PAIRUP-MS Documentation and Example
version 1.0 (11/8/2017)

Yu-Han Hsu, Harvard Medical School

Introduction

PAIRUP-MS (Pathway Analysis and Imputation to Relate Unknowns in Mass Spectrometry-based metabolite data) is a suite of computational methods for analyzing unknown signals in mass spectrometry (MS)-based untargeted metabolomics datasets. PAIRUP-MS consists of three main components: (1) a data processing pipeline for cleaning, processing, and preparing the metabolite data for downstream analyses, (2) an imputation-based approach for pairing up unknown signals across two datasets, which would enable meta-analysis of matched signals across studies, and (3) a pathway annotation and enrichment analysis framework that links the unknown signals to plausible biological functions, without needing to confirm their chemical identities.

PAIRUP-MS source code on GitHub (https://github.com/yuhanhsu/PAIRUP-MS) is split into three sub-directories, including *DataProcessing*, *ImputationMatching*, and *PathwayAnalysis*, corresponding to each of the components described above. A brief description of all scripts and an example of how to run them are provided in this documentation. In general, all input and output files are tab-delimited text files unless specified otherwise, except for plots that are stored as PDFs. The example uses randomized data generated for testing purpose only and does not capture real biological relationships.

We note that while the methods in PAIRUP-MS were designed to be run in a streamlined manner, each component can be used independently depending on the characteristics of your datasets and your analysis needs. For example, you can perform data processing using an alternative approach and still apply the matching method to your version of cleaned data. It is also possible to apply the pathway method to non-MS-based dataset without any m/z information.

Software and Hardware Requirements

All scripts were tested using R v3.2 or Python v2.7. The R scripts make use of the following R packages (and their dependencies): moments (v0.14), lawstat (v3.0), plyr (v1.8.4), mice (v2.25), abind (v1.4-3), ggplot2 (v2.2.1), and ROCR (v1.0-7). Some Python scripts require scipy (v0.16.0).

Memory and run time will depend on the size of your datasets. For reference, for our largest dataset (containing ~580 samples and ~15,000 metabolite signals), *DataProcessing* scripts took a total of ~1 day to run, *ImputationMatching* took < 10 hours, and *PathwayAnalysis* took ~2 days (with “ConstructAnnotationMatrix.py” taking most of the time). Maximum memory requirement was < 14 GB.

1. Data Processing

Overview

The *DataProcessing* pipeline consists of four main steps: (1) quality control (QC) that adjusts for measurement variation associated with technical artifacts and remove samples, metabolite signals, and data points with noisy trends, (2) remove samples and signals based on percentage of missingness, (3) impute any remaining missing values, and (4) adjust for covariates and perform inverse normal transformation step. While QC should be separately performed on data generated by each profiling experiment, the other three steps are applied to the combined dataset after merging QC'ed data from different profiling experiments/methods. An important note is that our QC procedure requires the presence of internal standards and interspersed control samples in the raw profiling data (i.e. standard design used by Broad Metabolomics Platform), so it may not be applicable to all profiling platforms. However, the procedures after QC are not platform-dependent.

DataProcessing Scripts and Other Provided Files

Location	File Name	Description
src/	QC_submission_script.r	R script for running QC procedure
	QC_step1_normalization.r	R script for sample and signal normalization (called by QC_submission_script.r)
	Smoothed_Spline_Normalization_Breaks.r	R script for spline-based normalization function (called by QC_step1_normalization.r)
	QC_step2_postnormFiltering.r	R script for post-normalization filtering (called by QC_submission_script.r)
	QC_filteredInfo.r	R script for summarizing data filtered during QC
	MissingnessFilter.r	R script for filtering based on missingness
	MissingValueImputation.r	R script for missing value imputation
	AdjustCov_InvNorm.r	R script for covariate adjustment and inverse normal transformation
example/	example_raw_data_C8-pos.txt	Raw metabolite profiling data (C8-pos method)
	example_raw_data_C18-neg.txt	Raw metabolite profiling data (C18-neg method)
	example_sample_run_window.txt	Sample run window annotation
	example_phenotype_data.txt	Sample phenotype data
	example_QC_C8-pos_param.r	Config file for QC_submission_script.r (C8-pos)
	example_QC_C18-neg_param.r	Config file for QC_submission_script.r (C18-neg)
	example_MissingnessFilter_param.r	Config file for MissingnessFilter.r
	example_MissingValueImputation_param.r	Config file for MissingValueImputation.r
	example_AdjustCov_InvNorm_param.r	Config file for AdjustCov_InvNorm.r
	example_DataProcessing_submission.sh	Shell script to test run all *DataProcessing* scripts using the example data and config files

Example Commands and Output Description

The following command line commands (also in "example_DataProcessing_submission.sh") should be executed in *DataProcessing/example/* directory.

Step 1: Running QC (data normalization and noise filtering)

Command:

```
$ Rscript ../src/QC_submission_script.r example_QC_C8-pos_param.r
$ Rscript ../src/QC_submission_script.r example_QC_C18-neg_param.r
```

Output: The following files (or directories) can be found in the output directory (e.g. *QC_output_C8-pos/*).

File/Directory Name	Description
logfile.txt	Log file for QC procedure
Metabolite_info.txt	List of all input signals with their mass-to-charge ratio (m/z) and retention time (RT) information
Metabolite_raw_summaries.txt	Summary statistics of all input signals
BreakpointDetection/	Directory containing text files and plots describing locations of detected breakpoints for each signal
spline_plots/ (optional)	Directory containing plots overlaying control sample spline fits on metabolite abundance vs. run order
XISPPnorm_C8-pos.txt	Internal standard (IS) and control sample normalized metabolite data
Biorec_Var.txt	Variance of second set of control samples (not used for normalization) in normalized data
Filtering_samples.txt (not generated in example)	List of filtered samples (with outlier IS values)
Filtering_metabolites.txt	List of filtered signals and reasons for removing them
Filtering_points.txt	List of filtered data points and reasons for removing them
Filtering_windows.txt	List of filtered windows (with outlier mean/variance)
X_QCed_C8-pos_withcontrols.txt	QC'ed metabolite data for control + experimental samples
X_QCed_C8-pos.txt ***	QC'ed metabolite data for experimental samples
Metabolites_QCpass.txt	List of signals that passed QC
QCed_plots/ (optional)	Directory containing QC'ed metabolite abundance vs. run order plots
All_metabolites_mz_vs_rt_Known.pdf	Plot of m/z vs. RT for known vs. unknown signals
Filtered_metabolites_mz_vs_rt.pdf Filtered_metabolites_mz_vs_rt_50miss.pdf Filtered_metabolites_mz_vs_rt_cvPP.pdf Filtered_metabolites_mz_vs_rt_Var_diff.pdf Filtered_metabolites_mz_vs_rt_Breaks.pdf (not all generated in example)	Plots of m/z vs. RT for filtered signals removed for different reasons
Filtered_metabolites_abundance.pdf	Plot comparing abundance of filtered vs. unfiltered signals
Filtered_metabolites_count.txt	Number of known vs. unknown signals filtered for different reasons

*** "X_QCed_C8-pos.txt" is used as input of next step and contains a signal x sample data matrix. Missing values are stored as "NA".

Step 2: Filtering samples and signals based on missingness

Command:

```
$ Rscript ../src/MissingnessFilter.r example_MissingnessFilter_param.r
```

Step 3: Missing value imputation

Command:

```
$ Rscript ../src/MissingValueImputation.r example_MissingValueImputation_param.r
```

Step 4: Covariate adjustment and inverse normal transformation

Command:

```
$ Rscript ../src/AdjustCov_InvNorm.r example_AdjustCov_InvNorm_param.r
```

The outputs of Steps 2-4 are "example_spQC_miss0.25.txt", "example_spQC_miss0.25_medianMiceImp.txt", and "example_spQC_miss0.25_medianMiceImp_adjAgeSex_invNorm.txt", respectively, and each contains a sample x signal data matrix. Missing values are stored as "NA".

2. Imputation-based Matching

Overview

The *ImputationMatching* pipeline consists of six steps: (1) impute the abundance of Dataset 1 signals in Dataset 2 samples and vice versa, (2) pair up signals across the two datasets using m/z + imputation-based correlation, (3) pair up signals using m/z + RT (for comparison against step 2), (4) generate "combined" adduct ion setting matches, (5) generate "unique" or "reciprocal" match type matches, and (6) calculate statistics of matched shared known metabolites (can be used to evaluate matching performance). Not all steps have to be run sequentially (e.g. step 3 is independent from steps 1 and 2) and some steps may need to be repeated using different parameter settings before moving on to the next step (e.g. step 4 requires both "noAdduct" and "adduct" matches to have been generated in previous steps). Please see Methods in the PAIRUP-MS paper for detailed descriptions of all relevant parameters for this pipeline.

ImputationMatching Scripts and Other Provided Files

Location	File Name	Description
src/	ImputeSignals.r	R script for imputing signals across datasets
	MatchSignalsByImp.r	R script for imputation-based matching
	MatchSignalsByPredRt.r	R script for RT-based matching
	Match_MZ_Function.r	R script for function to check for m/z agreement (called by MatchSignalsByImp.r and MatchSignalsByPredRt.r)
	GetCombinedMatches.py	Python script for generating matches with "combined" adduct ion setting
	GetUniqueReciprocalMatches.py	Python script for generating matches with "unique" and "reciprocal" match types
	GetMatchedKnownStats.r	R script for statistics of matched shared known metabolites
example/	example_met_data_1.txt	Sample x signal abundance z-score matrix for Dataset 1
	example_met_data_2.txt	Sample x signal abundance z-score matrix for Dataset 2
	example_met_names_1.txt	List of signals in Dataset 1 file above
	example_met_names_2.txt	List of signals in Dataset 2 file above
	example_shared_knowns.txt	Known metabolites shared by the two datasets
	example_met_info_1.txt	m/z and RT for Dataset 1 signals
	example_met_info_2.txt	m/z and RT for Dataset 2 signals
	example_ImputeSignals_param.r	Config file for ImputeSignals.r
	example_MatchSignalsByImp_noAdduct_param.r example_MatchSignalsByImp_adduct_param.r	Config files for MatchSignalsByImp.r (only differ in adduct ion setting)
	example_MatchSignalsByPredRt_noAdduct_param.r example_MatchSignalsByPredRt_adduct_param.r	Config files for MatchSignalsByPredRt.r (only differ in adduct ion setting)
	example_GetCombinedMatches.cfg	Config file for GetCombinedMatches.py
	example_GetUniqueReciprocalMatches.cfg	Config file for GetUniqueReciprocalMatches.py
	example_GetMatchedKnownStats_param.r	Config file for GetMatchedKnownStats.r
	example_ImputationMatching_submission.sh	Shell script to test run all *ImputationMatching* scripts using the example data and config files

Example Commands and Output Description

The following command line commands (also in "example_ImputationMatching_submission.sh") should be executed in *ImputationMatching/example/* directory.

Step 1: Imputing signals across datasets

Command:

```
$ Rscript ../src/ImputeSignals.r example_ImputeSignals_param.r
```

Output: "example_imp_data_1.txt" and "example_imp_data_2.txt" contain imputed signal abundance for Dataset 1 (Dataset 1 samples x Dataset 2 signals) and Dataset 2 (Dataset 2 samples x Dataset 1 signals), respectively.

Step 2: Matching signals using imputation

Command:

```
$ Rscript ../src/MatchSignalsByImp.r example_MatchSignalsByImp_noAdduct_param.r
$ Rscript ../src/MatchSignalsByImp.r example_MatchSignalsByImp_adduct_param.r
```

Output: The following files with *(X)* = "noAdduct" or "adduct" are generated.

File Name	**Description**
example_1-2_MatchedSignals_ByImp1_*(X)*MzMode.txt	Matching from Dataset 1 to 2 using imputation-based correlation calculated across Dataset 1 samples
example_2-1_MatchedSignals_ByImp1_*(X)*MzMode.txt	Matching from Dataset 2 to 1 using imputation-based correlation calculated across Dataset 1 samples
example_1-2_MatchedSignals_ByImp2_*(X)*MzMode.txt	Matching from Dataset 1 to 2 using imputation-based correlation calculated across Dataset 2 samples
example_2-1_MatchedSignals_ByImp2_*(X)*MzMode.txt	Matching from Dataset 2 to 1 using imputation-based correlation calculated across Dataset 2 samples
example_1-2_MatchedSignals_ByImpAll_*(X)*MzMode.txt	Matching from Dataset 1 to 2 using imputation-based correlation calculated across all samples
example_2-1_MatchedSignals_ByImpAll_*(X)*MzMode.txt	Matching from Dataset 2 to 1 using imputation-based correlation calculated across all samples

Each row in the files is a pair of matched signals. The columns/headers of the files include: "Metabolite1", "Method1", "Ion1", "MZ1", "RT1": name, profiling method, ionization mode, m/z, and RT of signal in the reference dataset (first dataset in file name); "Metabolite2", "Method2", "Ion2", "MZ2", "RT2": name, profiling method, ionization mode, m/z, and RT of signal in the matched dataset (second dataset in file name); "DeltaMZ", "DeltaRT": differences in m/z and RT between the matched signal pair; "Match_Corr": imputation-based correlation between the matched signal pair.

Step 3: Matching signals using RT

Command:

```
$ Rscript ../src/MatchSignalsByPredRt.r example_MatchSignalsByPredRt_noAdduct_param.r
$ Rscript ../src/MatchSignalsByPredRt.r example_MatchSignalsByPredRt_adduct_param.r
```

Output: "example_1-2_MatchedSignals_ByPredRt_*(X)*MzMode.txt" and "example_2-1_MatchedSignals_ByPredRt_*(X)*MzMode.txt" (with *(X)* = "noAdduct" or "adduct") have similar format as the output from Step 2. The only difference is that the last two columns/headers are "PredRT" and "DeltaPredRT" (instead of "DeltaRT" and "Match_Corr"), which are the predicted RT for the reference signal in the matched dataset and the difference between the predicted RT and the RT of the matched signal.

Step 4: Generating "combined" matches

Command:

```
$ python ../src/GetCombinedMatches.py example_GetCombinedMatches.cfg
```

Output: “example_1-2_MatchedSignals_ByImp1_combinedMzMode.txt” has the same format as output from Step 2.

*** NOTE: This step can take output from Step 2 or 3 as input, but requires the appropriate “noAdduct” and “adduct” matching files to both have been generated.

Step 5: Generating “unique” and “reciprocal” matches

Command:
```
$ python ../src/GetUniqueReciprocalMatches.py example_GetUniqueReciprocalMatches.cfg
```

Output: “example_1-2_UNIQUE_MatchedSignals_ByPredRt_noAdductMzMode.txt” and “example_1-2_RECIPROCAL_MatchedSignals_ByPredRt_noAdductMzMode.txt” have the same format as output from Step 3.

*** NOTE: This step can take output from Step 2, 3, or 4 as input.

Step 6: Calculating statistics for matched shared knowns

Command:
```
$ Rscript ../src/GetMatchedKnownStats.r example_GetMatchedKnownStats_param.r
```

Output: The following files are generated.

File Name	Description
example_1-2_MatchedKnowns_ByImp1_noAdductMzMode.txt example_2-1_MatchedKnowns_ByImp1_noAdductMzMode.txt example_1-2_MatchedKnowns_ByImp2_noAdductMzMode.txt example_2-1_MatchedKnowns_ByImp2_noAdductMzMode.txt example_1-2_MatchedKnowns_ByImpAll_noAdductMzMode.txt example_2-1_MatchedKnowns_ByImpAll_noAdductMzMode.txt example_1-2_MatchedKnowns_ByPredRt_noAdductMzMode.txt example_2-1_MatchedKnowns_ByPredRt_noAdductMzMode.txt	Subsets of matches found in the corresponding “MatchedSignals” files described above. Only contain matches made for shared known metabolites (which were treated as unshared/unmatched during matching). Format is described below.
example_1-2_MatchedKnowns_ByImp1_noAdductMzMode_Plots.pdf example_2-1_MatchedKnowns_ByImp1_noAdductMzMode_Plots.pdf example_1-2_MatchedKnowns_ByImp2_noAdductMzMode_Plots.pdf example_2-1_MatchedKnowns_ByImp2_noAdductMzMode_Plots.pdf example_1-2_MatchedKnowns_ByImpAll_noAdductMzMode_Plots.pdf example_2-1_MatchedKnowns_ByImpAll_noAdductMzMode_Plots.pdf	Plots comparing R^2 between “Known” (“Metabolite1”), “Match” (“Metabolite2”), and “Real” (“RealMetabolite”) metabolites (see explanations below).

The columns/headers of the “MatchedKnowns” text files are the same as their “MatchedSignals” counterparts except for a few additional columns: “RealMetabolite”: true match for Metabolite1 in the second dataset; “CorrectMatch”: whether Metabolite2 == RealMetabolite; “Real_Corr”: imputation-based correlation between Metabolite1 and RealMetabolite (only present in imputation-based matching files); “Match_Real_ObsCorr”: observed correlation between Metabolite2 and RealMetabolite in second dataset.

*** NOTE: This step can take output from Steps 2-5 as input.

3. Pathway Annotation and Enrichment Analysis

Overview

The *PathwayAnalysis* pipeline can be split into two parts. Part A deals with generating pathway/metabolite set annotations for metabolite signals and include the following steps: (1) create metabolite sets using CPDB pathway annotations, (2) derive metabolic components (MCs) that capture correlation patterns in a metabolite profiling dataset, (3) calculate MC-metabolite set enrichment, (4) construct signal x metabolite set annotation matrix (i.e. reconstitution), (5) generate label confidence scores for reconstituted metabolite sets, and (6) perform ROC/AUC analysis to assess robustness of annotation matrix. Part B uses the generated annotations to perform pathway enrichment analysis and include the following steps: (1) identify trait-associated signals (in a second dataset) and generate null lists of signals ranked by association with permuted traits, (2) convert signal lists for second dataset into (shared known or matched) signal lists for first dataset, (3) calculate nominal metabolite set enrichment p-values, (4) calculate permutation p-values by comparing observed vs. null results, and (5) calculate false discovery rate (FDR) and output final metabolite set enrichment results. Note that Part A only needs to be performed once to generate pathway annotations for any given reference dataset, while Part B can be repeated many times to perform enrichment analyses for different traits measured in different datasets.

PathwayAnalysis Scripts and Other Provided Files

Location	File Name	Description
src/	CPDB/CPDB_pathways_metabolites_release31.tab CPDB/CPDB_pathways_genes_release31.tab	Metabolic pathway annotations downloaded from ConsensusPathDB ("genes" file not used but is provided for reference)
	CreateMetaboliteSets.py	Python script for creating metabolite sets
	DeriveMetabolicComponents.r	R script for deriving MCs
	CalculateMcMetSetEnrichment.py	Python script for calculating MC-metabolite set enrichment
	ConstructAnnotationMatrix.py	Python script for constructing signal x metabolite set annotation matrix
	CalculatePostReconEnrichment.py	Python script for calculating label confidence scores for reconstituted metabolite sets
	GenerateRocCurveData.py	Python script to generate input data file for ROC/AUC analysis in R
	PlotRocCurvesWithAuc.r	R script for generating ROC curve and AUC
	IdTraitSignals_Pearson.py	Python script to identify signals associated with a continuous trait
	IdNullTraitSignals_Pearson.py	Python script to generate null lists of signals ranked by association with permuted continuous traits
	IdTraitSignals_Ranksum.py	Python script to identify signals associated with a dichotomous/binary trait
	IdNullTraitSignals_Ranksum.py	Python script to generate null lists of signals ranked by association with permuted binary traits
	GetMatchedSignalLists.py	Python script to convert signal lists in one dataset to (shared known or matched) signal lists in another dataset
	CalculateRanksumP.py	Python script to calculate nominal metabolite set enrichment p-values for list(s) of signals
	CalculateFisherP.py	Python script to calculate nominal metabolite set over-representation p-values for list(s) of known metabolites
	CalculatePermP.py	Python script to calculate permutation p-values for list(s) of nominal p-values

	CalculateFDR.py	Python script to calculate FDR for different permutation p-values and output final metabolite set enrichment results
example/	example_met_data_1.txt	Sample x signal abundance z-score matrix for Dataset 1
	example_met_data_1_knowns.txt	List of known metabolites in Dataset 1
	example_met_data_1_knownIDs.txt	Table of known metabolite database IDs
	example_met_data_2_covAdj.txt	Sample x signal covariate-adjusted abundance z-score matrix for Dataset 2
	example_phenotype_data_2.txt	Sample phenotype data for Dataset 2
	example_2-1_shared_knowns.txt	Known metabolites shared by the two datasets
	example_2-1_matched_signals.txt	Matched signals between the two datasets
	example_CreateMetaboliteSets.cfg	Config file for CreateMetaboliteSets.py
	example_DeriveMetabolicComponents_param.r	Config file for DeriveMetabolicComponents.r
	example_CalculateMcMetSetEnrichment.cfg	Config file for CalculateMcMetSetEnrichment.py
	example_ConstructAnnotationMatrix.cfg	Config file for ConstructAnnotationMatrix.py
	example_CalculatePostReconEnrichment.cfg	Config file for CalculatePostReconEnrichment.py
	example_GenerateRocCurveData.cfg	Config file for GenerateRocCurveData.r
	example_PlotRocCurvesWithAuc_param.r	Config file for PlotRocCurvesWithAuc.r
	example_IdTraitSignals_Pearson.cfg	Config file for IdTraitSignals_Pearson.py
	example_IdNullTraitSignals_Pearson.cfg	Config file for IdNullTraitSignals_Pearson.py
	example_IdTraitSignals_Ranksum.cfg	Config file for IdTraitSignals_Ranksum.py
	example_IdNullTraitSignals_Ranksum.cfg	Config file for IdNullTraitSignals_Ranksum.py
	example_GetMatchedSignalLists.cfg example_GetMatchedSignalLists_sharedKnowns.cfg	Config files for GetMatchedSignalLists.py (second file only outputs shared knowns)
	example_CalculateRanksumP.cfg example_CalculateRanksumP_null.cfg	Config files for CalculateRanksumP.py (for observed and null p-values, respectively)
	example_CalculateFisherP.cfg	Config file for CalculateFisherP.py
	example_CalculatePermP.cfg example_CalculatePermP_null.cfg	Config files for CalculatePermP.py (for observed and null p-values, respectively)
	example_CalculateFDR.cfg	Config file for CalculateFDR.py
	example_PathwayAnalysis.sh	Shell script to test run all *PathwayAnalysis* scripts using the example data and config files

Example Commands and Output Description

The following command line commands (also in "example_PathwayAnalysis_submission.sh") should be executed in *PathwayAnalysis/example/* directory.

Step A-1: Creating metabolite sets

Command:

```
$ python ../src/CreateMetaboliteSets.py example_CreateMetaboliteSets.cfg
```

Output: "example_data_1_CPDB_2Mets_MetSet-Metabolites-Pathways.txt" contains three columns/headers: "SET_ID", "Metabolites", and "Pathways". Metabolites are separated by ";" and pathways are separated by "|".

Step A-2: Deriving metabolic components

Command:

```
$ Rscript ../src/DeriveMetabolicComponents.r example_DeriveMetabolicComponents_param.r
```

Output: “example_data_1_MetaboliteCorMatrix.txt.gz” is an optional, gzipped file containing the signal-signal correlation matrix; “example_data_1_McStatistics.txt” contains statistics for each MC (i.e. standard deviation, proportion and cumulative proportion of variance explained); “example_data_1_McScoreMatrix.txt” contains the signal x MC score matrix that is used for reconstitution below.

Step A-3: Calculating MC-metabolite set enrichment

Command:

```
$ python ../src/CalculateMcMetSetEnrichment.py example_CalculateMcMetSetEnrichment.cfg
```

Output: “example_data_1_MC_CPDB_2Mets_SetEnrichment_fdr20.txt” lists the most enriched metabolite set(s) (“BestP”, “BestFDR”, “BestSets”) and all enriched sets at 5% FDR (“FDR0.05_Sets”) for each MC.

Step A-3.5: Getting list of MCs to use for reconstitution

Command:

```
$ tail -n +2 example_data_1_MC_CPDB_2Mets_SetEnrichment_fdr20.txt | cut -f 1 | head -10 > example_data_1_MC_CPDB_2Mets_SetEnrichment_top10MCs.txt
```

Output: “example_data_1_MC_CPDB_2Mets_SetEnrichment_top10MCs.txt” lists the first 10 MCs.

*** NOTE: The example command simply takes the first 10 MCs. But when using real data, you should be able to filter for MCs enriched for metabolite sets using the output from Step 3.

Step A-4: Constructing signal x metabolite set annotation matrix

Command:

```
$ python ../src/ConstructAnnotationMatrix.py example_ConstructAnnotationMatrix.cfg
```

Output: “example_data_1_SignalMetSetAnnotationMatrix_2Mets_top10MCs.txt” contains the signal x metabolite set annotation matrix (i.e. reconstituted metabolite sets).

Step A-5: Calculating post-reconstitution metabolite set label confidence scores

Command:

```
$ python ../src/CalculatePostReconEnrichment.py example_CalculatePostReconEnrichment.cfg
```

Output: “example_data_1_SignalMetSetAnnotationMatrix_2Mets_top10MCs_labelRankSumP.txt” contains the label confidence scores (rank-sum p-values) for each reconstituted metabolite set.

Step A-6: Generating ROC curve and AUC using annotation matrix

Command:

```
$ python ../src/GenerateRocCurveData.py example_GenerateRocCurveData.cfg
$ Rscript ../src/PlotRocCurvesWithAuc.r example_PlotRocCurvesWithAuc_param.r
```

Output:

(a) “example_data_1_SignalMetSetAnnotationMatrix_2Mets_top10MCs_confidentSets_rocData.txt” lists the reconstituted metabolite set scores and original binary membership labels of known metabolites.

(b) "example_data_1_SignalMetSetAnnotationMatrix_2Mets_top10MCs_confidentSets_ROC-AUC.pdf" is a plot of the ROC curve generated by classifying known metabolites, with AUC value shown in legend.

Step B-1: Identifying trait-associated signals + generating null signal lists

Command:

```
$ python ../src/IdTraitSignals_Pearson.py example_IdTraitSignals_Pearson.cfg
$ python ../src/IdNullTraitSignals_Pearson.py example_IdNullTraitSignals_Pearson.cfg
```

Output:

(a) "example_2_Trait_PearsonStats.txt" contains Pearson correlation statistics between the continuous example trait and all Dataset 2 signals; "example_2_Trait_Pearson_p0.2_signals.txt" is a list of "significant" trait-associated Dataset 2 signals separated by ";".
(b) "example_2_NullTrait_Pearson_signals_55.txt" contains lists (rows) of Dataset 2 signals ranked by association with permuted trait values, also separated by ";".

Alternative Command: (same output format but for binary traits)

```
$ python ../src/IdTraitSignals_Ranksum.py example_IdTraitSignals_Ranksum.cfg
$ python ../src/IdNullTraitSignals_Ranksum.py example_IdNullTraitSignals_Ranksum.cfg
```

Step B-2: Converting Dataset 2 signal lists to Dataset 1 signal lists

Command:

```
$ python ../src/GetMatchedSignalLists.py example_GetMatchedSignalLists.cfg
```

Output: "example_2-1_Trait_Pearson_p0.2_signals.txt" and "example_2-1_NullTrait_Pearson_signals_55.txt" are lists of Dataset 1 signals corresponding to the Dataset 2 lists from the previous step; "example_2-1_allMappedSignalList.txt" is a list of all mapped (shared known or matched) Dataset 1 signals.

Alternative Command: (same output format but restricted to shared knowns)

```
$ python ../src/GetMatchedSignalLists.py example_GetMatchedSignalLists_sharedKnowns.cfg
```

Step B-3: Calculating nominal metabolite set enrichment p-values

Command:

```
$ python ../src/CalculateRanksumP.py example_CalculateRanksumP.cfg
$ python ../src/CalculateRanksumP.py example_CalculateRanksumP_null.cfg
```

Output:

(a) "example_2-1_Trait_signals_MetSetRanksumP.txt" is a tab-delimited list of nominal p-values (corresponding to each metabolite set) calculated using observed trait-associated signals
(b) "example_2-1_NullTrait_signals_MetSetRanksumP_55.txt" contains tab-delimited lists (rows) of nominal p-values calculated using null signal lists.

Alternative Command: (same output format but test for "over-representation" using only known metabolites)

```
$ python ../src/CalculateFisherP.py example_CalculateFisherP.cfg
```

Step B-4: Calculating permutation p-values

Command:

```
$ python ../src/CalculatePermP.py example_CalculatePermP.cfg
$ python ../src/CalculatePermP.py example_CalculatePermP_null.cfg
```

Output:

(a) “example_2-1_Trait_signals_MetSetPermP.txt” is a tab-delimited list of permutation p-values calculated using observed vs. null nominal p-values.

(b) “example_2-1_NullTrait_signals_MetSetPermP_5.txt” contains tab-delimited lists (rows) of permutation p-values calculated using null vs. null nominal p-values.

Step B-5: Calculating FDR + Outputting final metabolite set enrichment results

Command:

```
$ python ../src/CalculateFDR.py example_CalculateFDR.cfg
```

Output: “example_2-1_Trait_signals_MetSetFdrEnrichment.txt” is a table summarizing the metabolite set enrichment results with columns/headers: “SET_ID”, “Nominal_P”, “Perm_P”, “FDR”, “Label_P” (label confidence socre), and “CPDB_Pathways” (original pathway labels for the metabolite set). Metabolite sets are first sorted by permutation p-value/FDR then by nominal p-value.