inferelator

Release v0.5.5

May 03, 2021

Contents

1 Workflows 3

2 Model Selection & Regression Modules 13

3 Result Objects 17

4 Inferelator Tutorial 19

5 Examples 23

6 References 25

7 Change Log 27

Python Module Index 31

Index 33

The inferelator is a package for gene regulatory network inference that is based on regularized regression. It is maintained by the Bonneau lab in the Systems Biology group of the Flatiron Institute.

This repository is the actively developed inferelator package for python. It works for both single-cell and bulk transcriptome experiments. Includes AMuSR (Castro et al 2019), elements of InfereCLaDR (Tchourine et al 2018), and single-cell workflows (Jackson et al 2020).

We recommend installing this package from PyPi using `python -m pip install inferelator`. If running locally, also install `pathos` by `python -m pip install pathos` for parallelization. If running on a cluster, also install `dask` by `python -m pip install dask[complete] dask_jobqueue` for dask-based parallelization.

This package can also be installed from the github repository. Clone the inferelator GitHub repository and run `python setup.py install`.

Documentation is available at https://inferelator.readthedocs.io, and basic workflows for ****Bacillus subtilis**** and ****Saccharomyces cerevisiae**** are included with a tutorial.

All current example data and scripts are available from Zenodo

CHAPTER 1

Workflows

1.1 Workflow Constructor

Construct inferelator workflows from preprocessing, postprocessing, and regression modules

inferelator.workflow.**inferelator_workflow**(*regression=<class 'inferelator.regression.base_regression._RegressionWorkflowMixin'>*, *workflow=<class 'inferelator.workflow.WorkflowBase'>*)

Create and instantiate an Inferelator workflow.

Parameters

- **regression** (*str, RegressionWorkflow subclass*) – A class object which implements the run_regression and run_bootstrap methods for a specific regression strategy. This can be provided as a string.

  ”base” loads a non-functional regression stub.

  ”bbsr” loads Bayesian Best Subset Regression.

  ”elasticnet” loads Elastic Net Regression.

  ”sklearn” loads scikit-learn Regression.

  ”stars” loads the StARS stability Regression.

  ”amusr” loads AMuSR Regression. This requires multitask workflow.

  ”bbsr-by-task” loads Bayesian Best Subset Regression for multiple tasks. This requires multitask workflow.

  ”elasticnet-by-task” loads Elastic Net Regression for multiple tasks. This requires multitask workflow.

  Defaults to “base”.

- **workflow** (*str, WorkflowBase subclass*) – A class object which implements the necessary data loading and preprocessing to create design & response data for the regression strategy, and then the postprocessing to turn regression betas into a network. This can be provided as a string.

  "base" loads a non-functional workflow stub.

  "tfa" loads the TFA-based workflow.

  "single-cell" loads the Single Cell TFA-based workflow.

  "multitask" loads the multitask workflow.

  Defaults to "base".

Returns This returns an initialized object which has both the regression workflow and the preprocessing/postprocessing workflow. This object can then have settings assigned to it, and can be run with *.run()*

Return type Workflow instance

1.2 Common Workflow

class inferelator.workflow.**WorkflowBaseLoader**

WorkflowBaseLoader is the class to load raw data. It does no processing; it only takes data from files.

append_to_path(*var_name*, *to_append*)

Add a string to an existing path variable

Parameters

- **var_name** (*str*) – The name of the path variable (*input_dir* or *output_dir*)
- **to_append** (*str*) – The path to join to the end of the existing path variable

print_file_loading_arguments(*file_name*)

Print the settings that will be used to load a given file name.

Parameters **file_name** (*str*) – The name of the variable containing the file name (from *set_file_properties*)

set_expression_file(*tsv=None*, *hdf5=None*, *h5ad=None*, *tenx_path=None*, *mtx=None*, *mtx_barcode=None*, *mtx_feature=None*, *h5_layer=None*)

Set the type of expression data file. Current loaders include TSV, hdf5, h5ad (AnnData), and MTX sparse files. Only one of these loaders can be used; passing arguments for multiple loaders will raise a ValueError.

Parameters

- **tsv** (*str, optional*) – A path to a TSV (or tsv.gz) file which can be loaded by pandas.read_csv()
- **hdf5** (*str, optional*) – A path to a hdf5 file which can be loaded by pandas.HDFStore
- **h5ad** (*str, optional*) – A path to an AnnData hd5 file
- **tenx_path** (*Path, optional*) – A path to the folder containing the 10x mtx, barcode, and feature files
- **mtx** (*str, optional*) – A path to an mtx file
- **mtx_barcode** (*str, optional*) – A path to a list of observation names (i.e. barcodes, etc) for the mtx file

- **mtx_feature** (*str, optional*) – A path to a list of gene names for the mtx file
- **h5_layer** (*str, optional*) – The layer (in an AnnData h5) or the store key (in an hdf5) file to use. Defaults to using the first key.

set_file_loading_arguments(*file_name*, ***kwargs*)

Update the settings for a given file name. By default we assume all files can be read in as TSV files. Any arguments provided here will be passed to *pandas.read_csv()* for the file name provided.

set_file_loading_arguments('expression_matrix_file', sep=",") will read the expression_matrix_file as a CSV.

Parameters

- **file_name** (*str*) – The name of the variable containing the file name (from *set_file_properties*)
- **kwargs** – Arguments to be passed to *pandas.read_csv()*

set_file_paths(*input_dir=None*, *output_dir=None*, *expression_matrix_file=None*, *tf_names_file=None*, *meta_data_file=None*, *priors_file=None*, *gold_standard_file=None*, *gene_metadata_file=None*, *gene_names_file=None*)

Set the file paths necessary for the inferelator to run

Parameters

- **input_dir** (*str*) – A path containing the input files
- **output_dir** (*str, optional*) – A path to put the output files
- **expression_matrix_file** (*str*) – Path to the expression data If set here, this expression file will be assumed to be a TSV file. Use set_expression_file() for other file types
- **meta_data_file** (*str, optional*) – Path to the meta data TSV file
- **tf_names_file** (*str*) – Path to a list of regulator names to include in the model
- **priors_file** (*str*) – Path to a prior data file TSV file [Genes x Regulators]
- **gold_standard_file** (*str*) – Path to a gold standard data TSV file [Genes x Regulators]
- **gene_metadata_file** (*str, optional*) – Path to a genes annotation file
- **gene_names_file** (*str, optional*) – Path to a list of genes to include in the model (optional)

set_file_properties(*extract_metadata_from_expression_matrix=None*, *expression_matrix_metadata=None*, *expression_matrix_columns_are_genes=None*, *gene_list_index=None*, *metadata_handler=None*)

Set properties associated with the input data files

Parameters

- **extract_metadata_from_expression_matrix** (*bool, optional*) – A boolean flag that should be set to True if there is non-expression data in the expression matrix. If True, *expression_matrix_metadata* must be provided. Defaults to False.
- **expression_matrix_metadata** (*list(str), optional*) – A list of columns which, if provided, will be removed from the expression matrix file and kept as metadata. Defaults to None.

- **expression_matrix_columns_are_genes** (*bool, optional*) – A boolean flag indicating the orientation of the expression matrix. False reads the expression matrix as genes on rows, samples on columns. True reads the expression matrix as samples on rows, genes on columns. Defaults to False.
- **gene_list_index** (*str, optional*) – The column name in the gene metadata file which corresponds to the gene labels in the expression and prior data files. Defaults to None. Must be provided if *gene_metadata_file* was provided to *set_file_paths()*.
- **metadata_handler** (*str*) – A string which identifies the specific metadata parsing method to use. Options include "branching" or "nonbranching". Defaults to "branching".

set_network_data_flags(*use_no_prior=None*, *use_no_gold_standard=None*)

Set flags to skip using existing network data. Note that these flags will be ignored if network data is provided

Parameters

- **use_no_prior** (*bool*) – Flag to indicate the inferelator should be run without existing prior data. Will create a mock prior with no information. Highly inadvisable. Defaults to False
- **use_no_gold_standard** (*bool*) – Flag to indicate the inferelator should be run without existing gold standard data. Will create a mock gold standard with no information. Highly inadvisable. Defaults to False

class inferelator.workflow.**WorkflowBase**

WorkflowBase handles crossvalidation, shuffling, and validating priors and gold standards

run()

Execute workflow, after all configuration.

set_crossvalidation_parameters(*split_gold_standard_for_crossvalidation=None*, *cv_split_ratio=None*, *cv_split_axis=None*)

Set parameters for crossvalidation.

Parameters

- **split_gold_standard_for_crossvalidation** (*bool*) – Boolean flag indicating if the gold standard should be split. Must be set to True for other crossvalidation settings to have an effect. Defaults to False.
- **cv_split_ratio** (*float*) – The proportion of the gold standard which should be retained for scoring. The rest will be used to train the model. Must be set betweeen 0 and 1.
- **cv_split_axis** (*int, None*) – How to split the gold standard. If 0, split genes; this will take all the data for certain genes and keep it in the gold standard. These genes will be removed from the prior. If 1, split regulators; this will take all the data for certain regulators and keep it in the gold standard. These regulators will be removed from the prior. Splitting regulators is inadvisable. If None, the prior will be replaced with a downsampled gold standard. Setting this to 0 is generally the best choice. Defaults to None.

static set_output_file_names(*network_file_name=''*, *confidence_file_name=''*, *nonzero_coefficient_file_name=''*, *pdf_curve_file_name=''*, *curve_data_file_name=''*)

Set output file names. File names that end in '.gz' will be gzipped.

Parameters

- **network_file_name** (*str*) – Long-format network TSV file with TF->Gene edge information. Default is "network.tsv".

- **confidence_file_name** (*str*) – Genes x TFs TSV with confidence scores for each edge. Default is "combined_confidences.tsv"
- **nonzero_coefficient_file_name** (*str*) – Genes x TFs TSV with the number of non-zero model coefficients for each edge. Default is None (this file is not produced).
- **pdf_curve_file_name** (*str*) – PDF file with plotted curve(s). Default is "combined_metrics.pdf".
- **curve_data_file_name** (*str*) – TSV file with the data used to plot curves. Default is None (this file is not produced).

set_postprocessing_parameters(*gold_standard_filter_method=None*, *metric=None*)
Set parameters for the postprocessing engine

Parameters

- **gold_standard_filter_method** (*str*) – A flag that determines if the gold standard should be shrunk to the size of the produced model. "overlap" will only score on overlap between the gold standard and the inferred gene regulatory network. "keep_all_gold_standard" will score on the entire gold standard. Defaults to "keep_all_gold_standard".
- **metric** (*str*) – The model metric to use for scoring. Supports "precision-recall", "mcc", "f1", and "combined" Defaults to "combined".

set_run_parameters(*num_bootstraps=None*, *random_seed=None*, *use_mkl=None*)
Set parameters used during runtime

Parameters

- **num_bootstraps** (*int*) – The number of bootstraps to run. Defaults to 2.
- **random_seed** (*int*) – The random number seed to use. Defaults to 42.
- **use_mkl** (*bool*) – A flag to indicate if the intel MKL library should be used for matrix multiplication

set_shuffle_parameters(*shuffle_prior_axis=None*, *make_data_noise=None*)
Set parameters for shuffling labels on a prior axis. This is useful to establish a baseline.

Parameters

- **shuffle_prior_axis** (*int, None*) – The axis for shuffling prior labels. 0 shuffles gene labels. 1 shuffles regulator labels. None means labels will not be shuffled. Defaults to None.
- **make_data_noise** – Replace loaded data with simulated data that is entirely random. This retains type; integer data remains integer, float remains float. Gene distributions should be centered around the mean of gene expression in the original data, but is otherwise random.

1.3 Transcription Factor Activity (TFA) Workflow

Implementation for the Transcription Factor Activity (TFA) based Inferelator workflow. This workflow also has a design driver which will incorporate timecourse data. This is the standard workflow for most applications.

class inferelator.tfa_workflow.**TFAWorkFlow**
Bases: *inferelator.workflow.WorkflowBase*

TFAWorkFlow runs the timecourse driver and the TFA driver prior to regression.

run()
Execute workflow, after all configuration.

set_design_settings(*timecourse_response_driver=True*, *delTmin=None*, *delTmax=None*, *tau=None*)
Set the parameters used in the timecourse design-response driver.

Parameters

- **timecourse_response_driver** (*bool*) – A flag to indicate that the timecourse calculations should be performed. If set False, no other timecourse settings will have any effect. Defaults to True.
- **delTmin** (*int, float*) – The minimum allowed time difference between timepoints to model as a time series. Provide in the same units as the metadata time column (usually minutes). Defaults to 0.
- **delTmax** (*int, float*) – The maximum allowed time difference between timepoints to model as a time series. Provide in the same units as the metadata time column (usually minutes). Defaults to 120.
- **tau** (*int, float*) – The tau parameter. Provide in the same units as the metadata time column (usually minutes). Defaults to 45.

set_tfa(*tfa_driver=None*, *tfa_output_file=None*, *tfa_input_file=None*, *tfa_input_file_type=None*)
Perform or skip the TFA calculations; by default the design matrix will be transcription factor activity. If this is called with *tfa_driver = False*, the design matrix will be transcription factor expression. It is not necessary to call this function unless setting *tfa_driver = False*.

Parameters

- **tfa_driver** (*bool*) – A flag to indicate that the TFA calculations should be performed. Defaults to True
- **tfa_output_file** (*str, optional*) – A path to a TSV file which will be created with the calculated TFAs. Note that this file may contain TF expression if the TFA cannot be calculated for that TF. If None, no output file will be produced. Defaults to None
- **tfa_input_file** – A path to a TFA file which will be loaded and used in place of activity calculations If set, all TFA-related settings will be irrelevant. TSV file MUST be Samples X TFA. If None, the inferelator will calculate TFA Defaults to None
- **tfa_input_file_type** – A string which identifies file type. Accepts "tsv" and "h5ad". If None, assume the file is a TSV Defaults to None

1.4 Single-Cell Workflow

Run Single Cell Network Inference. This is the same network inference with some extra preprocessing functionality.

class inferelator.single_cell_workflow.**SingleCellWorkflow**
Bases: *inferelator.tfa_workflow.TFAWorkFlow*

SingleCellWorkflow has some additional preprocessing prior to calculating TFA and running regression

add_preprocess_step(*fun*, ***kwargs*)
Add a preprocessing step after count filtering but before calculating TFA or regression.

Parameters

- **fun** (str, *preprocessing.single_cell* function) – Preprocessing function. Can be provided as a string or as a function in *preprocessing.single_cell*.

"log10" will take the log10 of pseudocounts

"ln" will take the natural log of pseudocounts

"log2" will take the log2 of pseudocounts

"fft" will do the Freeman-Tukey transform

- **kwargs** – Additional arguments to the preprocessing function

run()
Execute workflow, after all configuration.

set_count_minimum(*count_minimum=None*)
Set the minimum count value for each gene (averaged over all samples)

Parameters **count_minimum** (*float*) – The mean expression value which is required to retain a gene for modeling. Data that has already been normalized should probably be filtered during normalization, not now. Defaults to None (disabled).

1.5 Multi-Task AMuSR Workflow

Run Multitask Network Inference with TFA-AMuSR.

class inferelator.amusr_workflow.**MultitaskLearningWorkflow**
Bases: *inferelator.single_cell_workflow.SingleCellWorkflow*

Class that implements multitask learning. Handles loading and validation of multiple data packages.

create_task(*task_name=None*, *input_dir=None*, *expression_matrix_file=None*, *meta_data_file=None*, *tf_names_file=None*, *priors_file=None*, *gene_names_file=None*, *gene_metadata_file=None*, *workflow_type='single-cell'*, ***kwargs*)
Create a task object and set any arguments to this function as attributes of that task object. TaskData objects are stored internally in _task_objects.

Parameters

- **task_name** (*str*) – A descriptive name for this task
- **input_dir** (*str*) – A path containing the input files
- **expression_matrix_file** (*str*) – Path to the expression data
- **meta_data_file** (*str, optional*) – Path to the meta data
- **tf_names_file** (*str*) – Path to a list of regulator names to include in the model
- **priors_file** (*str*) – Path to a prior data file
- **gene_metadata_file** (*str, optional*) – Path to a genes annotation file
- **gene_names_file** (*str, optional*) – Path to a list of genes to include in the model (optional)
- **workflow_type** (str, *inferelator.BaseWorkflow* subclass) – The type of workflow for data preprocessing. "tfa" uses the TFA workflow, "single-cell" uses the Single-Cell TFA workflow
- **kwargs** – Any additional arguments are assigned to the task object.

Returns Returns a task reference which can be additionally modified by calling any valid Workflow function to set task parameters

Return type TaskData instance

set_task_filters(*regulator_expression_filter=None*, *target_expression_filter=None*)
Set the filtering criteria for regulators and targets between tasks

Parameters

- **regulator_expression_filter** (*str, optional*) – "union" includes regulators which are present in any task, "intersection" includes regulators which are present in all tasks
- **target_expression_filter** (*str, optional*) – "union" includes targets which are present in any task, "intersection" includes targets which are present in all tasks

1.6 Cross-Validation Workflow Wrapper

This is a manager which will take an Inferelator workflow and repeatedly run it with different parameters. This is implemented using deep copies; it is therefore memory-intensive.

class inferelator.crossvalidation_workflow.**CrossValidationManager**(*workflow_object=None*)
Bases: object

Crossvalidate an Inferelator Workflow

__init__(*workflow_object=None*)
Create a new CrossValidationManager instance and give it a workflow

Parameters **workflow_object** (*Workflow*) – The workflow to run crossvalidation with

add_gridsearch_parameter(*param_name*, *param_vector*)
Set a parameter to search through by exhaustive grid search

Parameters

- **param_name** (*str*) – The workflow parameter to change for each run
- **param_vector** (*iterable*) – An iterable with values to use for the parameter

add_grouping_dropin(*metadata_column_name*, *group_size=None*, *seed=42*)
Run modeling on each group (defined by a metadata column) individually.

Parameters

- **metadata_column_name** (*str*) – Metadata column which has different values for each group
- **group_size** (*int, None*) – The maximum size of each group. Groups will be downsampled to the same size if this is not set to None. Default is None.
- **seed** (*int*) – The random seed to use for the group downsampling (this is not the same as the seed passed to the workflow)

add_grouping_dropout(*metadata_column_name*, *group_size=None*, *seed=42*)
Drop each group (defined by a metadata column) and run modeling on all of the other groups.

Parameters

- **metadata_column_name** (*str*) – Metadata column which has different values for each group
- **group_size** (*int, None*) – The maximum size of each group. Groups will be downsampled to the same size if this is not set to None. Default is None.
- **seed** (*int*) – The random seed to use for the group downsampling (this is not the same as the seed passed to the workflow)

add_size_subsampling(*size_vector*, *stratified_column_name=None*, *with_replacement=False*, *seed=42*, *size_sample_only=None*)

Resample expression data to a ratio of the original data.

Parameters

- **size_vector** (*iterable(floats)*) – An iterable with numeric ratios for downsampling. These values must be between 0 and 1.
- **stratified_column_name** (*str, None*) – Set this to stratify sampling (to maintain group size ratios). If None, do not maintain group size ratios. Default is None.
- **with_replacement** (*bool*) – Do sampling with or without replacement. Defaults to False
- **seed** – The random seed to use when selecting observations (this is not the same as the seed passed to the workflow)
- **seed** – int

CHAPTER 2

Model Selection & Regression Modules

2.1 BBSR

class inferelator.regression.bbsr_python.**BBSRRegressionWorkflowMixin**

Bayesian Best Subset Regression (BBSR)

https://doi.org/10.15252/msb.20156236

set_regression_parameters(*prior_weight=None*, *no_prior_weight=None*, *bsr_feature_num=None*, *clr_only=False*, *ordinary_least_squares_only=None*)

Set regression parameters for BBSR

Parameters

- **prior_weight** (*float*) – Weight for edges that are present in the prior network. Defaults to 1.
- **no_prior_weight** (*float*) – Weight for edges that are not present in the prior network. Defaults to 1.
- **bsr_feature_num** (*int*) – The number of features to include in best subset regression. Defaults to 10.
- **clr_only** (*bool*) – Only use Context Likelihood of Relatedness to select features for BSR, not prior edges. Defaults to False.
- **ordinary_least_squares_only** (*bool*) – Use OLS instead of Bayesian regression, for testing. Defaults to False.

2.2 AMuSR

class inferelator.regression.amusr_regression.**AMUSRRegressionWorkflowMixin**

Multi-Task AMuSR regression

https://doi.org/10.1371/journal.pcbi.1006591

set_regression_parameters(*prior_weight=None*, *lambda_Bs=None*, *lambda_Ss=None*, *heuristic_Cs=None*, *tol=None*, *relative_tol=None*)

Set regression parameters for AmUSR.

Parameters

- **prior_weight** (*numeric*) – Weight for edges that are present in the prior network. Non-prior edges have a weight of 1. Set this to 1 to weight prior and non-prior edges equally. Defaults to 1.
- **lambda_Bs** (*list(floats) or np.ndarray(floats)*) – Lambda_B values to search during model selection. If not set, lambda_B will be chosen using the heuristic lambda_b = c * sqrt(d log p / n) from Castro 2019 Defaults to not set. Must be provided if lambda_S is set.
- **lambda_Ss** (*list(floats) or np.ndarray(floats)*) – Lambda_S values to search during model selection. If not set, lambda_S will be chosen using the heuristic 0.5 < lambda_s/lambda_b < 1 from Castro 2019 Defaults to not set.
- **heuristic_Cs** (*list(floats) or np.ndarray(floats)*) – c values to search during model selection. Values of c to calculate lambda_b = c * sqrt(d log p / n), Defaults to np.logspace(np.log10(0.01), np.log10(10), 20)[::-1]. Does not have an effect if lambda_B is provided.
- **tol** (*float*) – Convergence tolerance for amusr regression
- **relative_tol** (*float*) – Relative convergence tolerance for amusr regression

2.3 Scikit-Learn

class inferelator.regression.sklearn_regression.**SKLearnWorkflowMixin**(**args*, ***kwargs*)

Use any scikit-learn regression module

set_regression_parameters(*model=None*, *add_random_state=None*, ***kwargs*)

Set parameters to use a sklearn model for regression

Parameters

- **model** (*BaseEstimator subclass*) – A scikit-learn model class
- **add_random_state** (*bool*) – Flag to include workflow random seed as "random_state" in the model
- **kwargs** (*any*) – Any arguments which should be passed to the scikit-learn model class instantiation

2.4 Elastic-Net

class inferelator.regression.elasticnet_python.**ElasticNetWorkflowMixin**(**args*, ***kwargs*)

Set default parameters to run scikit-learn ElasticNetCV

set_regression_parameters(*model=None*, *add_random_state=None*, ***kwargs*)

Set parameters to use a sklearn model for regression

Parameters

- **model** (*BaseEstimator subclass*) – A scikit-learn model class
- **add_random_state** (*bool*) – Flag to include workflow random seed as "random_state" in the model
- **kwargs** (*any*) – Any arguments which should be passed to the scikit-learn model class instantiation

2.5 StARS-Lasso

class inferelator.regression.stability_selection.**StARSWorkflowMixin**(**args*, ***kwargs*)

Stability Approach to Regularization Selection (StARS)-LASSO. StARS-Ridge is implemented on an experimental basis.

https://arxiv.org/abs/1006.3316 https://doi.org/10.1016/j.immuni.2019.06.001

set_regression_parameters(*alphas=None*, *num_subsamples=None*, *method=None*, ***kwargs*)

Set regression parameters for StARS-LASSO

Parameters

- **alphas** (*list(float)*) – A list of alpha (L1 term) values to search. Defaults to logspace between 0. and 10.
- **num_subsamples** (*int*) – The number of groups to break data into. Defaults to 20.
- **method** (*str*) – The model to use. Can choose from 'lasso' or 'ridge'. Defaults to 'lasso'. If 'ridge' is set, ridge_threshold should also be passed. Any value below ridge_threshold will be set to 0.
- **kwargs** (*any*) – Any additional arguments will be passed to the LASSO or Ridge scikit-learn object at instantiation

CHAPTER 3

Result Objects

3.1 Network File

```
network_file_name = "network.tsv"
```

The network.tsv is a long-format TSV file containing Regulator -> Target edges. This TSV file is sorted by the confidence score of the regulator (TF) -> target (gene) edge, from largest to smallest.:

```
target          regulator       combined_confidences    gold_standard   precision       ␣
↪ recall        MCC             F1
BSU24750        BSU04730        0.999986                1               1               ␣
↪ 0.00165       0.04057         0.003295
BSU13020        BSU04730        0.999984
BSU09690        BSU04730        0.99998
BSU06590        BSU04730        0.999978
BSU18510        BSU04730        0.999976
BSU25800        BSU25810        0.999975
```

If the gene and TF are in the gold standard, the gold standard for this edge is reported (1 if present, 0 if not present), and the model performance is calculated. The Precision, Recall, MCC, and F1 scores are calculated assuming that all edges above a row (with greater confidence scores) are predicted TF -> Gene interactions, and all values below are predicted to not be TF -> Gene interactions. Rows which do not contain any gold standard (either 1 or 0) indicate that the regulator or the target are not in the Genes x TFs gold standard matrix. These rows will not be scored.

Also included is a column indicating if the network edge was in the prior (1, 0, or not present if the gene or TF were not present in the prior network). The `beta.sign.sum` column is the number of times the model coefficient occurred and the sign (positive model coefficients will be reported as a positive value, and negative model coefficients will be reported as a negative value). The `var.exp.median` column reports the median amount of variance in the gene explained by the regulator.

3.2 InferelatorResults

class inferelator.postprocessing.**InferelatorResults**(*network_data*, *betas_stack*, *combined_confidences*, *metric_object*, *betas_sign=None*, *betas=None*)

For network analysis, the results produced in the output_dir are sufficient. Model development and comparisons may require to values that are not written to files. An InferelatorResults object is returned by the `workflow.run()` methods (A list of InferelatorResults objects is returned by the `CrossValidationManager.run()` method).

This object allows access to most of the internal values created by the inferelator.

name
Results name, usually set to task name. Defaults to None.

network
Network dataframe, usually written to network.tsv.gz

betas_sign
The aggregate sign of non-zero betas. This is a dataframe which is Genes x TFs

betas_stack
Count of non-zero betas. This is a dataframe which is Genes x TFs

combined_confidences
Confidence scores for tf-gene network edges. This is a dataframe which is Genes x TFs

tasks
Task result objects if there were multiple tasks. None if there were not. This is a dict, keyed by task ID

CHAPTER 4

Inferelator Tutorial

4.1 Input Data

All data provided to the inferelator should be in TSV format.

The inferelator package **requires** two data structures to function:

- A gene expression matrix which contains some expression data for *G* genes and *N* samples. Any unit is generally acceptable provided all samples are the same unit and are reasonably normalized together.
- A text list of *K* genes which should be modeled as regulators (like Transcription Factors)

The performance with no additional data is extremely poor, however. In addition to the two required data elements, there is other data which can be provided.

The most important of these additional elements is some known knowledge about regulatory connections.

- A prior knowledge connectivity matrix *[G x K]* which links the genes *G* to the regulators *K*. This matrix should have a zero where a gene is not regulated by a regulator. It should have a non-zero value where a gene is known to be regulated by a regulator. This can be as simple as a boolean matrix, but sign and magnitude will affect calculation of regulator activity.
- A gold standard connectivity matrix *[G x K]* which links the genes *G* to the regulators *K*. This matrix should have a zero where a gene is not regulated by a regulator. It should have a non-zero value where a gene is known to be regulated by a regulator. It will be interpreted as a boolean matrix, so sign and magnitude of non-zeros is not considered.

Also important is sample metadata. This is necessary if there is a time element to the samples, or if there is some grouping criteria. If time series data is included, there are two supported formats for this data. If time series data is not included, any metadata structure is valid.

The first format is **branching** and has 5 columns:

```
isTs | is1stLast | prevCol | del.t | condName
==========================================
TRUE |     f     |   NA    |   NA  |   A-1
```

(continues on next page)

(continued from previous page)

```
TRUE |      m      |    A-1    |    15   |    A-2
TRUE |      l      |    A-2    |    15   |    A-3
```

- **isTs** is TRUE or FALSE and indicates if this sample is in a time series.
- **is1stLast** is **f** if this sample is the first sample in a time series. It is **m** if this sample is a middle sample in a time series. It is **l** if this sample is the last sample in a time series. It is **NA** if this sample is not in a time series
- **prevCol** is the name of the sample which comes before this sample
- **del.t** is the time elapsed since the sample which comes before this sample
- **condName** is the name of this sample. It must match the sample name in the expression data.

The second format is **nonbranching** and has 3 columns:

```
condName | strain | time
=======================
   A-1   |    A   |  0
   A-2   |    A   |  15
   A-3   |    A   |  30
```

- **condName** is the name of this sample. It must match the sample name in the expression data.
- **strain** is the name of the sample group.
- **time** is the absolute time elapsed during this sample group's experiment.

Finally, gene metadata can also be provided. This is currently used to restrict modeling to just some genes.

4.2 Workflow setup

The inferelator is implemented on a workflow model. The first step is to create a workflow object. At this stage, the type of regression model and workflow must be chosen:

```
from inferelator import inferelator_workflow

worker = inferelator_workflow(regression="bbsr", workflow="tfa")
```

- Valid options for regression include "bbsr", "elastic-net", and "amusr".
- Valid options for workflow include "tfa", "single-cell", and "multitask".

The next step is to set the location of input data files:

```
worker.set_file_paths(input_dir=".",
                      output_dir="./output_inferelator",
                      expression_matrix_file="expression.tsv",
                      tf_names_file="regulators.tsv",
                      meta_data_file="meta_data.tsv",
                      priors_file="priors.tsv",
                      gold_standard_file="gold_standard.tsv")
```

The input directory will be added to all file locations which are not absolute paths. The output directory will be created if it does not exist.

Finally, run parameters should be set:

```
worker.set_run_parameters(num_bootstraps=5, random_seed=42)
```

This worker can now be run with:

```
network_result = worker.run()
```

4.3 Multitask Workflows

The inferelator supports inferring networks from multiple separate "tasks" at the same time. Several modeling options exist, but all must use the multitask workflow:

```
worker = inferelator_workflow(regression="amusr", workflow="multitask")
```

- **amusr** regression is a multitask learning model that shares information during regression.
- **bbsr-by-task** regression learns separate networks using the BBSR model, and then aggregates them into a joint network.
- **elasticnet-by-task** regression learns separate networks using the Elastic Net model, and then aggregates them into a joint network.

After creating a workflow, only the input, output and gold standard file location should be provided directly:

```
worker.set_file_paths(input_dir=".", output_dir="./output_network", gold_standard_
↪file="gold_standard.tsv.gz")
```

Other information should be provided to each separate task. These can be created by calling the `.create_task()` function. This function returns a task reference which can be used to set additional task properties:

```
task_1 = worker.create_task(task_name="Bsubtilis_1",
                            input_dir=".",
                            tf_names_file='tf_names.tsv',
                            meta_data_file='GSE67023_meta_data.tsv',
                            priors_file='gold_standard.tsv.gz',
                            workflow_type="tfa")
task_1.set_expression_file(tsv='GSE67023_expression.tsv.gz')

task_2 = worker.create_task(task_name="Bsubtilis_2",
                            input_dir=".",
                            tf_names_file='tf_names.tsv',
                            meta_data_file='meta_data.tsv',
                            priors_file='gold_standard.tsv.gz',
                            workflow_type="tfa")
task_2.set_expression_file(tsv='expression.tsv.gz')
```

Additional parameters can be set on the main workflow. Task references made with `.create_task()` are automatically included when the workflow is started. The workflow can then be started with `.run()`:

```
worker.set_run_parameters(num_bootstraps=5, random_seed=42)
worker.run()
```

4.4 Parallelization

The inferelator supports three major parallelization options. These can be set using a controller class. Calling the multiprocessing environment should be protected with the `if __name__ == '__main__'` pragma. This is necessary to prevent a specific error in creating new processes that occurs when `os.fork()` is unavailable. Multiprocessing options should be set prior to creating and running workflows. It is not necessary to set multiprocessing more then once per session:

```
from inferelator import MPControl

if __name__ == '__main__':
    MPControl.set_multiprocess_engine("multiprocessing")
    MPControl.client.processes = 12
    MPControl.connect()
```

- **multiprocessing** engine uses the pathos implementation of python's multiprocessing. It creates multiple processes on one computer.
- **local** engine uses no multiprocessing and runs from a single process. In some cases, python libraries (like numpy) may use multiple threads within this process.
- **dask-cluster** engine uses the dask scheduler-worker library in combination with the dask_jobqueue cluster-management library to manage processes through a job scheduler. Currently, only SLURM is supported. Correctly configuring this for your cluster may be a challenge.

CHAPTER 5

Examples

Example scripts are currently available on GitHub.

CHAPTER 6

References

- R. Bonneau et al., "The Inferelator: an algorithm for learning parsimonious regulatory networks from systems-biology data sets de novo," Genome Biology, vol. 7, p. R36, May 2006.
- A. Madar, A. Greenfield, E. Vanden-Eijnden, and R. Bonneau, "DREAM3: Network Inference Using Dynamic Context Likelihood of Relatedness and the Inferelator," PLOS ONE, vol. 5, no. 3, p. e9803, Mar. 2010.
- A. Greenfield, A. Madar, H. Ostrer, and R. Bonneau, "DREAM4: Combining Genetic and Dynamic Information to Identify Biological Networks and Dynamical Models," PLOS ONE, vol. 5, no. 10, p. e13397, Oct. 2010.
- M. Ciofani et al., "A Validated Regulatory Network for Th17 Cell Specification," Cell, vol. 151, no. 2, pp. 289–303, Oct. 2012.
- A. Greenfield, C. Hafemeister, and R. Bonneau, "Robust data-driven incorporation of prior knowledge into the inference of dynamic regulatory networks," Bioinformatics, vol. 29, no. 8, pp. 1060–1067, Apr. 2013.
- M. L. Arrieta-Ortiz et al., "An experimentally supported model of the Bacillus subtilis global transcriptional regulatory network," Molecular Systems Biology, vol. 11, no. 11, p. 839, Nov. 2015.
- O. Wilkins et al., "EGRINs (Environmental Gene Regulatory Influence Networks) in Rice That Function in the Response to Water Deficit, High Temperature, and Agricultural Environments," The Plant Cell, vol. 28, no. 10, pp. 2365–2384, Oct. 2016.
- K. Tchourine, C. Vogel, and R. Bonneau, "Condition-Specific Modeling of Biophysical Parameters Advances Inference of Regulatory Networks," Cell Reports, vol. 23, no. 2, pp. 376–388, Apr. 2018.
- D. M. Castro, N. R. de Veaux, E. R. Miraldi, and R. Bonneau, "Multi-study inference of regulatory networks for more accurate models of gene regulation," PLOS Computational Biology, vol. 15, no. 1, p. e1006591, Jan. 2019.
- E. R. Miraldi et al., "Leveraging chromatin accessibility for transcriptional regulatory network inference in T Helper 17 Cells," Genome Res., vol. 29, no. 3, pp. 449–463, Mar. 2019.
- C. A. Jackson, D. M. Castro, G.-A. Saldi, R. Bonneau, and D. Gresham, "Gene regulatory network reconstruction using single-cell RNA sequencing of barcoded genotypes in diverse environments," bioRxiv, p. 581678, Apr. 2019.

CHAPTER 7

Change Log

7.1 Inferelator v0.5.5 *April 29, 2021*

New Functionality:

- Added `.set_regression_parameters(tol=None)` to parameterize tolerances in AMuSR regression

Code Refactoring:

- Profiled and optimized AMuSR code

7.2 Inferelator v0.5.4 *April 23, 2021*

Bug Fixes:

- Fixed bug in multitask prior processing
- Fixed bug in dask cluster setup
- Suppressed stdout warning when output network MCC is not finite

7.3 Inferelator v0.5.3 *March 22, 2021*

New Functionality:

- Added the ability to control threads-per-process when using dask

Bug Fixes:

- Fixed bug in result dataframe that failed to create columns in older versions of pandas

7.4 Inferelator v0.5.2 *January 29, 2021*

New Functionality:

- Added flag `.set_shuffle_parameters(make_data_noise=True)` to model on randomly generated noise
- Output TSV files are gzipped by default
- Added `.set_output_file_names()` as interface to change output file names
- Added `.set_regression_parameters(lambda_Bs=None, lambda_Ss=None, heuristic_Cs=None)` for AMuSR regression

Bug Fixes:

- Fixed bug(s) with dask cluster scaling
- Fixed float precision bug in mutual information

Code Refactoring:

- Added additional tests
- Refactored AMuSR code

7.5 Inferelator v0.5.1 *November 22, 2020*

Bug Fixes:

- Fixed bug that prevented PDF summary figure generation

7.6 Inferelator v0.5.0 *November 14, 2020*

New Functionality:

- Changed output to include additional performance metrics (Matthews Correlation Coefficient and F1)

Bug Fixes:

- Fixed several bugs around data loading
- Fixed several float tolerance bugs

Code Refactoring:

- Added additional tests
- Improved dask cluster configurations
- Improved documentation

7.7 Inferelator v0.4.1 *August 4, 2020*

New Functionality:

- Added a regression module based on stability selection
- Added a regression module that can apply any scikit-learn regression model

Bug Fixes:

- Fixed row labels in matrix outputs

Code Refactoring:

- Added additional tests

7.8 Inferelator v0.4.0 *April 7, 2020*

New Functionality:

- Support for sparse data structures
- Support for h5 and mtx input files
- Added several flags that can change behavior of BBSR (clr_only, ols_only)

Bug Fixes:

- Changed behavior of precision-recall to average the precision of ties instead of randomly ordering

Code Refactoring:

- Refactored the core data structures from pandas to AnnData backed by numpy or scipy arrays
- Data matrices are loaded and maintained as OBS x VAR throughout the workflow. Data files which are in GENE x SAMPLE orientation can be loaded if `.set_file_properties(expression_matrix_columns_are_genes=False)` is set.
- Use sparse_dot_mkl with the intel Math Kernel Library to handle sparse (dot) dense multiplication
- Improved memory usage
- Added unit tests for dask-related functionality
- Changed a number of error messages to improve clarity

7.9 Inferelator v0.3.2 *December 19, 2019*

New Functionality:

- Improved error messages associated with misaligned data structures
- Added example script and data for the multitask workflows

Bug Fixes:

- Corrected several bugs when using the CrossValidationManager on multitask workflows

Code Refactoring:

- This is the final release which will be fully py2.7 compatible
- Additional unit testing

7.10 Inferelator v0.3.1 *December 10, 2019*

New Functionality:

- Created a CrossValidationManager which handles parameter searches on workflows. Replaces the single_cell_cv_workflow which did not generalize well.
- Workflow parameters are now set through functional setters like set_file_paths(), instead of through setting (cryptic) instance variables
- Calculated transcription factor activities can be saved to a file prior to inference. This is set with *workflow.set_tfa(tfa_output_file = "Filename.tsv")*

Bug Fixes:

- Many

Code Refactoring:

- Rebuilt the multitask workflow with TaskData objects instead managing data in many lists of things.

7.11 Inferelator v0.3.0 *July 30, 2019*

New Functionality:

- Created a MultiprocessingManger for abstract control of multiprocessing.
- Implemented a scheduler-worker model through the dask package for cluster computing.
- Implemented a map model through the pathos implementation of multiprocessing for local computing.
- Example scripts and datasets are now provided

Bug Fixes:

- Many

Code Refactoring:

- Rebuilt the core workflow
- Workflow assembly by inheritance is managed with a factory function
- Refactored regression to act as a mapped function for easier multiprocessing

Python Module Index

i

inferelator.amusr_workflow, 9
inferelator.crossvalidation_workflow, 10
inferelator.single_cell_workflow, 8
inferelator.tfa_workflow, 7
inferelator.workflow, 3

Index

Symbols

__init__() (*inferelator.crossvalidation_workflow.CrossValidationManager method*), 10

A

add_gridsearch_parameter() (*inferelator.crossvalidation_workflow.CrossValidationManager method*), 10
add_grouping_dropin() (*inferelator.crossvalidation_workflow.CrossValidationManager method*), 10
add_grouping_dropout() (*inferelator.crossvalidation_workflow.CrossValidationManager method*), 10
add_preprocess_step() (*inferelator.single_cell_workflow.SingleCellWorkflow method*), 8
add_size_subsampling() (*inferelator.crossvalidation_workflow.CrossValidationManager method*), 10
AMUSRRegressionWorkflowMixin (*class in inferelator.regression.amusr_regression*), 13
append_to_path() (*inferelator.workflow.WorkflowBaseLoader method*), 4

B

BBSRRegressionWorkflowMixin (*class in inferelator.regression.bbsr_python*), 13
betas_sign (*inferelator.postprocessing.InferelatorResults attribute*), 18
betas_stack (*inferelator.postprocessing.InferelatorResults attribute*), 18

C

combined_confidences (*inferelator.postprocessing.InferelatorResults attribute*), 18
create_task() (*inferelator.amusr_workflow.MultitaskLearningWorkflow method*), 9
CrossValidationManager (*class in inferelator.crossvalidation_workflow*), 10

E

ElasticNetWorkflowMixin (*class in inferelator.regression.elasticnet_python*), 14

I

inferelator.amusr_workflow (*module*), 9
inferelator.crossvalidation_workflow (*module*), 10
inferelator.single_cell_workflow (*module*), 8
inferelator.tfa_workflow (*module*), 7
inferelator.workflow (*module*), 3
inferelator_workflow() (*in module inferelator.workflow*), 3
InferelatorResults (*class in inferelator.postprocessing*), 18

M

MultitaskLearningWorkflow (*class in inferelator.amusr_workflow*), 9

N

name (*inferelator.postprocessing.InferelatorResults attribute*), 18
network (*inferelator.postprocessing.InferelatorResults attribute*), 18

P

print_file_loading_arguments() (*inferelator.workflow.WorkflowBaseLoader method*), 4

R

run() (*inferelator.single_cell_workflow.SingleCellWorkflow method*), 9
run() (*inferelator.tfa_workflow.TFAWorkFlow method*), 7
run() (*inferelator.workflow.WorkflowBase method*), 6

S

set_count_minimum() (*inferelator.single_cell_workflow.SingleCellWorkflow method*), 9
set_crossvalidation_parameters() (*inferelator.workflow.WorkflowBase method*), 6
set_design_settings() (*inferelator.tfa_workflow.TFAWorkFlow method*), 8
set_expression_file() (*inferelator.workflow.WorkflowBaseLoader method*), 4
set_file_loading_arguments() (*inferelator.workflow.WorkflowBaseLoader method*), 5
set_file_paths() (*inferelator.workflow.WorkflowBaseLoader method*), 5
set_file_properties() (*inferelator.workflow.WorkflowBaseLoader method*), 5
set_network_data_flags() (*inferelator.workflow.WorkflowBaseLoader method*), 6
set_output_file_names() (*inferelator.workflow.WorkflowBase static method*), 6
set_postprocessing_parameters() (*inferelator.workflow.WorkflowBase method*), 7
set_regression_parameters() (*inferelator.regression.amusr_regression.AMUSRRegressionWorkflowMixin method*), 14
set_regression_parameters() (*inferelator.regression.bbsr_python.BBSRRegressionWorkflowMixin method*), 13
set_regression_parameters() (*inferelator.regression.elasticnet_python.ElasticNetWorkflowMixin method*), 14
set_regression_parameters() (*inferelator.regression.sklearn_regression.SKLearnWorkflowMixin method*), 14
set_regression_parameters() (*inferelator.regression.stability_selection.StARSWorkflowMixin method*), 15
set_run_parameters() (*inferelator.workflow.WorkflowBase method*), 7
set_shuffle_parameters() (*inferelator.workflow.WorkflowBase method*), 7
set_task_filters() (*inferelator.amusr_workflow.MultitaskLearningWorkflow method*), 9
set_tfa() (*inferelator.tfa_workflow.TFAWorkFlow method*), 8
SingleCellWorkflow (*class in inferelator.single_cell_workflow*), 8
SKLearnWorkflowMixin (*class in inferelator.regression.sklearn_regression*), 14
StARSWorkflowMixin (*class in inferelator.regression.stability_selection*), 15

T

tasks (*inferelator.postprocessing.InferelatorResults attribute*), 18
TFAWorkFlow (*class in inferelator.tfa_workflow*), 7

W

WorkflowBase (*class in inferelator.workflow*), 6
WorkflowBaseLoader (*class in inferelator.workflow*), 4