
ODDT Documentation

Release 0.1.1

Maciej Wojcikowski

May 07, 2015

1	oddt package	3
1.1	Subpackages	3
1.2	Submodules	15
1.3	oddt.interactions module	15
1.4	oddt.metrics module	20
1.5	oddt.spatial module	24
1.6	oddt.virtualscreening module	27
1.7	Module contents	29
2	Open Drug Discovery Toolkit	31
3	Indices and tables	33
	Bibliography	35
	Python Module Index	37

Contents:

oddt package

1.1 Subpackages

1.1.1 oddt.docking package

Submodules

oddt.docking.autodock_vina module

class oddt.docking.autodock_vina.**autodock_vina** (*protein=None, size=(10, 10, 10), center=(0, 0, 0), auto_ligand=None, exhaustiveness=8, num_modes=9, energy_range=3, seed=None, prefix_dir='/tmp', n_cpu=1, executable=None, autocleanup=True*)

Bases: object

Attributes

tmp_dir

Methods

<i>clean()</i>	
<i>dock</i> (ligands[, protein, single])	Automated docking procedure.
<i>score</i> (ligands[, protein, single])	Automated scoring procedure.
<i>set_protein</i> (protein)	Change protein to dock to.

clean()

dock (*ligands, protein=None, single=False*)
Automated docking procedure.

Parameters **ligands:** iterable of oddt.toolkit.Molecule objects

Ligands to dock

protein: oddt.toolkit.Molecule object or None Protein object to be used. If None,

then the default one is used, else the protein is new default.

single: bool (default=False) A flag to indicate single ligand docking (performance reasons (eg. there is no need for subdirectory for one ligand))

Returns ligands : array of oddt.toolkit.Molecule objects

Array of ligands (scores are stored in mol.data method)

score (*ligands*, *protein=None*, *single=False*)

Automated scoring procedure.

Parameters ligands: iterable of oddt.toolkit.Molecule objects

Ligands to score

protein: oddt.toolkit.Molecule object or None Protein object to be used. If None, then the default one is used, else the protein is new default.

single: bool (default=False) A flag to indicate single ligand scoring (performance reasons (eg. there is no need for subdirectory for one ligand))

Returns ligands : array of oddt.toolkit.Molecule objects

Array of ligands (scores are stored in mol.data method)

set_protein (*protein*)

Change protein to dock to.

Parameters protein: oddt.toolkit.Molecule object

Protein object to be used.

tmp_dir

`oddt.docking.autodock_vina.parse_vina_docking_output` (*output*)

Function parsing Autodock Vina docking output to a dictionary

Parameters output : string

Autodock Vina standard output (STDOUT).

Returns out : dict

dictionary containing scores computed by Autodock Vina

`oddt.docking.autodock_vina.parse_vina_scoring_output` (*output*)

Function parsing Autodock Vina scoring output to a dictionary

Parameters output : string

Autodock Vina standard output (STDOUT).

Returns out : dict

dictionary containing scores computed by Autodock Vina

`oddt.docking.autodock_vina.random` () → x in the interval [0, 1).

Module contents

1.1.2 oddt.scoring package

Subpackages

oddt.scoring.descriptors package

Submodules

oddt.scoring.descriptors.binana module Internal implementation of binana software (<http://nbc.ucs.d.edu/data/sw/hosted/binana/>)

class `oddt.scoring.descriptors.binana.binana_descriptor` (*protein=None*)
Bases: `object`

Methods

<code>build</code> (ligands[, protein])	Descriptor building method
<code>set_protein</code> (protein)	One function to change all relevant proteins

build (*ligands*, *protein=None*)
Descriptor building method

Parameters **ligands:** array-like

An array of generator of `oddt.toolkit.Molecule` objects for which the descriptor is computed

protein: `oddt.toolkit.Molecule` object (default=None) Protein object to be used while generating descriptors. If none, then the default protein (from constructor) is used. Otherwise, protein becomes new global and default protein.

Returns `descs`: numpy array, shape=[*n_samples*, 351]

An array of binana descriptors, aligned with input ligands

set_protein (*protein*)
One function to change all relevant proteins

Parameters **protein:** `oddt.toolkit.Molecule` object

Protein object to be used while generating descriptors. Protein becomes new global and default protein.

Module contents

`oddt.scoring.descriptors.atoms_by_type` (*atom_dict*, *types*, *mode='atomic_nums'*)

Returns atom dictionaries based on given criteria. Currently we have 3 types of atom selection criteria:

- atomic numbers [`'atomic_nums'`]
- Sybyl Atom Types [`'atom_types_sybyl'`]

- AutoDock4 atom types ['atom_types_ad4'] (<http://autodock.scripps.edu/faqs-help/faq/where-do-i-set-the-autodock-4-force-field-parameters>)

Parameters **atom_dict**: `oddt.toolkit.Molecule.atom_dict`

Atom dictionary as implemented in `oddt.toolkit.Molecule` class

types: array-like List of atom types/numbers wanted.

Returns out: dictionary of shape=[len(types)]

A dictionary of queried atom types (types are keys of the dictionary). Values are of `oddt.toolkit.Molecule.atom_dict` type.

```
class oddt.scoring.descriptors.close_contacts (protein=None, cutoff=4,
                                              mode='atomic_nums', ligand_types=None,
                                              protein_types=None, aligned_pairs=False)
```

Bases: object

Methods

build(ligands[, protein, single]) Builds descriptors for series of ligands

build (ligands, protein=None, single=False)

Builds descriptors for series of ligands

Parameters **ligands**: iterable of `oddt.toolkit.Molecules` or `oddt.toolkit.Molecule`

A list or iterable of ligands to build the descriptor or a single molecule.

protein: `oddt.toolkit.Molecule` or `None` (default=None) Default protein to use as reference

single: bool (default=False) Flag indicating if the ligand is single.

```
class oddt.scoring.descriptors.fingerprints (fp='fp2', toolkit='ob')
```

Bases: object

Methods

build(mols[, single])

build (mols, single=False)

oddt.scoring.functions package

Submodules

`oddt.scoring.functions.NNScore` module

`oddt.scoring.functions.RFScore` module

Module contents

oddt.scoring.models package

Submodules

oddt.scoring.models.classifiers module

`oddt.scoring.models.classifiers.randomforest`
 alias of `RandomForestClassifier`
`oddt.scoring.models.classifiers.svm`
 alias of `SVC`

oddt.scoring.models.neuralnetwork module

oddt.scoring.models.regressors module

Module contents

Module contents

`oddt.scoring.cross_validate(model, cv_set, cv_target, n=10, shuffle=True, n_jobs=1)`

class `oddt.scoring.ensemble_model(models)`
 Bases: `object`

Methods

*`fit(X, y, *args, **kwargs)`*

*`predict(X, *args, **kwargs)`*

*`score(X, y, *args, **kwargs)`*

fit (*X*, *y*, *args, **kwargs)

predict (*X*, *args, **kwargs)

score (*X*, *y*, *args, **kwargs)

class `oddt.scoring.scorer(model_instances, descriptor_generator_instances, score_title='score')`
 Bases: `object`

Methods

<i><code>fit</code></i> (ligands, target, *args, **kwargs)	Trains model on supplied ligands and target values
<i><code>load</code></i> (filename)	Loads scoring function from a pickle file.
<i><code>predict</code></i> (ligands, *args, **kwargs)	Predicts values (eg.
<i><code>predict_ligand</code></i> (ligand)	Local method to score one ligand and update it's scores.
<i><code>predict_ligands</code></i> (ligands)	Method to score ligands lazily

Continued on next page

Table 1.7 – continued from previous page

<code>save(filename)</code>	Saves scoring function to a pickle file.
<code>score(ligands, target, *args, **kwargs)</code>	Methods estimates the quality of prediction as squared correlation coefficient (R^2)
<code>set_protein(protein)</code>	Proxy method to update protein in all relevant places.

fit (*ligands, target, *args, **kwargs*)

Trains model on supplied ligands and target values

Parameters **ligands:** array-like of ligands

Ground truth (correct) target values.

target: array-like of shape = [n_samples] or [n_samples, n_outputs] Estimated target values.

classmethod load (*filename*)

Loads scoring function from a pickle file.

Parameters **filename:** string

Pickle filename

Returns sf: scorer-like object

Scoring function object loaded from a pickle

predict (*ligands, *args, **kwargs*)

Predicts values (eg. affinity) for supplied ligands

Parameters **ligands:** array-like of ligands

Ground truth (correct) target values.

target: array-like of shape = [n_samples] or [n_samples, n_outputs] Estimated target values.

Returns predicted: np.array or array of np.arrays of shape = [n_ligands]

Predicted scores for ligands

predict_ligand (*ligand*)

Local method to score one ligand and update it's scores.

Parameters **ligand:** oddt.toolkit.Molecule object

Ligand to be scored

Returns ligand: oddt.toolkit.Molecule object

Scored ligand with updated scores

predict_ligands (*ligands*)

Method to score ligands lazily

Parameters **ligands:** iterable of oddt.toolkit.Molecule objects

Ligands to be scored

Returns ligand: iterator of oddt.toolkit.Molecule objects

Scored ligands with updated scores

save (*filename*)

Saves scoring function to a pickle file.

Parameters filename: string

Pickle filename

score (*ligands, target, *args, **kwargs*)Methods estimates the quality of prediction as squared correlation coefficient (R^2)**Parameters ligands: array-like of ligands**

Ground truth (correct) target values.

target: array-like of shape = [n_samples] or [n_samples, n_outputs] Estimated target values.**Returns r2: float**Squared correlation coefficient (R^2) for prediction**set_protein** (*protein*)

Proxy method to update protein in all relevant places.

Parameters protein: oddt.toolkit.Molecule object

New default protein

1.1.3 oddt.toolkits package

Submodules

oddt.toolkits.ob module

class oddt.toolkits.ob.**Residue** (*OBResidue*)

Bases: object

Represent a Pybel residue.

Required parameter: OBResidue – an Open Babel OBResidue**Attributes:** atoms, idx, name.

(refer to the Open Babel library documentation for more info).

The original Open Babel atom can be accessed using the attribute: OBResidue

Attributes

atomsidxname**atoms****idx****name**oddt.toolkits.ob.**pickle_mol** (*self*)oddt.toolkits.ob.**readfile** (*format, filename, opt=None, lazy=False*)

```
oddt.toolkits.ob.unpickle_mol(source)
```

oddt.toolkits.rdk module

rdkit - A Cinfony module for accessing the RDKit from CPython

Global variables: Chem and AllChem - the underlying RDKit Python bindings
informats - a dictionary of supported input formats
outformats - a dictionary of supported output formats
descs - a list of supported descriptors
fps - a list of supported fingerprint types
forcefields - a list of supported forcefields

class `oddt.toolkits.rdk.Atom` (*Atom*)

Bases: object

Represent an rdkit Atom.

Required parameters: Atom – an RDKit Atom

Attributes: atomicnum, coords, formalcharge

The original RDKit Atom can be accessed using the attribute: Atom

Attributes

<i>atomicnum</i>
<i>coords</i>
<i>formalcharge</i>
<i>idx</i>
<i>neighbors</i>
<i>partialcharge</i>

atomicnum

coords

formalcharge

idx

neighbors

partialcharge

class `oddt.toolkits.rdk.Fingerprint` (*fingerprint*)

Bases: object

A Molecular Fingerprint.

Required parameters: fingerprint – a vector calculated by one of the fingerprint methods

Attributes: fp – the underlying fingerprint object
bits – a list of bits set in the Fingerprint

Methods: The “|” operator can be used to calculate the Tanimoto coeff. For example, given two Fingerprints ‘a’, and ‘b’, the Tanimoto coefficient is given by:

$$\text{tanimoto} = a \mid b$$

Attributes

raw

raw

class `oddt.toolkits.rdk.Molecule` (*Mol=None, source=None, protein=False*)

Bases: object

Represent an rdkit Molecule.

Required parameter: Mol – an RDKit Mol or any type of cinfony Molecule

Attributes: atoms, data, formula, molwt, title

Methods: `addh()`, `calcfp()`, `calcdesc()`, `draw()`, `localopt()`, `make3D()`, `removeh()`, `write()`

The underlying RDKit Mol can be accessed using the attribute: Mol

Attributes

<i>Mol</i>	
<i>atom_dict</i>	
<i>atoms</i>	
<i>canonic_order</i>	Returns np.array with canonic order of heavy atoms in the molecule
<i>charges</i>	
<i>clone</i>	
<i>coords</i>	
<i>data</i>	
<i>formula</i>	
<i>molwt</i>	
<i>num_rotors</i>	
<i>res_dict</i>	
<i>ring_dict</i>	
<i>sssr</i>	
<i>title</i>	

Methods

<i>addh()</i>	Add hydrogens.
<i>calcdesc</i> ([descnames])	Calculate descriptor values.
<i>calcfp</i> ([fptype, opt])	Calculate a molecular fingerprint.
<i>clone_coords</i> (source)	
<i>draw</i> ([show, filename, update, usecoords])	Create a 2D depiction of the molecule.
<i>localopt</i> ([forcefield, steps])	Locally optimize the coordinates.
<i>make3D</i> ([forcefield, steps])	Generate 3D coordinates.
<i>removeh()</i>	Remove hydrogens.
<i>write</i> ([format, filename, overwrite])	Write the molecule to a file or return a string.

Mol

addh ()

Add hydrogens.

atom_dict

atoms

calcdesc (*descnames=[]*)

Calculate descriptor values.

Optional parameter: descnames – a list of names of descriptors

If descnames is not specified, all available descriptors are calculated. See the descs variable for a list of available descriptors.

calcfp (*fptype='rdkit', opt=None*)

Calculate a molecular fingerprint.

Optional parameters:

fptype – the fingerprint type (default is “rdkit”). See the fps variable for a list of available fingerprint types.

opt – a dictionary of options for fingerprints. Currently only used for radius and bitInfo in Morgan fingerprints.

canonic_order

Returns np.array with canonic order of heavy atoms in the molecule

charges

clone

clone_coords (*source*)

coords

data

draw (*show=True, filename=None, update=False, usecoords=False*)

Create a 2D depiction of the molecule.

Optional parameters: show – display on screen (default is True) filename – write to file (default is None)

update – update the coordinates of the atoms to those

determined by the structure diagram generator (default is False)

usecoords – don’t calculate 2D coordinates, just use the current coordinates (default is False)

Aggdraw or Cairo is used for 2D depiction. Tkinter and Python Imaging Library are required for image display.

formula

localopt (*forcefield='uff', steps=500*)

Locally optimize the coordinates.

Optional parameters:

forcefield – default is “uff”. See the forcefields variable for a list of available forcefields.

steps – default is 500

If the molecule does not have any coordinates, make3D() is called before the optimization.

make3D (*forcefield='uff', steps=50*)

Generate 3D coordinates.

Optional parameters:

forcefield – default is “uff”. See the **forcefields variable** for a list of available forcefields.

steps – default is 50

Once coordinates are generated, a quick local optimization is carried out with 50 steps and the UFF force-field. Call `localopt()` if you want to improve the coordinates further.

molwt

num_rotors

removeh()

Remove hydrogens.

res_dict

ring_dict

sssr

title

write (*format='smi', filename=None, overwrite=False, **kwargs*)

Write the molecule to a file or return a string.

Optional parameters:

format – see the **informats variable for a list of available** output formats (default is “smi”)

filename – default is None **overwrite** – if the output file already exists, should it

be overwritten? (default is False)

If a filename is specified, the result is written to a file. Otherwise, a string is returned containing the result.

To write multiple molecules to the same file you should use the `Outputfile` class.

class `oddt.toolkits.rdk.MoleculeData (Mol)`

Bases: `object`

Store molecule data in a dictionary-type object

Required parameters: `Mol` – an RDKit `Mol`

Methods and accessor methods are like those of a dictionary except that the data is retrieved on-the-fly from the underlying `Mol`.

Example: `>>> mol = readfile("sdf", 'head.sdf').next() >>> data = mol.data >>> print data {'Comment': 'CORINA 2.61 0041 25.10.2001', 'NSC': '1'} >>> print len(data), data.keys(), data.has_key("NSC") 2 ['Comment', 'NSC'] True >>> print data['Comment'] CORINA 2.61 0041 25.10.2001 >>> data['Comment'] = 'This is a new comment' >>> for k,v in data.iteritems(): ... print k, "→", v Comment → This is a new comment NSC → 1 >>> del data['NSC'] >>> print len(data), data.keys(), data.has_key("NSC") 1 ['Comment'] False`

Methods

`clear()`

`has_key(key)`

`items()`

`iteritems()`

`keys()`

`update(dictionary)`

`values()`

```
clear()
has_key(key)
items()
iteritems()
keys()
update(dictionary)
values()
```

class `oddt.toolkits.rdk.Outputfile` (*format, filename, overwrite=False*)
Bases: `object`

Represent a file to which *output* is to be sent.

Required parameters:

format - see the `outformats` variable for a list of available output formats
filename

Optional parameters:

overwrite – if the output file already exists, should it be overwritten? (default is `False`)

Methods: `write(molecule)` `close()`

Methods

<code>close()</code>	Close the Outputfile to further writing.
<code>write(molecule)</code>	Write a molecule to the output file.

close()
Close the Outputfile to further writing.

write (*molecule*)
Write a molecule to the output file.

Required parameters: `molecule`

class `oddt.toolkits.rdk.Smarts` (*smartspattern*)
Bases: `object`

A Smarts Pattern Matcher

Required parameters: `smartspattern`

Methods: `findall(molecule)`

Example: `>>> mol = readstring("smi","CCN(CC)CC") # triethylamine >>> smarts = Smarts("[#6][#6]") #`
Matches an ethyl group `>>> print smarts.findall(mol) [(0, 1), (3, 4), (5, 6)]`

The numbers returned are the indices (starting from 0) of the atoms that match the SMARTS pattern. In this case, there are three matches for each of the three ethyl groups in the molecule.

Methods

`findall(molecule)` Find all matches of the SMARTS pattern to a particular molecule.

findall (*molecule*)

Find all matches of the SMARTS pattern to a particular molecule.

Required parameters: molecule

`oddt.toolkits.rdk.descs = []`

A list of supported descriptors

`oddt.toolkits.rdk.forcefields = ['uff']`

A list of supported forcefields

`oddt.toolkits.rdk.fps = ['rdkit', 'layered', 'maccs', 'atompairs', 'torsions', 'morgan']`

A list of supported fingerprint types

`oddt.toolkits.rdk.informats = {'inchi': 'InChI', 'mol2': 'Tripos MOL2 file', 'sdf': 'MDL SDF file', 'smi': 'SMILES', ...}`

A dictionary of supported input formats

`oddt.toolkits.rdk.outformats = {'inchikey': 'InChIKey', 'sdf': 'MDL SDF file', 'can': 'Canonical SMILES', 'smi': 'SMILES', ...}`

A dictionary of supported output formats

`oddt.toolkits.rdk.readfile (format, filename, *args, **kwargs)`

Iterate over the molecules in a file.

Required parameters:

format - see the **informats** variable for a list of available input formats

filename

You can access the first molecule in a file using the `next()` method of the iterator:

```
mol = readfile("smi", "myfile.smi").next()
```

You can make a list of the molecules in a file using: `mols = list(readfile("smi", "myfile.smi"))`

You can iterate over the molecules in a file as shown in the following code snippet: `>>> atomtotal = 0 >>> for mol in readfile("sdf", "head.sdf"): ... atomtotal += len(mol.atoms) ... >>> print atomtotal 43`

`oddt.toolkits.rdk.readstring (format, string, **kwargs)`

Read in a molecule from a string.

Required parameters:

format - see the **informats** variable for a list of available input formats

string

Example: `>>> input = "C1=CC=CS1" >>> mymol = readstring("smi", input) >>> len(mymol.atoms) 5`

Module contents

1.2 Submodules

1.3 oddt.interactions module

Module calculates interactions between two molecules (proein-protein, protein-ligand, small-small). Currently following interactions are implemented:

- hydrogen bonds
- halogen bonds
- pi stacking (parallel and perpendicular)
- salt bridges
- hydrophobic contacts
- pi-cation
- metal coordination
- pi-metal

`oddt.interactions.close_contacts(x, y, cutoff, x_column='coords', y_column='coords')`

Returns pairs of atoms which are within close contact distance cutoff.

Parameters `x, y` : atom_dict-type numpy array

Atom dictionaries generated by `oddt.toolkit.Molecule` objects.

cutoff [float] Cutoff distance for close contacts

x_column, y_column [string, (default='coords')] Column containing coordinates of atoms (or pseudo-atoms, i.e. ring centroids)

Returns `x_, y_` : atom_dict-type numpy array

Aligned pairs of atoms in close contact for further processing.

`oddt.interactions.hbond_acceptor_donor(mol1, mol2, cutoff=3.5, base_angle=120, tolerance=30)`

Returns pairs of acceptor-donor atoms, which meet H-bond criteria

Parameters `mol1, mol2` : `oddt.toolkit.Molecule` object

Molecules to compute H-bond acceptor and H-bond donor pairs

cutoff [float, (default=3.5)] Distance cutoff for A-D pairs

base_angle [int, (default=120)] Base angle determining allowed direction of hydrogen bond formation, which is divided by the number of neighbors of acceptor atom to establish final directional angle

tolerance [int, (default=30)] Range (+/- tolerance) from perfect direction ($\text{base_angle}/n_{\text{neighbors}}$) in which H-bonds are considered as strict.

Returns `a, d` : atom_dict-type numpy array

Aligned arrays of atoms forming H-bond, firstly acceptors, secondly donors.

strict [numpy array, dtype=bool] Boolean array align with atom pairs, informing whether atoms form 'strict' H-bond (pass all angular cutoffs). If false, only distance cutoff is met, therefore the bond is 'crude'.

`oddt.interactions.hbond(mol1, mol2, *args, **kwargs)`

Calculates H-bonds between molecules

Parameters `mol1, mol2` : `oddt.toolkit.Molecule` object

Molecules to compute H-bond acceptor and H-bond donor pairs

cutoff [float, (default=3.5)] Distance cutoff for A-D pairs

base_angle [int, (default=120)] Base angle determining allowed direction of hydrogen bond formation, which is divided by the number of neighbors of acceptor atom to establish final directional angle

tolerance [int, (default=30)] Range (+/- tolerance) from perfect direction (base_angle/n_neighbors) in which H-bonds are considered as strict.

Returns **mol1_atoms, mol2_atoms** : atom_dict-type numpy array

Aligned arrays of atoms forming H-bond

strict [numpy array, dtype=bool] Boolean array align with atom pairs, informing whether atoms form 'strict' H-bond (pass all angular cutoffs). If false, only distance cutoff is met, therefore the bond is 'crude'.

```
oddt.interactions.halogenbond_acceptor_halogen(mol1, mol2, base_angle_acceptor=120,
                                                base_angle_halogen=180, tolerance=30, cutoff=4)
```

Returns pairs of acceptor-halogen atoms, which meet halogen bond criteria

Parameters **mol1, mol2** : oddt.toolkit.Molecule object

Molecules to compute halogen bond acceptor and halogen pairs

cutoff [float, (default=4)] Distance cutoff for A-H pairs

base_angle_acceptor [int, (default=120)] Base angle determining allowed direction of halogen bond formation, which is divided by the number of neighbors of acceptor atom to establish final directional angle

base_angle_halogen [int (default=180)] Ideal base angle between halogen bond and halogen-neighbor bond

tolerance [int, (default=30)] Range (+/- tolerance) from perfect direction (base_angle/n_neighbors) in which halogen bonds are considered as strict.

Returns **a, h** : atom_dict-type numpy array

Aligned arrays of atoms forming halogen bond, firstly acceptors, secondly halogens

strict [numpy array, dtype=bool] Boolean array align with atom pairs, informing whether atoms form 'strict' halogen bond (pass all angular cutoffs). If false, only distance cutoff is met, therefore the bond is 'crude'.

```
oddt.interactions.halogenbond(mol1, mol2, **kwargs)
```

Calculates halogen bonds between molecules

Parameters **mol1, mol2** : oddt.toolkit.Molecule object

Molecules to compute halogen bond acceptor and halogen pairs

cutoff [float, (default=4)] Distance cutoff for A-H pairs

base_angle_acceptor [int, (default=120)] Base angle determining allowed direction of halogen bond formation, which is divided by the number of neighbors of acceptor atom to establish final directional angle

base_angle_halogen [int (default=180)] Ideal base angle between halogen bond and halogen-neighbor bond

tolerance [int, (default=30)] Range (+/- tolerance) from perfect direction (base_angle/n_neighbors) in which halogen bonds are considered as strict.

Returns **mol1_atoms, mol2_atoms** : atom_dict-type numpy array

Aligned arrays of atoms forming halogen bond

strict [numpy array, dtype=bool] Boolean array align with atom pairs, informing whether atoms form 'strict' halogen bond (pass all angular cutoffs). If false, only distance cutoff is met, therefore the bond is 'crude'.

`oddt.interactions.pi_stacking(mol1, mol2, cutoff=5, tolerance=30)`

Returns pairs of rings, which meet pi stacking criteria

Parameters **mol1, mol2** : oddt.toolkit.Molecule object

Molecules to compute ring pairs

cutoff [float, (default=5)] Distance cutoff for Pi-stacking pairs

tolerance [int, (default=30)] Range (+/- tolerance) from perfect direction (parallel or perpendicular) in which pi-stackings are considered as strict.

Returns **r1, r2** : ring_dict-type numpy array

Aligned arrays of rings forming pi-stacking

strict_parallel [numpy array, dtype=bool] Boolean array align with ring pairs, informing whether rings form 'strict' parallel pi-stacking. If false, only distance cutoff is met, therefore the stacking is 'crude'.

strict_perpendicular [numpy array, dtype=bool] Boolean array align with ring pairs, informing whether rings form 'strict' perpendicular pi-stacking (T-shaped, T-face, etc.). If false, only distance cutoff is met, therefore the stacking is 'crude'.

`oddt.interactions.salt_bridge_plus_minus(mol1, mol2, cutoff=4)`

Returns pairs of plus-minus atoms, which meet salt bridge criteria

Parameters **mol1, mol2** : oddt.toolkit.Molecule object

Molecules to compute plus and minus pairs

cutoff [float, (default=4)] Distance cutoff for A-H pairs

Returns **plus, minus** : atom_dict-type numpy array

Aligned arrays of atoms forming salt bridge, firstly plus, secondly minus

`oddt.interactions.salt_bridges(mol1, mol2, *args, **kwargs)`

Calculates salt bridges between molecules

Parameters **mol1, mol2** : oddt.toolkit.Molecule object

Molecules to compute plus and minus pairs

cutoff [float, (default=4)] Distance cutoff for plus-minus pairs

Returns **mol1_atoms, mol2_atoms** : atom_dict-type numpy array

Aligned arrays of atoms forming salt bridges

`oddt.interactions.hydrophobic_contacts(mol1, mol2, cutoff=4)`

Calculates hydrophobic contacts between molecules

Parameters **mol1, mol2** : oddt.toolkit.Molecule object

Molecules to compute hydrophobe pairs

cutoff [float, (default=4)] Distance cutoff for hydrophobe pairs

Returns **mol1_atoms, mol2_atoms** : atom_dict-type numpy array

Aligned arrays of atoms forming hydrophobic contacts

`oddt.interactions.pi_cation(mol1, mol2, cutoff=5, tolerance=30)`

Returns pairs of ring-cation atoms, which meet pi-cation criteria

Parameters **mol1, mol2** : oddt.toolkit.Molecule object

Molecules to compute ring-cation pairs

cutoff [float, (default=5)] Distance cutoff for Pi-cation pairs

tolerance [int, (default=30)] Range (+/- tolerance) from perfect direction (perpendicular) in which pi-cation are considered as strict.

Returns **r1** : ring_dict-type numpy array

Aligned rings forming pi-stacking

plus2 [atom_dict-type numpy array] Aligned cations forming pi-cation

strict_parallel [numpy array, dtype=bool] Boolean array align with ring-cation pairs, informing whether they form 'strict' pi-cation. If false, only distance cutoff is met, therefore the interaction is 'crude'.

`oddt.interactions.acceptor_metal(mol1, mol2, base_angle=120, tolerance=30, cutoff=4)`

Returns pairs of acceptor-metal atoms, which meet metal coordination criteria Note: This function is directional (mol1 holds acceptors, mol2 holds metals)

Parameters **mol1, mol2** : oddt.toolkit.Molecule object

Molecules to compute acceptor and metal pairs

cutoff [float, (default=4)] Distance cutoff for A-M pairs

base_angle [int, (default=120)] Base angle determining allowed direction of metal coordination, which is divided by the number of neighbors of acceptor atom to establish final directional angle

tolerance [int, (default=30)] Range (+/- tolerance) from perfect direction (base_angle/n_neighbors) in metal coordination are considered as strict.

Returns **a, d** : atom_dict-type numpy array

Aligned arrays of atoms forming metal coordination, firstly acceptors, secondly metals.

strict [numpy array, dtype=bool] Boolean array align with atom pairs, informing whether atoms form 'strict' metal coordination (pass all angular cutoffs). If false, only distance cutoff is met, therefore the interaction is 'crude'.

`oddt.interactions.pi_metal(mol1, mol2, cutoff=5, tolerance=30)`

Returns pairs of ring-metal atoms, which meet pi-metal criteria

Parameters **mol1, mol2** : `oddt.toolkit.Molecule` object

Molecules to compute ring-metal pairs

cutoff [float, (default=5)] Distance cutoff for Pi-metal pairs

tolerance [int, (default=30)] Range (+/- tolerance) from perfect direction (perpendicular) in which pi-metal are considered as strict.

Returns **r1** : ring_dict-type numpy array

Aligned rings forming pi-metal

m [atom_dict-type numpy array] Aligned metals forming pi-metal

strict_parallel [numpy array, dtype=bool] Boolean array align with ring-metal pairs, informing whether they form 'strict' pi-metal. If false, only distance cutoff is met, therefore the interaction is 'crude'.

1.4 oddt.metrics module

Metrics for estimating performance of drug discovery methods implemented in ODDT

`oddt.metrics.roc(y_true, y_score, pos_label=None, sample_weight=None)`

Compute Receiver operating characteristic (ROC)

Note: this implementation is restricted to the binary classification task.

Parameters **y_true** : array, shape = [n_samples]

True binary labels in range {0, 1} or {-1, 1}. If labels are not binary, pos_label should be explicitly given.

y_score : array, shape = [n_samples]

Target scores, can either be probability estimates of the positive class or confidence values.

pos_label : int

Label considered as positive and others are considered negative.

sample_weight : array-like of shape = [n_samples], optional

Sample weights.

Returns **fpr** : array, shape = [>2]

Increasing false positive rates such that element i is the false positive rate of predictions with score \geq thresholds[i].

tpr : array, shape = [>2]

Increasing true positive rates such that element i is the true positive rate of predictions with score \geq thresholds[i].

thresholds : array, shape = [n_thresholds]

Decreasing thresholds on the decision function used to compute fpr and tpr. *thresholds[0]* represents no instances being predicted and is arbitrarily set to $\max(y_score) + 1$.

See also:

roc_auc_score Compute Area Under the Curve (AUC) from prediction scores

Notes

Since the thresholds are sorted from low to high values, they are reversed upon returning them to ensure they correspond to both `fpr` and `tpr`, which are sorted in reversed order during their calculation.

References

[R1]

Examples

```
>>> import numpy as np
>>> from sklearn import metrics
>>> y = np.array([1, 1, 2, 2])
>>> scores = np.array([0.1, 0.4, 0.35, 0.8])
>>> fpr, tpr, thresholds = metrics.roc_curve(y, scores, pos_label=2)
>>> fpr
array([ 0. ,  0.5,  0.5,  1. ])
>>> tpr
array([ 0.5,  0.5,  1. ,  1. ])
>>> thresholds
array([ 0.8 ,  0.4 ,  0.35,  0.1 ])
```

`oddt.metrics.auc(x, y, reorder=False)`

Compute Area Under the Curve (AUC) using the trapezoidal rule

This is a general function, given points on a curve. For computing the area under the ROC-curve, see `roc_auc_score()`.

Parameters **x** : array, shape = [n]

x coordinates.

y : array, shape = [n]

y coordinates.

reorder : boolean, optional (default=False)

If True, assume that the curve is ascending in the case of ties, as for an ROC curve. If the curve is non-ascending, the result will be wrong.

Returns **auc** : float

See also:

roc_auc_score Computes the area under the ROC curve

precision_recall_curve Compute precision-recall pairs for different probability thresholds

Examples

```
>>> import numpy as np
>>> from sklearn import metrics
>>> y = np.array([1, 1, 2, 2])
>>> pred = np.array([0.1, 0.4, 0.35, 0.8])
>>> fpr, tpr, thresholds = metrics.roc_curve(y, pred, pos_label=2)
>>> metrics.auc(fpr, tpr)
0.75
```

`oddt.metrics.roc_auc(y_true, y_score, average='macro', sample_weight=None)`

Compute Area Under the Curve (AUC) from prediction scores

Note: this implementation is restricted to the binary classification task or multilabel classification task in label indicator format.

Parameters **y_true** : array, shape = [n_samples] or [n_samples, n_classes]

True binary labels in binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]

Target scores, can either be probability estimates of the positive class, confidence values, or binary decisions.

average : string, [None, 'micro', 'macro' (default), 'samples', 'weighted']

If None, the scores for each class are returned. Otherwise, this determines the type of averaging performed on the data:

'micro': Calculate metrics globally by considering each element of the label indicator matrix as a label.

'macro': Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

'weighted': Calculate metrics for each label, and find their average, weighted by support (the number of true instances for each label).

'samples': Calculate metrics for each instance, and find their average.

sample_weight : array-like of shape = [n_samples], optional

Sample weights.

Returns **auc** : float

See also:

average_precision_score Area under the precision-recall curve

roc_curve Compute Receiver operating characteristic (ROC)

References

[R2]

Examples

```
>>> import numpy as np
>>> from sklearn.metrics import roc_auc_score
>>> y_true = np.array([0, 0, 1, 1])
>>> y_scores = np.array([0.1, 0.4, 0.35, 0.8])
>>> roc_auc_score(y_true, y_scores)
0.75
```

`oddt.metrics.roc_log_auc(y_true, y_score, pos_label=None, log_min=0.001, log_max=1.0)`

Computes area under semi-log ROC for random distribution.

Parameters `y_true` : array, shape=[n_samples]

True binary labels, in range {0,1} or {-1,1}. If positive label is different than 1, it must be explicitly defined.

`y_score` [array, shape=[n_samples]] Scores for tested series of samples

pos_label: int Positive label of samples (if other than 1)

log_min [float (default=0.001)] Minimum logarithm value for estimating AUC

log_max [float (default=1.)] Maximum logarithm value for estimating AUC.

Returns `auc` : float

semi-log ROC AUC

`oddt.metrics.enrichment_factor(y_true, y_score, percentage=1, pos_label=None)`

Computes enrichment factor for given percentage, i.e. EF_1% is enrichment factor for first percent of given samples.

Parameters `y_true` : array, shape=[n_samples]

True binary labels, in range {0,1} or {-1,1}. If positive label is different than 1, it must be explicitly defined.

`y_score` [array, shape=[n_samples]] Scores for tested series of samples

percentage [int or float] The percentage for which EF is being calculated

pos_label: int Positive label of samples (if other than 1)

Returns `ef` : float

Enrichment Factor for given percentage

`oddt.metrics.random_roc_log_auc(log_min=0.001, log_max=1.0)`

Computes area under semi-log ROC for random distribution.

Parameters `log_min` : float (default=0.001)

Minimum logarithm value for estimating AUC

log_max [float (default=1.)] Maximum logarithm value for estimating AUC.

Returns `auc` : float

semi-log ROC AUC for random distribution

`oddt.metrics.rmse(y_true, y_pred)`

Compute Root Mean Squared Error (RMSE)

Parameters `y_true` : array-like of shape = [n_samples] or [n_samples, n_outputs]

Ground truth (correct) target values.

`y_pred` [array-like of shape = [n_samples] or [n_samples, n_outputs]] Estimated target values.

Returns `rmse` : float

A positive floating point value (the best value is 0.0).

1.5 oddt.spatial module

Spatial functions included in ODDT Mainly used by other modules, but can be accessed directly.

`oddt.spatial.angle(p1, p2, p3)`

Returns an angle from a series of 3 points (point #2 is centroid). Angle is returned in degrees.

Parameters `p1, p2, p3` : numpy arrays, shape = [n_points, n_dimensions]

Triplets of points in n-dimensional space, aligned in rows.

Returns `angles` : numpy array, shape = [n_points]

Series of angles in degrees

`oddt.spatial.angle_2v(v1, v2)`

Returns an angle from a series of 3 points (point #2 is centroid). Angle is returned in degrees.

Parameters `v1, v2` : numpy arrays, shape = [n_vectors, n_dimensions]

Pairs of vectors in n-dimensional space, aligned in rows.

Returns `angles` : numpy array, shape = [n_vectors]

Series of angles in degrees

`oddt.spatial.dihedral(p1, p2, p3, p4)`

Returns an dihedral angle from a series of 4 points. Dihedral is returned in degrees. Function distinguishes clockwise and antyclockwise dihedrals.

Parameters `p1, p2, p3, p4` : numpy arrays, shape = [n_points, n_dimensions]

Quadruplets of points in n-dimensional space, aligned in rows.

Returns `angles` : numpy array, shape = [n_points]

Series of angles in degrees

`oddt.spatial.distance(XA, XB, metric='euclidean', p=2, V=None, VI=None, w=None)`

Computes distance between each pair of the two collections of inputs.

The following are common calling conventions:

1.Y = `cdist(XA, XB, 'euclidean')`

Computes the distance between m points using Euclidean distance (2-norm) as the distance metric between the points. The points are arranged as m n -dimensional row vectors in the matrix X.

2.Y = `cdist(XA, XB, 'minkowski', p)`

Computes the distances using the Minkowski distance $\|u - v\|_p$ (p -norm) where $p \geq 1$.

```
3.Y = cdist(XA, XB, 'cityblock')
```

Computes the city block or Manhattan distance between the points.

```
4.Y = cdist(XA, XB, 'seuclidean', V=None)
```

Computes the standardized Euclidean distance. The standardized Euclidean distance between two n-vectors u and v is

$$\sqrt{\sum (u_i - v_i)^2 / V[x_i]}.$$

V is the variance vector; $V[i]$ is the variance computed over all the i 'th components of the points. If not passed, it is automatically computed.

```
5.Y = cdist(XA, XB, 'sqeuclidean')
```

Computes the squared Euclidean distance $\|u - v\|_2^2$ between the vectors.

```
6.Y = cdist(XA, XB, 'cosine')
```

Computes the cosine distance between vectors u and v ,

$$1 - \frac{u \cdot v}{\|u\|_2 \|v\|_2}$$

where $\|*\|_2$ is the 2-norm of its argument $*$, and $u \cdot v$ is the dot product of u and v .

```
7.Y = cdist(XA, XB, 'correlation')
```

Computes the correlation distance between vectors u and v . This is

$$1 - \frac{(u - \bar{u}) \cdot (v - \bar{v})}{\|(u - \bar{u})\|_2 \|(v - \bar{v})\|_2}$$

where \bar{v} is the mean of the elements of vector v , and $x \cdot y$ is the dot product of x and y .

```
8.Y = cdist(XA, XB, 'hamming')
```

Computes the normalized Hamming distance, or the proportion of those vector elements between two n-vectors u and v which disagree. To save memory, the matrix X can be of type boolean.

```
9.Y = cdist(XA, XB, 'jaccard')
```

Computes the Jaccard distance between the points. Given two vectors, u and v , the Jaccard distance is the proportion of those elements $u[i]$ and $v[i]$ that disagree where at least one of them is non-zero.

```
10.Y = cdist(XA, XB, 'chebyshev')
```

Computes the Chebyshev distance between the points. The Chebyshev distance between two n-vectors u and v is the maximum norm-1 distance between their respective elements. More precisely, the distance is given by

$$d(u, v) = \max_i |u_i - v_i|.$$

```
11.Y = cdist(XA, XB, 'canberra')
```

Computes the Canberra distance between the points. The Canberra distance between two points u and v is

$$d(u, v) = \sum_i \frac{|u_i - v_i|}{|u_i| + |v_i|}.$$

```
12.Y = cdist(XA, XB, 'braycurtis')
```

Computes the Bray-Curtis distance between the points. The Bray-Curtis distance between two points u and v is

$$d(u, v) = \frac{\sum_i (u_i - v_i)}{\sum_i (u_i + v_i)}$$

```
13.Y = cdist(XA, XB, 'mahalanobis', VI=None)
```

Computes the Mahalanobis distance between the points. The Mahalanobis distance between two points u and v is $(u - v)(1/V)(u - v)^T$ where $(1/V)$ (the `VI` variable) is the inverse covariance. If `VI` is not `None`, `VI` will be used as the inverse covariance matrix.

```
14.Y = cdist(XA, XB, 'yule')
```

Computes the Yule distance between the boolean vectors. (see `yule` function documentation)

```
15.Y = cdist(XA, XB, 'matching')
```

Computes the matching distance between the boolean vectors. (see `matching` function documentation)

```
16.Y = cdist(XA, XB, 'dice')
```

Computes the Dice distance between the boolean vectors. (see `dice` function documentation)

```
17.Y = cdist(XA, XB, 'kulsinski')
```

Computes the Kulsinski distance between the boolean vectors. (see `kulsinski` function documentation)

```
18.Y = cdist(XA, XB, 'rogerstanimoto')
```

Computes the Rogers-Tanimoto distance between the boolean vectors. (see `rogerstanimoto` function documentation)

```
19.Y = cdist(XA, XB, 'russellrao')
```

Computes the Russell-Rao distance between the boolean vectors. (see `russellrao` function documentation)

```
20.Y = cdist(XA, XB, 'sokalmichener')
```

Computes the Sokal-Michener distance between the boolean vectors. (see `sokalmichener` function documentation)

```
21.Y = cdist(XA, XB, 'sokalsneath')
```

Computes the Sokal-Sneath distance between the vectors. (see `sokalsneath` function documentation)

```
22.Y = cdist(XA, XB, 'wminkowski')
```

Computes the weighted Minkowski distance between the vectors. (see `sokalsneath` function documentation)

```
23.Y = cdist(XA, XB, f)
```

Computes the distance between all pairs of vectors in X using the user supplied 2-arity function f . For example, Euclidean distance between the vectors could be computed as follows:

```
dm = cdist(XA, XB, lambda u, v: np.sqrt(((u-v)**2).sum()))
```

Note that you should avoid passing a reference to one of the distance functions defined in this library. For example,:

```
dm = cdist(XA, XB, sokalsneath)
```

would calculate the pair-wise distances between the vectors in X using the Python function `sokalsneath`. This would result in `sokalsneath` being called $\binom{n}{2}$ times, which is inefficient. Instead, the optimized C version is more efficient, and we call it using the following syntax.:

```
dm = cdist(XA, XB, 'sokalsneath')
```

Parameters XA : ndarray

An m_A by n array of m_A original observations in an n -dimensional space.

XB : ndarray

An m_B by n array of m_B original observations in an n -dimensional space.

metric : string or function

The distance metric to use. The distance function can be 'braycurtis', 'canberra', 'chebyshev', 'cityblock', 'correlation', 'cosine', 'dice', 'euclidean', 'hamming', 'jaccard', 'kulsinski', 'mahalanobis', 'matching', 'minkowski', 'rogerstanimoto', 'russellrao', 'seuclidean', 'sokalmichener', 'sokalsneath', 'sqeuclidean', 'wminkowski', 'yule'.

w : ndarray

The weight vector (for weighted Minkowski).

p : double

The p -norm to apply (for Minkowski, weighted and unweighted)

V : ndarray

The variance vector (for standardized Euclidean).

VI : ndarray

The inverse of the covariance matrix (for Mahalanobis).

Returns Y : ndarray

A m_A by m_B distance matrix is returned. For each i and j , the metric `dist(u=XA[i], v=XB[j])` is computed and stored in the ij th entry.

Raises An exception is thrown if “ XA ” and “ XB ” do not have the same number of columns.

1.6 oddt.virtualscreening module

ODDT pipeline framework for virtual screening

```
class oddt.virtualscreening.virtualscreening(n_cpu=-1, verbose=False)
```

Methods

<code>apply_filter(expression[, filter_type, ...])</code>	Filtering method, can use raw expressions (strings to be eveled in if statement, can u
<code>dock(engine, protein, *args, **kwargs)</code>	Docking procedure.
<code>fetch()</code>	
<code>load_ligands(file_type, ligands_file)</code>	Loads file with ligands.
<code>score(function, protein, *args, **kwargs)</code>	Scoring procedure.
<code>write(fmt, filename[, csv_filename])</code>	Outputs molecules to a file
<code>write_csv(csv_filename[, keep_pipe])</code>	Outputs molecules to a csv file

apply_filter (*expression, filter_type='expression', soft_fail=0*)

Filtering method, can use raw expressions (strings to be eveled in if statement, can use oddt.toolkit.Molecule methods, eg. 'mol.molwt < 500') Currently supported presets:

- Lipinski Rule of 5 ('r5' or 'l5')
- Fragment Rule of 3 ('r3')

Parameters expression: string or list of strings

Expresion(s) to be used while filtering.

filter_type: 'expression' or 'preset' (default='expression') Specify filter type: 'expression' or 'preset'. Default strings are treated as expressions.

soft_fail: int (default=0) The number of faulures molecule can have to pass filter, aka. soft-fails.

dock (*engine, protein, *args, **kwargs*)

Docking procedure.

Parameters engine: string

Which docking engine to use.

fetch ()

load_ligands (*file_type, ligands_file*)

Loads file with ligands.

Parameters file_type: string

Type of molecular file

ligands_file: string Path to a file, which is loaded to pipeline

score (*function, protein, *args, **kwargs*)

Scoring procedure.

Parameters function: string

Which scoring function to use.

protein: oddt.toolkit.Molecule Default protein to use as reference

write (*fmt*, *filename*, *csv_filename=None*, ***kwargs*)

Outputs molecules to a file

Parameters **file_type: string**

Type of molecular file

ligands_file: string Path to a output file

csv_filename: string Optional path to a CSV file

write_csv (*csv_filename*, *keep_pipe=False*, ***kwargs*)

Outputs molecules to a csv file

Parameters **csv_filename: string**

Optional path to a CSV file

keep_pipe: bool (default=False) If set to True, the ligand pipe is sustained.

1.7 Module contents

1.7.1 Open Drug Discovery Toolkit

Universal and easy to use resource for various drug discovery tasks, ie docking, virtual screening, rescoring.

toolkit [module,] Toolkits backend module, currently OpenBabel [ob] and RDKit [rdk]. This setting is toolkit-wide, and sets given toolkit as default

Open Drug Discovery Toolkit

Universal and easy to use resource for various drug discovery tasks, ie docking, virtual screening, rescoring.

toolkit [module,] Toolkits backend module, currently OpenBabel [ob] and RDKit [rdk]. This setting is toolkit-wide, and sets given toolkit as default

Indices and tables

- `genindex`
- `modindex`
- `search`

Bibliography

- [R1] [Wikipedia entry for the Receiver operating characteristic](#)
[R2] [Wikipedia entry for the Receiver operating characteristic](#)

O

- `oddt`, [29](#)
- `oddt.docking`, [5](#)
- `oddt.docking.autodock_vina`, [3](#)
- `oddt.interactions`, [15](#)
- `oddt.metrics`, [20](#)
- `oddt.scoring`, [7](#)
- `oddt.scoring.descriptors`, [5](#)
- `oddt.scoring.descriptors.binana`, [5](#)
- `oddt.scoring.models`, [7](#)
- `oddt.scoring.models.classifiers`, [7](#)
- `oddt.spatial`, [24](#)
- `oddt.toolkits`, [15](#)
- `oddt.toolkits.ob`, [9](#)
- `oddt.toolkits.rdk`, [10](#)
- `oddt.virtualscreening`, [27](#)

A

acceptor_metal() (in module oddt.interactions), 19
addh() (oddt.toolkits.rdk.Molecule method), 11
angle() (in module oddt.spatial), 24
angle_2v() (in module oddt.spatial), 24
apply_filter() (oddt.virtualscreening.virtualscreening method), 28
Atom (class in oddt.toolkits.rdk), 10
atom_dict (oddt.toolkits.rdk.Molecule attribute), 11
atomicnum (oddt.toolkits.rdk.Atom attribute), 10
atoms (oddt.toolkits.ob.Residue attribute), 9
atoms (oddt.toolkits.rdk.Molecule attribute), 12
atoms_by_type() (in module oddt.scoring.descriptors), 5
auc() (in module oddt.metrics), 21
autodock_vina (class in oddt.docking.autodock_vina), 3

B

binana_descriptor (class in oddt.scoring.descriptors.binana), 5
build() (oddt.scoring.descriptors.binana.binana_descriptor method), 5
build() (oddt.scoring.descriptors.close_contacts method), 6
build() (oddt.scoring.descriptors.fingerprints method), 6

C

calcdesc() (oddt.toolkits.rdk.Molecule method), 12
calcfp() (oddt.toolkits.rdk.Molecule method), 12
canonic_order (oddt.toolkits.rdk.Molecule attribute), 12
charges (oddt.toolkits.rdk.Molecule attribute), 12
clean() (oddt.docking.autodock_vina.autodock_vina method), 3
clear() (oddt.toolkits.rdk.MoleculeData method), 14
clone (oddt.toolkits.rdk.Molecule attribute), 12
clone_coords() (oddt.toolkits.rdk.Molecule method), 12
close() (oddt.toolkits.rdk.Outputfile method), 14
close_contacts (class in oddt.scoring.descriptors), 6
close_contacts() (in module oddt.interactions), 16
coords (oddt.toolkits.rdk.Atom attribute), 10
coords (oddt.toolkits.rdk.Molecule attribute), 12

cross_validate() (in module oddt.scoring), 7

D

data (oddt.toolkits.rdk.Molecule attribute), 12
descs (in module oddt.toolkits.rdk), 15
dihedral() (in module oddt.spatial), 24
distance() (in module oddt.spatial), 24
dock() (oddt.docking.autodock_vina.autodock_vina method), 3
dock() (oddt.virtualscreening.virtualscreening method), 28
draw() (oddt.toolkits.rdk.Molecule method), 12

E

enrichment_factor() (in module oddt.metrics), 23
ensemble_model (class in oddt.scoring), 7

F

fetch() (oddt.virtualscreening.virtualscreening method), 28
findall() (oddt.toolkits.rdk.Smarts method), 15
Fingerprint (class in oddt.toolkits.rdk), 10
fingerprints (class in oddt.scoring.descriptors), 6
fit() (oddt.scoring.ensemble_model method), 7
fit() (oddt.scoring.scorer method), 8
forcefields (in module oddt.toolkits.rdk), 15
formalcharge (oddt.toolkits.rdk.Atom attribute), 10
formula (oddt.toolkits.rdk.Molecule attribute), 12
fps (in module oddt.toolkits.rdk), 15

H

halogenbond() (in module oddt.interactions), 17
halogenbond_acceptor_halogen() (in module oddt.interactions), 17
has_key() (oddt.toolkits.rdk.MoleculeData method), 14
hbond() (in module oddt.interactions), 16
hbond_acceptor_donor() (in module oddt.interactions), 16
hydrophobic_contacts() (in module oddt.interactions), 19

I

idx (oddt.toolkits.ob.Residue attribute), 9
 idx (oddt.toolkits.rdk.Atom attribute), 10
 informats (in module oddt.toolkits.rdk), 15
 items() (oddt.toolkits.rdk.MoleculeData method), 14
 iteritems() (oddt.toolkits.rdk.MoleculeData method), 14

K

keys() (oddt.toolkits.rdk.MoleculeData method), 14

L

load() (oddt.scoring.scorer class method), 8
 load_ligands() (oddt.virtualscreening.virtualscreening method), 28
 localopt() (oddt.toolkits.rdk.Molecule method), 12

M

make3D() (oddt.toolkits.rdk.Molecule method), 12
 Mol (oddt.toolkits.rdk.Molecule attribute), 11
 Molecule (class in oddt.toolkits.rdk), 11
 MoleculeData (class in oddt.toolkits.rdk), 13
 molwt (oddt.toolkits.rdk.Molecule attribute), 13

N

name (oddt.toolkits.ob.Residue attribute), 9
 neighbors (oddt.toolkits.rdk.Atom attribute), 10
 num_rotors (oddt.toolkits.rdk.Molecule attribute), 13

O

oddt (module), 29
 oddt.docking (module), 5
 oddt.docking.autodock_vina (module), 3
 oddt.interactions (module), 15
 oddt.metrics (module), 20
 oddt.scoring (module), 7
 oddt.scoring.descriptors (module), 5
 oddt.scoring.descriptors.binana (module), 5
 oddt.scoring.models (module), 7
 oddt.scoring.models.classifiers (module), 7
 oddt.spatial (module), 24
 oddt.toolkits (module), 15
 oddt.toolkits.ob (module), 9
 oddt.toolkits.rdk (module), 10
 oddt.virtualscreening (module), 27
 outformats (in module oddt.toolkits.rdk), 15
 Outputfile (class in oddt.toolkits.rdk), 14

P

parse_vina_docking_output() (in module oddt.docking.autodock_vina), 4
 parse_vina_scoring_output() (in module oddt.docking.autodock_vina), 4
 partialcharge (oddt.toolkits.rdk.Atom attribute), 10

pi_cation() (in module oddt.interactions), 19
 pi_metal() (in module oddt.interactions), 20
 pi_stacking() (in module oddt.interactions), 18
 pickle_mol() (in module oddt.toolkits.ob), 9
 predict() (oddt.scoring.ensemble_model method), 7
 predict() (oddt.scoring.scorer method), 8
 predict_ligand() (oddt.scoring.scorer method), 8
 predict_ligands() (oddt.scoring.scorer method), 8

R

random() (in module oddt.docking.autodock_vina), 4
 random_roc_log_auc() (in module oddt.metrics), 23
 randomforest (in module oddt.scoring.models.classifiers), 7
 raw (oddt.toolkits.rdk.Fingerprint attribute), 11
 readfile() (in module oddt.toolkits.ob), 9
 readfile() (in module oddt.toolkits.rdk), 15
 readstring() (in module oddt.toolkits.rdk), 15
 removeh() (oddt.toolkits.rdk.Molecule method), 13
 res_dict (oddt.toolkits.rdk.Molecule attribute), 13
 Residue (class in oddt.toolkits.ob), 9
 ring_dict (oddt.toolkits.rdk.Molecule attribute), 13
 rmse() (in module oddt.metrics), 23
 roc() (in module oddt.metrics), 20
 roc_auc() (in module oddt.metrics), 22
 roc_log_auc() (in module oddt.metrics), 23

S

salt_bridge_plus_minus() (in module oddt.interactions), 18
 salt_bridges() (in module oddt.interactions), 18
 save() (oddt.scoring.scorer method), 8
 score() (oddt.docking.autodock_vina.autodock_vina method), 4
 score() (oddt.scoring.ensemble_model method), 7
 score() (oddt.scoring.scorer method), 9
 score() (oddt.virtualscreening.virtualscreening method), 28
 scorer (class in oddt.scoring), 7
 set_protein() (oddt.docking.autodock_vina.autodock_vina method), 4
 set_protein() (oddt.scoring.descriptors.binana.binana_descriptor method), 5
 set_protein() (oddt.scoring.scorer method), 9
 Smarts (class in oddt.toolkits.rdk), 14
 sssr (oddt.toolkits.rdk.Molecule attribute), 13
 svm (in module oddt.scoring.models.classifiers), 7

T

title (oddt.toolkits.rdk.Molecule attribute), 13
 tmp_dir (oddt.docking.autodock_vina.autodock_vina attribute), 4

U

`unpickle_mol()` (in module `oddt.toolkits.ob`), [9](#)

`update()` (`oddt.toolkits.rdk.MoleculeData` method), [14](#)

V

`values()` (`oddt.toolkits.rdk.MoleculeData` method), [14](#)

`virtualsecreening` (class in `oddt.virtualsecreening`), [27](#)

W

`write()` (`oddt.toolkits.rdk.Molecule` method), [13](#)

`write()` (`oddt.toolkits.rdk.Outputfile` method), [14](#)

`write()` (`oddt.virtualsecreening.virtualsecreening` method),
[28](#)

`write_csv()` (`oddt.virtualsecreening.virtualsecreening`
method), [29](#)