`idpp.probability.trees`

Module with data structures that facilitate identification probability analysis.

```
# type aliases
type PropertyTree = Union[MzTree, RtTree, CcsTree, Ms2Tree]
# map cmpd_id (int) to matching cmpd_ids (set(int))
type QueryResult = Dict[int, Set[int]]
# map adduct_id (int) to similarity contributions from other adducts
# similarity contributions: data arrays for COO matrix
type Similarities = Tuple[
    npt.NDArray[np.int32],
    npt.NDArray[np.int32],
    npt.NDArray[np.float32]
]
```

`MzTree`

class **`idpp.probability.trees.MzTree`**(*mzs: ndarray[Any, dtype[float64]], cmpd_ids: ndarray[Any, dtype[int32]], leaf_size: int = 128*)

a KDTree subclass for querying m/z values

Attributes:

mzs : `numpy.ndarray(float)`

input array of m/zs

cmpd_ids : `numpy.ndarray(int)`

input array of cmpd_ids

Methods

`get_arrays` ()	Get data and node arrays.
`get_n_calls` ()	Get number of calls.
`get_tree_stats` ()	Get tree status.
`kernel_density` (X, h[, kernel, atol, rtol, ...])	Compute the kernel density estimate at points X with the
`load_attrs` (*attrs)	load the extra attributes that didnt get pickled automatic
`query` (X[, k, return_distance, dualtree, ...])	query the tree for the k nearest neighbors
`query_all` (ppm)	Search all of self.mzs using tolerance computed from spe search tolerance ppm
`query_all_gen` (ppm)	just like query_all() method but yields one search result a
`query_radius` (X, r[, return_distance, ...])	query the tree for neighbors within a radius r
`reset_n_calls` ()	Reset number of calls to 0.
`save` (dir, dataset_id)	save this *MzTree* instance to file, load again using the *load*
`two_point_correlation` (X, r[, dualtree])	Compute the two-point correlation function

Methods

`idpp.probability.trees.MzTree.__init__`(*self, mzs: ndarray[Any, dtype[float64]], cmpd_ids: ndarray[Any, dtype[int32]], leaf_size: int = 128*)

create a new MzTree instance from an array of m/zs

Parameters:

mzs : `numpy.ndarray(float)`

input array of m/zs

cmpd_ids : `numpy.ndarray(int)`

input array of cmpd_ids

leaf_size : `int`

TODO

`idpp.probability.trees.MzTree.query_all`(*self, ppm: float*)→ QueryResult

Search all of self.mzs using tolerance computed from specified ppm, returns query result Parameters ——–- ppm : `float`

search tolerance ppm

Returns:

result : `QueryResult`

dict mapping cmpd_ids to sets of matching cmpd_ids

`idpp.probability.trees.MzTree.query_all_gen`(*self, ppm: float*)

just like query_all() method but yields one search result at a time

`idpp.probability.trees.MzTree.save`(*self, dir: str, dataset_id: int*)→ None

save this *MzTree* instance to file, load again using the *load_tree* function

Parameters:

dir : `str`

directory to save the tree instance into

dataset_id : `int`

dataset identifier, used to generate file name

`idpp.probability.trees.MzTree.load_attrs`(*self, *attrs*)→ None

load the extra attributes that didnt get pickled automatically

`CcsTree`

class **`idpp.probability.trees.CcsTree`**(*ccss: ndarray[Any, dtype[float64]], cmpd_ids: ndarray[Any, dtype[int32]], leaf_size: int = 256*)

a KDTree subclass for querying CCS values

Attributes:

ccss : `numpy.ndarray(float)`

input array of CCS values

cmpd_ids : `numpy.ndarray(int)`

input array of cmpd_ids

Methods

`get_arrays` ()	Get data and node arrays.
`get_n_calls` ()	Get number of calls.
`get_tree_stats` ()	Get tree status.
`kernel_density` (X, h[, kernel, atol, rtol, ...])	Compute the kernel density estimate at points X with the
`load_attrs` (*attrs)	load the extra attributes that didnt get pickled automatic
`query` (X[, k, return_distance, dualtree, ...])	query the tree for the k nearest neighbors
`query_all` (percent)	Search all of self.ccs_qry using tolerance computed from
`query_radius` (percent)	Search all of self.ccs_qry against the KDTree using a radiu
`query_radius_single` (ccs, percent)	Query a single CCS value using a specified radius and ret
`reset_n_calls` ()	Reset number of calls to 0.
`save` (dir, dataset_id)	save this *CcsTree* instance to file, load again using the *loac*
`two_point_correlation` (X, r[, dualtree])	Compute the two-point correlation function

Methods

`idpp.probability.trees.CcsTree.__init__`(*self, ccss: ndarray[Any, dtype[float64]], cmpd_ids: ndarray[Any, dtype[int32]], leaf_size: int = 256*)

create a new MzTree instance from an array of m/zs

Parameters:

ccss : `numpy.ndarray(float)`

input array of CCS values

cmpd_ids : `numpy.ndarray(int)`

input array with corresponding commpound IDs

leaf_size : `int` , default=64

TODO

`idpp.probability.trees.CcsTree.query_radius_single`(*self, ccs: float, percent: float*)→ Set[int]

Query a single CCS value using a specified radius and return the set of adduct IDs within that radius

Parameters:

ccs : `float`

query CCS value

percent : `float`

query tolerance (as a percent)

Returns:

adduct_ids : `set(int)`

matching adduct IDs

idpp.probability.trees.CcsTree.query_radius(*self, percent: float*)→ ndarray[Any, dtype[Any]]

Search all of self.ccs_qry against the KDTree using a radius (tolerance computed from specified percent),returns an array of all matching indices for each element in self.ccs_qry

> Note
>
> This method is a thin wrapper around the `KDTree.query_radius(...)` method, and it returns the same array of arrays where each index in the first array contains an array of matching indices from the query. This is not so useful for my ultimate goal of coordinating the query results across multiple trees. So instead of using this method directly, the `query_all(...)` method should be used instead which will take the output from this method and convert that into a dictionary that will incorporate cmpd_id info as well.

Parameters: percent : `float`

search tolerance percent

Returns: TODO

idpp.probability.trees.CcsTree.query_all(*self, percent: float*)→ QueryResult

Search all of self.ccs_qry using tolerance computed from specified percent returns query result

Parameters: percent : `float`

search tolerance percent

Returns: result : `QueryResult`

dict mapping cmpd_ids to sets of matching cmpd_ids

idpp.probability.trees.CcsTree.save(*self, dir: str, dataset_id: int*)→ None

save this *CcsTree* instance to file, load again using the *load_ccs_tree* function

Parameters: dir : `str`

directory to save the tree instance into

dataset_id : `int`

dataset identifier, used to generate file name

idpp.probability.trees.CcsTree.load_attrs(*self, *attrs*)→ None

load the extra attributes that didnt get pickled automatically

`RtTree`

class **idpp.probability.trees.RtTree**(*rts: ndarray[Any, dtype[float64]], cmpd_ids: ndarray[Any, dtype[int32]], leaf_size: int = 256*)

a KDTree subclass for querying RT values

Attributes: rts : `numpy.ndarray(float)`

input array of RTs

cmpd_ids : `numpy.ndarray(int)`

input array of cmpd_ids

Methods

`get_arrays` ()	Get data and node arrays.
`get_n_calls` ()	Get number of calls.
`get_tree_stats` ()	Get tree status.
`kernel_density` (X, h[, kernel, atol, rtol, ...])	Compute the kernel density estimate at points X with the
`load_attrs` (*attrs)	load the extra attributes that didnt get pickled automatic
`query` (X[, k, return_distance, dualtree, ...])	query the tree for the k nearest neighbors
`query_all` (tol)	Search all of self.rts using tolerance in min.
`query_radius` (tol)	Search all of self.rts against the KDTree using a radius (to
`query_radius_single` (rt, tol)	Query a single RT value using a specified radius and retur
`reset_n_calls` ()	Reset number of calls to 0.
`save` (dir, dataset_id)	save this *RtTree* instance to file, load again using the *load_*
`two_point_correlation` (X, r[, dualtree])	Compute the two-point correlation function

Methods

`idpp.probability.trees.RtTree.__init__`*(self, rts: ndarray[Any, dtype[float64]], cmpd_ids: ndarray[Any, dtype[int32]], leaf_size: int = 256)*

create a new RtTree instance from an array of m/zs

Parameters:

rts : `numpy.ndarray(float)`

input array of RTs

cmpd_ids : `numpy.ndarray(int)`

input array of cmpd_ids

leaf_size : `int`

TODO

`idpp.probability.trees.RtTree.query_radius_single`*(self, rt: float, tol: float)*→ Set[int]

Query a single RT value using a specified radius and return the set of adduct IDs within that radius

Parameters:

rt : `float`

query RT value

tol : `float`

query tolerance

Returns:

adduct_ids : `set(int)`

matching adduct IDs

`idpp.probability.trees.RtTree.query_radius`(*self*, *tol: float*)→ ndarray[Any, dtype[Any]]

Search all of self.rts against the KDTree using a radius (tolerance in min.), returns an array of all matching indices for each element in self.rts

> ❗ Note
>
> This method is a thin wrapper around the `KDTree.query_radius(...)` method, and it returns the same array of arrays where each index in the first array contains an array of matching indices from the query. This is not so useful for my ultimate goal of coordinating the query results across multiple trees. So instead of using this method directly, the `query_all(...)` method should be used instead which will take the output from this method and convert that into a dictionary that will incorporate cmpd_id info as well.

Parameters: tol : `float`

search tolerance (in min.)

Returns: TODO

`idpp.probability.trees.RtTree.query_all`(*self*, *tol: float*)→ QueryResult

Search all of self.rts using tolerance in min. returns search result

Parameters: tol : `float`

search tolerance (in min.)

Returns: result : `QueryResult`

dict mapping cmpd_ids to sets of matching cmpd_ids

`idpp.probability.trees.RtTree.save`(*self*, *dir: str*, *dataset_id: int*)→ None

save this *RtTree* instance to file, load again using the *load_tree* function

Parameters: dir : `str`

directory to save the tree instance into

dataset_id : `int`

dataset identifier, used to generate file name

`idpp.probability.trees.RtTree.load_attrs`(*self*, **attrs*)→ None

load the extra attributes that didnt get pickled automatically

`Ms2Tree`

class **`idpp.probability.trees.Ms2Tree`**(*frag_imzs: ndarray[Any, dtype[int32]]*, *frag_iis: ndarray[Any, dtype[int32]]*, *adduct_ids: ndarray[Any, dtype[int32]]*, *cmpd_ids: ndarray[Any, dtype[int32]]*)

a class for querying MS2 spectra with similar interface to KDTree

Attributes: TODO

cmpd_ids : `numpy.ndarray(int)`

input array of cmpd_ids

Methods

`load_attrs`(*attrs)	load the extra attributes that didnt get pickled automatic
`precompute_similarities`([imz_tol, debug])	precompute a matrix of similarities between all spectra s
`query_all`(similarity_threshold)	Return query results for all spectra stored in this object u
`save`(dir, dataset_id)	save this *Ms2Tree* instance to file, load again using the *lo*

Methods

`idpp.probability.trees.Ms2Tree.__init__`*(self, frag_imzs: ndarray[Any, dtype[int32]], frag_iis: ndarray[Any, dtype[int32]], adduct_ids: ndarray[Any, dtype[int32]], cmpd_ids: ndarray[Any, dtype[int32]])*

create a new instance of Ms2Tree from array of ms2 fragments and associated cmpd_ids

Parameters:

frag_imzs : `numpy.ndarray(int)`

input array of framemt m/zs (in integer representation)

frag_iis : `numpy.ndarray(int)`

input array of fragment abundances (in integer representation)

cmpd_ids : `numpy.ndarray(int)`

input array of cmpd_ids

`idpp.probability.trees.Ms2Tree.precompute_similarities`*(self, imz_tol: int = 2000, debug: bool = False)*→ None

precompute a matrix of similarities between all spectra stored in this object

Parameters:

imz_tol : `int`, default=2000

specify the tolerance (in Da, integer representation) for combining fragment mzs, the default is 2000 which corresponds to 20 mDa, the same that was used in the publication for flash entropy searchs

`idpp.probability.trees.Ms2Tree.query_all`*(self, similarity_threshold: float)*→ QueryResult

Return query results for all spectra stored in this object using a specified spectral entropy similarity threshold.

❗ Note

Must use `precompute_similarities(...)` method before performing queries

Parameters:

similarity_threshold : `float`

spectral entropy similarity threshold

Returns:

result : `QueryResult`

dict mapping cmpd_ids to sets of matching cmpd_ids `idpp.probability.trees.Ms2Tree.save`*(self, dir: str, dataset_id: int)*→ None

save this *Ms2Tree* instance to file, load again using the *load_tree* function

Parameters: dir : `str`

directory to save the tree instance into

dataset_id : `int`

dataset identifier, used to generate file name

`idpp.probability.trees.Ms2Tree.load_attrs`*(self, *attrs)*→ None

load the extra attributes that didnt get pickled automatically

Utility

class `idpp.probability.trees.DatasetQueries`*(mz_qry: Tuple[str, str], rt_qry: str, ccs_qry: str, ms2_qry: Tuple[str, str])*

Store a set of queries needed for selecting a complete dataset for identification probability analysis

Methods

`to_json` ()	return a string with JSON representation of this object

`idpp.probability.trees.load_tree`*(tree_file: str)*→ PropertyTree

load an already constructed *MzTree*, *CcsTree*, *Ms2Tree* or *RtTree* instance from file

Parameters: file : `str`

file name for saved instance

Returns: tree : `PropertyTree`

loaded tree instance

`idpp.probability.trees.construct_ccs_tree`*(db: IdPPdb, queries: DatasetQueries)*→ CcsTree

Select a dataset for identification probability analysis using a set of input queries, then construct and return corresponding instance of *CcsTree* for performing the analysis.

Parameters: db : `IdPPdb`

interface for IdPP database

queries : `idpp.probability.analysis.DatasetQueries`

Instance of DatasetQueries dataclass containing dataset selection querys

Returns: tree : `CcsTree`

instance of`CcsTree` constructed using the input query

`idpp.probability.trees.construct_rt_tree`*(db: IdPPdb, queries: DatasetQueries)*→ RtTree

Select a dataset for identification probability analysis using a set of input queries, then construct and return corresponding instance of *RtTree* for performing the analysis.

Parameters:

db : `IdPPdb`

interface for IdPP database

queries : `idpp.probability.analysis.DatasetQueries`

Instance of DatasetQueries dataclass containing dataset selection querys

Returns:

tree : `RtTree`

instance of`RtTree` constructed using the input query

`idpp.probability.trees.construct_ms2_tree`(*db: IdPPdb, queries: DatasetQueries*)→ Ms2Tree | None

Select a dataset for identification probability analysis using a set of input queries, then construct and return corresponding instance of *Ms2Tree* for performing the analysis.

Parameters:

db : `IdPPdb`

interface for IdPP database

queries : `idpp.probability.analysis.DatasetQueries`

Instance of DatasetQueries dataclass containing dataset selection querys

Returns:

tree : `Ms2Tree` or None

instance of`Ms2Tree` constructed using the input query, or None if no spectra

`idpp.probability.trees.construct_ms2_tree_for_adduct_ids`(*db: IdPPdb, adduct_ids: int, precompute_similarities: bool = True*)→ Ms2Tree | None

Alternate function for constructing an Ms2Tree from spectra based on a specified list of adduct IDs If there are no MS/MS spectra associated with the adduct_ids, returns None

Parameters:

db : `IdPPdb`

interface for IdPP database

adduct_ids : `list(int)`

adduct identifiers to attempt to gather spectra from

precompute_similarities : `bool` , default=True

flag specifying whether to precompute spectra similarity matrix

Returns:

ms2t : `Ms2Tree` or `None`

instance of Ms2Tree if there were MS/MS spectra associated with the input adduct_id, or else None

`idpp.probability.trees.construct_property_trees`(*db: IdPPdb, queries: DatasetQueries*)→ Tuple[int, Tuple[MzTree, RtTree, CcsTree, Ms2Tree]]

Select a dataset for identification probability analysis using a set of input queries, then construct and return corresponding instances of *MzTree*, *RtTree*, *CcsTree* and *Ms2Tree* for performing the analysis.

Parameters:

db : `IdPPdb`

interface for IdPP database

queries : `idpp.probability.analysis.DatasetQueries`

Instance of DatasetQueries dataclass containing dataset selection querys

Returns:

trees : `tuple(MzTree, RtTree, CcsTree, Ms2Tree)`

instances of *MzTree*, *RtTree*, *CcsTree* and *Ms2Tree* constructed from data fetched using the input query