Hypercluster

Release 0.0.2

Ruggleslab

Feb 25, 2020

CONTENTS

CHAPTER

ONE

HYPERCLUSTER PACKAGE

class hypercluster.**AutoClusterer**(*clusterer_name: Optional[str] = 'KMeans'*, *params_to_optimize: Optional[dict] = None*, *random_search: bool = False*, *random_search_fraction: Optional[float] = 0.5*, *param_weights: dict = {}*, *clus_kwargs: Optional[dict] = None*, *labels_: Optional[pandas.core.frame.DataFrame] = None*, *evaluation_: Optional[pandas.core.frame.DataFrame] = None*, *data: Optional[pandas.core.frame.DataFrame] = None*, *labels_df: Optional[pandas.core.frame.DataFrame] = None*, *evaluation_df: Optional[pandas.core.frame.DataFrame] = None*)

Bases: *hypercluster.classes.Clusterer*

Main hypercluster object.

clusterer_name
String name of clusterer.

Type str

params_to_optimize
Dictionary with possibilities for different parameters. Ex format - {'parameter_name':[1, 2, 3, 4, 5]}. If None, will optimize default selection, given in hypercluster.constants.variables_to_optimize. Default None.

Type dict

random_search
Whether to search a random selection of possible parameters or all possibilities. Default True.

Type bool

random_search_fraction
If random_search is True, what fraction of the possible parameters to search. Default 0.5.

Type float

param_weights
Dictionary of str: dictionaries. Ex format - { 'parameter_name':{'param_option_1':0.5, 'param_option_2':0.5}}.

Type dict

clus_kwargs
Additional kwargs to pass into given clusterer, but not to be optimized. Default None.

Type dict

labels_
If already fit, labels DataFrame fit to data.

Type Optional[DataFrame]

evaluation_
If already fit and evalute, evaluations per label.

Type Optional[DataFrame]

data
Data to fit, will not fit by default even if passed data.

Type Optional[DataFrame]

evaluate(*methods: Optional[Iterable[str]] = None*, *metric_kwargs: Optional[dict] = None*, *gold_standard: Optional[Iterable[T_co]] = None*)
Evaluate labels with given metrics.

Parameters

- **methods** (*Optional[Iterable[str]]*) – List of evaluation methods to use.
- **metric_kwargs** (*Optional[dict]*) – Additional kwargs per evaluation metric. Structure of {'metric_name':{'param1':value, 'param2':val2}.
- **gold_standard** (*Optional[Iterable]*) – Gold standard labels, if available. Only needed if using a metric that needs ground truth.

Returns (AutoClusterer): self with attribute .evaluation_; a DataFrame with all eval values per labels.

fit(*data: pandas.core.frame.DataFrame*)
Fits clusterer to data with each parameter set.

Parameters **data** (*DataFrame*) – DataFrame with elements to cluster as index and features as columns.

Returns (AutoClusterer): self

generate_param_sets()
Uses info from init to make a Dataframe of all parameter sets that will be tried.

Returns (AutoClusterer): self

class hypercluster.**MultiAutoClusterer**(*algorithm_names: Union[Iterable[T_co], str, None] = None, algorithm_parameters: Optional[Dict[str, dict]] = None, random_search: bool = False, random_search_fraction: Optional[float] = 0.5, algorithm_param_weights: Optional[dict] = None, algorithm_clus_kwargs: Optional[dict] = None, data: Optional[pandas.core.frame.DataFrame] = None, evaluation_methods: Optional[List[str]] = None, metric_kwargs: Optional[Dict[str, dict]] = None, gold_standard: Optional[Iterable[T_co]] = None, autoclusterers: Iterable[hypercluster.classes.AutoClusterer] = None, labels_: Dict[str, hypercluster.classes.AutoClusterer] = None, evaluation_: Dict[str, hypercluster.classes.AutoClusterer] = None, labels_df: Optional[pandas.core.frame.DataFrame] = None, evaluation_df: Optional[pandas.core.frame.DataFrame] = None*)

Bases: *hypercluster.classes.Clusterer*

Object for training multiple clustering algorithms.

algorithm_names
List of algorithm names to test OR name of category of clusterers from hypercluster.constants.categories, OR None. If None, default is hypercluster.constants.variables_to_optimize.keys().

Type Optional[Union[Iterable, str]]

algorithm_parameters
Dictionary of hyperparameters to optimize. Example format: {'clusterer_name1':{'hyperparam1':[val1, val2]}}.

Type Optional[Dict[str, dict]]

random_search
Whether to search a random subsample of possible conditions.

Type bool

random_search_fraction
If random_search, what fraction of conditions to search.

Type float

algorithm_param_weights
If random_search, and you want to give probability weights to certain parameters, dictionary of probability weights. Example format: {'clusterer1': {'hyperparam1':{val1:probability1, val2:probability2}}}.

Type Dict[str, Dict[str, dict]]

algorithm_clus_kwargs
Dictionary of additional keyword args for any clusterer. Example format: {'clusterer1':{'param1':val1}}.

Type Dict[str, dict]

data
Optional, data to fit. Will not fit even if passed, need to call fit method.

Type Optional[DataFrame]

evaluation_methods
List of metrics with which to evaluate. If None, will use hypercluster.constants.inherent_metrics. Default is None.

Type Optional[List[str]]

metric_kwargs
Additional keyword args for any metric function. Example format: {'metric1':{'param1':value}}.

Type Optional[Dict[str, dict]]

gold_standard
If using methods that need ground truth, vector of correct labels. Can also pass in during evaluate.

Type Optional[Iterable]

autoclusterers
If building from initialized AutoClusterer objects, can give a list of them here. If these are given, it will override anything

Type Iterable[*AutoClusterer*]

passed to labels_ and evaluation_.

labels_
Dictionary of label DataFrames per clusterer, if already fit. Example format: {'clusterer1': labels_df}.

Type Optional[Dict[str, DataFrame]]

evaluation_
Dictionary of evaluation DataFrames per clusterer, if already fit and evaluated. Example format: {'clusterer1': evaluation_df}.

Type Optional[Dict[str, DataFrame]]

labels_df
Combined DataFrame of all labeling results.

Type Optional[DataFrame]

evaluation_df
Combined DataFrame of all evaluation results.

Type Optional[DataFrame]

evaluate(*evaluation_methods: Optional[list] = None*, *metric_kwargs: Optional[dict] = None*, *gold_standard: Optional[Iterable[T_co]] = None*)

fit(*data: Optional[pandas.core.frame.DataFrame] = None*)

1.1 hypercluster.classes module

class hypercluster.classes.**AutoClusterer**(*clusterer_name: Optional[str] = 'KMeans', params_to_optimize: Optional[dict] = None, random_search: bool = False, random_search_fraction: Optional[float] = 0.5, param_weights: dict = {}, clus_kwargs: Optional[dict] = None, labels_: Optional[pandas.core.frame.DataFrame] = None, evaluation_: Optional[pandas.core.frame.DataFrame] = None, data: Optional[pandas.core.frame.DataFrame] = None, labels_df: Optional[pandas.core.frame.DataFrame] = None, evaluation_df: Optional[pandas.core.frame.DataFrame] = None*)

Bases: *hypercluster.classes.Clusterer*

Main hypercluster object.

clusterer_name
String name of clusterer.

Type str

params_to_optimize
Dictionary with possibilities for different parameters. Ex format - {'parameter_name':[1, 2, 3, 4, 5]}. If None, will optimize default selection, given in hypercluster.constants.variables_to_optimize. Default None.

Type dict

random_search
Whether to search a random selection of possible parameters or all possibilities. Default True.

Type bool

random_search_fraction
If random_search is True, what fraction of the possible parameters to search. Default 0.5.

Type float

param_weights
Dictionary of str: dictionaries. Ex format - { 'parameter_name':{'param_option_1':0.5, 'param_option_2':0.5}}.

Type dict

clus_kwargs
Additional kwargs to pass into given clusterer, but not to be optimized. Default None.

Type dict

labels_
If already fit, labels DataFrame fit to data.

Type Optional[DataFrame]

evaluation_
If already fit and evalute, evaluations per label.

Type Optional[DataFrame]

data
Data to fit, will not fit by default even if passed data.

Type Optional[DataFrame]

evaluate(*methods: Optional[Iterable[str]] = None*, *metric_kwargs: Optional[dict] = None*, *gold_standard: Optional[Iterable[T_co]] = None*)
Evaluate labels with given metrics.

Parameters

- **methods** (*Optional[Iterable[str]]*) – List of evaluation methods to use.
- **metric_kwargs** (*Optional[dict]*) – Additional kwargs per evaluation metric. Structure of {'metric_name':{'param1':value, 'param2':val2}.
- **gold_standard** (*Optional[Iterable]*) – Gold standard labels, if available. Only needed if using a metric that needs ground truth.

Returns (AutoClusterer): self with attribute .evaluation_; a DataFrame with all eval values per labels.

fit(*data: pandas.core.frame.DataFrame*)
Fits clusterer to data with each parameter set.

Parameters **data** (*DataFrame*) – DataFrame with elements to cluster as index and features as columns.

Returns (AutoClusterer): self

generate_param_sets()
Uses info from init to make a Dataframe of all parameter sets that will be tried.

Returns (AutoClusterer): self

class hypercluster.classes.**Clusterer**
Bases: object

Meta class for shared methods for both AutoClusterer and MultiAutoClusterer.

fit_predict(*data: Optional[pandas.core.frame.DataFrame], parameter_set_name, method, min_of_max*)

pick_best_labels(*method: Optional[str] = None*, *min_or_max: Optional[str] = None*)

visualize_evaluations(*savefig: bool = False*, *output_prefix: str = 'evaluations'*, ***heatmap_kws*) → List[matplotlib.axes._axes.Axes]

visualize_for_picking_labels(*method: Optional[str] = None*, *savefig_prefix: Optional[str] = None*)

visualize_label_agreement(*method: Optional[str] = None*, *savefig: bool = False*, *output_prefix: Optional[str] = None*, ***heatmap_kws*) → List[matplotlib.axes._axes.Axes]

visualize_sample_label_consistency(*savefig: bool = False*, *output_prefix: Optional[str] = None*, ***heatmap_kws*) → List[matplotlib.axes._axes.Axes]

class hypercluster.classes.**MultiAutoClusterer**(*algorithm_names: Union[Iterable[T_co], str, None] = None, algorithm_parameters: Optional[Dict[str, dict]] = None, random_search: bool = False, random_search_fraction: Optional[float] = 0.5, algorithm_param_weights: Optional[dict] = None, algorithm_clus_kwargs: Optional[dict] = None, data: Optional[pandas.core.frame.DataFrame] = None, evaluation_methods: Optional[List[str]] = None, metric_kwargs: Optional[Dict[str, dict]] = None, gold_standard: Optional[Iterable[T_co]] = None, autoclusterers: Iterable[hypercluster.classes.AutoClusterer] = None, labels_: Dict[str, hypercluster.classes.AutoClusterer] = None, evaluation_: Dict[str, hypercluster.classes.AutoClusterer] = None, labels_df: Optional[pandas.core.frame.DataFrame] = None, evaluation_df: Optional[pandas.core.frame.DataFrame] = None*)

Bases: *hypercluster.classes.Clusterer*

Object for training multiple clustering algorithms.

algorithm_names
List of algorithm names to test OR name of category of clusterers from hypercluster.constants.categories, OR None. If None, default is hypercluster.constants.variables_to_optimize.keys().

Type Optional[Union[Iterable, str]]

algorithm_parameters
Dictionary of hyperparameters to optimize. Example format: {'clusterer_name1':{'hyperparam1':[val1, val2]}}.

Type Optional[Dict[str, dict]]

random_search
Whether to search a random subsample of possible conditions.

Type bool

random_search_fraction
If random_search, what fraction of conditions to search.

Type float

algorithm_param_weights
If random_search, and you want to give probability weights to certain parameters, dictionary of probability weights. Example format: {'clusterer1': {'hyperparam1':{val1:probability1, val2:probability2}}}.

Type Dict[str, Dict[str, dict]]

algorithm_clus_kwargs
Dictionary of additional keyword args for any clusterer. Example format: {'clusterer1':{'param1':val1}}.

Type Dict[str, dict]

data
Optional, data to fit. Will not fit even if passed, need to call fit method.

Type Optional[DataFrame]

evaluation_methods
List of metrics with which to evaluate. If None, will use hypercluster.constants.inherent_metrics. Default is None.

Type Optional[List[str]]

metric_kwargs
Additional keyword args for any metric function. Example format: {'metric1':{'param1':value}}.

Type Optional[Dict[str, dict]]

gold_standard
If using methods that need ground truth, vector of correct labels. Can also pass in during evaluate.

Type Optional[Iterable]

autoclusterers
If building from initialized AutoClusterer objects, can give a list of them here. If these are given, it will override anything

Type Iterable[*AutoClusterer*]

passed to labels_ and evaluation_.

labels_
Dictionary of label DataFrames per clusterer, if already fit. Example format: {'clusterer1': labels_df}.

Type Optional[Dict[str, DataFrame]]

evaluation_
Dictionary of evaluation DataFrames per clusterer, if already fit and evaluated. Example format: {'clusterer1': evaluation_df}.

Type Optional[Dict[str, DataFrame]]

labels_df
Combined DataFrame of all labeling results.

Type Optional[DataFrame]

evaluation_df
Combined DataFrame of all evaluation results.

Type Optional[DataFrame]

evaluate(*evaluation_methods: Optional[list] = None, metric_kwargs: Optional[dict] = None, gold_standard: Optional[Iterable[T_co]] = None*)

fit(*data: Optional[pandas.core.frame.DataFrame] = None*)

1.2 hypercluster.utilities module

hypercluster.utilities.**calculate_row_weights**(*row: Iterable[T_co], param_weights: dict, vars_to_optimize: dict*) → float
Used to select random rows of parameter combinations using individual parameter weights.

Parameters

- **row** (*Iterable*) – Series of parameters, with parameter names as index.

- **param_weights** (*dict*) – Dictionary of str: dictionaries. Ex format - {'parameter_name':{ 'param_option_1':0.5, 'param_option_2':0.5}}.
- **vars_to_optimize** (*Iterable*) – Dictionary with possibilities for different parameters. Ex format - {'parameter_name':[1, 2, 3, 4, 5]}.

Returns (float): Float representing the probability of seeing that combination of parameters, given their individual weights.

hypercluster.utilities.**cluster**(*clusterer_name: str*, *data: pandas.core.frame.DataFrame*, *params: dict = {}*)

Runs a given clusterer with a given set of parameters.

Parameters

- **clusterer_name** (*str*) – String name of clusterer.
- **data** (*DataFrame*) – Dataframe with elements to cluster as index and examples as columns.
- **params** (*dict*) – Dictionary of parameter names and values to feed into clusterer. Default {}

Returns Instance of the clusterer fit with the data provided.

hypercluster.utilities.**convert_to_multiind**(*key: str*, *df: pandas.core.frame.DataFrame*) → pandas.core.frame.DataFrame

Takes columns from a single clusterer from Clusterer.labels_df or .evaluation_df and converts to a multiindexed rather than collapsed into string. Equivalent to grabbing Clusterer.labels[clusterer] or .evaluations[clusterer]. Opposite of generate_flattened_df.

Parameters

- **key** (*str*) – Name of clusterer, must match beginning of columns to convert.
- **df** (*DataFrame*) – Dataframe to grab chunk from.

Returns Subset DataFrame with multiindex.

hypercluster.utilities.**evaluate_one**(*labels: Iterable[T_co]*, *method: str = 'silhouette_score'*, *data: Optional[pandas.core.frame.DataFrame] = None*, *gold_standard: Optional[Iterable[T_co]] = None*, *metric_kwargs: Optional[dict] = None*) → dict

Uses a given metric to evaluate clustering results.

Parameters

- **labels** (*Iterable*) – Series of labels.
- **method** (*str*) – Str of name of evaluation to use. Default is silhouette.
- **data** (*DataFrame*) – If using an inherent metric, must provide DataFrame with which to calculate the metric.
- **gold_standard** (*Iterable*) – If using a metric that compares to ground truth, must provide a set of gold standard labels.
- **metric_kwargs** (*dict*) – Additional kwargs to use in evaluation.

Returns (float): Metric value

hypercluster.utilities.**generate_flattened_df**(*df_dict: Dict[str, pandas.core.frame.DataFrame]*) → pandas.core.frame.DataFrame

Takes dictionary of results from many clusterers and makes 1 DataFrame. Opposite of convert_to_multiind.

Parameters **df_dict** (*Dict[str, DataFrame]*) – Dictionary of dataframes to flatten. Can be .labels_ or .evaluations_ from MultiAutoClusterer.

Returns Flattened DataFrame with all data.

hypercluster.utilities.**pick_best_labels**(*evaluation_results_df: pandas.core.frame.DataFrame*, *clustering_labels_df: pandas.core.frame.DataFrame*, *method: Optional[str] = None*, *min_or_max: Optional[str] = None*) → Iterable[T_co]

From evaluations and a metric to minimize or maximize, return all labels with top pick.

Parameters

- **evaluation_results_df** (*DataFrame*) – Evaluations DataFrame from optimize_clustering.
- **clustering_labels_df** (*DataFrame*) – Labels DataFrame from optimize_clustering.
- **method** (*str*) – Method with which to choose the best labels.
- **min_or_max** (*str*) – Whether to minimize or maximize the metric. Must be 'min' or 'max'.

Returns (DataFrame): DataFrame of all top labels.

1.3 hypercluster.visualize module

hypercluster.visualize.**compute_order**(*df*, *dist_method: str = 'euclidean'*, *cluster_method: str = 'average'*)

Gives hierarchical clustering order for the rows of a DataFrame

Parameters

- **df** (*DataFrame*) – DataFrame with rows to order.
- **dist_method** (*str*) – Distance method to pass to scipy.cluster.hierarchy.linkage.
- **cluster_method** (*str*) – Clustering method to pass to scipy.spatial.distance.pdist.

Returns (pandas.Index): Ordered row index.

hypercluster.visualize.**visualize_evaluations**(*evaluations_df: pandas.core.frame.DataFrame*, *savefig: bool = False*, *output_prefix: str = 'evaluations'*, ***heatmap_kws*) → List[matplotlib.axes._axes.Axes]

Makes a z-scored visualization of all evaluations.

Parameters

- **evaluations_df** (*DataFrame*) – Evaluations dataframe from clustering.optimize_clustering
- **output_prefix** (*str*) – If saving a figure, file prefix to use.

- **savefig** (*bool*) – Whether to save a pdf
- ****heatmap_kws** – Additional keyword arguments to pass to seaborn.heatmap.

Returns (List[matplotlib.axes.Axes]): List of all matplotlib axes.

hypercluster.visualize.**visualize_for_picking_labels**(*evaluation_df: pandas.core.frame.DataFrame*, *method: Optional[str] = None*, *savefig_prefix: Optional[str] = None*)

Generates graphs similar to a scree graph for PCA for each parameter and each clusterer.

Parameters

- **evaluation_df** (*DataFrame*) – DataFrame of evaluations to visualize. Clusterer.evaluation_df.
- **method** (*str*) – Which metric to visualize.
- **savefig_prefix** (*str*) – If not None, save a figure with give prefix.

Returns matplotlib axes.

hypercluster.visualize.**visualize_label_agreement**(*labels: pandas.core.frame.DataFrame*, *method: Optional[str] = None*, *savefig: bool = False*, *output_prefix: Optional[str] = None*, ***heatmap_kws*) → List[matplotlib.axes._axes.Axes]

Visualize similarity between clustering results given an evaluation metric.

Parameters

- **labels** (*DataFrame*) – Labels DataFrame, e.g. from optimize_clustering or **AutoClusterer.labels_**
- **method** (*str*) – Method with which to compare labels. Must be a metric like the ones in constants.need_ground_truth, which takes two sets of labels.
- **savefig** (*bool*) – Whether to save a pdf.
- **output_prefix** (*str*) – If saving a pdf, file prefix to use.
- ****heatmap_kws** – Additional keywords to pass to seaborn.heatmap

Returns (List[matplotlib.axes.Axes]): List of matplotlib axes

hypercluster.visualize.**visualize_pairwise**(*df: pandas.core.frame.DataFrame*, *savefig: bool = False*, *output_prefix: Optional[str] = None*, *method: Optional[str] = None*, ***heatmap_kws*) → List[matplotlib.axes._axes.Axes]

Visualize symmetrical square DataFrames.

Parameters

- **df** (*DataFrame*) – DataFrame to visualize.
- **savefig** (*bool*) – Whether to save a pdf.
- **output_prefix** (*str*) – If saving a pdf, file prefix to use.

- **method** (*str*) – Label for cbar, if relevant.
- ****heatmap_kws** – Additional keywords to pass to seaborn.heatmap

Returns (List[matplotlib.axes.Axes]): List of matplotlib axes for figure.

hypercluster.visualize.**visualize_sample_label_consistency**(*labels: pandas.core.frame.DataFrame*, *savefig: bool = False*, *output_prefix: Optional[str] = None*, ***heatmap_kws*) → List[matplotlib.axes._axes.Axes]

Visualize how often two samples are labeled in the same group across conditions. Interpret with care–if you use more conditions for some type of clusterers, e.g. more n_clusters for KMeans, those cluster more similarly across conditions than between clusterers. This means that more agreement in labeling could be due to the choice of clusterers rather than true similarity between samples.

Parameters

- **labels** (*DataFrame*) – Labels DataFrame, e.g. from optimize_clustering or **AutoClusterer.labels_**
- **savefig** (*bool*) – Whether to save a pdf.
- **output_prefix** (*str*) – If saving a pdf, file prefix to use.
- ****heatmap_kws** – Additional keywords to pass to seaborn.heatmap

Returns (List[matplotlib.axes.Axes]): List of matplotlib axes

hypercluster.visualize.**zscore**(*df*)

Row zscores a DataFrame, ignores np.nan

Parameters **df** (*DataFrame*) – DataFrame to z-score

Returns (DataFrame): Row-zscored DataFrame.

1.4 hypercluster.constants module

hypercluster.constants.**param_delim**

delimiter between hyperparameters for snakemake file labels and labels DataFrame columns.

hypercluster.constants.**val_delim**

delimiter between hyperparameter label and value for snakemake file labels and labels DataFrame columns.

hypercluster.constants.**categories**

Convenient groups of clusterers to use. If all samples need to be clustered, 'partitioners' is a good choice. If there are millions of samples, 'fastest' might be a good choice.

hypercluster.constants.**variables_to_optimize**

Some default hyperparameters to optimize and value ranges for a selection of commonly used clustering algoirthms from sklearn. Used as deafults for clustering.AutoClusterer and clustering.optimize_clustering.

hypercluster.constants.**need_ground_truth**

list of sklearn metrics that need ground truth labeling. "adjusted_rand_score", "adjusted_mutual_info_score", "homogeneity_score", "completeness_score", "fowlkes_mallows_score", "mutual_info_score", "v_measure_score"

hypercluster.constants.**inherent_metrics**
list of sklearn metrics that need original data for calculation. “silhouette_score”, “calinski_harabasz_score”, “davies_bouldin_score”, “smallest_largest_clusters_ratio”, “number_of_clusters”, “smallest_cluster_size”, “largest_cluster_size”

hypercluster.constants.**min_or_max**
establishing whether each sklearn metric is better when minimized or maximized for clustering.pick_best_labels.

1.5 hypercluster.additional_clusterers module

Additonal clustering classes can be added here, as long as they have a ‘fit’ method.

hypercluster.additional_clusterers.**HDBSCAN**
See hdbscan

Type clustering class

class hypercluster.additional_clusterers.**LeidenCluster**(*adjacency_method: str = 'MNN', k: int = 20, resolution: float = 0.8, adjacency_kwargs: Optional[dict] = None, partition_type: str = 'RBConfigurationVertexPartition', **leiden_kwargs*)

Bases: object

Leidein clustering on graph derived from an adjacency matrix. See reference for more info

Parameters

- **adjacency_method** – Method to use to construct adjacency matrix, which is used to construct graph that will be clustered. Valid methods are any metric valid in scipy.spatial.distance.pdist, or MNN, for mutual nearest neighbors and CNN for common nearest neighbors. Both use sklearn.neighbors.NearestNeighbors at a given k to calculate NNs. MNN then uses whether points i and j are each others NNs as edge weights. CNN uses the count of how many NNs i and j have in common as the edge weight.
- **k** – If using CNN or MNN, k to use to construct the NearestNeighbors matrix.
- **resolution** – If using ‘RBConfigurationVertexPartition’, ‘CPMVertexPartition’ which resolution to use. If using other partitioners, this is ignored but any other kwargs for those partitioners can be passed too.
- **adjacency_kwargs** – Additional keyword arguments to pass to sklearn.neighbors.NearestNeighbors or scipy.spatial.distance.pdist to construct the adjacency matrix.
- **partition_type** – Which partition to use for leiden clustering, see leidenalg for more info.
- ****leiden_kwargs** – Additional kwargs to be passed to **‘find_partition‘_**

fit(*data: pandas.core.frame.DataFrame*)

class hypercluster.additional_clusterers.**LouvainCluster**(*adjacency_method: str = 'MNN', k: int = 20, resolution: float = 0.8, adjacency_kwargs: Optional[dict] = None, partition_type: str = 'RBConfigurationVertexPartition', **louvain_kwargs*)

Bases: object

Louvain clustering on graph derived from an adjacency matrix.

Parameters

- **adjacency_method** – Method to use to construct adjacency matrix, which is used to construct graph that will be clustered. Valid methods are any metric valid in scipy.spatial.distance.pdist, or MNN, for mutual nearest neighbors and CNN for common nearest neighbors. Both use sklearn.neighbors.NearestNeighbors at a given k to calculate NNs. MNN then uses whether points i and j are each others NNs as edge weights. CNN uses the count of how many NNs i and j have in common as the edge weight.
- **k** – If using CNN or MNN, k to use to construct the NearestNeighbors matrix.
- **resolution** – If using 'RBConfigurationVertexPartition', 'CPMVertexPartition' which resolution to use. If using other partitioners, this is ignored but any other kwargs for those partitioners can be passed too.
- **adjacency_kwargs** – Additional keyword arguments to pass to sklearn.neighbors.NearestNeighbors or scipy.spatial.distance.pdist to construct the adjacency matrix.
- **partition_type** – Which partition to use for louvain clustering, see louvain-igraph for more info.
- ****louvain_kwargs** – Additional kwargs to be passed to **'find_partition'_**

fit(*data: pandas.core.frame.DataFrame*)

class hypercluster.additional_clusterers.**NMFCluster**(*n_clusters: int = 8, **nmf_kwargs*)

Bases: object

Uses non-negative factorization from sklearn to assign clusters to samples, based on the maximum membership score of the sample per component.

Parameters

- **n_clusters** – The number of clusters to find. Used as n_components when fitting.
- ****nmf_kwargs** –

fit(*data*)

If negative numbers are present, creates one data matrix with all negative numbers zeroed. Create another data matrix with all positive numbers zeroed and the signs of all negative numbers reversed. Concatenate both matrices resulting in a data matrix twice as large as the original, but with positive values only and zeros and hence appropriate for NMF. Uses decomposed matrix H, which is nxk (with n=number of samples and k=number of components) to assign cluster membership. Each sample is assigned to the cluster for which it has the highest membership score. See sklearn.decomposition.NMF

Parameters **data** (*DataFrame*) – Data to fit with samples as rows and features as columns.

Returns self with labels_ attribute.

1.6 hypercluster.additional_metrics module

More functions for evaluating clustering results. Additional metric evaluations can be added here, as long as the second argument is the labels to evaluate

hypercluster.additional_metrics.**largest_cluster_size**(_, *labels: Iterable[T_co]*) → float

Number in largest cluster

Parameters

- _ – Dummy, pass anything or None
- **labels** (*Iterable*) – Vector of sample labels.

Returns (int): Number of samples in largest cluster.

hypercluster.additional_metrics.**number_clustered**(_, *labels: Iterable[T_co]*) → float

Returns the number of clustered samples.

Parameters

- _ – Dummy, pass anything or None.
- **labels** (*Iterable*) – Vector of sample labels.

Returns (int): The number of clustered labels.

hypercluster.additional_metrics.**number_of_clusters**(_, *labels: Iterable[T_co]*) → float

Number of total clusters.

Parameters

- _ – Dummy, pass anything or None
- **labels** (*Iterable*) – Vector of sample labels.

Returns (int): Number of clusters.

hypercluster.additional_metrics.**smallest_cluster_ratio**(_, *labels: Iterable[T_co]*) → float

Number in the smallest cluster over the total samples.

Parameters

- _ – Dummy, pass anything or None.
- **labels** (*Iterable*) – Vector of sample labels.

Returns (float): Ratio of number of members in smallest over all samples.

hypercluster.additional_metrics.**smallest_cluster_size**(_, *labels: Iterable[T_co]*) → float

Number in smallest cluster

Parameters

- _ – Dummy, pass anything or None
- **labels** (*Iterable*) – Vector of sample labels.

Returns (int): Number of samples in smallest cluster.

hypercluster.additional_metrics.**smallest_largest_clusters_ratio**(_, *labels: Iterable[T_co]*) → float

Number in the smallest cluster over the number in the largest cluster.

Parameters

- _ – Dummy, pass anything or None.
- **labels** (*Iterable*) – Vector of sample labels.

Returns (float): Ratio of number of members in smallest over largest cluster.

CHAPTER

TWO

HYPERCLUSTER SNAKEMAKE PIPELINE

2.1 Line-by-line explanation of config.yml

Table 1: Explanation for config.yml

config.yml parameter	Explanation	Example from scRNA-seq workflow
`input_data_folder`	Path to folder in which input data can be found. No / at the end.	`/input_data`
`input_data_files`	List of prefixes of data files. Exclude extension, .csv, .tsv and .txt allowed.	`['input_data1', 'input_data2']`
`gold_standard_file`	File name of gold_standard_file. Must have same pandas.read_csv kwargs as the corresponding input file. Must be in input_data_folder.	`{'input_data': 'gold_standard_file.txt'}`
`read_csv_kwargs`	Per input data file, keyword args to put into pandas.read_csv. **If specifying multiindex, also put the same in output_kwargs['labels']**	`{'test_input': {'index_col':[0]}}`
`output_folder`	Path to folder in which results will be written. No / at the end.	`/hypercluster_results`
`intermediates_folder`	Name of the folder within the output_folder to put intermediate results, such as labels and evaluations per condition. No need to change this usually.	`clustering_intermediates`
`clustering_results`	Name of the folder within the output_folder to put final results. No need to change this usually.	`clustering`

**Note: Formatting of lists and dictionaries can be in python syntax (like above) or yaml syntax, or a mixture, like below. **

2.2 config.yml example from scRNA-seq workflow

```
input_data_folder: '.'
input_data_files:
  - sc_data
gold_standards:
  test_input: 'gold_standard.csv'
read_csv_kwargs:
  test_input: {'index_col':[0]}

output_folder: 'results'
intermediates_folder: 'clustering_intermediates'
clustering_results: 'clustering'

clusterer_kwargs: {}
generate_parameters_addtl_kwargs: {}

evaluations:
  - silhouette_score
  - calinski_harabasz_score
  - davies_bouldin_score
  - number_clustered
  - smallest_largest_clusters_ratio
  - smallest_cluster_ratio
eval_kwargs: {}

metric_to_choose_best: silhouette_score
metric_to_compare_labels: adjusted_rand_score
compare_samples: true

output_kwargs:
  evaluations:
    index_col: [0]
  labels:
    index_col: [0]
heatmap_kwargs: {}

optimization_parameters:
  HDBSCAN:
    min_cluster_size: &id002
    - 2
    - 3
    - 4
    - 5
  KMeans:
    n_clusters: &id001
    - 5
    - 6
    - 7
  MiniBatchKMeans:
    n_clusters: *id001
  OPTICS:
    min_samples: *id002
```

(continues on next page)

(continued from previous page)

```
  NMFCluster:
    n_clusters: *id001
  LouvainCluster: &id003
    resolution:
    - 0.2
    - 0.4
    - 0.6
    - 0.8
    - 1.0
    - 1.2
    - 1.4
    - 1.6
    k:
    - 10
    - 15
    - 20
    - 40
    - 80
    - 120
  LeidenCluster: *id003
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

FOUR

INSTALLATION AND LOGISTICS

4.1 Installation

Available via pip:

```
pip install hypercluster
```

Or bioconda:

```
conda install hypercluster
 # or
conda install -c conda-forge -c bioconda hypercluster
```

If you are having problems installing with conda, try changing your channel priority. Priority of conda-forge > bioconda > defaults is recommended.

To check channel priority: `conda config --get channels`

It should look like:

```
--add channels 'defaults'   # lowest priority
--add channels 'bioconda'
--add channels 'conda-forge'   # highest priority
```

If it doesn't look like that, try:

```
conda config --add channels bioconda
conda config --add channels conda-forge
```

4.2 Quick reference for clustering and evaluation

Table 1: Clustering algorithms

Clusterer	Type
KMeans/MiniBatch KMeans	Partitioner
Affinity Propagation	Partitioner
Mean Shift	Partitioner
DBSCAN	Clusterer
OPTICS	Clusterer
Birch	Partitioner
OPTICS	Clusterer
HDBSCAN	Clusterer
NMF	Partitioner
LouvainCluster	Partitioner
LeidenCluster	Partitioner

Table 2: Evaluations

Metric	Type
adjusted_rand_score	Needs ground truth
adjusted_mutual_info_score	Needs ground truth
homogeneity_score	Needs ground truth
completeness_score	Needs ground truth
fowlkes_mallows_score	Needs ground truth
mutual_info_score	Needs ground truth
v_measure_score	Needs ground truth
silhouette_score	Inherent metric
calinski_harabasz_score	Inherent metric
davies_bouldin_score	Inherent metric
smallest_largest_clusters_ratio	Inherent metric
number_of_clusters	Inherent metric
smallest_cluster_size	Inherent metric
largest_cluster_size	Inherent metric

4.3 Quickstart and examples

4.3.1 With snakemake:

```
snakemake -s hypercluster.smk --configfile config.yml --config input_data_files=test_
↪data input_data_folder=.
```

4.3.2 With python:

```
import pandas as pd
from sklearn.datasets import make_blobs
import hypercluster
```

(continues on next page)

(continued from previous page)

```
data, labels = make_blobs()
data = pd.DataFrame(data)
labels = pd.Series(labels, index=data.index, name='labels')

# With a single clustering algorithm
clusterer = hypercluster.AutoClusterer()
clusterer.fit(data).evaluate(
  methods = hypercluster.constants.need_ground_truth+hypercluster.constants.inherent_
↪metrics,
  gold_standard = labels
  )

clusterer.visualize_evaluations()

# With a range of algorithms

clusterer = hypercluster.MultiAutoClusterer()
clusterer.fit(data).evaluate(
  methods = hypercluster.constants.need_ground_truth+hypercluster.constants.inherent_
↪metrics,
  gold_standard = labels
  )

clusterer.visualize_evaluations()
```

Example work flows for both python and snakemake are here

Source code is available here

PYTHON MODULE INDEX

h

INDEX

G

H

I

L

M

N

P

R

S

V

Z