resulting RV-coefficient.

## profiles\_correlation\_reps(self,profiles)

Calls self.load\_prot\_profiles\_and\_split\_into\_reps and self.load\_met\_profiles\_and\_split\_into\_reps to get experimental repetitions for protein and metabolite profiles, respectively. Then the function excludes columns containing NaNs and calls self.rv\_coeff to calculate RV-coefficients for protein and metabolite profiles. The resulting RV-coefficents are returned as a list (first entry for proteins, second for metabolites).

## profiles\_correlation\_ogs(self,xset,profiles,method)

For a given dataframe with COGs and their included protein UniProt IDs, this function calculates the correlation of protein elution profiles within an orthologous group and returns a dataframe with all correlations which can be further displayed in a histogram. Additionally as second output this function returns a dataframe containing the number of proteins present in the datasets per orthologous group. The argument profiles defines which profiles need to be loaded from the excel sheet (default “raw”) and the argument method defines the method how protein elution profiles are pooled over repetitions.

## profiles\_correlation\_proteins(self,proteins,profiles,method)

For a given list of protein UniProt IDs this function calculates the correlation between their elution profiles pooled over repetitions by the argument “method”. “profiles” defines the type of profiles loaded from the excel sheet.

## average\_meta\_and\_prot\_profiles(self,ogs,metabolites,profiles,normalized)

Averages elution profiles over repetitions for a list of metabolites (argument “metabolites”) and for proteins belonging to orthologous groups defined by the dataframe argument “ogs”. This function was used to observe the coverage of all SEC fractions. The function returns an average protein and average metabolite profile.

## change\_excel\_sheets(self)

This function was used to exclude elution profiles of proteins and metabolites with a sum of zero. It loads the excel file self.inputfile in self.experimental and overwrites the existing file.

## helpfun()

This function describes the initiation of a class object.

# DBFileInteractions()

A class to collect functions for the communication between the Stitch database and excel file experimental datasets necessary for the creation of the positive subset of the gold standard.

## \_\_init\_\_(self,expfilename,simulation,experimental,databases)

self.simulation can be set to True for testing the creation of the positive subset of the gold standard

self.experimental is the folder containing the excel files with experimental datasets

self.databases is the folder with experiment specific databases

self.expfile is the FileHandler obect to the path and filename of the current excel file dataset (set by calling self.get\_expfile(“expfilename”)

self.database is the DBHandler object for the organism included in “expfilename” by calling self.get\_database(“expfilename”).

## get\_expfile(self,expfilename)

Returns the FileHandler object initiated with the “expfilename” and passing the class attributes.

## get\_database(self,expfilename)

Returns a DBHandlder object by passing class attributes and “expfilename”

## intersect(self,dbset,expdataset)

Returns the intersection of the two sets “dbset” and “epdataset”.

## union(self,dbset,expdataset)

Returns the union of the two sets “dbset” and “expdataset”.

## find\_positive\_candidates(self,pooling)

This function calls the function specified by pooling (e.g. self.intersect or self.union) to find and return a list of the intersecting protein UniProt IDs between the proteins of the database (self.database.get\_proteins) and the experimental dataset (self.expfile.get\_proteins).

## extract\_and\_save\_orthologs(self,uniprot\_set,outfilename,overwrite,include\_metas)

Translates the UniProt IDs “uniprot\_set” to String IDs and gets their Orthologous Groups. If include\_metas is True, it will add interacting metabolites using self.database.find\_metabolites. The output is not returned but saved to “outfilename”. If overwrite is set to False, it will try to read the file “outfilename” and create it if not existent. Overwrite set to True will create it anyway.

# XSet(IDTranslations)

class that collect methods used by positive and negative subset of the training set as well as the set on which predictions are made.

## \_\_init\_\_(self,simulation,overwrite,experimental,databases,analyses,feature,methods,normalized,normalized2,proteinwise)

self.simulation is set to True to test the class

self.overwrite is set to True to overwrite previously created files

self.experimental is the folder containing experimental dataset excel files.

self.databases is the folder containing databases trimmed to the experimental datasets

self.analyses is the folder for made analyses on the datasets

self.feature is a string describing the feature engineering

self.normalized is a string describing the normalization of the elution profiles

self.normalized2 is a string describing the normalization of over repetition pooled profiles

self.proteinwise is set to True to work on protein-metabolite interaction and False to work on OG-metabolite interactions (if datasets derive from different organisms)

self.protcol specifies the coloumn name for the proteins/OGs depending on self.proteinwise

self.mode is a substring of self.feature specifying the calculation of the cross-correlation

self.methods is a list of methods used for profile pooling

self.expfiles is a list of experimental datasets in self.expfiles created by calling IDTranslations.get\_expfiles

## set\_feature(self,feature)

Sets self.feature to the argument “feature”.

## find\_orthologs\_between\_files(self)

For all experimental datasets in self.expfiles this function reads the file mapping COGs to String IDs or if not existent calls DBHandler.get\_orthologs to create it and returns union of all COGs across the datasets.

## meta\_intersect\_files(self)

Returns the intersection of all metabolites across all datasets in self.expfiles.

## prot\_intersect\_files(self)

Returns the intersection of all proteins across all datasets in self.expfiles.

## meta\_union\_files(self)

Returns the union of all metabolites across all datasets in self.expfiles.

## save\_set(self,tabfile,df)

Abstract method to generate and save the final set, which is further defined by inheriting classes.

## load\_set(self,tabfile,method)

Tries to load the set specified by “tabfile” and if not present or requested by self.overwrite calls self.save\_set to create it. The final set is returned as a dataframe.

## collect\_uniprot\_IDs(self,xset,predictions)

Writes a text file containing all UniProt IDs from the either passed “xset” (dataframe with COGs as index and UniProt IDs for datasets as columns) or “predictions” (dataframe with multi-index consisting of protein and metabolite). This file can be used on uniprot.org to query EC numbers and Gene Ontology terms.

## retrieve\_EC\_and\_GO(self)

Prints instructions on how to retrieve EC numbers and GO-Terms to annotate proteins for further analysis.

## classify\_proteins(self,uniprot\_ids,org)

For a given KEGG organism identifier provided by the argument “org” this function translates the UniProt\_IDs “uniprot\_ids” to KEGG IDs and classifies them to BRITE classes and KEGG pathways using the KEGG API.

## assign\_classes\_and\_pathways\_to\_OGs(self,complete\_set,kegg\_df)

“complete\_set” is a dataframe with COGs as index and columns with UniProt IDs for every dataset in the experiment. With the KEGG IDs provided by “kegg\_df” this function assigns to every COG the intersecting pathways and BRITE classes which are returned as a dataframe.

## assign\_EC\_numbers\_to\_OGs(self,complete\_set)

Takes as input the “complete\_set” returned by self.assign\_classes\_and\_pathways\_to\_OGs to annotate intersecting and union EC numbers to COGs.

## find\_pathways\_metabolites(self,kegg\_ids)

Uses the KEGG API to annotate pathways to metabolite KEGG IDs (argument “kegg\_ids”)

The resulting dataframe is saved to self.analyses + metabolite\_kegg\_pathways.tsv and returned.

## retrieve\_CID\_classes(self,cids)

Queries the CIDs provided by “cids” on the PubChem REST API to retrieve metabolite classes, which are saved to self.analyses + metabolite\_classifications.tsv and returned by the dataframe. The resulting dataframe needs to be checked manually.

## stack\_profiles(self,xset,method,profiles)

Iterates through experimental datasets as FileHandler objects and concatenates the via “xset” provided protein and metabolite elution profiles pooled with FileHandler.extract\_prot\_profiles and FileHandler.extract\_met\_profiles. The function returns a dataframe with multi-index for protein/OG and metabolite and the corresponding profiles.

## normalize\_concat\_profiles\_per\_org(self,xset\_profiles)

Normalizes the pooled and concatenated profiles from self.stack\_profiles as defined by self.normalized2 and returns them.

## average\_meta\_and\_prot\_profiles\_per\_file(self,xset,profiles)

For a given dataframe “xset” with COG index and protein UniProt IDs per dataset column and a coloumn with interacting CIDs this function calls per experimental dataset FileHandler.average\_meta\_and\_prot\_profiles to create an average protein and an average metabolite profile to inspect fractional coverage.

## refeaturize\_x\_set(self,x\_set)

For a given set with elution profiles and multi-index for protein/OG – metabolite pairs this function performs the feature engineering as specified by self.feature and returns the feature engineered dataframe.

# XSet\_sampled(XSet)

Class that creates subsets of gold standard by randomly drawing profiles for training to select those always predicted the same.

## \_\_init\_\_(self,x,y\_set,balanced,simulation,overwrite,experimental,databases,analyses,feature,normalized,normalized2,proteinwise)

Passes simulation, overwrite, experimental, databases, analyses, feature, normalized, normalized2 and proteinwise to the XSet constructor.

self.x defines the subset of the gold standard, e.g. “positive”

self.balanced: set to True if the metabolites in the negative subset should also be present in the positive subset and vice versa.

self.y\_set is the opposite gold standard subset

## sample\_xs(self,remaining\_set,n\_samples)

Randomly draws “n\_samples” profiles from “remaining\_set” and returns them as dataframe of the same shape as remaining\_set.

## rf\_clf(self)

Function returning a sklearn RandomForestClassifier object.

## svm\_clf(self,C=1,kernel,param)

Function returning a sklearn SVC object.

## train\_to\_get\_xs(self,n\_trainings,classifiers,method)

Trains classifier several times with sampled negative/positive (self.x) training data and returns consistently predicted as “x”. “n\_training” defines the number of trainings and “classifiers” is a list of classifiers to train on.

## plot\_n\_trainings(self,n\_vec,save,method)

Plots and saves a figure displaying the size of the “x”-subset of the gold standard over the amount of trainings, showing if the number of samplings is sufficient.

## save\_set(self,tabfile,n\_trainings,method)

Reads the set of all protein-metabolite pairs and creates a sampled training subset by calling self.train\_to\_get\_xs passing “n\_trainings” and “method” and saves the resulting dataframe to “tabfile”.

## intersect\_xs\_from\_sampling\_and\_db(self,method)

creates and characterizes the intersection between the sampled intersecting “x” subset of the gold standard and the database derived set.

# NegativeSet(XSet)

## \_\_init(self,simulation,overwrite,experimental,database,analyses,feature,db\_organism,normalized,normalized2,proteinwise)

Passes the arguments to the constructor of XSet except of db\_organism.

self.db\_organism is a tuple of the STRING organism identifier as string and the organism name

## open\_cog\_db(self)

Returns the STRING database providing COG mappings to proteins. If not found in the folder “databases” it will open a file dialog.

## get\_orthologs\_in\_human(self,cogs\_set)

For a given set of COGs “cogs\_set”, or if not provided the set from XSet.find\_orthologs\_between\_files. this function finds STRING IDs for the organism specified by self.db\_organism by querying the “cogs\_set” in the STRING database from self.open\_cog\_db.

## collect\_all\_string\_ids(self,df)

Takes as input the dataframe from self.get\_orthologs\_in\_human or if none provided calls self.get\_orthologs\_in\_human and adds STRING IDs from experimental datasets into one dataframe, which is then returned.

## translate\_string\_df\_to\_uniprot(self,df\_string)

This function translates the dataframe “df\_string” from self.collect\_all\_string\_ids to UniProt\_IDs and returns it.

## find\_nonint\_cids\_in\_pubchem(self,uniprot\_ids)

Queries a list of UniProt IDs “uniprot\_ids” on pubchem bioassays to find not interacting CIDs and returns a dataframe with the UniProt IDs and PubChem CIDs.

## find\_CIDs(self,df\_uniprot)

For a given Dataframe with UniProt IDs for COGs, or if not provided loaded from self.translate\_string\_df\_to\_uniprot. this function calls find\_nonint\_cids\_in\_pubchem for every protein from self.db\_organism per orthologous group to retrieve not interacting CIDs. Only CIDs are considered which are found for all proteins belonging to an orthologous group. The dataframe is saved to self.databases as “neg\_uniprot\_og\_cid” +self.db\_organism[1] + “.tsv” and returned.

## padd\_df(self,df\_uniprot)

For a given dataframe “df\_uniprot” as returned by self.find\_CIDs this function pads zeros in front of every CID to translate them to Stitch CIDs.

## meta\_intersect\_bioassay\_files(self,df\_uniprot)

Translates the PubChem CIDs in the dataframe “df\_uniprot” as returned by self.find\_CIDs to Stitch CIDs and excludes those not present in the intersecton over experimental datasets. This function returns a dataframe.

## proteinwise\_x\_set(self,x\_set,method,corr\_thresh)

From the dataframe with not interacting COG-CID mappings this function returns protein-CID mappings. If the elution profiles of proteins correlate more than provided by “corr\_thresh”, only one protein is considered (else the gold standard subset would be over represented by similar profiles and therefore biased). The argument “x\_set” provides COG-protein mappings and “method” describes the pooling method for combination of profiles from repetitions.

## save\_set(self,tabfile,df,method)

This function calls if “df” is not provided self.meta\_intersect\_bioassay\_files with a given profile pooling method “method”, saves it to “tabfile” and returns the set as a dataframe

# PositiveSet(XSet)

Class to construct the positive subset of the gold standard (interacting protein-metabolite pairs).

## get\_orthologs\_between\_species(self)

Calls DBFileInteractions.find\_positive\_candidates and DBFileInteractions.extract\_and\_save\_orhtologs for every experimental data set to find all COGs across the datasets, which are returned as a set.

## get\_metabolites\_for\_all\_orthologs(self,orthologs)

For a given set of COGs or if not provided retrieved from self.get\_orthologs\_between\_species this function gets interacting CIDs from the Stitch database for all present STRING IDs, saves them to self.databases+"positive\_set\_raw.tsv" and returns the dataframe. This is used if self.proteinwise is False as in analysis over datasets from different species.

## protwise\_pos\_set(self)

Creates and returns a dataframe with protein (STRING ID) – metabolite interactions present in the Stitch database. This is used if self.proteinwise is True for analysis within a species.

## filter\_stereospecific\_cids(self,metabolites,df)

Filters the dataframe obtained from self.protwise\_pos\_set or self.get\_metabolites\_for\_all\_orthologs to stereospecific metabolites present in the intersection over datasets and characterizes, saves and returns a dataframe with interacting CIDs and protein STRING IDs.

## trim\_biolip\_to\_orglist(self,organisms)

Under Construction. For a given list of organisms. this function is intended to query the BioLiP database for interacting protein-metabolite pairs to extend the positive subset of the gold standard. The function was not finished, because most of the ligands found were not biologically relevant.

## save\_set(self,tabfile,df,method)

For a given dataframe “df” or if not provided retrieved from self.filter\_stereospecific\_cids) this function builds, saves to “tabfile” and returns a dataframe with UniPot IDs and interacting Stitch CIDs.

# PositiveSet\_PubChem(NegativeSet)

Constructs a positive subset of the gold standard by querying PubChem bioassays for interacting protein-metabolite interactions similar as done by NegativeSet, which can be merged with the positive set retrieved from PositiveSet.

## find\_int\_cids\_pubchem(self,uniprot\_ids)

For a given list of uniprot IDs this function queries PubChem bioassays to retrieve interacting metabolites which are returned as a dataframe.

## find\_CIDs(self,df\_uniprot)

Fills a dataframe with UniProt IDs for orthologous groups as returned by NegativeSet.translate\_string\_df\_to\_uniprot with interacting CIDs cia self.find\_int\_cids\_in\_pubchem and saves and returns the dataframe to self.databases+"pos\_uniprot\_og\_cid\_"+self.db\_organism[1]+".tsv".

## save\_set(self,tabfile,df,method)

Calls, if not provided by “df”, self.meta\_intersect\_bioassay\_files to create and return the dataframe with the interacting proteins and metabolites and saves it to “tabfile”.

# TrainingSet(XSet)

Class to join negative and positive subsets to the gold standard (training set).

## \_\_init\_\_(self,db\_orglist,simulation,overwrite,experimental,databases,analyses,feature,methods,normalized,normalized2,proteinwise)

Passes arguments to the constructor of XSet and sets self.db\_orglist to db\_orglist, which provides a list of tuples with organism identifiers and names used for the creation of the training subsets retrieved from PubChem.

## merge\_positive\_set\_with\_pubchems(self)

Merges the positive sets from PositiveSet and from PositiveSet\_PubChem to one and extracts, normalizes and performs feature engineering on the corresponding elution profiles. The final set is saved to self.databases.

## merge\_negative\_set\_with\_pubchems(self,feature)

Merges the negative sets from NegativeSet for different organisms to one, normalizes elution profiles and performs feature engineering on them to create the final negative subset of the gold standard.

## load\_merged\_x\_set(self,method,x,feature)

Tries to read the “x” set (either positive or negative) and if not present calls self.merge\_positive\_set\_with\_pubchems or self.merge\_negative\_set\_with\_pubchems to create it and returns the subset of the gold standard.

## load\_x\_set(self,method,x,metabolite\_extension,manual\_selection,confidence)

Calls self.load\_merged\_x\_set to load and return the subset of the gold standard specified by “x” (either positive or negative) passing “method”. Further if manual\_selection is True it trims the set to manually selected interactions by calling self.trim\_x\_set\_by\_manual\_selection passing “confidence” and if metabolite\_extension is set to True it expands the set by calling self.expand\_xset\_with\_similar\_metabolites.

## get\_unbalanced\_training\_set(self,method,extend\_with\_sampling,metabolite\_extension,manual\_selection,confidence)

Merges the negative and positive subsets of the training set by calling self.load\_x\_set passing the arguments. If “extend\_with\_sampling” is set to True, this function creates a XSet\_sampled object to extend the smaller subset. Both subsets are trimmed to the size of the smaller subset such that they become equal size. This function also calls self.split\_training\_set\_into\_xsets, returns and saves the training set.

## balance\_training\_set(self,positive\_set,negative\_set,balance\_to)

Balances the training set from its subsets “positive\_set” and “negative\_set” to the same metabolites as present in “balance\_to” set.

## get\_balanced\_db\_set(self,method,metabolite\_extension,manual\_selection,confidence)

Merges the negative and positive subsets of the training set by calling self.load\_x\_set passing the arguments. Both subsets are trimmed to the size of the smaller subset such that they become equal size and protein-metabolite pairs are present in both subsets such that every metabolite is present the same amount in each subset. This function also calls self.split\_training\_set\_into\_xsets, returns and saves the training set.

## get\_balanced\_sampled\_set(self,method,metabolite\_extension,manual\_selection,confidence)

Merges the negative and positive subsets of the training set by calling self.load\_x\_set passing the arguments. If “extend\_with\_sampling” is set to True, this function creates a XSet\_sampled object to extend the smaller subset. Both subsets are trimmed to the size of the smaller subset such that they become equal size and protein-metabolite pairs are present in both subsets such that every metabolite is present the same amount in each subset. This function also calls self.split\_training\_set\_into\_xsets, returns and saves the training set.

## split\_training\_set\_into\_xsets(self,training\_set,approach,method)

Splits the “training\_set” into positive and negative subsets which are saved as well as the OG-protein mappings if self.proteinwise is False. “approach” defines the appendix how to save the sets.

## extract\_antibalanced\_set(self,xset,training\_set)

Returns “training\_set” without “xset”.

## metabolite\_profile\_correlation(self,method,threshold,plot)

Finds similar metabolites by the criteria of correlated elution profiles with correlation higher “threshold” and returns two dataframes (one with the correlations and one with the metabolite IDs and names).

## metabolite\_profile\_correlation\_all(self,colormap1,colormap2,method)

Creates a heatmap with profile correlations and tanimoto coefficients among all metabolites in the experiment.

## metabolites\_tanimoto\_index(self)

Provides the user with instructions on how to get tanimoto coefficients for the metabolites of the experiment.

## get\_coeluting\_similar\_metabolites(self,T\_thresh,corr\_thresh)

Saves metabolite to self.databases+"similar\_metabolites.tsv" which show profile correlation above corr\_thresh and tanimodo coefficients above T\_thresh.

## expand\_xset\_with\_similar\_metabolites(self,xset,method)

Adds interactions to the training set for similar metabolites according to self.get\_coeluting\_similar\_metabolites.

## plot\_protein\_metabolite\_overlay(self,method,metabolite\_positive\_set,protein\_positive\_set,metabolite\_negative\_set,protein\_negative\_set,separators,savefigs,plot\_feature\_engineered)

Plots protein and metabolite profiles overlayed to trim the gold standard. As inputs the positive and negative training subsets separated by protein and metabolite profiles are given (metabolite\_positive\_set, protein\_positive\_set, metabolite\_negative\_set,protein\_negative\_set) and “separators” defining the positions of vertical lines between experiments. If these five arguments are not provided, the training subsets are loaded via self.load\_merged\_x\_set. If “plot\_feature\_engineered” is set to True, the feature engineered profiles are loaded and displayed.

## save\_pair(self,confidences,method,savefigs,plot\_feature\_engineered)

Saves chosen index from self.pos\_index to self.analyses+”/overlay\_plots/positive\_indices\_”+method” +"\_"+self.normalized+self.normalized2+"profiles\_"+confidence+"\_confidence.txt" If “savefigs is set to True, the figure self.f is also saved to self.analyses+”overlay\_plots/”. If “plot\_feature\_engineered is set to False, the negative index self.neg\_index is also saved. “confidences” is a list of strings, in such way different gold standards can be created simultaneously with different confidences about the trimming.

## trim\_x\_set\_manually(self,x,method,x\_set,savefigs)

“x” describes the subset which is trimmed and passed to self.load\_merged\_x\_set if the set “x\_set” is not provided. This function opens a GUI setting confidence and calling self.plotloop and self.save\_pair to loop through the subset and save via the GUI selected indices.

## plotloop(self,x\_set,positive\_set,negative\_set,method,metabolite\_positive\_set,protein\_positive\_set,metabolite\_negative\_set,protein\_negative\_set,separators,savefigs,plot\_feature\_engineered)

Loops through the “x\_set”, sets self.pos\_index and self.neg\_index and calls self.plot\_protein\_metabolite\_overlay to plot the overlaying protein and metabolite profiles.

## trim\_x\_set\_by\_manual\_selection(self,x,method,x\_set,confidence,mean\_as\_default)

Loads self.analyses+"overlay\_plots/"+x+"\_indices\_"+”method"+\_+self.normalized+self.normalized2+profiles\_+confidence+\_confidence.txt or if not present first calls self.trim\_x\_set\_manually and returns the trimmed subset.

## plot\_training\_set(self,method,indices,num\_profiles,save)

Plots selected indices of positive and negative training subset specified by the list argument “indices”,

## distribution\_of\_maxcrosscorr(self,method)

For a given profile pooling method “method” this function shows a histogram of the maxima of crosscorrelation feature vectors for positive and negative set among experiments to estimate dimensions.

## fig\_pmrelation\_example(self,pos\_index,neg\_index,method,feature)

For a given positive and negative index “pos\_index” and “neg\_index” this function plots and saves a figure with the overlays for positive and negative subset as well as their feature engineered profile (like the figure in the manuscript).

# CompleteSet(XSet)

Class to build the set to make predictions on (all protein-metabolite pairs present in the intersection over all experimental datasets)

## get\_orthologs\_between\_species(self)

Assigns orthologous groups to all proteins over the datasets via DBFileInteractions.extract\_and\_save\_orthologs for all experimental datasets.

## get\_proteins\_for\_all\_orhtologs(self,orthologs)

For the set of all orthologous groups retrieved via self.get\_orthologs\_between\_species this function assigns proteins from the experimental data, saves and returns the dataframe.

## create\_complete\_set\_profiles(self,complete\_set)

In the case of self.proteinwise = False this function loads the COG-protein mappings via self.load\_set and pairs the COGs with metabolites present in the intersection over all experimental datasets. In the case of self.proteinwise = True it pairs the proteins from self.prot\_intersect\_files with the metabolites from self.meta\_intersect\_files.

## save\_set(self,tabfile,df,method)

Translates STRING IDs to UniProt IDs and saves and returns the set of protein/COG-metabolite pairs.

# MLClassifiers(IDTranslations)

Class to perform supervised machine learning on the complete set with the gold standard from TrainingSet and evaluate predictions.

## \_\_init\_\_(self,experimental,databases,analyses,approach,feature,normalized,proteinwise)

self.experimental defines the folder containing experimental dataset files

self.databases defines the folder containing experiment specific databases as well as gold standard subsets

self.analyses defines the folder containing results from the analyzed predictions as well as predictions

self.approach defines the gold standard

self.feature defines the feature engineering procedure

self.normalized is a string containing the two methods for normalization of raw and pooled profiles.

self.proteinwise is set to True for analysis of protein-metabolite interactions and set to False for COG-metabolite interactions as used if experimental datasets come from different organisms

self.protcol defines the coloumn name specific due to self.proteinwise

self.expfiles is the output from IDTranslations.get\_expfiles

## load\_known\_interactions(self,overwrite,method,profiles,trim,random\_indices,filtered\_ogs)

Reconstructs a training set for selected COGs if “filtered\_ogs” is provided and returns the gold standard.

## load\_known\_set(self,method,randomized,complete\_randomized,filtered\_ogs)

Loads and returns the training set. “randomized” set to True randomly shuffles labels, complete\_randomized set to True randomly draws profiles with random assigned labels from the complete set and if “filtered\_ogs” is provided as a list the gold standard is trimmed with self.load\_known\_interactions.

## rf\_clf(self)

Returns a sklearn RandomForestClassifier object.

## svm\_clf(self,C,kernel,degree,param)

Returns a sklearn SVC object passing the arguments.

## fit\_with\_training\_and\_test\_data(self,clf,known\_set,training\_size)

Splits the gold standard “known\_set” into training and test data as specified by “training\_size” without cross-validation and fits the classifier specified by “clf”. This function returns the precision, recall and F1 score.

## fit\_and\_test\_trainset(self,clf,known\_set,training\_size)

Splits the gold standard “known\_set” into training and test data as specified by “training\_size” without cross-validation and fits the classifier specified by “clf”. This function returns accuracy.

## fit\_with\_cross\_validation(self,clf,known\_set,num\_kfolds,train\_size,test\_size)

Splits the gold standard “known\_set” into training and test data as specified by “training\_size” or “test\_size” with cross-validation (“num\_kfolds” defines the k-fold validation) and fits the classifier specified by “clf”. This function returns the precision, recall, F1 score, accuracy, sensitivity and specificity.

## measure\_table(self,methods,calssifiers,reps)

Creates a summary table with performance measures retrieved viea self.fit\_with\_cross\_validation for different pooling methods “methods” and different classifiers as specified by the list “classifiers”. “reps” defines the number of repetitions which are averaged to obtain performance measures. The result is returned and saved to self.analyses+"analysis\_summary"+self.feature+self.approach+".tsv".

## trim\_known\_set(self,known\_set,trim)

Trims the gold standard “known\_set” to a smaller set as specified relatively by “trim” via randomly drawing profiles from the positive and negative subset.

## create\_learning\_curves(self,methods,classifiers,training\_sizes,plot,filtered\_ogs,randomized,complete\_randomized,reps,appendix,legend,title,len\_trainset,prob\_int)

Plots and saves learning curves for different pooling methods “methods” and classifiers “classifiers”. “training\_sizes” is a list with relative sizes of the training set. “plot” set to True displays the learning curves before saving. “filtered\_ogs” trims the training set using self.load\_known\_interactions, “randomized” and “complete\_randomized” define whether to use a randomized training set. “reps” defines the number of repetitions used to average performance (e.g. for randomization). The output ist saved to self.analyses.

## create\_learning\_ROC\_curves\_comparison(self,method,classifier,training\_sizes,reps,appendix,test\_size)

Creates a figure with learning and ROC curve on the same test set for random and trained classifiers. The figure as well as dataframes with the underlaying data are saved to self.analyses.

## my\_roc\_curve(self,test\_res,test\_scores)

Function to calculate ROC curves, takes as input a list of true results on the test set and test scores from the classifier and returns a list of false positive rates, true positive rates and thresholds used to calculate them.

## roc\_auc(self,methods,classifiers,training\_size,len\_trainset,reps,appendix,plot,legend,randomized,complete\_randomized,prob\_int)

Calculates and plots ROC curves and area under the curves using self.my\_roc\_curve for different profile pooling methods “methods”, different classifiers “classifiers”. “training\_size” specifies the relative size gold standard assigned to the training set, “len\_trainset” the absolute size of the training set used for sampling, “reps” the number of repetitions to average the ROC curves, “appendix” a suffix added to the saved figure, “plot” set to True displays the figure, “legend” set to True displays a legend, “randomized” and “complete\_randomized” specify randomization procedure if not trained on the true gold standard. “prob\_int” specifies the probability for an interaction used for randomized gold standard.

## corr\_rep(self,overwrite)

Constructs and saves a figure displaying the RV-coefficients between repetitions as retrieved b FileHandler.profiles\_correlation\_reps.

## count\_og\_assignments(self)

Counts for every protein in every experimental dataset the number of assigned orthologous Groups, saves and returns a dataframe.

## corr\_og\_hist(self,xsets,method,profiles,appendix)

Creates a figure with histograms showing frequency of Pearson correlation of protein profiles per COG by calling FileHandler.profiles\_correlation\_ogs, pie charts with number of proteins per COG and pie chart with number of OGs annotated per protein by calling self.coung\_og\_assignments for the given list of dataframes with protein/COG-metabolite profiles “xsets”. The figure is displayed and saved to self.analyses +"corr\_ogs\_histograms\_"+appendix+"og.png".

## merge\_pos\_neg\_orthos(self,method)

Returns for a given profile pooling method “method” the mapping of COGs to proteins present in the training set.

## corr\_boxplots(self,xset,profiles,method)

For a given set of profiles “xset” this function calculates boxplots with correlation of protein profiles belonging to one COG and saves the figure as self.analyses+"boxplots\_ogs\_reps"+self.approach+".pdf" as well as the underlying data to self.analyses+"correlations\_inside\_og.tsv".

## filter\_unimodal\_ogs(self,corr\_df,corr\_thresh)

Returns a list of COGs with correlation of protein profiles below “corr\_thresh” for a given “corr\_df” which if not passed is retrieved via self.corr\_boxplots(self.merge\_pos\_eg\_orthos).

## train\_with\_unimodal\_ogs(self,methods,classifiers,training\_sizes,filtered\_ogs,appendix)

Calls self.create\_learning\_curves passing all arguments. If “filtered\_ogs” not provided, retrieves it calling self.filter\_unimodal\_ogs.

## load\_prediction\_set(self,which,method)

Returns the feature engineered profiles to make predictions on. If which is not None, it takes the output of self.exclude\_underrepresented\_profiles\_from\_prediction\_data passing which and method.

## make\_predictions(self,methods,classifiers,known\_set,reps,trim)

Makes predictions on the set retrieved from self.load\_prediction\_set for all provided profile pooling methods “methods” and all classifiers “classifiers” trained on “known\_set”. If “known\_set” is not provided, retrieves the training set with self.load\_known\_set. “reps” defines the number of repetitions used for averaging predictions of random forest classifiers. The predictions are saved to self.analyses+"predictions"+trim+"\_"+method+self.feature+self.approach+".tsv".

## my\_assortativity(self,g)

For a given igraph graph “g” this function returns the assortativity of a bipartite network relative to the node number per partition.

## make\_randompredictions(self,method,classifier,reps)

Averages over “reps” repetitions random predictions for training on the gold standard with randomized labels and randomly drawn protein/COG – metabolite pairs from the set to make predictions on (from self.load\_prediction\_set) and saves them to self.analyses+"predictions\_random\_ts\_"+method+self.feature+self.approach+"\_"+classifier+".tsv" and self.analyses+"predictions\_random\_cs\_"+method.\_\_name\_\_+self.feature+self.approach+"\_"+classifier+".tsv", respectively. This function also calculates average assortativity and accuracy on the random networks, which are saved to self.analyses+"comparison\_to\_random\_ts\_cs\_assortativity\_"+method+self.feature+self.approach+"\_"+classifier+".tsv".

## make\_predictoin\_probabilities(self,methods,classifiers,reps)

Saves for the given profile pooling methods “methods” and classifiers “classifiers” the prediction probabilites from sklearn.predict\_proba to self.analyses+"prediction\_probabilities\_"+method+self.feature+self.approach+".tsv"

## rank\_proteins(self,methods,classifiers,classifier1,top,predictions,appendix)

Ranks the proteins/COGs by the highest amount of interacting metabolites for the profile pooling methods “methods” and classifiers “classifiers” sort on “classifier1” and saves them to self.analyses+self.protcol+"\_ranks\_"+method+self.feature+self.approach+appendix+".tsv". Additionally saves the highest “top” ranked proteins/COGs to self.analyses+top+"\_highest\_ranked\_"+self.protcol+"s\_"+method+classifier+appendix+".tsv"

## rank\_metabolites(self,methods,classifiers,classifier1,predictions,appendix)

Ranks the metabolites by the highest amount of interacting proteins/COGs for the profile pooling methods “methods” and classifiers “classifiers” sort on “classifier1” and saves them to self.analyses+"metabolite\_ranks\_"+method+self.feature+self.approach+appendix+".tsv".

## check\_consistency(self,methods,classifiers)

Creates and saves a figure with a histogram of predicted probability of interactions and a heatmap of correlations among ten prediction repetitions to evaluate the consistency of random forest predictions for different random forests and to get insight on the distribution of prediction probabilities.

## compare\_metabolite\_occur\_training\_pred(self,method)

Constructs a barplot comparing occurences of metabolites in positive and negative subset of the gold standard and the complete set to make predictions on.

## corr\_preds(self,methods,classifiers)

Plots and saves heatmaps of correlation among predictions from various profile pooling methods “methods” and/or classifiers “classifiers”.

## compare\_fractional\_coverage(self,profiles)

Averages elution profiles of metabolites and proteins/OGs present in the gold standard and set to make predictions on using XSet.average\_meta\_and\_prot\_profiles\_per\_file.

## pearson\_affinity(self,M):

returns the Pearson Correlation as distance metric (<https://stackoverflow.com/questions/32303217/how-to-use-pearson-correlation-as-distance-metric-in-scikit-learn-agglomerative>)

## determine\_n\_clusters\_for\_profile\_clustering(self,method,n\_clusters\_max,clustertype)

Creates and saves a scree plot to determine the number of clusters to group elution profiles. “n\_clusters\_max” is the maximum number of clusters and “clustertype one of "hierarchical distance-based", "hierarchical correlation-based" or "K-Means".

## determine\_n\_clusters\_for\_profile\_hierarchical\_clustering(self,method,trim\_last)

Creates a figure with dendrograms from hierarchical clustering to determine the number of clusters to group elution profiles.

## profile\_clustering(self,method,clustertype,n\_clusters)