sumo Documentation

Release 0.3.0

Karolina Sienkiewicz

Jul 06, 2022

Contents:

CHAPTER 1

Example usage

In this example, we will use SUMO to stratify patients diagnosed with Acute Myeloid Leukemia (LAML) into subgroups based on gene-expression, micro-RNA expression, and methylation datasets. We will describe steps that should be taken for pre-processing of these data, use SUMO to detect the various sub-groups of patients, and identify the features e.g., genes and methylation probes, that drive each of those subgroups.

We will use gene expression, methylation and miRNA expression data from UCSC XENA browser in this python vignette.

1.1 Data preprocessing

While preprocessing data for SUMO analysis we strongly suggest the following steps:

1. **Filtering your data** - SUMO handles missing values to some extent, however removing features and samples with a large fraction of missing values (>10%) has been shown to improve the classification. You can additionally choose to remove features that are likely to be noise e.g., low expressed genes in all samples.
2. **Data normalization/transformation** - We advise the use of a log transform or a variant stabilizing transform when using count data as input. This step is omitted in the code below, as the downloaded data is already log normalized. When using methylation data, we prefer the use of M-values over beta values.
3. **Data standardization** - Lastly, each feature should be standardized before being input to SUMO.

Here is code in python that we can use to perform the preprocessing for the RNA-seq and the miRNA-sequence datasets

```
import numpy as np
import pandas as pd
from sklearn import preprocessing

def preprocess_logfpkm(inputfile, outputfile):
    """Filter the input log2(fpkm+1) values and write the output"""

    # read the log2(fpkm+1) values
```

(continues on next page)

(continued from previous page)

```
    norm_fpkm = pd.read_csv(inputfile, sep='\t', header=0, index_col=0)
    print("Read %s" % inputfile)

    # remove genes where less then two samples have FPKM higher then zero
    norm_fpkm = norm_fpkm[np.sum(norm_fpkm > 1, axis=1) > 1]

    # standardize the norm_fpkm matrix
    scaler = preprocessing.StandardScaler()
    scaled_fpkm = scaler.fit_transform(norm_fpkm.T)
    scaled_fpkm = scaled_fpkm.T
    scaled_fpkm = pd.DataFrame(scaled_fpkm, index=list(norm_fpkm.index),␣
↪columns=list(norm_fpkm.columns))
        # write the file
        scaled_fpkm.to_csv(outputfile, sep='\t', index=True, na_rep="NA")
        print("Wrote %s" % outputfile)

preprocess_logfpkm("TCGA-LAML.htseq_fpkm.tsv.gz", "TCGA-LAML.htseq_fpkm.flt.tsv.gz")
preprocess_logfpkm("TCGA-LAML.mirna.tsv.gz", "TCGA-LAML.mirna.flt.tsv.gz")
```

Here is the python code to perform preprocessing of the methylation dataset.

```
import numpy as np
import pandas as pd
from sklearn import preprocessing

# read in the beta values
beta = pd.read_csv("TCGA-LAML.methylation27.tsv.gz", sep='\t', header=0, index_col=0)

# remove rows where we do not have information from any sample
beta = beta.dropna(axis=0, how='all')

# convert each beta value to the corresponding M values
def convert(B):
    eps = np.spacing(1)
    return(np.log2((B + eps)/(1. - B + eps)))

M = beta.applymap(convert)
print("Converted to M values")

scaler = preprocessing.StandardScaler()
scaled_M = scaler.fit_transform(M.T)
scaled_M = scaled_M.T
scaled_M = pd.DataFrame(scaled_M, index=list(M.index), columns=list(M.columns))
print("Standardization complete")

scaled_M.to_csv("TCGA-LAML.methylation27.flt.tsv.gz", sep='\t', index=True, na_rep="NA
↪")
```

1.2 Running SUMO

SUMO provides four modes allowing for molecular subtyping of multi-omic data (*prepare* and *run*), as well as comprehensive analysis that includes identification of molecular features driving classification (*interpret*) and comparison with existing subtype classifications (*evaluate*).

1.2.1 sumo prepare

In this mode, SUMO calculates the pairwise similarity between the samples for each separate input file containing a feature matrix with omic data (in this case gene expression, methylation and miRNA expression).

```
sumo prepare -plot LAML.png TCGA-LAML.htseq_fpkm.flt.tsv.gz,TCGA-LAML.methylation27.
↪flt.tsv.gz,TCGA-LAML.mirna.flt.tsv.gz prepared.LAML.npz
```

The above command creates a multiplex network file 'prepared.LAML.npz' containing:

- the pairwise similarities organized as network adjacency matrices in order of input files (arrays: '0', '1', '2')
- input feature matrices (arrays: 'f0', 'f1', 'f2')
- list of sample identifiers in order corresponding to rows/columns of adjacency matrices ('sample' array)

Thanks to the -plot flag SUMO also creates three .png files with plots of the adjacency matrices for each omic datatype.

1.2.2 sumo run

In this mode, SUMO applies symmetric non-negative matrix tri-factorization on the similarity matrices to identify the clusters of samples. Estimating the best number of clusters remains a challenging problem, but we recommend that the user supply a range of values to use with SUMO.

```
sumo run prepared.LAML.npz 2,4 LAML
```

When the above command is run, SUMO creates an output directory named 'LAML'. In that directory, SUMO creates a sub-directory for each *k* (the number of clusters) that contains the factorization results in the form of .npz files, and a 'clusters.tsv' file with sample labels. A 'plots' sub-directory is also created, where we provide several plots that can assist in selection of the best number of subtypes in the dataset. A stable clustering result is characterized by a high value of cophenetic correlation coefficient (plotted in LAML/plots/cophenet.png) and low proportion of ambiguous clusterings (plotted in LAML/plots/pac.png).

The complete directory structure generated after running the above command is shown below.

```
LAML
├── k2
│   ├── clusters.tsv
│   ├── eta_0.1.log
│   ├── eta_0.1.npz
│   └── sumo_results.npz -> eta_0.1.npz
├── k3
│   ├── clusters.tsv
│   ├── eta_0.1.log
│   ├── eta_0.1.npz
│   └── sumo_results.npz -> eta_0.1.npz
├── k4
│   ├── clusters.tsv
│   ├── eta_0.1.log
│   ├── eta_0.1.npz
│   └── sumo_results.npz -> eta_0.1.npz
└── plots
    ├── consensus_k2.png
    ├── consensus_k3.png
    ├── consensus_k4.png
    ├── cophenet.png
    └── pac.png
```

To make subtyping results more robust SUMO uses a resampling-based approach in conjunction with consensus clustering. In this mode, the factorization is repeated multiple times (set by -n flag), with a fraction of samples (set by -subsample flag) removed from each run. Next, we use random subsets of runs to create multiple (set by -rep flag) weighted consensus matrices, that are utilized for the robust assessment of the factorization results and derivation of final clustering labels.

As presented in the directory structure above SUMO creates an .npz result file for each (k, eta) pair, where k is a set number of clusters and eta is a factorization sparsity value (can be modified by -sparsity flag). Each such file contains:

- calculated clustering stability metrics for each consensus matrix: the proportion of ambiguous clusterings and cophenetic correlation coefficient ('pac' and 'cophenet' arrays respectively)
- quality metric assessing the within-cluster similarities based on final sample labels, used for sparsity parameter selection ('quality' array)
- selected consensus matrix ('unfiltered_consensus' array) and its copy used for final sample label assignment after the noise filtering ('consensus' array)
- final sample labels ('clusters' array)
- number of iterations/steps reached in each solver run ('steps' array)
- simulation parameters ('config' array)

Adding -log DEBUG flag when running SUMO 'run' mode, results in additional arrays (saved in .npz files) and plots displaying the number of iterations reached by sumo for each "k" (saved to 'plots' directory).

Following additional arrays are added to each .npz file:

- every weighted consensus matrix used for the calculation of stability metrics (arrays: 'pac_consensus_0', 'pac_consensus_1'...)
- indices of solver runs used to create each consensus matrix (arrays: 'runs_0', 'runs_1', ...)
- **results of each factorization run:**
  - final cost function value (array 'costi' for run 'i')
  - final H matrix (array 'hi' for run 'i')
  - final S matrix for each data type (array 'sij' for data type 'i' and run 'j')
  - indices of fraction of samples used in the factorization run (arrays: 'samples0', 'samples1', ...)

1.2.3 sumo interpret

Use SUMO *interpret* mode to investigate which features drive obtained clustering results.

```
sumo interpret LAML/k4/sumo_results.npz TCGA-LAML.htseq_fpkm.flt.tsv.gz,TCGA-LAML.
↪methylation27.flt.tsv.gz,TCGA-LAML.mirna.flt.tsv.gz LAML_features
```

The above command generates a file two files "LAML_features.tsv" and "LAML_features.hits.tsv" which report the importance of each feature in supporting cluster separation. Briefly, we train a LightGBM model (https://github.com/microsoft/LightGBM) based on the clusters identified by SUMO, and the results from this mode are the SHAP (SHapley Additive exPlanations) feature importance deduced using that model.

For example, here the results shows that the following top 10 features support various clusters:

Group 0		Group 1	
cg27497900	30.21	cg21299958	36.01
cg05934874	21.77	cg16907075	13.5
hsa-mir-574	17.65	cg14142521	12.92
hsa-mir-450a-1	12.71	ENSG00000135404.10	11.87
ENSG00000173599.12	7.795	ENSG00000185875.11	9.78
cg23705973	6.13	cg24995240	4.21
cg26450541	4.56	ENSG00000114942.12	3.97
ENSG00000173482.15	4.5	ENSG00000271270.4	3.59
hsa-mir-450b	2.76	ENSG00000113272.12	3.51
ENSG00000154122.11	1.96	cg06540636	3.175

Group 2		Group 3	
hsa-mir-199a-2	24.34	cg14178895	18.385
ENSG00000153786.11	14.95	cg00617305	12.89
hsa-let-7e	10.81	ENSG00000269845.1	11.81
ENSG00000229816.1	9.43	ENSG00000196705.7	11.61
ENSG00000281016.1	8.96	cg09891761	11.535
ENSG00000177731.14	7.75	ENSG00000160229.10	7.88
cg18959422	7.57	ENSG00000255730.3	4.84
ENSG00000206841.1	4.71	ENSG00000269399.2	2.88
hsa-mir-128-2	4.63	hsa-mir-4473	2.735
hsa-mir-106a	3.645	ENSG00000270876.1	2.68

1.3 Including somatic mutations in SUMO analysis

Calculation of similarity distances is challenging for sparse data types such as somatic mutation. Feature selection e.g., limiting to the genes that are known to play a role in the disease, or feature transformation, e.g. mapping to known pathways instead of individual genes can reduce sparsity and improve distance calculations. Another approach is to convert the somatic data into a continuous data matrix appropriate for SUMO.

One way to efficiently convert somatic mutation data into a continuous matrix is to not only consider the gene's self-characteristic (in this case a mutation presence/absence) but also its influence on the regulatory/interaction network.

In this example, we use a random walk with restart on the somatic mutation data from LAML patients (see the previous example) on a protein-protein interaction network, creating a continuous matrix that can be included in the SUMO data integration workflow.

In the below R vignettes we use MC3 public somatic mutation data. The file was subsetted to contain only LAML samples.

```
library(biomaRt)
library(igraph)
library(STRINGdb)
library(annotables)
library(RandomWalkRestartMH)
library(tidyverse)
library(ComplexHeatmap)

# Load the somatic mutation data
mutations <- read_tsv("mc3.v0.2.8.PUBLIC.LAML.maf.gz", guess_max=1000000)
```

(continues on next page)

(continued from previous page)

```
patient_ids <- sapply(mutations$Tumor_Sample_Barcode,
                  function(x){spx = strsplit(x, '-')[[1]]; paste(spx[1], spx[2],␣
↪spx[3],sep="-")})
mutations$patient_id <- patient_ids
```

Fetch the protein-protein interactions for H. Sapiens from the STRING data base.

```
string_db <- STRINGdb$new( version="10", species=9606, score_threshold=400, input_
↪directory="")
db <- string_db$get_graph()
```

Create the gene-peptide identifier mapping with BioMart.

```
mart <- useMart(biomart = "ensembl", dataset = "hsapiens_gene_ensembl")
mapping <- getBM(attributes=c("ensembl_gene_id", "ensembl_transcript_id", "ensembl_
↪peptide_id"), mart=mart) %>%
           mutate(ensembl_peptide_id = paste0("9606.", ensembl_peptide_id)) %>%
           filter(ensembl_peptide_id %in% names(V(db)))
```

Here we apply an exemplary filtering of the mutation types and possible artifacts based on the MC3 filters.

```
classes_toignore <- c("Intron", "Silent", "5'Flank", "3'UTR", "5'UTR", "3'Flank", "IGR
↪", "RNA")
mutations <- mutations %>%
    filter(!Variant_Classification %in% classes_toignore) %>% # subset the mutation␣
↪types
    filter(FILTER != "oxog,wga") # filter out possible artifacts
```

Create one layer protein-protein interaction (PPI) network.

```
PPI_MultiplexObject <- create.multiplex(db,Layers_Name=c("PPI"))
# To apply the Random Walk with Restart (RWR) on this monoplex network,
# we need to compute the adjacency matrix of the network and normalize it by column.
AdjMatrix_PPI <- compute.adjacency.matrix(PPI_MultiplexObject)
AdjMatrixNorm_PPI <- normalize.multiplex.adjacency(AdjMatrix_PPI)
```

Apply the Random Walk with Restart (RWR) for each patient based on PPI network with seeds set according to somatic mutation data.

```
score_patient <- function(tbl) {
    seedgenes <- tbl$ensembl_peptide_id
    RWR_PPI_Results <- Random.Walk.Restart.Multiplex(AdjMatrixNorm_PPI, PPI_
↪MultiplexObject,seedgenes)
    seeddf <- tibble(NodeNames = seedgenes, Score = 1/length(seedgenes))
    outdf <- RWR_PPI_Results$RWRM_Results %>% as_tibble()
    outdf <- bind_rows(outdf, seeddf)
    return(outdf)
}

scores <- mutations %>%
    select(patient_id, Gene) %>%
    distinct() %>%
    left_join(mapping, by=c("Gene"="ensembl_gene_id")) %>%
    select(patient_id, ensembl_peptide_id) %>%
    na.omit() %>%
    group_by(patient_id) %>%
```

(continues on next page)

(continued from previous page)

```
    nest() %>%
    mutate(df = map(data, score_patient)) %>%
    select(-data) %>%
    unnest(cols=c(df)) %>%
    ungroup()
```

Save the result.

```
df <- scores %>% spread(NodeNames, Score, fill=0)
mat <- data.matrix(df %>% select(-patient_id))
rownames(mat) <- df$patient_id

write.table(t(mat), file="mutation_scores.tsv", quote=F, sep="\t")
```

We now have a continuous matrix that can be used as input to *sumo prepare*. The choice of the interaction network can be important, and tissue-specific networks such as those from HumanBase lead to an improvement in results.

CHAPTER 2

SUMO modes

class sumo.modes.mode.**SumoMode**(***kwargs*)

Defines modes of sumo

run()

Run mode specific functionality

2.1 prepare

2.1.1 SumoPrepare Class

class sumo.modes.prepare.prepare.**SumoPrepare**(***kwargs*)

Sumo mode for data pre-processing and creation of multiplex network files. Constructor args are set in 'prepare' subparser.

Args:

infiles (list): comma-delimited list of paths to input files, containing standardized feature matrices, with samples in columns and features in rows (supported types of files: ['.txt', '.txt.gz', '.txt.bz2', '.tsv', '.tsv.gz', '.tsv.bz2'])

outfile (str): path to output .npz file

method (list): comma-separated list of methods for every layer (available methods: ['euclidean', 'cosine', 'pearson', 'spearman'])

k (float): fraction of nearest neighbours to use for sample similarity calculation using Euclidean distance similarity

alpha (float): hypherparameter of RBF similarity kernel, for Euclidean distance similarity

missing (list): acceptable fraction of available (not missing) values for assessment of distance/similarity between pairs of samples, either one value or different values for every layer

atol (float): if input files have continuous values, sumo checks if data is standardized feature-wise, meaning all features should have mean close to zero, with standard deviation around one; use this parameter to set tolerance of standardization checks

sn (int): index of row with sample names for .txt input files

fn (int): index of column with feature names for .txt input files

df (float): if percentage of missing values for feature exceeds this value, remove feature

ds (float): if percentage of missing values for sample (that remains after feature dropping) exceeds this value, remove sample

logfile (str): path to save log file, if set to None stdout is used

log (str): sets the logging level from ['DEBUG', 'INFO', 'WARNING']

plot (str): path to save adjacency matrix heatmap(s), if set None plots are displayed on screen

load_all_data()

Load all of input files

Returns: list of tuples, every containing file name (str) and filtered feature matrix (pandas.DataFrame))

run()

Generate similarity matrices for samples based on biological data

2.1.2 Similarity Metrics

sumo.modes.prepare.similarity.**correlation**(*a: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c6e780>, b: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c6e7b8>, missing: float, method='pearson'*)

Calculate correlation similarity between two vectors

sumo.modes.prepare.similarity.**cosine_sim**(*a: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c6e5c0>, b: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c6e550>, missing: float*)

Calculate cosine similarity between two vectors

sumo.modes.prepare.similarity.**euclidean_dist**(*a: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c6e6d8>, b: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c6e4a8>, missing: float*)

Calculate euclidean distance between two vectors of continuous variables

sumo.modes.prepare.similarity.**feature_rbf_similarity**(*f: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c6e828>, missing: float = 0.1, n: float = 0.1, alpha: float = 0.5, distance=<function euclidean_dist>*)

Generate similarity matrix using RBF kernel and supplied distance function

Args: f (Numpy.ndarray): Feature matrix (n x k, where 'n' - samples, 'k' - measurements) n (float): fraction of nearest neighbours to use for samples similarity calculation missing (float): acceptable fraction of values for assessment of distance/similarity between two samples alpha (float): hyperparameter of RBF kernel distance: distance function accepting two vectors and missing parameter (default of Euclidean distance)

Returns: w (Numpy.ndarray): symmetric matrix describing similarity between samples (n x n)

sumo.modes.prepare.similarity.**feature_to_adjacency**(*f: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c6e7f0>, missing: float = 0.1, method: str = 'euclidean', n: float = None, alpha: float = None*)

Generate similarity matrix from genomic assay

Args: f (Numpy.ndarray): Feature matrix (n x k, where 'n' - samples, 'k' - measurements) missing (float): acceptable fraction of values for assessment of distance/similarity between two samples (default of 0.1, means that up to 90 % of missing values is acceptable) method (str): similarity method selected from: ['euclidean', 'cosine', 'pearson', 'spearman'] n (float): parameter of euclidean similarity method, fraction of nearest neighbours of sample alpha (float): parameter of euclidean similarity method, RBF kernel hyperparameter

Returns: sim (Numpy.ndarray): symmetric matrix describing similarity between samples (n x n)

2.2 run

2.2.1 SumoRun Class

class sumo.modes.run.run.**SumoRun**(***kwargs*)

Sumo mode for factorization of multiplex network to identify molecular subtypes. Constructor args are set in 'run' subparser.

Args:

infile (str): input .npz file containing adjacency matrices for every network layer and sample names (file created by running program with mode "prepare") - consecutive adjacency arrays in file are indexed in following way: "0", "1" ... and index of sample name vector is "samples"

k (int): number of clusters

outdir (str) path to save output files

sparsity (list): list of sparsity penalty values for H matrix (if multiple values sumo will try all and select the best results

n (int): number of repetitions

method (str): method of cluster extraction, selected from ['max_value', 'spectral']

max_iter (int): maximum number of iterations for factorization

tol (float): if objective cost function value fluctuation is smaller than this value, stop iterations before reaching max_iter

subsample (float): fraction of samples randomly removed from each run, cannot be greater then 0.5

calc_cost (int): number of steps between every calculation of objective cost function

logfile (str): path to save log file, if set to None stdout is used

log (str): sets the logging level from ['DEBUG', 'INFO', 'WARNING']

h_init (int): index of adjacency matrix to use for H matrix initialization, if set to None average adjacency matrix is used

t (int): number of threads

rep (int): number of times consensus matrix is created for the purpose of assessing clustering quality

run()

Cluster multiplex network using non-negative matrix tri-factorization

2.2.2 NMF Solvers

class sumo.modes.run.solver.**SumoNMFResults**(*graph: sumo.network.MultiplexNet*, *h: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c7a278>*, *s: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c7a2b0>*, *objval: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c7a320>*, *steps: int*, *logger: logging.Logger*, *sample_ids: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c7ab38>*, ***kwargs*)

Wrapper class for SumoNMF factorization results

extract_clusters(*method: str*)

Extract cluster labels using selected method

Args: method (str): either "max_value" for extraction based on maximum value in each row of h matrix or "spectral" for spectral clustering on h matrix values

class sumo.modes.run.solver.**SumoSolver**(*graph: sumo.network.MultiplexNet*, *nbins: int*, *bin_size: int = None*, *rseed: int = None*)

Defines solver of sumo

Args:

graph (MultiplexNet): network object, containing data about connections between nodes in each layer in form of adjacency matrices

nbins (int): number of bins, to distribute samples into

bin_size (int): size of bin, if None set to number of samples

calculate_avg_adjacency() → <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c7a6a0>

Creates average adjacency matrix

create_sample_bins() → list

Separate samples randomly into bins of set size, while making sure that each sample is allocated in at least one bin.

Returns: list of arrays containing indices of samples allocated to the bin

factorize(*sparsity_penalty: float*, *k: int*, *max_iter: int*, *tol: float*, *calc_cost: int*, *logger_name: str*, *bin_id: int*) → sumo.modes.run.solver.SumoNMFResults

Run solver specific factorization

Unsupervised SUMO

class sumo.modes.run.solvers.unsupervised_sumo.**UnsupervisedSumoNMF**(*graph: sumo.network.MultiplexNet*, *nbins: int*, *bin_size: int = None*, *rseed: int = None*)

Unsupervised SUMO solver (A(i)=HS(i)H^T formulation)

factorize(*sparsity_penalty: float, k: int, max_iter: int = 500, tol: float = 1e-05, calc_cost: int = 1, h_init: int = None, logger_name: str = None, bin_id: int = None*) → sumo.modes.run.solver.SumoNMFResults

Run tri-factorization

Args: sparsity_penalty (float): 'η' value, corresponding to sparsity penalty for H k (int): expected number of clusters max_iter (int): maximum number of iterations tol (float): if objective cost function value fluctuation is smaller than 'stop_val', stop iterations before reaching max_iter calc_cost (int): number of steps between every calculation of objective cost function h_init (int): index of adjacency matrix to use for H matrix initialization or None for initialization using average adjacency logger_name (str): name of existing logger object, if not supplied new main logger is used bin_id (int): id of sample bin created in SumoNMF constructor (default of None, means clustering all samples instead of samples in given bin)

Returns: SumoNMFResults object (with result feature matrix / soft cluster indicator matrix (H array), and the list of result S matrices for each graph layer)

Supervised SUMO

class sumo.modes.run.solvers.supervised_sumo.**SupervisedSumoNMF**(*graph: sumo.network.MultiplexNet, priors: <sphinx.ext.autodoc.importer._MockObj object at 0x7f4466c7d390>, nbins: int, bin_size: int = None, rseed: int = None*)

Supervised SUMO solver (A(i)=HS(i)H^T formulation)

create_sample_bins() → list

Separate samples randomly into bins of set size, while making sure that each sample is allocated in at least one bin and each prior label in represented equally.

Returns: list of arrays containing indices of samples allocated to the bin

factorize(*sparsity_penalty: float, k: int, max_iter: int = 500, tol: float = 1e-05, calc_cost: int = 1, logger_name: str = None, bin_id: int = None*) → sumo.modes.run.solver.SumoNMFResults

Run tri-factorization

Args: sparsity_penalty (float): 'η' value, corresponding to sparsity penalty for H k (int): expected number of clusters max_iter (int): maximum number of iterations tol (float): if objective cost function value fluctuation is smaller than 'stop_val', stop iterations before reaching max_iter calc_cost (int): number of steps between every calculation of objective cost function logger_name (str): name of existing logger object, if not supplied new main logger is used bin_id (int): id of sample bin created in SumoNMF constructor (default of None, means clustering all samples instead of samples in given bin)

Returns: SumoNMFResults object (with result feature matrix / soft cluster indicator matrix (H array), and the list of result S matrices for each graph layer)

2.3 evaluate

2.3.1 SumoEvaluate Class

class sumo.modes.evaluate.evaluate.**SumoEvaluate**(***kwargs*)

Sumo mode for evaluating accuracy of clustering. Constructor args are set in 'evaluate' subparser.

Args:

infile (str): input .tsv file containing sample names in 'sample' and clustering labels in 'label' column (clusters.tsv file created by running sumo with mode 'run')

labels (str): .tsv of the same structure as input file

metric (str): one of metrics (['NMI', 'purity', 'ARI']) for accuracy evaluation, if set to None all metrics are calculated

logfile (str): path to save log file, if set to None stdout is used

log (str): sets the logging level from ['DEBUG', 'INFO', 'WARNING']

load_tsv(*fname: str*)

Load .tsv file

run()

Evaluate clustering results, given set of labels

2.4 interpret

2.4.1 SumoInterpret Class

class sumo.modes.interpret.interpret.**SumoInterpret**(***kwargs*)

Sumo mode for interpreting clustering results. Constructor args are set in 'interpret' subparser.

Args:

sumo_results (str): path to sumo_results.npz (created by running program with mode "run")

infiles (list): comma-delimited list of paths to input files, containing standardized feature matrices, with samples in columns and features in rows(supported types of files: ['.txt', '.txt.gz', '.txt.bz2', '.tsv', '.tsv.gz', '.tsv.bz2'])

output_prefix (str): prefix of output files - sumo will create two output files (1) .tsv file containing matrix (features x clusters), where the value in each cell is the importance of the feature in that cluster; (2) .hits.tsv file containing features of most importance

hits (int): sets number of most important features for every cluster, that are logged in .hits.tsv file

max_iter (int): maximum number of iterations, while searching through hyperparameter space

n_folds (int): number of folds for model cross validation

t (int): number of threads

seed (int): random state

sn (int): index of row with sample names for .txt input files

fn (int): index of column with feature names for .txt input files

df (float): if percentage of missing values for feature exceeds this value, remove feature

ds (float): if percentage of missing values for sample (that remains after feature dropping) exceeds this value, remove sample

logfile (str): path to save log file, if set to None stdout is used

log (str): sets the logging level from ['DEBUG', 'INFO', 'WARNING']

create_classifier(*x: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c67908>*, *y: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c67940>*)

Create a gradient boosting method classifier

Args: x (Numpy.ndarray): input feature matrix y (Numpy.ndarray): one dimensional array of labels in classification

Returns: LGBM classifier

run()

Find features that support clusters separation

CHAPTER 3

Multiplex Network

class sumo.network.**MultiplexNet**(*adj_matrices: list*, *node_labels: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466c7aac8>*)

Multiplex network representation

CHAPTER 4

Utilities

sumo.utils.**adjusted_rand_index**(*cl: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb518>, org: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb550>*)

Clustering accuracy measure calculated by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the predicted and true clusterings

sumo.utils.**check_accuracy**(*cl: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb5f8>, org: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb630>, method='purity'*)

Check clustering accuracy

Args: cl (Numpy.ndarray): one dimensional array containing computed clusters ids for every node org (Numpy.ndarray): one dimensional array containing true classes ids for every node method (str): accuracy assessment function from ['NMI', 'purity', 'ARI']

sumo.utils.**check_categories**(*a: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb3c8>*)

Check categories in data

sumo.utils.**check_matrix_symmetry**(*m: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb2b0>, tol=1e-08, equal_nan=True*)

Check symmetry of numpy array, after removal of missing samples

sumo.utils.**close_logger**(*logger*)

Remove all handlers of logger

sumo.utils.**docstring_formatter**(**args, **kwargs*)

Decorator allowing for printing variable values in docstrings

sumo.utils.**extract_max_value**(*h: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb438>*)

Select clusters based on maximum value in feature matrix H for every sample/row

Args:

h (Numpy.ndarray): feature matrix from optimization algorithm run of shape (n,k), where 'n' is a number of nodes and 'k' is a number of clusters

Returns: one dimensional array containing clusters ids for every node

sumo.utils.**extract_ncut**(*a: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb2e8>*, *k: int*)
Select clusters using normalized cut based on graph similarity matrix

Args: a (Numpy.ndarray): symmetric similarity matrix k (int): number of clusters

Returns: one dimensional array containing clusters ids for every node

sumo.utils.**extract_spectral**(*h: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb470>*, *assign_labels: str = 'kmeans'*, *n_neighbors: int = 10*, *n_clusters: int = None*)
Select clusters using spectral clustering of feature matrix H

Args: h (Numpy.ndarray): feature matrix from optimization algorithm run of shape (n,k), where 'n' is a number of nodes and 'k' is a number of clusters assign_labels : {'kmeans', 'discretize'}, strategy to use to assign labels in the embedding space n_neighbors (int): number of neighbors to use when constructing the affinity matrix n_clusters (int): number of clusters, if not set use number of columns of 'a'

Returns: one dimensional array containing clusters ids for every node

sumo.utils.**filter_features_and_samples**(*data: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb6a0>*, *drop_features: float = 0.1*, *drop_samples: float = 0.1*)
Filter data frame features and samples

Args: data (pandas.DataFrame): data frame (with samples in columns and features in rows) drop_features (float): if percentage of missing values for feature exceeds this value, remove this feature drop_samples (float): if percentage of missing values for sample (that remains after feature dropping) exceeds this value, remove this sample

Returns: filtered data frame

sumo.utils.**is_standardized**(*a: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb668>*, *axis: int = 1*, *atol: float = 0.001*)
Check if matrix values are standardized (have mean equal 0 and standard deviation equal 1)

Args: a (Numpy.ndarray):feature matrix axis: either 0 (column-wise standardization) or 1 (row-wise standardization) atol (float): absolute tolerance

Returns: is_standard (bool): True if data is standardized mean (float): maximum and minimum mean of columns/rows std (float): maximum and minimum standard deviation of columns/rows

sumo.utils.**load_data_text**(*file_path: str*, *sample_names: int = None*, *feature_names: int = None*, *drop_features: float = 0.1*, *drop_samples: float = 0.1*)
Loads data from text file (with samples in columns and features in rows) into pandas.DataFrame

Args: file_path (str): path to the tab delimited .txt file sample_names (int): index of row with sample names feature_names (int): index of column with feature names drop_features (float): if percentage of missing values for feature exceeds this value, remove this feature drop_samples (float): if percentage of missing values for sample (that remains after feature dropping) exceeds this value, remove this sample

Returns: data (pandas.DataFrame): data frame loaded from file, with missing values removed

sumo.utils.**load_npz**(*file_path: str*)
Load data from .npz file

Args: file_path (str): path to .npz file

Returns: dictionary with arrays as values and their indices used during saving to .npz file as keys

sumo.utils.**normalized_mutual_information**(*cl: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb588>, org: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb5c0>*)

Clustering accuracy measure, which takes into account mutual information between two clusterings and entropy of each cluster

sumo.utils.**plot_metric**(*x: list, y: list, xlabel='x', ylabel='y', title='', file_path: str = None, color='blue', allow_omit_xticks: bool = False*)

Create plot of median metric values, with ribbon between min and max values for each x

sumo.utils.**purity**(*cl: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb4a8>, org: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4466cdb4e0>*)

Clustering accuracy measure representing percentage of total number of nodes classified correctly

sumo.utils.**save_arrays_to_npz**(*data: Union[dict, list], file_path: str*)

Save numpy arrays to .npz file

Args: data (dict/list): list of numpy arrays or dictionary with specified keywords for every array file_path (str): optional path to output file

sumo.utils.**setup_logger**(*logger_name*, *level='INFO'*, *log_file: str = None*)

Create and configure logging object

sumo is a command-line tool to identify molecular subtypes in multi-omics datasets. It implements a novel nonnegative matrix factorization (NMF) algorithm to identify groups of samples that share molecular signatures, and provides additional modules to evaluate such assignments and identify features that drive the classification.

To see how **sumo** performs the joint factorization of patient-similarity networks and through integration of multi-omic data identifies significantly different molecular subtypes of LGG read our publication in CellReportsMethods:

> Sienkiewicz, K., Chen, J., Chatrath, A., Lawson, J. T., Sheffield, N. C., Zhang, L., & Ratan, A. (2022). Detecting molecular subtypes from multi-omics datasets using SUMO. In Cell Reports Methods (Vol. 2, Issue 1, p. 100152). Elsevier BV. https://doi.org/10.1016/j.crmeth.2021.100152

For practical details about **sumo** analysis pipeline, examples of downstream analysis and troubleshooting please refer to our published STAR Protocol and/or package documentation available at https://python-sumo.readthedocs.io.

CHAPTER 5

Installation

You can install **sumo** from PyPI, by executing command below. Please note that we require **python 3.6+**.

```
pip install python-sumo
```

(March 2021): We have noted an installation issue with the *llvmlite* package (required for one of **sumo** dependencies). To avoid errors with installation, **upgrade pip to a >19.0 version**.

CHAPTER 6

License

MIT

CHAPTER 7

Usage

sumo consists of four subroutines. A typical workflow includes running *prepare* mode for preparation of similarity matrices from feature matrices, followed by factorization of produced multiplex network (mode *run*). Third mode *evaluate* can be used for comparison of created cluster labels against biologically significant labels. A fourth mode *interpret* can be used to detect the importance of each feature in driving the classification.

(February 2022) As of SUMO v0.3, a semi-supervised classification of samples is now supported. This allows the inclusion of "a priori" knowledge about labels of fraction of samples to improve the factorization results. The supervised version of solver is automatically enabled in *sumo run*, when the '-labels' parameter is used.

7.1 prepare

Generates similarity matrices for samples based on biological data and saves them into multiplex network files.

```
usage: sumo prepare [-h] [-method METHOD] [-k K] [-alpha ALPHA] [-missing MISSING]
                    [-atol ATOL] [-sn SN] [-fn FN] [-df DF] [-ds DS] [-logfile␣
↪LOGFILE]
                    [-log {DEBUG,INFO,WARNING}] [-plot PLOT]
                    infile1,infile2,... outfile.npz

positional arguments:
  infile1,infile2,...   comma-delimited list of paths to input files, containing
                        standardized feature matrices, with samples in columns and
                        features in rows (supported types of files: ['.txt', '.txt.gz
↪',
                        '.txt.bz2', '.tsv', '.tsv.gz', '.tsv.bz2'])
  outfile.npz           path to output .npz file

optional arguments:
  -h, --help            show this help message and exit
  -method METHOD        either one method of sample-sample similarity calculation, or
                        comma-separated list of methods for every layer (available
                        methods: ['euclidean', 'cosine', 'pearson', 'spearman'],␣
↪default
```

(continues on next page)

(continued from previous page)

```
                        of euclidean)
  -k K                  fraction of nearest neighbours to use for sample similarity
                        calculation using Euclidean distance similarity (default of 0.
↪1)
  -alpha ALPHA          hypherparameter of RBF similarity kernel, for Euclidean␣
↪distance
                        similarity (default of 0.5)
  -missing MISSING      acceptable fraction of available values for assessment of
                        distance/similarity between pairs of samples - either one␣
↪value
                        or comma-delimited list for every layer (default of [0.1])
  -atol ATOL            if input files have continuous values, sumo checks if data is
                        standardized feature-wise, meaning all features should have␣
↪mean
                        close to zero, with standard deviation around one; use this
                        parameter to set tolerance of standardization checks (default␣
↪of
                        0.01)
  -sn SN                index of row with sample names for input files (default of 0)
  -fn FN                index of column with feature names for input files (default␣
↪of 0)
  -df DF                if percentage of missing values for feature exceeds this␣
↪value,
                        remove feature (default of 0.1)
  -ds DS                if percentage of missing values for sample (that remains after
                        feature dropping) exceeds this value, remove sample (default␣
↪of
                        0.1)
  -logfile LOGFILE      path to save log file, by default stdout is used
  -log {DEBUG,INFO,WARNING}
                        sets the logging level (default of INFO)
  -plot PLOT            path to save adjacency matrix heatmap(s), by default plots are
                        displayed on screen
```

Example

```
sumo prepare -plot plot.png methylation.txt,expression.txt prepared.data.npz
```

7.2 run

Cluster multiplex network using non-negative matrix tri-factorization to identify molecular subtypes.

```
usage: sumo run [-h] [-sparsity SPARSITY] [-labels labels.tsv] [-n N]
                [-method {max_value,spectral}] [-max_iter MAX_ITER] [-tol TOL]
                [-subsample SUBSAMPLE] [-calc_cost CALC_COST] [-logfile LOGFILE]
                [-log {DEBUG,INFO,WARNING}] [-h_init H_INIT] [-t T] [-rep REP]
                [-seed SEED]
                infile.npz k outdir

positional arguments:
  infile.npz            input .npz file containing adjacency matrices for every␣
↪network
                        layer and sample names (file created by running program with␣
↪mode
```

(continues on next page)

(continued from previous page)

```
                        "run") - consecutive adjacency arrays in file are indexed in
                        following way: "0", "1" ... and index of sample name vector is
                        "samples"
  k                     either one value describing number of clusters or coma-
↪delimited
                        range of values to check (sumo will suggest cluster structure
                        based on cophenetic correlation coefficient)
  outdir                path to save output files

optional arguments:
  -h, --help            show this help message and exit
  -sparsity SPARSITY    either one value or coma-delimited list of sparsity penalty
                        values for H matrix (sumo will try different values and select
                        the best results; default of [0.1])
  -labels labels.tsv    optional path to .tsv file containing some of known sample␣
↪labels
                        to be included as prior knowledge during the factorization
                        (inclusion of this parameter enables the 'supervised' mode of
                        sumo), the file should contain sample names in 'sample' and␣
↪labels
                        in 'label' column
  -n N                  number of repetitions (default of 60)
  -method {max_value,spectral}
                        method of cluster extraction (default of "max_value")
  -max_iter MAX_ITER    maximum number of iterations for factorization (default of␣
↪500)
  -tol TOL              if objective cost function value fluctuation (|Δ|) is smaller
                        than this value, stop iterations before reaching max_iter
                        (default of 1e-05)
  -subsample SUBSAMPLE  fraction of samples randomly removed from each run, cannot be
                        greater then 0.5 (default of 0.05)
  -calc_cost CALC_COST  number of steps between every calculation of objective cost
                        function (default of 20)
  -logfile LOGFILE      path to save log file (by default printed to stdout)
  -log {DEBUG,INFO,WARNING}
                        set the logging level (default of INFO)
  -h_init H_INIT        index of adjacency matrix to use for H matrix initialization␣
↪(by
                        default using average adjacency), only for unsupervised
                        classification (when no "-labels" are set)
  -t T                  number of threads (default of 1)
  -rep REP              number of times consensus matrix is created for the purpose of
                        assessing clustering quality (default of 5)
  -seed SEED            random state (none by default)
```

Example

```
sumo run -t 8 prepared.data.npz 2,5 results_dir
```

7.3 evaluate

Evaluate clustering results, given set of labels.

```
usage: sumo evaluate [-h] [-metric {NMI,purity,ARI}] [-logfile LOGFILE]
                     [-log {DEBUG,INFO,WARNING}]
                     infile.tsv labels


positional arguments:
  infile.tsv            input .tsv file containing sample names in 'sample' and
                        clustering labels in 'label' column (clusters.tsv file␣
↪created by
                        running sumo with mode 'run')
  labels                .tsv of the same structure as input file

optional arguments:
  -h, --help            show this help message and exit
  -metric {NMI,purity,ARI}
                        metric for accuracy evaluation (by default all metrics are
                        calculated)
  -logfile LOGFILE      path to save log file (by default printed to stdout)
  -log {DEBUG,INFO,WARNING}
                        sets the logging level (default of INFO)
```

Example

```
sumo evaluate results_dir/k3/clusters.tsv labels.tsv
```

7.4 interpret

Find features that support clusters separation.

```
usage: sumo interpret [-h] [-logfile LOGFILE] [-log {DEBUG,INFO,WARNING}] [-hits HITS]
                      [-max_iter MAX_ITER] [-n_folds N_FOLDS] [-t T] [-seed SEED]
                      [-sn SN] [-fn FN] [-df DF] [-ds DS]
                      sumo_results.npz infile1,infile2,... output_prefix

positional arguments:
  sumo_results.npz      path to sumo_results.npz (created by running program with mode
                        "run")
  infile1,infile2,...   comma-delimited list of paths to input files, containing
                        standardized feature matrices, with samples in columns and
                        features in rows(supported types of files: ['.txt', '.txt.gz',
                        '.txt.bz2', '.tsv', '.tsv.gz', '.tsv.bz2'])
  output_prefix         prefix of output files - sumo will create two output files (1)
                        .tsv file containing matrix (features x clusters), where the
                        value in each cell is the importance of the feature in that
                        cluster; (2) .hits.tsv file containing features of most
                        importance

optional arguments:
  -h, --help            show this help message and exit
  -logfile LOGFILE      path to save log file (by default printed to stdout)
  -log {DEBUG,INFO,WARNING}
                        sets the logging level (default of INFO)
  -hits HITS            sets number of most important features for every cluster, that
                        are logged in .hits.tsv file
  -max_iter MAX_ITER    maximum number of iterations, while searching through
```

(continues on next page)

(continued from previous page)

```
                        hyperparameter space
  -n_folds N_FOLDS      number of folds for model cross validation (default of 5)
  -t T                  number of threads (default of 1)
  -seed SEED            random state (default of 1)
  -sn SN                index of row with sample names for input files (default of 0)
  -fn FN                index of column with feature names for input files (default␣
↪of 0)
  -df DF                if percentage of missing values for feature exceeds this␣
↪value,
                        remove feature (default of 0.1)
  -ds DS                if percentage of missing values for sample (that remains after
                        feature dropping) exceeds this value, remove sample (default␣
↪of
                        0.1)
```

Example

```
sumo interpret -t 8 results_dir/k3/sumo_results.npz methylation.txt,expression.txt␣
↪interpret_results
```

Python Module Index

S

Index

R

S

U