nachos

Release 0.0.1

Matthew Wiesner

Mar 30, 2023

CONTENTS:

1 nachos.constraints package **1**
- 1.1 Submodules . . . 1
- 1.2 nachos.constraints.Constraints module . . . 1
- 1.3 nachos.constraints.abstract_constraint module . . . 2
- 1.4 nachos.constraints.kl module . . . 2
- 1.5 nachos.constraints.mean module . . . 3
- 1.6 nachos.constraints.mean_tuple module . . . 3
- 1.7 nachos.constraints.sum module . . . 4
- 1.8 nachos.constraints.sum_tuple module . . . 5
- 1.9 Module contents . . . 5

2 nachos.data package **7**
- 2.1 Submodules . . . 7
- 2.2 nachos.data.Data module . . . 7
- 2.3 nachos.data.Input module . . . 12
- 2.4 Module contents . . . 13

3 nachos package **15**
- 3.1 Subpackages . . . 15
- 3.2 Submodules . . . 20
- 3.3 nachos.utils module . . . 20
- 3.4 Module contents . . . 20

4 Indices and tables **21**

Python Module Index **23**

Index **25**

CHAPTER

ONE

NACHOS.CONSTRAINTS PACKAGE

1.1 Submodules

1.2 nachos.constraints.Constraints module

class nachos.constraints.Constraints.**Constraints**(*fns*, *weights*)

Bases: object

classmethod build(*conf*)

__init__(*fns*, *weights*)

__call__(*u*, *s*, *n=None*)

Summary: This function computes the discompatibility score according to predined constraints (self.fns) of a split s.

Parameters

- **u** (Dataset) – A dataset
- **s** (Tuple[set, set]) – A proposed split
- **n** (*Optional[int]*) – The index of the constraint with respect to which to compute the discompatibility score. None means compute the weighted sum of all constraints

Returns The discompatibility score

Return type float

stats(*u*, *s*)

Compute the "stats" associated with each constraint on the split.

Parameters

- **u** (Dataset) – The Dataset from which a subset is drawn
- **s** (*set*) – The proposed subset of the dataset

Returns dictionary of the scores for the set s according to the constraints specified in this class

Return type dict

1.3 nachos.constraints.abstract_constraint module

class nachos.constraints.abstract_constraint.**AbstractConstraint**

Bases: abc.ABC

abstract classmethod build(*conf*)

__init__()

abstract __call__(*c1*, *c2*)

Call self as a function.

Return type float

1.4 nachos.constraints.kl module

class nachos.constraints.kl.**KL**(*smooth=1e-06*, *direction='forward'*)

Bases: *nachos.constraints.abstract_constraint.AbstractConstraint*

Summary: Defines the constraint on the categorical distribution over values between two datasets. The cost of mismatch is computed as the kl-divergence between two sets. In general, the smaller set is the test set and we would like it to have specific characteristics w/r to the large (training) set. The forward kl, i.e.,

The forward KL, i.e.,

$$kl\left(p||q_\theta\right)$$

is mean seeking

cost = KL(d1_train || d2_test)

This will encourge selecting data with good coverage of the dataset, including data points that may have been seen only occasionally in the training data. See ReverseKL, Jeffrys for more information.

Reverse KL is

$$kl\left(q_\theta||p\right)$$

cost = KL(d2_test || d1_train)

This encourages mode seeking behavior.

The Jeffry's divergence symmetrizes the KL divergence as

$$\frac{1}{2}\left[KL\left(p||q_\theta\right) + KL\left(q_\theta||p\right)\right]$$

classmethod build(*conf*)

__init__(*smooth=1e-06*, *direction='forward'*)

__call__(*c1*, *c2*)

Summary: Computes the KL divergence between the empircal distributions defined by values in c1 and values in c2.

Parameters

- **c1** (*Union[list, Generator]*) – the values to constrain seen in dataset 1

- **c2** (*Union[list, Generator]*) – the values to constrain seen in dataset 2

Returns how closely (0 is best) the sets c1, c2 satisfy the constraint

Return type float

1.5 nachos.constraints.mean module

class nachos.constraints.mean.**Mean**

Bases: *nachos.constraints.abstract_constraint.AbstractConstraint*

Summary: Defines a constraint on the mean value of a factor. The constraint is that the mean between two Datasets (defined by the Dataset class) should be the same. This class just computes the difference between the means and returns that as a float. Instead of working with the Dataset class directly, this class works on the constraint values in that class.

classmethod build(*conf*)

__call__(*c1*, *c2*)

Summary: Computes

$$|\frac{1}{|c1|}\sum c1 - \frac{1}{|c2|}\sum c2|$$

Parameters

- **c1** (*Union[list, Generator]*) – the list of values to constrain associated with dataset 1
- **c2** (*Union[list, Generator]*) – the list of values to constrain associated with dataset 2

Returns the constraint score (how close the constraints are met)

Return type float

stat(*c1*)

Summary: computes the mean of the values in c1.

Parameters **c1** (Union[list, Generator]) – the list of values over which to compute the mean

Return type float

1.6 nachos.constraints.mean_tuple module

class nachos.constraints.mean_tuple.**MeanTuple**(*s1_mean*, *s2_mean*)

Bases: *nachos.constraints.mean.Mean*

Summary: Defines the constraint on the mean value of a factor. The constraint is that the mean for two datasets should be close to a specified value.

classmethod build(*conf*)

__init__(*s1_mean*, *s2_mean*)

__call__(*c1*, *c2*)

Summary: Given a tuple

$$\mu = (\mu_1, \mu_2)$$

compute

$$|\frac{1}{|c1|}\sum c1 - \mu_1| + |\frac{1}{|c2|}\sum c2 - \mu_2|$$

Parameters

- **c1** (*Union[list, Generator]*) – the list of values to constrain associated with dataset 1
- **c2** (*Union[list, Generator]*) – the list of values to constrain associated with dataset 2

Returns the constraint score (how close the constraints are met)

Return type float

1.7 nachos.constraints.sum module

class nachos.constraints.sum.**Sum**

Bases: *nachos.constraints.abstract_constraint.AbstractConstraint*

Summary: Defines the constraint on the mean value of a factor. The constraint is that the mean for two datasets should be close to a specified value.

classmethod build(*conf*)

__call__(*c1*, *c2*)

Summary: Computes

$$|\sum c1 - \sum c2|$$

Parameters

- **c1** (*Union[list, Generator]*) – the list of values to constrain associated with dataset 1
- **c2** (*Union[list, Generator]*) – the list of values to constrain associated with dataset 2

Returns the constraint score (how close the constraints are met)

Return type float

stat(*c1*)

Summary: computes the sum of the values in c1.

Parameters **c1** (Union[list, Generator]) – the list of values over which to compute the sum

Return type float

1.8 nachos.constraints.sum_tuple module

class nachos.constraints.sum_tuple.**SumTuple**(*s1_sum*, *s2_sum*)

Bases: *nachos.constraints.sum.Sum*

Summary: Defines the constraint on the mean value of a factor. The constraint is that the mean for two datasets should be close to a specified value.

classmethod **build**(*conf*)

__init__(*s1_sum*, *s2_sum*)

__call__(*c1*, *c2*)

Summary: Computes

$$|\sum c1 - \mu_1| + |\sum c2 - \mu_2|$$

Parameters

- **c1** (*Union[list, Generator]*) – the list of values to constrain associated with dataset 1
- **c2** (*Union[list, Generator]*) – the list of values to constrain associated with dataset 2

Returns the constraint score (how close the constraints are met)

Return type float

1.9 Module contents

nachos.constraints.**register**(*name*)

nachos.constraints.**build_constraints**(*conf*)

CHAPTER

TWO

NACHOS.DATA PACKAGE

2.1 Submodules

2.2 nachos.data.Data module

class nachos.data.Data.**Data**(*id*, *factors*, *field_names=None*)

Bases: object

Summary: A structure to store the factors (including those that will be used as constraints) associated with records in a tsv file, dataframe, or lhotse manifest.

__init__(*id*, *factors*, *field_names=None*)

copy()

Return type *Data*

class nachos.data.Data.**Dataset**(*data*, *factor_idxs*, *constraint_idxs*)

Bases: object

Summary: A class to store and manipulate the data and their associated factors and constraints. The structure we ultimately want is similar to an inverted index.

factors = [

{ factor1_value1: [fid1, fid2, . . .], factor1_value2: [fids, . . .], . . .

}, {

factor2_value1: [. . .], factor2_value2: [. . .],

]

__init__(*data*, *factor_idxs*, *constraint_idxs*)

subset_from_data(*d*)

Summary: Create a new subset, with the same factors and constraints as self, from a subset of the data points.

Parameters **d** (*Iterable[Data]*) – The data points from which to create a Dataset

Returns A Dataset object representing the subset of points

Return type *Dataset*

subset_from_records(*r*)

Summary: Create a new subset, with the same factors and constraints as self, from a subset of the data points.

Return type *Dataset*

check_complete()

Summary: Checks if the graph is complete :return: True if complete, False otherwise :rtype: bool

Return type bool

check_disconnected()

Summary: Checks if the graph if there are M > 1 disconnected components in the graph.

Returns True is disconnected, False otherwise

Return type bool

make_graph(*simfuns*)

Summary: Makes the graph representation of the dataset. This assumes that the graph is undirected, an assumption which we may later break, depending on the kinds of similarity functiosn we will ultimately support. It also makes subgraphs corresponding to each individual factor value. This is like the inverted index. You can lookup the neighbors of a factors. The graph and factors are stored in self.graph and self.graphs respectively.

param simfuns the similarity functions (1 per factor) used to compare records (i.e., data points)

type simfuns nachos.SimilarityFunctions.SimilarityFunctions

Return type None

get_record(*i*)

Return type Any

export_graph(*filename*)

Summary: Exports graph to .gml file which in theory can be read for visualization.

Parameters **filename** (*str*) – the filename of the .gml file to create

Returns None

Return type None

get_constraints(*subset=None*, *n=None*)

Summary: Returns a generator over the dataset constraints.

Parameters

- **subset** (*Optional[Iterable] (Default is None) which means use all ids.*) – Iterable of subset of ids to use
- **n** (*Optional[int]*) – The constraint index to return. By default it is None, which means to return all the constraints.

Returns generator over constraints

Return type Generator

get_factors(*subset=None*, *n=None*)

Summary: Returns a generator over the dataset factors.

Parameters

- **subset** (*Optional[Iterable] (Default is None) which means use all ids.*) – Iterable of subset of ids to use
- **n** (*Optional[int]*) – The factor index to return. By default it is None, which means to return all factors.

Returns generator over factors

Return type Generator

make_constraint_inverted_index()

Summary: Sets the inverted index for the constraints. In other words inverted_index[n] = [value1, value2, ...], the set of value seen for the n-th constraint.

Return type None

make_factor_inverted_index()

Summary: Returns the inverted index for the factors. In other words inverted_index[n] = [value1, value2, ...], the set of value seen for the n-th factor. This is really not a particularly useful function, as the inverted index computed in this way only works for the set_intersection similarity method. For other types of similarity, such as cosine distance, self.make_graph() will make a the graphs corresponding to each factor, and is really a better version of the inverted index created in this function.

This function therefore exists mostly to mirror what the make_cosntraint_inverted_index function.

Return type None

draw_random_split_from_factor(*n*)

Summary: Return a set of Data point ids and its complement corresponding to the inclusion of a subset of values selected from the n-th factor into the "training" set. We also return the index of the set from the powerset of values that resulted in the split.

Parameters **n** (*int*) – the index of the factor in the list self.factor_idxs from which to select

Returns The tuple of the index of the set from the powerset of values and the datasets corresponding to the random split and it's complement resulting from that index

Return type Tuple[int, Tuple[set, set]]

draw_split_from_factor(*n*, *idx*)

Summary: Like draw_random_split from factor, but draws the split specified by an integer index, idx, which specifies the subset of values from the powerset of values from factor n to use.

Parameters

- **n** (*int*) – the index of the factor in the list self.factor_idxs from which to select
- **idx** (*int*) – The index in the powerset of the subset of values from the n-th factor to use.

Returns The tuple of the index of the set from the powerset of values and the datasets corresponding to the random split and it's complement resulting from that index

Return type Tuple[int, Tuple[set, set]]

draw_random_split()

Summary: Applies self.draw_random_split_from_factor() to each factor independently, and returns all of the splits.

Returns The keys (indices into the powersets of values for each factor), and the values (the selected Dataset and its complement) for each factor.

Return type Tuple[List[int], List[Tuple[*Dataset*, *Dataset*]]]

set_random_seed(*seed=0*)

Summary: Set the random seed of the random module

Parameters **seed** (*int*) – Default to 0. It's the random module's random seed

Return type None

nearby_splits(*idxs*, *split*)

Summary: Make a generator over "neaby splits". These are splits that are Hamming distance 1 away from the current split. By this we mean if you concatenated the bit strings representing the indices of the powersets of values for each factor, then any bit string that differs in a single value.

Parameters

- **idx** – The indices into the powersets of the subset corresponding to split
- **split** (*FactoredSplit*) – a split (a factored split actually) around which we want to find splits that are Hamming distance = 1 away

Returns a generator over the neighboring splits

Return type Generator[FactoredSplit]

`get_neighborhood`(*idxs*, *split*, *l*, *max_neighbors=2000*)

Summary: Return a generator over all of the neighbors at distance l from split.

Parameters

- **idxs** (*List[int]*) – The list of indices, for each factor into their respective powersets of the corresponding to the splits
- **split** (*FactoredSplit*) – The split whose neighbors at distance l we want to generate.
- **l** (*int*) – The distance from split of the neighbors we would like to generate
- **max_neighbors** (*int*) – The maximum number of neighbors to explore

Returns A generator over the neighbors at distance l from split

Return type Generator[FactoredSplit]

`shake`(*idxs*, *split*, *k*)

Summary: Return a random split from the neighborhood around split.

Parameters

- **idx** – The index o
- **split** (*FactoredSplit*) – The current split around which we will select a random neighbor
- **k** (*int*) – The distance from the split form which our new split, obtained by shaking will be drawn from. Kind of like a shake distance

Returns The randomly selected split from the neighborhood of split

Return type FactoredSplit

`draw_random_node_cut`()

Summary: Draw random, non-adjacenent verticies as source and target nodes, and compute the minimum st-vertex cut. This cut may result in > 2 components. In this case, randomly assign the components to different splits.

Returns the split of components

Return type Split

`make_overlapping_test_sets`(*split*)

Summary: Takes a split of the dataset, i.e., two subsets of the dataset that do not overlap in the specified factors, and from the remaining data in the dataset not included in the split, creates multiple test sets that have some overlap with respect to one or more factors in the first of the two subsets in split.

In general, there are 2^N different kinds of overlap when using N factors. By overlap, we mean factors that are considered under the similarity function used to create the graph. We can use the factors specific graphs for this purpose.

Parameters **split** (*Split*) – The split (i.e., two subsets of the data sets) with no factor overlap with respect to which we are making the additional test sets.

Returns Test sets

Return type List[set]

overlap_stats(*s1*, *s2*)

Summary: Compute the overlap s2 w/r s1 "stats" associated with each factor.

Parameters

- **s1** (*set*) – The set with respect to which overlap will be computed
- **s2** (*set*) – the set whose overlap is computed with respect to s1

Returns The dictionary of factors overlaps (s2 w/r to s1)

Return type dict

nachos.data.Data.**collapse_factored_split**(*split*)

Summary: Take a FactoredSplit and collapse it by intersecting all the selected set, and intersecting all of their complements to create a single selected set and a single other split with no overlap in any of the factors present in the selected set.

Parameters **split** (*FactoredSplit*) – The split to collapse

Returns the collapsed split

Return type Split

2.3 nachos.data.Input module

class nachos.data.Input.**TSVLoader**
Bases: object

static load(*fname*, *config*)

class nachos.data.Input.**PandasLoader**
Bases: object

__init__()

class nachos.data.Input.**LhotseLoader**
Bases: object

__init__()

2.4 Module contents

CHAPTER

THREE

NACHOS PACKAGE

3.1 Subpackages

3.1.1 nachos.similarity_functions package

Submodules

nachos.similarity_functions.SimilarityFunctions module

class nachos.similarity_functions.SimilarityFunctions.**SimilarityFunctions**(*fns*, *weights*)

Bases: object

classmethod build(*conf*)

__init__(*fns*, *weights*)

__call__(*u*, *v*, *n=None*)

Summary: This function is overloaded to operate with a few different kinds of data. It can either work to compare the similarities between two data points, between a data point and a dataset, or either of the previous two functions with respect to a single factor, n.

Parameters

- **u** (Dataset) – A data point (defined by the Dataset class)
- **v** (Dataset) – A data set
- **n** (*Optional[int]*) – The index of the factor with respect to which to compute similarity. None means use the sum of all factors

Returns The similarity score

Return type float

score(*u*, *v*, *n*)

Return type float

score_set(*u*, *v*, *n=None*)

Return type float

nachos.similarity_functions.abstract_similarity module

class nachos.similarity_functions.abstract_similarity.**AbstractSimilarity**
Bases: abc.ABC

abstract classmethod build(*conf*)

__init__()

abstract __call__(*f*, *g*)
Call self as a function.

Return type float

nachos.similarity_functions.boolean module

class nachos.similarity_functions.boolean.**Boolean**
Bases: *nachos.similarity_functions.abstract_similarity.AbstractSimilarity*

Summary: This class defines the boolean similarity between points. It assumes the points are categorical. It is overloaded to allow for sets of inputs, in which case the similarity (True or False) is decided by examining whether any element in the set is equal to any element in the other set.

classmethod build(*conf*)

__call__(*f*, *g*)

Summary: Computes the similarity bewtween f and g. Similarity is binary a binary value. f, g can be any object though the intention is for them to be categorical values that can be compared for equality.

Parameters

- **f** (Any) – a value (categorical) to be compared
- **g** (Any) – a value (categorical) to be compared

Returns the boolean similarity between f and g

Return type bool

nachos.similarity_functions.cosine module

class nachos.similarity_functions.cosine.**Cosine**(*t*)
Bases: *nachos.similarity_functions.abstract_similarity.AbstractSimilarity*

Summary: Defines the (thresholded) cosine similarity between two points. Each points are expected to be ndarrays. The cosine similarity is computed using the sklearn pairwise metrics package. If all pairwise distances are desired, then the ndarray can be Nxd, where N specifies the number of data points.

Using N > 1 is useful when defining similarities on sets, which this similarity function is automatically designed to do. It returns the largest pairwise similarity between any elements of the sets being compared.

classmethod build(*conf*)

__init__(*t*)

__call__(*f*, *g*)

Summary: Computes the thresholded cosine similarity between inputs f, g. f, g are assumed to be real valued vectors, generally representing embeddings which have been whitened.

Parameters

- **f** (*set*) – an ndarray representing a set of vectors to compare
- **g** (*set*) – an ndarray representing a set of vectors to compare

Returns returns the similarity score

Return type float

nachos.similarity_functions.gaussian module

class nachos.similarity_functions.gaussian.**Gaussian**(*t*)

Bases: *nachos.similarity_functions.abstract_similarity.AbstractSimilarity*

classmethod build(*conf*)

__init__(*t*)

__call__(*f*, *g*)

Summary: Computes the thresholded similarity score between inputs f, g. f, g are assumed to be real valued scalars, and the similarity is the Gaussian similarity between the values assuming unit variance.

Parameters

- **f** (*float*) – a float representing a real value to compare
- **g** (*float*) – a float representing a real value to compare

Returns returns the similarity score

Return type float

nachos.similarity_functions.set_intersection module

class nachos.similarity_functions.set_intersection.**SetIntersection**

Bases: *nachos.similarity_functions.abstract_similarity.AbstractSimilarity*

classmethod build(*conf*)

__call__(*f*, *g*)

Summary: Computes the similarity between inputs f and g. f, g are assumed to be multi-valued objects, i.e., represent sets of values. We use the size of the intersection of the elements as the similiarity.

Parameters

- **f** (*Union[Any, Iterable] I.e., a set or something which can be converted to a set*) – a value to compare
- **g** (*Union[Any, Iterable] I.e., a set or something which can be converted to a set*) – a value to compare

Returns returns the similarity score

Return type float

Module contents

nachos.similarity_functions.**register**(*name*)

nachos.similarity_functions.**build_similarity_functions**(*conf*)

3.1.2 nachos.splitters package

Submodules

Abstract Splitter module

class nachos.splitters.abstract_splitter.**AbstractSplitter**(*sim_fn*, *constraint_fn*)
Bases: abc.ABC

abstract classmethod build(*conf*)

__init__(*sim_fn*, *constraint_fn*)

abstract __call__(*d*)
Call self as a function.

Return type List[*Dataset*]

score(*u*, *s*)

Return type float

Disconnected Components Splitter

Minimum Node Cut Splitter

class nachos.splitters.min_node_cut.**MinNodeCut**(*sim_fn*, *constraints*, *max_iter=200*, *seed=0*)
Bases: *nachos.splitters.abstract_splitter.AbstractSplitter*

classmethod build(*conf*)

__init__(*sim_fn*, *constraints*, *max_iter=200*, *seed=0*)

__call__(*d*)

Summary: Given a dataset, split according to a search over minimum-st node cuts, picking the s-source and t-target vetrices that minimize the constraint cost function of the split.

Parameters **d** (Dataset) – The dataset to split

Returns The dataset split and scores

Return type Tuple[FactoredSplit, List[float]]

Random Search Splitter

class nachos.splitters.random.**Random**(*sim_fn*, *constraints*, *max_iter=100000*, *seed=0*)
Bases: *nachos.splitters.abstract_splitter.AbstractSplitter*

classmethod build(*conf*)

__init__(*sim_fn*, *constraints*, *max_iter=100000*, *seed=0*)

__call__(*d*)

Summary: Given a dataset, split according to the Random splitter algorithm. We draw random splits (a train and heldout split) keeping track of the one with the best score and return that split. We draw a random subset of values from each factor independently.

Parameters **d** (Dataset) – The dataset to split

Returns The dataset splits

Return type FactoredSplit

Spectral Clustering Splitter

Variable Neighborhood Search (VNS) splitter

class nachos.splitters.vns.**VNS**(*sim_fn*, *constraints*, *num_shake_neighborhoods=4*, *num_search_neighborhoods=10*, *max_iter=200*, *max_neighbors=2000*, *seed=0*)
Bases: *nachos.splitters.abstract_splitter.AbstractSplitter*

classmethod build(*conf*)

__init__(*sim_fn*, *constraints*, *num_shake_neighborhoods=4*, *num_search_neighborhoods=10*, *max_iter=200*, *max_neighbors=2000*, *seed=0*)

__call__(*d*)

Summary: Given a dataset, split according using a Variable Neighborhood Search method over feasible solutions. Feasible solutions are constructed by drawing subsets by selecting values from each factor independently (and including all associated data points), and then intersecting these sets. The intersection of these sets is guaranteed to be disjoint from the intersection of the complements of these sets.

Parameters **d** (Dataset) – The dataset to split

Returns The dataset splits and scores

Return type Tuple[FactoredSplit, List[float]]

Splitters

nachos.splitters.**register**(*name*)

nachos.splitters.**build_splitter**(*conf*)

3.2 Submodules

3.3 nachos.utils module

nachos.utils.**check_iterable**(*f*)

3.4 Module contents

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

nachos, 20
nachos.constraints, 5
nachos.constraints.abstract_constraint, 2
nachos.constraints.Constraints, 1
nachos.constraints.kl, 2
nachos.constraints.mean, 3
nachos.constraints.mean_tuple, 3
nachos.constraints.sum, 4
nachos.constraints.sum_tuple, 5
nachos.data, 13
nachos.data.Data, 7
nachos.data.Input, 12
nachos.similarity_functions, 18
nachos.similarity_functions.abstract_similarity, 16
nachos.similarity_functions.boolean, 16
nachos.similarity_functions.cosine, 16
nachos.similarity_functions.gaussian, 17
nachos.similarity_functions.set_intersection, 17
nachos.similarity_functions.SimilarityFunctions, 15
nachos.splitters, 20
nachos.splitters.abstract_splitter, 18
nachos.splitters.disconnected, 18
nachos.splitters.min_node_cut, 18
nachos.splitters.random, 19
nachos.splitters.spectral_clustering, 19
nachos.splitters.vns, 19
nachos.utils, 20

INDEX

Symbols

__call__() (*nachos.constraints.Constraints.Constraints method*), 1
__call__() (*nachos.constraints.abstract_constraint.AbstractConstraint method*), 2
__call__() (*nachos.constraints.kl.KL method*), 2
__call__() (*nachos.constraints.mean.Mean method*), 3
__call__() (*nachos.constraints.mean_tuple.MeanTuple method*), 4
__call__() (*nachos.constraints.sum.Sum method*), 4
__call__() (*nachos.constraints.sum_tuple.SumTuple method*), 5
__call__() (*nachos.similarity_functions.SimilarityFunctions.SimilarityFunctions method*), 15
__call__() (*nachos.similarity_functions.abstract_similarity.AbstractSimilarity method*), 16
__call__() (*nachos.similarity_functions.boolean.Boolean method*), 16
__call__() (*nachos.similarity_functions.cosine.Cosine method*), 16
__call__() (*nachos.similarity_functions.gaussian.Gaussian method*), 17
__call__() (*nachos.similarity_functions.set_intersection.SetIntersection method*), 17
__call__() (*nachos.splitters.abstract_splitter.AbstractSplitter method*), 18
__call__() (*nachos.splitters.min_node_cut.MinNodeCut method*), 18
__call__() (*nachos.splitters.random.Random method*), 19
__call__() (*nachos.splitters.vns.VNS method*), 19
__init__() (*nachos.constraints.Constraints.Constraints method*), 1
__init__() (*nachos.constraints.abstract_constraint.AbstractConstraint method*), 2
__init__() (*nachos.constraints.kl.KL method*), 2
__init__() (*nachos.constraints.mean_tuple.MeanTuple method*), 3
__init__() (*nachos.constraints.sum_tuple.SumTuple method*), 5
__init__() (*nachos.data.Data.Data method*), 7
__init__() (*nachos.data.Data.Dataset method*), 7
__init__() (*nachos.data.Input.LhotseLoader method*), 12
__init__() (*nachos.data.Input.PandasLoader method*), 12
__init__() (*nachos.similarity_functions.SimilarityFunctions.SimilarityFu method*), 15
__init__() (*nachos.similarity_functions.abstract_similarity.AbstractSimil method*), 16
__init__() (*nachos.similarity_functions.cosine.Cosine method*), 16
__init__() (*nachos.similarity_functions.gaussian.Gaussian method*), 17
__init__() (*nachos.splitters.abstract_splitter.AbstractSplitter method*), 18
__init__() (*nachos.splitters.min_node_cut.MinNodeCut method*), 18
__init__() (*nachos.splitters.random.Random method*), 19
__init__() (*nachos.splitters.vns.VNS method*), 19

A

AbstractConstraint (*class in nachos.constraints.abstract_constraint*), 2
AbstractSimilarity (*class in nachos.similarity_functions.abstract_similarity*), 16
AbstractSplitter (*class in nachos.splitters.abstract_splitter*), 18

B

Boolean (*class in nachos.similarity_functions.boolean*), 16
build() (*nachos.constraints.abstract_constraint.AbstractConstraint class method*), 2
build() (*nachos.constraints.Constraints.Constraints class method*), 1
build() (*nachos.constraints.kl.KL class method*), 2
build() (*nachos.constraints.mean.Mean class method*), 3
build() (*nachos.constraints.mean_tuple.MeanTuple class method*), 3
build() (*nachos.constraints.sum.Sum class method*), 4

build() (*nachos.constraints.sum_tuple.SumTuple class method*), 5
build() (*nachos.similarity_functions.abstract_similarity.AbstractSimilarity class method*), 16
build() (*nachos.similarity_functions.boolean.Boolean class method*), 16
build() (*nachos.similarity_functions.cosine.Cosine class method*), 16
build() (*nachos.similarity_functions.gaussian.Gaussian class method*), 17
build() (*nachos.similarity_functions.set_intersection.SetIntersection class method*), 17
build() (*nachos.similarity_functions.SimilarityFunctions.SimilarityFunctions class method*), 15
build() (*nachos.splitters.abstract_splitter.AbstractSplitter class method*), 18
build() (*nachos.splitters.min_node_cut.MinNodeCut class method*), 18
build() (*nachos.splitters.random.Random class method*), 19
build() (*nachos.splitters.vns.VNS class method*), 19
build_constraints() (*in module nachos.constraints*), 5
build_similarity_functions() (*in module nachos.similarity_functions*), 18
build_splitter() (*in module nachos.splitters*), 20

C

check_complete() (*nachos.data.Data.Dataset method*), 8
check_disconnected() (*nachos.data.Data.Dataset method*), 8
check_iterable() (*in module nachos.utils*), 20
collapse_factored_split() (*in module nachos.data.Data*), 12
Constraints (*class in nachos.constraints.Constraints*), 1
copy() (*nachos.data.Data.Data method*), 7
Cosine (*class in nachos.similarity_functions.cosine*), 16

D

Data (*class in nachos.data.Data*), 7
Dataset (*class in nachos.data.Data*), 7
draw_random_node_cut() (*nachos.data.Data.Dataset method*), 11
draw_random_split() (*nachos.data.Data.Dataset method*), 10
draw_random_split_from_factor() (*nachos.data.Data.Dataset method*), 9
draw_split_from_factor() (*nachos.data.Data.Dataset method*), 10

E

export_graph() (*nachos.data.Data.Dataset method*), 8

G

Gaussian (*class in nachos.similarity_functions.gaussian*), 17
get_constraints() (*nachos.data.Data.Dataset method*), 8
get_factors() (*nachos.data.Data.Dataset method*), 9
get_neighborhood() (*nachos.data.Data.Dataset method*), 11
get_record() (*nachos.data.Data.Dataset method*), 8

K

KL (*class in nachos.constraints.kl*), 2

L

LhotseLoader (*class in nachos.data.Input*), 12
load() (*nachos.data.Input.TSVLoader static method*), 12

M

make_constraint_inverted_index() (*nachos.data.Data.Dataset method*), 9
make_factor_inverted_index() (*nachos.data.Data.Dataset method*), 9
make_graph() (*nachos.data.Data.Dataset method*), 8
make_overlapping_test_sets() (*nachos.data.Data.Dataset method*), 11
Mean (*class in nachos.constraints.mean*), 3
MeanTuple (*class in nachos.constraints.mean_tuple*), 3
MinNodeCut (*class in nachos.splitters.min_node_cut*), 18
module
    nachos, 20
    nachos.constraints, 5
    nachos.constraints.abstract_constraint, 2
    nachos.constraints.Constraints, 1
    nachos.constraints.kl, 2
    nachos.constraints.mean, 3
    nachos.constraints.mean_tuple, 3
    nachos.constraints.sum, 4
    nachos.constraints.sum_tuple, 5
    nachos.data, 13
    nachos.data.Data, 7
    nachos.data.Input, 12
    nachos.similarity_functions, 18
    nachos.similarity_functions.abstract_similarity, 16
    nachos.similarity_functions.boolean, 16
    nachos.similarity_functions.cosine, 16
    nachos.similarity_functions.gaussian, 17
    nachos.similarity_functions.set_intersection, 17
    nachos.similarity_functions.SimilarityFunctions, 15
    nachos.splitters, 20
    nachos.splitters.abstract_splitter, 18

nachos.splitters.disconnected, 18
nachos.splitters.min_node_cut, 18
nachos.splitters.random, 19
nachos.splitters.spectral_clustering, 19
nachos.splitters.vns, 19
nachos.utils, 20

N

nachos
module, 20
nachos.constraints
module, 5
nachos.constraints.abstract_constraint
module, 2
nachos.constraints.Constraints
module, 1
nachos.constraints.kl
module, 2
nachos.constraints.mean
module, 3
nachos.constraints.mean_tuple
module, 3
nachos.constraints.sum
module, 4
nachos.constraints.sum_tuple
module, 5
nachos.data
module, 13
nachos.data.Data
module, 7
nachos.data.Input
module, 12
nachos.similarity_functions
module, 18
nachos.similarity_functions.abstract_similarity
module, 16
nachos.similarity_functions.boolean
module, 16
nachos.similarity_functions.cosine
module, 16
nachos.similarity_functions.gaussian
module, 17
nachos.similarity_functions.set_intersection
module, 17
nachos.similarity_functions.SimilarityFunctions
module, 15
nachos.splitters
module, 20
nachos.splitters.abstract_splitter
module, 18
nachos.splitters.disconnected
module, 18
nachos.splitters.min_node_cut
module, 18
nachos.splitters.random
module, 19
nachos.splitters.spectral_clustering
module, 19
nachos.splitters.vns
module, 19
nachos.utils
module, 20
nearby_splits() (*nachos.data.Data.Dataset method*), 10

O

overlap_stats() (*nachos.data.Data.Dataset method*), 12

P

PandasLoader (*class in nachos.data.Input*), 12

R

Random (*class in nachos.splitters.random*), 19
register() (*in module nachos.constraints*), 5
register() (*in module nachos.similarity_functions*), 18
register() (*in module nachos.splitters*), 20

S

score() (*nachos.similarity_functions.SimilarityFunctions.SimilarityFuncti method*), 15
score() (*nachos.splitters.abstract_splitter.AbstractSplitter method*), 18
score_set() (*nachos.similarity_functions.SimilarityFunctions.SimilarityF method*), 15
set_random_seed() (*nachos.data.Data.Dataset method*), 10
SetIntersection (*class in nachos.similarity_functions.set_intersection*), 17
shake() (*nachos.data.Data.Dataset method*), 11
SimilarityFunctions (*class in nachos.similarity_functions.SimilarityFunctions*), 15
stat() (*nachos.constraints.mean.Mean method*), 3
stat() (*nachos.constraints.sum.Sum method*), 4
stats() (*nachos.constraints.Constraints.Constraints method*), 1
subset_from_data() (*nachos.data.Data.Dataset method*), 7
subset_from_records() (*nachos.data.Data.Dataset method*), 7
Sum (*class in nachos.constraints.sum*), 4
SumTuple (*class in nachos.constraints.sum_tuple*), 5

T

TSVLoader (*class in nachos.data.Input*), 12

V

VNS (*class in nachos.splitters.vns*), 19