# CONTENTS

1 Installation .................................................. 3  
   1.1 Requirements ............................................. 3  
   1.2 Common installation problems ......................... 4  

2 Usage Instructions ........................................... 5  
   2.1 Atom, residues, bonds iteration ....................... 5  
   2.2 Reading molecules ........................................ 6  
   2.3 Numpy Dictionaries - store your molecule as an uniform structure ....... 6  
   2.4 Interaction Fingerprints ................................. 8  
   2.5 Molecular shape comparison ............................. 9  

3 ODDT command line interface (CLI) ....................... 11  

4 Development and contributions guide ..................... 13  

5 ODDT API documentation .................................. 15  
   5.1 oddt package ............................................. 15  

6 References .................................................... 75  

7 Documentation Indices and tables ......................... 77  

Bibliography .................................................... 79  

Python Module Index .......................................... 81  

Index ............................................................ 83
Contents

- Welcome to ODDT’s documentation!
  - Installation
    * Requirements
    * Common installation problems
  - Usage Instructions
    * Atom, residues, bonds iteration
    * Reading molecules
    * Numpy Dictionaries - store your molecule as an uniform structure
      · atom_dict
      · ring_dict
      · res_dict
    * Interaction Fingerprints
      · The most common usage
    * Molecular shape comparison
  - ODDT command line interface (CLI)
  - Development and contributions guide
  - ODDT API documentation
  - References
  - Documentation Indices and tables
1.1 Requirements

- Python 2.7+ or 3.4+
- OpenBabel (2.3.2+) or/and RDKit (2016.03)
- Numpy (1.8+)
- Scipy (0.14+)
- Sklearn (0.18+)
- joblib (0.8+)
- pandas (0.17.1+)
- Skimage (0.10+) (optional, only for surface generation)

Note: All installation methods assume that one of toolkits is installed. For detailed installation procedure visit toolkit’s website (OpenBabel, RDKit)

Most convenient way of installing ODDT is using PIP. All required python modules will be installed automatically, although toolkits, either OpenBabel (pip install openbabel) or RDKit need to be installed manually

```
pip install oddt
```

If you want to install cutting edge version (master branch from GitHub) of ODDT also using PIP

```
pip install git+https://github.com/oddt/oddt.git@master
```

Finally you can install ODDT straight from the source

```
wget https://github.com/oddt/oddt/archive/0.5.tar.gz
tar zxfv 0.5.tar.gz
cd oddt-0.5/
python setup.py install
```
1.2 Common installation problems
CHAPTER TWO

USAGE INSTRUCTIONS

You can use any supported toolkit united under common API (for reference see Pybel or Cinfony). All methods and software which based on Pybel/Cinfony should be drop in compatible with ODDT toolkits. In contrast to its predecessors, which were aimed to have minimalistic API, ODDT introduces extended methods and additional handles. This extensions allow to use toolkits at all its grace and some features may be backported from others to introduce missing functionalities. To name a few:

- coordinates are returned as Numpy Arrays
- atoms and residues methods of Molecule class are lazy, ie. not returning a list of pointers, rather an object which allows indexing and iterating through atoms/residues
- Bond object (similar to Atom)
- \textit{atom\_dict, ring\_dict, res\_dict} - comprehensive Numpy Arrays containing common information about given entity, particularly useful for high performance computing, ie. interactions, scoring etc.
- lazy Molecule (asynchronous), which is not converted to an object in reading phase, rather passed as a string and read in when underlying object is called
- pickling introduced for Pybel Molecule (internally saved to mol2 string)

2.1 Atom, residues, bonds iteration

One of the most common operation would be iterating through molecules atoms

```python
mol = oddt.toolkit.readstring('smi', 'c1cccc1')
for atom in mol:
    print(atom.idx)
```

Note: \texttt{mol.atoms}, returns an object (\texttt{AtomStack}) which can be access via indexes or iterated

Iterating over residues is also very convenient, especially for proteins

```python
for res in mol.residues:
    print(res.name)
```

Additionally residues can fetch atoms belonging to them:

```python
for res in mol.residues:
    for atom in res:
        print(atom.idx)
```
Bonds are also iterable, similar to residues:

```python
for bond in mol.bonds:
    print(bond.order)
    for atom in bond:
        print(atom.idx)
```

## 2.2 Reading molecules

Reading molecules is mostly identical to Pybel.

Reading from file

```python
for mol in oddt.toolkit.readfile('smi', 'test.smi'):
    print(mol.title)
```

Reading from string

```python
mol = oddt.toolkit.readstring('smi', 'c1ccccc1 benzene'):
    print(mol.title)
```

**Note:** You can force molecules to be read in asynchronously, aka “lazy molecules”. Current default is not to produce lazy molecules due to OpenBabel’s Memory Leaks in OBConverter. Main advantage of lazy molecules is using them in multiprocessing, then conversion is spreaded on all jobs.

Reading molecules from file in asynchronous manner

```python
for mol in oddt.toolkit.readfile('smi', 'test.smi', lazy=True):
    pass
```

This example will execute instantaneously, since no molecules were evaluated.

## 2.3 Numpy Dictionaries - store your molecule as an uniform structure

Most important and handy property of Molecule in ODDT are Numpy dictionaries containing most properties of supplied molecule. Some of them are straightforward, other require some calculation, ie. atom features. Dictionaries are provided for major entities of molecule: atoms, bonds, residues and rings. It was primarily used for interactions calculations, although it is applicable for any other calculation. The main benefit is marvelous Numpy broadcasting and subsetting.

Each dictionary is defined as a format in Numpy.
2.3.1 atom_dict

Atom basic information
- "coords", type: float32, shape: (3) - atom coordinates
- "charge", type: float32 - atom's charge
- "atomicnum", type: int8 - atomic number
- "atomtype", type: a4 - Sybyl atom's type
- "hybridization", type: int8 - atoms hybridization
- "neighbors", type: float32, shape: (4,3) - coordinates of non-H neighbors coordinates for angles (max of 4 neighbors should be enough)

Residue information for current atom
- "resid", type: int16 - residue ID
- "resnumber", type: int16 - residue number
- "resname", type: a3 - Residue name (3 letters)
- "isbackbone", type: bool - is atom part of backbone

Atom properties
- "isacceptor", type: bool - is atom H-bond acceptor
- "isdonor", type: bool - is atom H-bond donor
- "isdonorf", type: bool - is atom H-bond donor Hydrogen
- "ismetal", type: bool - is atom a metal
- "ishydrophobe", type: bool - is atom hydrophobic
- "isaromatic", type: bool - is atom aromatic
- "isminus", type: bool - is atom negatively charged/chargable
- "isplus", type: bool - is atom positively charged/chargable
- "ishalogen", type: bool - is atom a halogen

Secondary structure
- "isalpha", type: bool - is atom a part of alpha helix
- "isbeta", type: bool - is atom a part of beta strand

2.3.2 ring_dict

- "centroid", type: float32, shape: 3 - coordinates of ring’s centroid
- "vector", type: float32, shape: 3 - normal vector for ring
- "isalpha", type: bool - is ring a part of alpha helix
- "isbeta", type: bool - is ring a part of beta strand
2.3.3 res_dict

- ‘id’, type: int16 - residue ID
- ‘resnumber’, type: int16 - residue number
- ‘resname’, type: a3 - Residue name (3 letters)
- ‘N’, type: float32, shape: 3 - coordinates of backbone N atom
- ‘CA’, type: float32, shape: 3 - coordinates of backbone CA atom
- ‘C’, type: float32, shape: 3 - coordinates of backbone C atom
- ‘isalpha’, type: bool - is residue a part of alpha helix
- ‘isbeta’, type: bool - is residue a part of beta strand

Note: All aforementioned dictionaries are generated “on demand”, and are cached for molecule, thus can be shared between calculations. Caching of dictionaries brings incredible performance gain, since in some applications their generation is the major time consuming task.

Get all acceptor atoms:

```python
cache = mol.atom_dict['isacceptor']```

2.4 Interaction Fingerprints

Module, where interactions between two molecules are calculated and stored in fingerprint.

2.4.1 The most common usage

Firstly, loading files

```python
protein = next(oddt.toolkit.readfile('pdb', 'protein.pdb'))
protein.protein = True
ligand = next(oddt.toolkit.readfile('sdf', 'ligand.sdf'))
```

Note: You have to mark a variable with file as protein, otherwise You won’t be able to get access to e.g. ‘resname’, ‘resid’ etc. It can be done as above.

File with more than one molecule

```python
mols = list(oddt.toolkit.readfile('sdf', 'ligands.sdf'))
```

When files are loaded, You can check interactions between molecules. Let’s find out, which amino acids creates hydrogen bonds

```python
protein_atoms, ligand_atoms, strict = hbonds(protein, ligand)
print(protein_atoms['resname'])
```

Or check hydrophobic contacts between molecules
protein_atoms, ligand_atoms = hydrophobic_contacts(protein, ligand)
print(protein_atoms, ligand_atoms)

But instead of checking interactions one by one, You can use fingerprints module.

IFP = InteractionFingerprint(ligand, protein)
SIFP = SimpleInteractionFingerprint(ligand, protein)

Very often we’re looking for similar molecules. We can easily accomplish this by e.g.

results = []
reference = SimpleInteractionFingerprint(ligand, protein)
for el in query:
    fp_query = SimpleInteractionFingerprint(el, protein)
    # similarity score for current query
    cur_score = dice(reference, fp_query)
    # score is the lowest, required similarity
    if cur_score > score:
        results.append(el)
return results

2.5 Molecular shape comparison

Three methods for molecular shape comparison are supported: USR and its two derivatives: USRCAT and Electroshape.


Aside from spatial coordinates, atoms’ charges are also used as the fourth dimension to describe shape of the molecule.

To find most similar molecules from the given set, each of these methods can be used.

Loading files:

query = next(oddt.toolkit.readfile('sdf', 'query.sdf'))
database = list(oddt.toolkit.readfile('sdf', 'database.sdf'))

Example code to find similar molecules:

results = []
query_shape = usr(query)
for mol in database:
    mol_shape = usr(mol)
    similarity = usr_similarity(query_shape, mol_shape)
    if similarity > 0.7:
        results.append(mol)
To use another method, replace usr(mol) with usr_cat(mol) or electroshape(mol).
ODDT COMMAND LINE INTERFACE (CLI)

There is an `oddt` command to interface with Open Drug Discovery Toolkit from terminal, without any programming knowledge. It simply reproduces `oddt.virtualscreening.virtualscreening`. One can filter, dock and score ligands using methods implemented or compatible with ODDT. All positional arguments are treated as input ligands, whereas output must be assigned using `-O` option (following `obabel` convention). Input and output formats are defined using `-i` and `-o` accordingly. If output format is present and no output file is assigned, then molecules are printed to STDOUT.

To list all the available options issue `-h` option:

```
oddt_cli -h
```

1. Docking ligand using Autodock Vina (construct box using ligand from crystal structure) with additional RFscore v2 rescoring:

```
oddt_cli input_ligands.sdf --dock autodock_vina --receptor rec.mol2 --auto_ligand crystal_ligand.mol2 --score rfscore_v2 -O output_ligands.sdf
```

2. Filtering ligands using Lipinski RO5 and PAINS. Afterwards dock with Autodock Vina:

```
oddt_cli input_ligands.sdf --filter ro5 --filter pains --dock autodock_vina --receptor rec.mol2 --auto_ligand crystal_ligand.mol2 -O output_ligands.sdf
```

3. Dock with Autodock Vina, with precise box position and dimensions. Fix seed for reproducibility and increase exhaustiveness:

```
oddt_cli ampc/actives_final.mol2.gz --dock autodock_vina --receptor ampc/receptor.pdb --size '(8,8,8)' --center '(1,2,0.5)' --exhaustiveness 20 --seed 1 -O ampc_docked.sdf
```

4. Rescore ligands using 3 versions of RFscore and pre-trained scoring function (either pickle from ODDT or any other SF implementing `oddt.scoring.scorer` API):

```
oddt_cli docked_ligands.sdf --receptor rec.mol2 --score rfscore_v1 --score rfscore_v2 --score rfscore_v3 --score TrainedNN.pickle -O docked_ligands_rescored.sdf
```

```
```
1. Indicies All indicies within toolkit are 0-based, but for backward compatibility with OpenBabel there is \texttt{mol.idx} property. If you develop using ODDT you are encouraged to use 0-based indicies and/or \texttt{mol.idx0} and \texttt{mol.idx1} properties to be exact which convention you adhere to. Otherwise you can run into bags which are hard to catch, when writing toolkit independent code.
5.1 oddt package

5.1.1 Subpackages

oddt.docking package

Submodules

oddt.docking.AutodockVina module

class oddt.docking.AutodockVina.autodock_vina (protein=None, auto_ligand=None, size=(20, 20, 20), center=(0, 0, 0), exhaustiveness=8, num_modes=9, energy_range=3, seed=None, prefix_dir=None, n_cpu=1, executable=None, autocleanup=True, skip_bad_mols=True)

Bases: object

Autodock Vina docking engine, which extends it’s capabilities: automatic box (auto-centering on ligand).

Parameters

protein: oddt.toolkit.Molecule object (default=None)  Protein object to be used while generating descriptors.

auto_ligand: oddt.toolkit.Molecule object or string (default=None)  Ligand use to center the docking box. Either ODDT molecule or a file (opened based on extension and read to ODDT molecule). Box is centered on geometric center of molecule.

size: tuple, shape=[3] (default=(20, 20, 20))  Dimentions of docking box (in Angstroms)

center: tuple, shape=[3] (default=(0,0,0))  The center of docking box in cartesian space.

exhaustiveness: int (default=8)  Exhaustiveness parameter of Autodock Vina

num_modes: int (default=9)  Number of conformations generated by Autodock Vina. The maximum number of docked poses is 9 (due to Autodock Vina limitation).

energy_range: int (default=3)  Energy range cutoff for Autodock Vina

seed: int or None (default=None)  Random seed for Autodock Vina

prefix_dir: string or None (default=None)  Temporary directory for Autodock Vina files. By default (None) system temporary directory is used, for reference see tempfile.gettempdir.
executable: string or None (default=None)  Autodock Vina executable location in the system. It’s really necessary if autodetection fails.

autocleanup: bool (default=True)  Should the docking engine clean up after execution?

skip_bad_mols: bool (default=True)  Should molecules that crash Autodock Vina be skipped.

Attributes

tmp_dir

Methods

dock(ligands[, protein])  Automated docking procedure.
predict_ligand(ligand)  Local method to score one ligand and update it’s scores.
predict_ligands(ligands)  Method to score ligands lazily
score(ligands[, protein])  Automated scoring procedure.
set_protein(protein)  Change protein to dock to.

clean ()
dock (ligands, protein=None)  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.

Returns

ligands  [array of oddt.toolkit.Molecule objects]  Array of ligands (scores are stored in mol.data method)
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
score(ligands, protein=None)
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.

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.

property tmp_dir

oddt.docking.AutodockVina.parse_vina_docking_output(output)
Function parsing Autodock Vina docking output to a dictionary

Parameters

output [string] Autodock Vina standard ouptud (STDOUT).

Returns

out [dict] dictionary containing scores computed by Autodock Vina

oddt.docking.AutodockVina.parse_vina_scoring_output(output)
Function parsing Autodock Vina scoring output to a dictionary

Parameters

output [string] Autodock Vina standard ouptud (STDOUT).

Returns

out [dict] dictionary containing scores computed by Autodock Vina

oddt.docking.AutodockVina.write_vina_pdbqt(mol, directory, flexible=True, name_id=None)
Write single PDBQT molecule to a given directory. For proteins use flexible=False to avoid encoding torsions. Additionally an name ID can be appended to a name to avoid conflicts.

oddt.docking.internal module

ODDT’s internal docking/scoring engines

oddt.docking.internal.change_dihedral(coords, a1, a2, a3, a4, target_angle, rot_mask)

oddt.docking.internal.get_children(molecule, mother, restricted)

oddt.docking.internal.get_close_neighbors(molecule, a_idx, num_bonds=1)

oddt.docking.internal.num_rotors_pdbqt(lig)

class oddt.docking.internal.vina_docking(rec, lig=None, box=None, box_size=1.0, weights=None)
Bases: object

5.1. oddt package
Methods

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>correct_radius</td>
</tr>
<tr>
<td>score</td>
</tr>
<tr>
<td>score_inter</td>
</tr>
<tr>
<td>score_intra</td>
</tr>
<tr>
<td>score_total</td>
</tr>
<tr>
<td>set_box</td>
</tr>
<tr>
<td>set_coords</td>
</tr>
<tr>
<td>set_ligand</td>
</tr>
<tr>
<td>set_protein</td>
</tr>
<tr>
<td>weighted_inter</td>
</tr>
<tr>
<td>weighted_intra</td>
</tr>
<tr>
<td>weighted_total</td>
</tr>
</tbody>
</table>

**correct_radius**(atom_dict)

**score**(coords=None)

**score_inter**(coords=None)

**score_intra**(coords=None)

**score_total**(coords=None)

**set_box**(box)

**set_coords**(coords)

**set_ligand**(lig)

**set_protein**(rec)

**weighted_inter**(coords=None)

**weighted_intra**(coords=None)

**weighted_total**(coords=None)

**Class** oddt.docking.internal.vina_ligand(c0, num_rotors, engine, box_size=1)

Bases: object

**Methods**

**mutate**(x2, force=False)
Module contents

class oddt.docking.autodock_vina:

protein=None, auto_ligand=None, size=(20, 20, 20), center=(0, 0, 0),
exhaustiveness=8, num_modes=9, energy_range=3, seed=None, prefix_dir=None, n_cpu=1,
executable=None, autocleanup=True, skip_bad_mols=True

Bases: object

Autodock Vina docking engine, which extends its capabilities: automatic box (auto-centering on ligand).

Parameters

protein: oddt.toolkit.Molecule object (default=None) Protein object to be used while generating descriptors.

auto_ligand: oddt.toolkit.Molecule object or string (default=None) Ligand use to center the docking box. Either ODDT molecule or a file (opened based on extension and read to ODDT molecule). Box is centered on geometric center of molecule.


center: tuple, shape=[3] (default=(0,0,0)) The center of docking box in cartesian space.

exhaustiveness: int (default=8) Exhaustiveness parameter of Autodock Vina

num_modes: int (default=9) Number of conformations generated by Autodock Vina. The maximum number of docked poses is 9 (due to Autodock Vina limitation).

energy_range: int (default=3) Energy range cutoff for Autodock Vina

seed: int or None (default=None) Random seed for Autodock Vina

prefix_dir: string or None (default=None) Temporary directory for Autodock Vina files. By default (None) system temporary directory is used, for reference see `tempfile.gettempdir`.

executable: string or None (default=None) Autodock Vina executable location in the system. It’s really necessary if autodetection fails.

autocleanup: bool (default=True) Should the docking engine clean up after execution?

skip_bad_mols: bool (default=True) Should molecules that crash Autodock Vina be skipped.

Attributes

tmp_dir

Methods

dock(ligands[, protein]) Automated docking procedure.

predict_ligand(ligand) Local method to score one ligand and update its scores.

predict_ligands(ligands) Method to score ligands lazily

score(ligands[, protein]) Automated scoring procedure.

set_protein(protein) Change protein to dock to.

clean()
dock (ligands, protein=None)
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.

Returns

- **ligands**: array of oddt.toolkit.Molecule objects  Array of ligands (scores are stored in mol.data method)

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

score (ligands, protein=None)
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.

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.

property tmp_dir

Chapter 5. ODDT API documentation
oddt.scoring package

Subpackages

oddt.scoring.descriptors package

Submodules

oddt.scoring.descriptors.binana module

Internal implementation of binana software (http://nbcr.ucsd.edu/data/sw/hosted/binana/)

class oddt.scoring.descriptors.binana.binana_descriptor(protein=None)

Bases: object

Descriptor build from binana script (as used in NNScore 2.0

Parameters

protein: oddt.toolkit.Molecule object (default=None) Protein object to be used while generating descriptors.

Methods

build(ligands[, protein]) Descriptor building method

set_protein(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

class oddt.scoring.descriptors.autodock_vina_descriptor (protein=None, vina_scores=None)

Bases: object

Methods

build (ligands, protein=None)
set_protein (protein)

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

Bases: object

Close contacts descriptor which tallies atoms of type X in certain cutoff from atoms of type Y.

Parameters

protein: oddt.toolkit.Molecule or None (default=None) Default protein to use as reference
cutoff: int or list, shape=[n,] or shape=[n,2] (default=4) Cutoff for atoms in Angstroms
        given as an integer or a list of ranges, eg. [0, 4, 8, 12] or [[0,4],[4,8],[8,12]]. Upper bound
        is always inclusive, lower exclusive.
mode: string (default='atomic_nums') Method of atoms selection, as used in atoms_by_type
ligand_types: array List of ligand atom types to use
protein_types: array List of protein atom types to use
aligned_pairs: bool (default=False) Flag indicating should permutation of types should be
done, otherwise the atoms are treated as aligned pairs.

Methods

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

build (ligands, protein=None)

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

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

Bases: object
Methods

**build** *(mols)*

```python
class oddt.scoring.descriptors.oddt_vina_descriptor(protein=None, vina_scores=None)
```

Bases: object

Methods

**build**(ligands, protein=None)

**set_protein**(protein)

oddt.scoring.functions package

Submodules

oddt.scoring.functions.NNScore module

```python
class oddt.scoring.functions.NNScore.nnscore(protein=None, n_jobs=-1)
```

Bases: oddt.scoring.scorer


Parameters

- **protein** [oddt.toolkit.Molecule object] Receptor for the scored ligands
- **n_jobs**: int (default=-1) Number of cores to use for scoring and training. By default (-1) all cores are allocated.

References

[1], [2]
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fit(ligands, target, *args, **kwargs)</code></td>
<td>Trains model on supplied ligands and target values</td>
</tr>
<tr>
<td><code>load([filename, pdbbind_version])</code></td>
<td>Loads scoring function from a pickle file.</td>
</tr>
<tr>
<td><code>predict(ligands, *args, **kwargs)</code></td>
<td>Predicts values (eg.</td>
</tr>
<tr>
<td><code>predict_ligand(ligand)</code></td>
<td>Local method to score one ligand and update it’s scores.</td>
</tr>
<tr>
<td><code>predict_ligands(ligands)</code></td>
<td>Method to score ligands in a lazy fashion.</td>
</tr>
<tr>
<td><code>save(filename)</code></td>
<td>Saves scoring function to a pickle file.</td>
</tr>
<tr>
<td><code>score(...)</code></td>
<td></td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set_protein(protein)</code></td>
<td>Proxy method to update protein in all relevant places.</td>
</tr>
</tbody>
</table>

```
gen_training_data
```

```
train
```

```
home_dir=None, use_proteins=False)
classmethod load (filename=None, pdbbind_version=2016)
```

Loads scoring function from a pickle file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filename: string</code></td>
<td>Pickle filename</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Score</th>
<th>scorer-like object</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf: scorer-like object</td>
<td>Scoring function object loaded from a pickle</td>
</tr>
</tbody>
</table>

```
train (home_dir=None, sf_pickle=None, pdbbind_version=2016)
```

oddt.scoring.functions.PLECscore module

```
class oddt.scoring.functions.PLECscore.PLECscore (protein=None, n_jobs=-1, version='linear', depth_protein=5, depth_ligand=1, size=65536)
```

Bases: `oddt.scoring.scorer`

PLECscore - a novel scoring function based on PLEC fingerprints. The underlying model can be one of:

- linear regression
- neural network (dense, 200x200x200)
- random forest (100 trees)

The scoring function is trained on PDBbind v2016 database and even with linear model outperforms other machine-learning ones in terms of Pearson correlation coefficient on “core set”. For details see PLEC publication. PLECscore predicts binding affinity (pKi/d).

New in version 0.6.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>protein</code></td>
<td>[oddt.toolkit.Molecule object] Receptor for the scored ligands</td>
</tr>
</tbody>
</table>
n_jobs: int (default=-1) Number of cores to use for scoring and training. By default (-1) all cores are allocated.

version: str (default='linear') A version of scoring function ('linear', 'nn' or 'rf') - which model should be used for the scoring function.

depth_protein: int (default=5) The depth of ECFP environments generated on the protein side of interaction. By default 6 (0 to 5) environments are generated.

depth_ligand: int (default=1) The depth of ECFP environments generated on the ligand side of interaction. By default 2 (0 to 1) environments are generated.

size: int (default=65536) The final size of a folded PLEC fingerprint. This setting is not used to limit the data encoded in PLEC fingerprint (for that tune the depths), but only the final length. Setting it too low value will lead to many collisions.

Methods

**fit**(ligands, target, *args, **kwargs) Trains model on supplied ligands and target values

**load**(filename, version, pdbbind_version, ...) Loads scoring function from a pickle file.

**predict**(ligands, *args, **kwargs) Predicts values (e.g.

**predict_ligand**(ligand) Local method to score one ligand and update it’s scores.

**predict_ligands**(ligands) Method to score ligands in a lazy fashion.

**save**(filename) Saves scoring function to a pickle file.

**score**(...) Parameters

**set_protein**(protein) Proxy method to update protein in all relevant places.

<table>
<thead>
<tr>
<th>gen_json</th>
<th>gen_training_data</th>
<th>train</th>
</tr>
</thead>
</table>

**gen_json** (home_dir=None, pdbbind_version=2016)

**gen_training_data** (pdbbind_dir, pdbbind_versions=(2016), use_proteins=True, home_dir=None)

Classmethod **load** (filename=None, version='linear', pdbbind_version=2016, depth_protein=5, depth_ligand=1, size=65536)

Loads scoring function from a pickle file.

Parameters

**filename:** string Pickle filename

Returns

**sf:** scorer-like object Scoring function object loaded from a pickle

**train** (home_dir=None, sf_pickle=None, pdbbind_version=2016, ignore_json=False)
oddt.scoring.functions.RFScore module

class oddt.scoring.functions.RFScore.rfscore:

    Parameters

    n_jobs: int (default=-1) Number of cores to use for scoring and training. By default (-1) all
cores are allocated.
    version: int (default=1) Scoring function variant. The default is the simplest one (v1).
    spr: int (default=0) The minimum number of contacts in each pair of atom types in the training
    set for the column to be included in training. This is a way of removal of not frequent and
    empty contacts.

    References

    [1], [2], [3]

    Methods

    fit(ligands, target, *args, **kwargs) Trains model on supplied ligands and target values.
    load([filename, version, pdbbind_version]) Loads scoring function from a pickle file.
    predict(ligands, *args, **kwargs) Predicts values (eg.
    predict_ligand(ligand) Local method to score one ligand and update it’s
    scores.
    predict_ligands(ligands) Method to score ligands in a lazy fashion.
    save(filename) Saves scoring function to a pickle file.
    score(...) Proxy method to update protein in all relevant places.

    set_protein(protein)

    gen_training_data
    train

home_dir=None, use_proteins=False)

classmethod load(filename=None, version=1, pdbbind_version=2016)

    Loads scoring function from a pickle file.

    Parameters

    filename: string Pickle filename
Returns

sf: scorer-like object  Scoring function object loaded from a pickle

\texttt{train}(home\_dir=None, sf\_pickle=None, pdbbind\_version=2016)

Module contents

class oddt.scoring.functions.PLECscore(protein=None, n\_jobs=-1, version='linear',
    depth\_protein=5, depth\_ligand=1, size=65536)

Bases: oddt.scoring.scorer

PLECscore - a novel scoring function based on PLEC fingerprints. The underlying model can be one of:

- linear regression
- neural network (dense, 200x200x200)
- random forest (100 trees)

The scoring function is trained on PDBbind v2016 database and even with linear model outperforms other machine-learning ones in terms of Pearson correlation coefficient on “core set”. For details see PLEC publication. PLECscore predicts binding affinity (pKi/d).

New in version 0.6.

Parameters

protein  [oddt.toolkit.Molecule object] Receptor for the scored ligands

n\_jobs: int (default=-1) Number of cores to use for scoring and training. By default (-1) all cores are allocated.

version: str (default='linear') A version of scoring function (‘linear’, ‘nn’ or ‘rf’) - which model should be used for the scoring function.

depth\_protein: int (default=5) The depth of ECFP environments generated on the protein side of interaction. By default 6 (0 to 5) environments are generated.

depth\_ligand: int (default=1) The depth of ECFP environments generated on the ligand side of interaction. By default 2 (0 to 1) environments are generated.

size: int (default=65536) The final size of a folded PLEC fingerprint. This setting is not used to limit the data encoded in PLEC fingerprint (for that tune the depths), but only the final length. Setting it to too low value will lead to many collisions.

Methods

---

\texttt{gen\_json}(home\_dir=None, pdbbind\_version=2016)

\texttt{gen\_training\_data}(pdbbind\_dir, pdbbind\_versions=(2016), home\_dir=None, use\_proteins=True)

5.1. oddt package
class method load(filename=None, version='linear', pdbbind_version=2016, depth_protein=5, depth_ligand=1, size=65536)

Loads scoring function from a pickle file.

Parameters

    filename: string  Pickle filename

Returns

    sf: scorer-like object  Scoring function object loaded from a pickle

train(home_dir=None, sf_pickle=None, pdbbind_version=2016, ignore_json=False)

class oddt.scoring.functions.nnscore(protein=None, n_jobs=-1)

Bases: oddt.scoring.scorer


Parameters

    protein [oddt.toolkit.Molecule object] Receptor for the scored ligands

    n_jobs: int (default=-1)  Number of cores to use for scoring and training. By default (-1) all cores are allocated.

References

[1], [2]

Methods

fit(ligands, target, *args, **kwargs)  Trains model on supplied ligands and target values

load([filename, pdbbind_version])  Loads scoring function from a pickle file.

predict(ligands, *args, **kwargs)  Predicts values (eg.

predict_ligand(ligand)  Local method to score one ligand and update it’s scores.

predict_ligands(ligands)  Method to score ligands in a lazy fashion.

save(filename)  Saves scoring function to a pickle file.

score(...)  

Parameters

set_protein(protein)  Proxy method to update protein in all relevant places.

gen_training_data

train


class method load(filename=None, pdbbind_version=2016)

Loads scoring function from a pickle file.

Parameters

    filename: string  Pickle filename
Returns

**sf**: scorer-like object  Scoring function object loaded from a pickle

**train**  
(home_dir=None, sf_pickle=None, pdbbind_version=2016)

```python
class oddt.scoring.functions.rfscore (protein=None, n_jobs=-1, version=1, spr=0, **kwargs)
```

**Bases**: oddt.scoring.scorer

Scoring function implementing RF-Score variants. It predicts the binding affinity (pKi/d) of ligand in a complex utilizing simple descriptors (close contacts of atoms <12A) with sophisticated machine-learning model (random forest). The third variand supplements those contacts with Vina partial scores. For further details see RF-Score publications v1[R062ccc3ea4fa-1]_, v2[R062ccc3ea4fa-2]_, v3[R062ccc3ea4fa-3]_.

**Parameters**

- **protein**  [oddt.toolkit.Molecule object] Receptor for the scored ligands
- **n_jobs**: int (default=-1) Number of cores to use for scoring and training. By default (-1) all cores are allocated.
- **version**: int (default=1) Scoring function variant. The default is the simplest one (v1).
- **spr**: int (default=0) The minimum number of contacts in each pair of atom types in the training set for the column to be included in training. This is a way of removal of not frequent and empty contacts.

**References**

[1], [2], [3]

**Methods**

- `fit(ligands, target, *args, **kwargs)` Trains model on supplied ligands and target values
- `load([filename, version, pdbbind_version])` Loads scoring function from a pickle file.
- `predict(ligands, *args, **kwargs)` Predicts values (eg.
- `predict_ligand(ligand)` Local method to score one ligand and update it’s scores.
- `predict_ligands(ligands)` Method to score ligands in a lazy fashion.
- `save(filename)` Saves scoring function to a pickle file.
- `score(...)`

**Parameters**

- `set_protein(protein)` Proxy method to update protein in all relevant places.

```python
gen_training_data
train
```

```python
classmethod load (filename=None, version=1, pdbbind_version=2016)
```

**Parameters**
filename: string  Pickle filename

Returns

sf: scorer-like object  Scoring function object loaded from a pickle

class train( home_dir=None, sf_pickle=None, pdbbind_version=2016 )

oddt.scoring.models package

Submodules

oddt.scoring.models.classifiers module

class oddt.scoring.models.classifiers.neuralnetwork(*args, **kwargs)

Bases: oddt.scoring.models.classifiers.OddtClassifier

Assemble Neural network or SVM using sklearn pipeline

Methods

score(descs, target_values)  Return the mean accuracy on the given test data and labels.

fit
get_params
predict
predict_log_proba
predict_proba
set_params

oddt.scoring.models.classifiers.randomforest

alias of sklearn.ensemble._forest.RandomForestClassifier

class oddt.scoring.models.classifiers.svm(*args, **kwargs)

Bases: oddt.scoring.models.classifiers.OddtClassifier

Assemble Neural network or SVM using sklearn pipeline

Methods

score(descs, target_values)  Return the mean accuracy on the given test data and labels.

fit
get_params
predict
predict_log_proba
predict_proba
set_params
**oddt.scoring.models.regressors module**

Collection of regressors models

```python
oddt.scoring.models.regressors.mlrr
alias of sklearn.linear_model._base.LinearRegression
```

```python
class oddt.scoring.models.regressors.neuralnetwork(*args, **kwargs)
    Bases: oddt.scoring.models.regressors.OddtRegressor

    Assemble Neural network or SVM using sklearn pipeline
```

**Methods**

```python
score(descs, target_values)  # Return the coefficient of determination $R^2$ of the prediction.
```

```python
oddt.scoring.models.regressors.pls
alias of sklearn.cross_decomposition._pls.PLS Regression
```

```python
oddt.scoring.models.regressors.randomforest
alias of sklearn.ensemble._forest.RandomForestRegressor
```

```python
class oddt.scoring.models.regressors.svm(*args, **kwargs)
    Bases: oddt.scoring.models.regressors.OddtRegressor

    Assemble Neural network or SVM using sklearn pipeline
```

**Methods**

```python
score(descs, target_values)  # Return the coefficient of determination $R^2$ of the prediction.
```
Module contents

Module contents

oddt.scoring.cross_validate(model, cv_set, cv_target, n=10, shuffle=True, n_jobs=1)
Perform cross validation of model using provided data

Parameters

model: object  Model to be tested


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

n: integer (default = 10)  How many folds to be created from dataset

shuffle: bool (default = True)  Should data be shuffled before folding.

n_jobs: integer (default = 1)  How many CPUs to use during cross validation

Returns

r2: array of shape = [n]  R^2 score for each of generated folds

class oddt.scoring.ensemble_descriptor(descriptor_generators)
Proxy class to build an ensemble of descriptors with an API as one

Parameters

models: array  An array of models

Methods

build(mols, *args, **kwargs)
set_protein(protein)

class oddt.scoring.ensemble_model(models)
Proxy class to build an ensemble of models with an API as one

Parameters

models: array  An array of models
Methods

<table>
<thead>
<tr>
<th>fit</th>
<th>predict</th>
<th>score</th>
</tr>
</thead>
</table>

**fit** \((X, y, *\text{args}, **\text{kwargs})\)

**predict** \((X, *\text{args}, **\text{kwargs})\)

**score** \((X, y, *\text{args}, **\text{kwargs})\)

**class** `oddt.scoring.scorer` *(model_instance, descriptor_generator_instance, score_title=’score’)*

**Bases:** object

Scorer class is parent class for scoring functions.

**Parameters**

- **model_instance:** model  Model compatible with sklearn API (fit, predict and score methods)
- **descriptor_generator_instance:** array of descriptors  Descriptor generator object
- **score_title:** string  Title of score to be used.

**Methods**

| fit(ligands, target, *\text{args}, **\text{kwargs}) | Trains model on supplied ligands and target values |
| load(filename) | Loads scoring function from a pickle file. |
| predict(ligands, *\text{args}, **\text{kwargs}) | Predicts values (eg. |
| predict_ligand(ligand) | Local method to score one ligand and update it’s scores. |
| predict_ligands(ligands) | Method to score ligands in a lazy fashion. |
| save(filename) | Saves scoring function to a pickle file. |
| score(...) |  |

**Parameters**

- **set_protein**(protein)  Proxy method to update protein in all relevant places.

**fit** \((ligands, target, *\text{args}, **\text{kwargs})\)

Trains model on supplied ligands and target values

**Parameters**

- **ligands:** array-like of ligands  Molecules to featurize and feed into the model
- **target:** array-like of shape \([n_{\text{samples}}]\) or \([n_{\text{samples}}, n_{\text{outputs}}]\)  Ground truth (correct) 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 (e.g. affinity) for supplied ligands.

**Parameters**

  * **ligands**: array-like of ligands  
    Molecules to featurize and feed into the model

**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 in a lazy fashion.

**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** *(accuracy for classification or R^2 for regression)*

**Parameters**

  * **ligands**: array-like of ligands  
    Molecules to featurize and feed into the model

  * **target**: array-like of shape = [n_samples] or [n_samples, n_outputs]  
    Ground truth (correct) target values.

**Returns**

  * **s**: float  
    Quality score (accuracy or 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
**oddt.toolkits package**

**Subpackages**

**oddt.toolkits.extras package**

**Subpackages**

**oddt.toolkits.extras.rdkit package**

**Submodules**

**oddt.toolkits.extras.rdkit.fixer module**

---

**exception** oddt.toolkits.extras.rdkit.fixer.AddAtomsError

Bases: Exception

**oddt.toolkits.extras.rdkit.fixer.AddMissingAtoms**(protein, residue, amap, template)

Add missing atoms to protein molecule only at the residue according to template.

**Parameters**

- **protein**: rdkit.Chem.rdchem.RWMol
  - Mol with whole protein. Note that it is modified in place.

- **residue**: Mol with residue only

- **amap**: list
  - List mapping atom IDs in residue to atom IDs in whole protein (amap[i] = j means that i'th atom in residue corresponds to j'th atom in protein)

- **template**: Residue template

---

**Returns**

- **protein**: rdkit.Chem.rdchem.RWMol  Modified protein

- **visited_bonds**: list
  - Bonds that match the template

- **is_complete**: bool
  - Indicates whether all atoms in template were found in residue

**oddt.toolkits.extras.rdkit.fixer.ExtractPocketAndLigand**(mol, cutoff=12.0, expandResidues=True, ligand_residue=None, ligand_residue_blacklist=None, append_residues=None)

Function extracting a ligand (the largest HETATM residue) and the protein pocket within certain cutoff. The selection of pocket atoms can be expanded to contain whole residues. The single atom HETATM residues are attributed to pocket (metals and waters)

**Parameters**

- **mol**: rdkit.Chem.rdchem.Mol
  - Molecule with a protein ligand complex

- **cutoff**: float (default=12.)
  - Distance cutoff for the pocket atoms
expandResidues: bool (default=True) Expand selection to whole residues within cutoff.

ligand_residue: string (default None) Residue name which explicitly point to a ligand(s).

ligand_residue_blacklist: array-like, optional (default None) List of residues to ignore during ligand lookup.

append_residues: array-like, optional (default None) List of residues to append to pocket, even if they are HETATM, such as MSE, ATP, AMP, ADP, etc.

Returns

pocket: rdkit.Chem.rdchem.RWMol Pocket constructed of protein residues/atoms around ligand

ligand: rdkit.Chem.rdchem.RWMol Largest HETATM residue contained in input molecule

oddt.toolkits.extras.rdkit.fixer.FetchAffinityTable(pdbids, affinity_types)
Fetch affinity data from RCSB PDB server.

Parameters

pdbids: array-like List of PDB IDs of structures with protein-ligand complexes.

affinity_types: array-like List of types of affinity data to retrieve. Available types are: Ki, Kd, EC50, IC50, deltaG, deltaH, deltaS, Ka.

Returns

ligand_affinity: pd.DataFrame Table with protein-ligand binding affinities. Table contains following columns: structureId, ligandId, ligandFormula, ligandMolecularWeight + columns named after affinity types specified by the user.

oddt.toolkits.extras.rdkit.fixer.FetchStructure(pdbid, sanitize=False, removeHs=True, cache_dir=None)
Fetch the structure in PDB format from RCSB PDB server and read it with rdkit.

Parameters

pdbid: str PDB IDs of the structure

sanitize: bool, optional (default False) Toggles molecule sanitation

removeHs: bool, optional (default False) Indicates whether Hs should be removed during reading

Returns

mol: Chem.rdchem.Mol Retrieved molecule

exception oddt.toolkits.extras.rdkit.fixer.FixerError Bases: Exception

oddt.toolkits.extras.rdkit.fixer.GetAtomResidueId(atom)
Return (residue number, residue name, chain id) for a given atom
oddt.toolkits.extras.rdkit_fixer.GetResidues(mol, atom_list=None)
Create dictionary that maps residues to atom IDs: (res number, res name, chain id) -> [atom1 idx, atom2 idx, ...]

oddt.toolkits.extras.rdkit_fixer.IsResidueConnected(mol, atom_ids)
Check if residue with given atom IDs is connected to other residues in the molecule.

oddt.toolkits.extras.rdkit_fixer.MolToTemplates(mol)
Prepare set of templates for a given PDB residue.

oddt.toolkits.extras.rdkit_fixer.PrepareComplexes(pdbids, pocket_dist_cutoff=12.0, affinity_types=None, cache_dir=None)
Fetch structures and affinity data from RCSB PDB server and prepare ligand-pocket pairs for small molecules with known activities.

Parameters

pdbids: array-like
List of PDB IDs of structures with protein-ligand complexes.

pocket_dist_cutoff: float, optional (default 12.) Distance cutoff for the pocket atoms

affinity_types: array-like, optional (default None) List of types of affinity data to retrieve. Available types are: Ki, Kd, EC50, IC50, deltaG, deltaH, deltaS, Ka. If not specified Ki, Kd, EC50, and IC50 are used.

Returns

complexes: dict Dictionary with pocket-ligand pairs, structured as follows: {'pdbid': {'ligid': (pocket_mol, ligand_mol)}. Ligands have binding affinity data stored as properties.

oddt.toolkits.extras.rdkit_fixer.PreparePDBMol(mol, removeHs=True, removeHOHs=True, residue_whitelist=None, residue_blacklist=None, remove_incomplete=False, add_missing_atoms=False, custom_templates=None, replace_default_templates=False)
Prepares protein molecule by:

- Removing Hs by hard using atomic number [default=True]
- Removes HOH [default=True]
- Assign bond orders from smiles of PDB residues (over 24k templates)
- Removes bonds to metals

Parameters

mol: rdkit.Chem.rdchem.Mol
Mol with whole protein.

removeHs: bool, optional (default True) If True, hydrogens will be forcefully removed

removeHOHs: bool, optional (default True) If True, remove waters using residue name

residue_whitelist: array-like, optional (default None) List of residues to clean. If not specified, all residues present in the structure will be used.
residue_blacklist: array-like, optional (default None) List of residues to ignore during cleaning. If not specified, all residues present in the structure will be cleaned.

remove_incomplete: bool, optional (default False) If True, remove residues that do not fully match the template

add_missing_atoms: bool (default=False) Switch to add missing atoms accordingly to template SMILES structure.

custom_templates: str or dict, optional (default None) Custom templates for residues. Can be either path to SMILES file, or dictionary mapping names to SMILES or Mol objects

replace_default_templates: bool, optional (default False) Indicates whether default default templates should be replaced by custom ones. If False, default templates will be updated with custom ones. This argument is ignored if custom_templates is None.

Returns

new_mol: rdkit.Chem.rdchem.RWMol Modified protein

oddt.toolkits.extras.rdkit.fixer.PreparePDBResidue(protein, residue, amap, template)

Parameters

protein: rdkit.Chem.rdchem.RWMol Mol with whole protein. Note that it is modified in place.

residue: Mol with residue only

amap: list List mapping atom IDs in residue to atom IDs in whole protein (amap[i] = j means that i’th atom in residue corresponds to j’th atom in protein)

template: Residue template

Returns

———

protein: rdkit.Chem.rdchem.RWMol Modified protein

visited_bonds: list Bonds that match the template

is_complete: bool Indicates whether all atoms in template were found in residue

oddt.toolkits.extras.rdkit.fixer.ReadTemplates(filename, resnames) Load templates from file for specified residues

exception oddt.toolkits.extras.rdkit.fixer.SanitizeError

Bases: Exception

oddt.toolkits.extras.rdkit.fixer.SimplifyMol(mol) Change all bonds to single and discharge/dearomatize all atoms. The molecule is modified in-place (no copy is made).

exception oddt.toolkits.extras.rdkit.fixer.SubstructureMatchError

Bases: Exception

oddt.toolkits.extras.rdkit.fixer.UFFConstrainedOptimize(mol, moving_atoms=None, fixed_atoms=None, cutoff=5.0, verbose=False) Minimize a molecule using UFF forcefield with a set of moving/fixed atoms. If both moving and fixed atoms
are provided, fixed_atoms parameter will be ignored. The minimization is done in-place (without copying molecule).

**Parameters**

- **mol**: rdkit.Chem.rdchem.Mol
  Molecule to be minimized.

- **moving_atoms**: array-like (default=None) Indices of freely moving atoms. If None, fixed atoms are assigned based on fixed_atoms. These two arguments are mutually exclusive.

- **fixed_atoms**: array-like (default=None) Indices of fixed atoms. If None, fixed atoms are assigned based on moving_atoms. These two arguments are mutually exclusive.

- **cutoff**: float (default=10.) Distance cutoff for the UFF minimization

**Returns**

- **mol**: rdkit.Chem.rdchem.Mol  Molecule with minimized moving_atoms

---

**Module contents**

oddt.toolkits.extras.rdkit.\_AtomListToSubMol\( (mol, amap, includeConformer=False) \)

**Parameters**

- **mol**: rdkit.Chem.rdchem.Mol
  Molecule

- **amap**: array-like List of atom indices (zero-based)

- **includeConformer**: bool (default=True) Toggle to include atoms coordinates in submolecule.

**Returns**

- **submol**: rdkit.Chem.rdchem.RWMol  Submol determined by specified atom list

oddt.toolkits.extras.rdkit.\_MolFromPDBBlock\( (molBlock, sanitize=True, removeHs=True, flavor=0) \)

oddt.toolkits.extras.rdkit.\_MolFromPDBQTBlock\( (block, sanitize=True, removeHs=True) \)

Read PDBQT block to a RDKit Molecule

**Parameters**

- **block**: string
  Residue name which explicitly points to a ligand(s).

- **sanitize**: bool (default=True) Should the sanitization be performed

- **removeHs**: bool (default=True) Should hydrogens be removed when reading molecule.

**Returns**

- **mol**: rdkit.Chem.rdchem.Mol  Molecule read from PDBQT
**oddt.toolkits.extras.rdkit.MolToPDBQTBloc**k

Write RDKit Molecule to a PDBQT block

**Parameters**

- **mol**: rdkit.Chem.rdchem.Mol
  - Molecule with a protein ligand complex

- **flexible**: bool (default=True)
  - Should the molecule encode torsions. Ligands should be flexible, proteins in turn can be rigid.

- **addHs**: bool (default=False)
  - The PDBQT format requires at least polar Hs on donors. By default Hs are added.

- **computeCharges**: bool (default=False)
  - Should the partial charges be automatically computed. If the Hs are added the charges must and will be recomputed. If there are no partial charge information, they are set to 0.0.

**Returns**

- **block**: str
  - String wit PDBQT encoded molecule

**oddt.toolkits.extras.rdkit.PDBQTAtomLines**

Create a list with PDBQT atom lines for each atom in molecule. Donors and acceptors are given as a list of atom indices.

**oddt.toolkits.extras.rdkit.PathFromAtomList**

Module contents

**Submodules**

**oddt.toolkits.common module**

Code common to all toolkits

**oddt.toolkits.common.canonize_ring_path**

Make a canonic path - list of consecutive atom IDXs bonded in a ring sorted in an uniform fasion.

1) Move the smallest index to position 0
2) Look for the smallest first step (delta IDX)
3) Ff -1 is smallest, inverse the path and move min IDX to position 0

**Parameters**

- **path**: [list of integers]
  - A list of consecutive atom indices in a ring

**Returns**

- **canonic_path**: [list of integers]
  - Sorted list of atoms

**oddt.toolkits.common.detect_secondary_structure**

Detect alpha helices and beta sheets in res_dict by phi and psi angles
**oddt.toolkits.ob module**

**class oddt.toolkits.ob.Atom (OBAtom)**

- **Bases:** pybel.Atom

  **Attributes**
  - atomicmass
  - atomicnum
  - bonds
  - cidx
  - coordidx
  - coords
  - exactmass
  - formalcharge
  - heavyvalence
  - heterovalence
  - hyb
  - \textit{idx}

  **DEPRECATED:** RDKit is 0-based and OpenBabel is 1-based.

  - \textit{idx0}
    - Note that this index is 0-based and OpenBabel’s internal index in 1-based.
  - \textit{idx1}
    - Note that this index is 1-based as OpenBabel’s internal index.

  **implicitvalence**
  - isotope
  - neighbors
  - partialcharge
  - residue
  - spin
  - type
  - valence
  - vector

  **property bonds**

  **property idx**

  **DEPRECATED:** RDKit is 0-based and OpenBabel is 1-based. State which convention you desire and use \textit{idx0} or \textit{idx1}.

  - Note that this index is 1-based as OpenBabel’s internal index.

  **property idx0**

  Note that this index is 0-based and OpenBabel’s internal index in 1-based. Changed to be compatible with RDKit

  **property idx1**

  - Note that this index is 1-based as OpenBabel’s internal index.

  **property neighbors**
**property residue**

```python
class oddt.toolkits.ob.AtomStack(OBMol):
    Bases: object
```

**Class: oddt.toolkits.ob.Bond**

```python
class oddt.toolkits.ob.Bond(OBBond):
    Bases: object
```

**Attributes**
- `atoms`
- `isrotor`
- `order`

**property atoms**

**property isrotor**

**property order**

```python
class oddt.toolkits.ob.BondStack(OBMol):
    Bases: object
```

```python
class oddt.toolkits.ob.Fingerprint(fingerprint):
    Bases: pybel.Fingerprint
```

**Attributes**
- `bits`
- `raw`

**property raw**

```python
class oddt.toolkits.ob.Molecule(OBMol=None, source=None, protein=False):
    Bases: pybel.Molecule
```

**Attributes**
- `OBMol`
- `atom_dict`
- `atoms`
- `bonds`

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

- `charge`
- `charges`
- `clone`
- `conformers`
- `coords`
- `data`
- `dim`
- `energy`
- `exactmass`
- `formula`
molwt

num_rotors Number of strict rotatable protein molecules

protein A flag for identifying the protein molecules, for which atom_dict procedures may differ.

res_dict

calcdesc([descnames])

residues

calcfp([fptype])

ring_dict

smiles

spin

sssr

title

unitcell

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addh([only_polar])</td>
<td>Add hydrogens</td>
</tr>
<tr>
<td>calccharges([model])</td>
<td>Calculate partial charges for a molecule.</td>
</tr>
<tr>
<td>calcdesc([descnames])</td>
<td>Calculate descriptor values.</td>
</tr>
<tr>
<td>calcfp([fptype])</td>
<td>Calculate a molecular fingerprint.</td>
</tr>
<tr>
<td>convertdbonds()</td>
<td>Convert Dative Bonds.</td>
</tr>
<tr>
<td>draw([show, filename, update, usecoords])</td>
<td>Create a 2D depiction of the molecule.</td>
</tr>
<tr>
<td>localopt([forcefield, steps])</td>
<td>Locally optimize the coordinates.</td>
</tr>
<tr>
<td>make2D()</td>
<td>Generate 2D coordinates for molecule</td>
</tr>
<tr>
<td>make3D([forcefield, steps])</td>
<td>Generate 3D coordinates</td>
</tr>
<tr>
<td>removeh()</td>
<td>Remove hydrogens</td>
</tr>
<tr>
<td>write([format, filename, overwrite, opt, size])</td>
<td>Write the molecule to a file or return a string.</td>
</tr>
</tbody>
</table>

property OBMol

addh (only_polar=False)

Add hydrogens

property atom_dict

property atoms

property bonds

calccharges (model='gasteiger')

Calculate partial charges for a molecule. By default the Gasteiger charge model is used.

Parameters

model [str (default="gasteiger")] Method for generating partial charges. Supported models:

* gasteiger
* mmff94
* others supported by OpenBabel (obabel -L charges)

property canonic_order

Returns np.array with canonic order of heavy atoms in the molecule
property charges
property clone
clonedcoords(source)
property coords
make2D()
  Generate 2D coordinates for molecule
make3D(forcefield='mmff94', steps=50)
  Generate 3D coordinates

property num_rotors
  Number of strict rotatable

property protein
  A flag for identifying the protein molecules, for which atom_dict procedures may differ.
removeh()
  Remove hydrogens

property res_dict
property residues
property ring_dict
property smiles
write(format='smi', filename=None, overwrite=False, opt=None, size=None)
  Write the molecule to a file or return a string.

Optional parameters:

format -- see the informsats 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)

opt -- a dictionary of format specific options For format options with no parameters, specify the value as None.

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.ob.MoleculeData(obmol)
Bases: pybel.MoleculeData

Methods
**to_dict()**

class oddt.toolkits.ob.Outputfile *(format, filename, overwrite=False, opt=None)*
    Bases: pybel.Outputfile

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>close()</td>
<td>Close the Outputfile to further writing.</td>
</tr>
<tr>
<td>write(molecule)</td>
<td>Write a molecule to the output file.</td>
</tr>
</tbody>
</table>

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**

```
    atoms  List of Atoms in the Residue
    chain  Residue chain ID
    idx    DEPRECATED: Use idx0 instead.
    idx0   Internal index (0-based) of the Residue
    name   Residue name
    number Residue number
```

**property atoms**
    List of Atoms in the Residue

**property chain**
    Residue chain ID

**property idx**
    DEPRECATED: Use idx0 instead.
    Internal index (0-based) of the Residue

**property idx0**
    Internal index (0-based) of the Residue

**property name**
    Residue name

**property number**
    Residue number

class oddt.toolkits.ob.ResidueStack *(OBMol)*
    Bases: object

---

5.1. oddt package 45
class oddt.toolkits.ob.Smarts(smartspattern)
    Bases: pybel.Smarts

    Initialise with a SMARTS pattern.

    Methods

    findall(molecule[, unique])
        Find all matches of the SMARTS pattern to a particular molecule

    match(molecule)
        Checks if there is any match.

    findall(molecule, unique=True)
        Find all matches of the SMARTS pattern to a particular molecule

    match(molecule)
        Checks if there is any match. Returns True or False

oddt.toolkits.ob.diverse_conformers_generator(mol, n_conf=10, method='confab', seed=None, **kwargs)

    Produce diverse conformers using current conformer as starting point. Returns a generator. Each conformer is a copy of original molecule object.

    New in version 0.6.

    Parameters

    mol [oddt.toolkit.Molecule object] Molecule for which generating conformers

    n_conf [int (default=10)] Target number of conformers

    method [string (default='confab')] Method for generating conformers. Supported methods: * confab * ga

    seed [None or int (default=None)] Random seed

    mutability [int (default=5)] The inverse of probability of mutation. By default 5, which translates to 1/5 (20%) chance of mutation. This setting only works with genetic algorithm method ("ga").

    convergence [int (default=5)] The number of generations with unchanged fitness, should the algorithm converge. This setting only works with genetic algorithm method ("ga").

    rmsd [float (default=0.5)] The conformers are pruned unless their RMSD is higher than this cutoff. This setting only works with Confab method ("confab").

    nconf [int (default=10000)] The number of initial conformers to generate before energy pruning. This setting only works with Confab method ("confab").

    energy_gap [float (default=5000.)] Energy gap from the lowest energy conformer to the highest possible. This setting only works with Confab method ("confab").

    Returns

    mols [list of oddt.toolkit.Molecule objects] Molecules with diverse conformers

oddt.toolkits.ob.readfile(format, filename, opt=None, lazy=False)
**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

```python
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
    ----------
    atomicnum
    bonds
    coords
    formalcharge

    idx  DEPRECATED: RDKit is 0-based and OpenBabel is 1-based.
    idx0 Note that this index is 0-based as RDKit’s
    idx1 Note that this index is 1-based and RDKit’s internal index in 0-based.

    neighbors
    partialcharge

    property atomicnum
    property bonds
    property coords
    property formalcharge

    property idx
        DEPRECATED: RDKit is 0-based and OpenBabel is 1-based. State which convention you desire and use idx0 or idx1.

        Note that this index is 1-based and RDKit’s internal index in 0-based. Changed to be compatible with OpenBabel

    property idx0
        Note that this index is 0-based as RDKit’s

    property idx1
        Note that this index is 1-based and RDKit’s internal index in 0-based. Changed to be compatible with OpenBabel

    property neighbors
    property partialcharge
```

---

5.1. oddt package 47
class oddt.toolkits.rdk.AtomStack(Mol)
    Bases: object

class oddt.toolkits.rdk.Bond(Bond)
    Bases: object

    Attributes
    atoms
    isrotor
    order

    property atoms
    property isrotor
    property order

class oddt.toolkits.rdk.BondStack(Mol)
    Bases: object

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:

        tanimoto = a | b

    Attributes
    raw

    property raw

class oddt.toolkits.rdk.Molecule(Mol=-1, source=None, *args, **kwargs)
    Bases: object

    Trap RDKit molecules which are ‘None’

    Attributes
    Mol
    atom_dict
    atoms
    bonds

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

    charges
    clone
    coords
    data
formula
molwt
num_rotors

protein A flag for identifying the protein molecules, for which atom_dict procedures may differ.
res_dict
residues
ring_dict
smiles
sssr
title

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addh(only_polar=False, **kwargs)</code></td>
<td>Add hydrogens.</td>
</tr>
<tr>
<td><code>calccharges([model])</code></td>
<td>Calculate partial charges for a molecule.</td>
</tr>
<tr>
<td><code>calcdesc([descnames])</code></td>
<td>Calculate descriptor values.</td>
</tr>
<tr>
<td><code>calcfp([fptype, opt])</code></td>
<td>Calculate a molecular fingerprint.</td>
</tr>
<tr>
<td><code>localopt([forcefield, steps])</code></td>
<td>Locally optimize the coordinates.</td>
</tr>
<tr>
<td><code>make2D()</code></td>
<td>Generate 2D coordinates for molecule</td>
</tr>
<tr>
<td><code>make3D([forcefield, steps])</code></td>
<td>Generate 3D coordinates.</td>
</tr>
<tr>
<td><code>removeh(**kwargs)</code></td>
<td>Remove hydrogens.</td>
</tr>
<tr>
<td><code>write([format, filename, overwrite, size])</code></td>
<td>Write the molecule to a file or return a string.</td>
</tr>
</tbody>
</table>

property Mol

addh (only_polar=False, **kwargs)
Add hydrogens.

property atom_dict

property atoms

property bonds

calccharges (model='gasteiger')
Calculate partial charges for a molecule. By default the Gasteiger charge model is used.

Parameters

model [str (default="gasteiger")]] Method for generating partial charges. Supported models:
* gasteiger * mmff94

calcdesc (descnames=None)
Calculate descriptor values.

Optional parameter: descnames – a list of names of descriptors

5.1. oddt package
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 of available fingerprint types.

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

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

property charges

property clone

clone_coords(source)

property coords

property data

property 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.

make2D()
    Generate 2D coordinates for molecule

make3D (forcefield='mmff94', 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 forcefield. Call localopt() if you want to improve the coordinates further.

property molwt

property num_rotors

property protein
    A flag for identifying the protein molecules, for which atom_dict procedures may differ.

removeh(**kwargs)
    Remove hydrogens.

property res_dict

property residues
property ring_dict
property smiles
property sssr
property title

\texttt{write}(format='smi', filename=None, overwrite=False, size=None, **kwargs)

Write the molecule to a file or return a string.

Optional parameters:

- \texttt{format} – see the \texttt{informat} variable for a list of available output formats (default is “smi”)
- \texttt{filename} – default is None
- \texttt{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.

\texttt{class oddt.toolkits.rdk.MoleculeData(Mol)}

\texttt{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 = next(readfile("sdf", 'head.sdf')) >>> 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.items(): ... 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

<table>
<thead>
<tr>
<th>clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>has_key</td>
</tr>
<tr>
<td>items</td>
</tr>
<tr>
<td>iteritems</td>
</tr>
<tr>
<td>keys</td>
</tr>
<tr>
<td>to_dict</td>
</tr>
<tr>
<td>update</td>
</tr>
<tr>
<td>values</td>
</tr>
</tbody>
</table>

\texttt{clear()}

\texttt{has_key(key)}

\texttt{items()}

\texttt{iteritems()}

\texttt{keys()}

\texttt{to_dict()}

5.1. oddt package
**update** *(dictionary)*

**values()**

class oddt.toolkits.rdk.Outputfile *(format, filename, overwrite=False, **kwargs)*

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**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>close()</td>
<td>Close the Outputfile to further writing.</td>
</tr>
<tr>
<td>write(molecule)</td>
<td>Write a molecule to the output file.</td>
</tr>
</tbody>
</table>

```python
close()  # Close the Outputfile to further writing.
write(molecule) # Write a molecule to the output file.
```

class oddt.toolkits.rdk.Residue *(ParentMol, atom_path, idx=0)*

Bases: object

Represent a RDKit residue.

Required parameter: ParentMol – Parent molecule (Mol) object path – atoms path of a residue

Attributes: atoms, idx, name.

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

The Mol object constucted of residues’ atoms can be accessed using the attribute: Residue

Attributes

- **atoms** List of Atoms in the Residue
- **chain** Residue chain ID
- **idx** DEPRECATED: Use `idx0` instead.
- **idx0** Internal index (0-based) of the Residue
- **name** Residue name
- **number** Residue number

**property atoms**

List of Atoms in the Residue
property chain
  Residue chain ID

property idx
  DEPRECATED: Use idx0 instead.
  Internal index (0-based) of the Residue

property idx0
  Internal index (0-based) of the Residue

property name
  Residue name

property number
  Residue number

class oddt.toolkits.rdk.ResidueStack (Mol, paths)
  Bases: object

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

  Initialise with a SMARTS pattern.

Methods

findall (molecule[, unique])
  Find all matches of the SMARTS pattern to a particular molecule.

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

findall (molecule, unique=True)
  Find all matches of the SMARTS pattern to a particular molecule.

  Required parameters: molecule

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

  Required parameters: molecule

class oddt.toolkits.rdk.base_feature_factory = <rdkit.Chem.rdMolChemicalFeatures.MolChemicalFeatureFactory object>
  Global feature factory based on BaseFeatures.fdef

class oddt.toolkits.rdk.descs = ['MaxEStateIndex', 'MinEStateIndex', 'MaxAbsEStateIndex', 'MinAbsEStateIndex', 'qed', 'MolWt', 'HeavyAtomMolWt',...
  A list of supported descriptors

class oddt.toolkits.rdk.diverse_conformers_generator (mol, n_conf=10, method='etkdg', seed=None, rmsd=0.5)
  Produce diverse conformers using current conformer as starting point. Each conformer is a copy of original molecule object.
  New in version 0.6.

  Parameters
    mol [oddt.toolkit.Molecule object] Molecule for which generating conformers
    n_conf [int (default=10)] Targer number of conformers
method [string (default='etkdg')]
Method for generating conformers. Supported methods:
“etkdg”, “etdg”, “kdg”, “dg”.

seed [None or int (default=None)]
Random seed

rmsd [float (default=0.5)]
The minimum RMSD that separates conformers to be retained (other- 
wise, they will be pruned).

Returns

mols [list of oddt.toolkit.Molecule objects] Molecules with diverse conformers

```python
oddt.toolkits.rdk.forcefields = ['uff', 'mmff94']
# A list of supported forcefields
```

```python
oddt.toolkits.rdk.fps = ['rdkit', 'layered', 'maccs', 'atompairs', 'torsions', 'morgan']
# A list of supported fingerprint types
```

```python
oddt.toolkits.rdk.informats = {'inchi': 'InChI', 'mol': 'MDL MOL file', 'mol2': 'Tripos'
# A dictionary of supported input formats
```

```python
oddt.toolkits.rdk.outformats = {'can': 'Canonical SMILES', 'inchi': 'InChI', 'inchikey':
# A dictionary of supported output formats
```

```python
oddt.toolkits.rdk.readfile (format, filename, lazy=False, opt=None, **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 = next(readfile("smi", "myfile.smi"))
```

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

```python
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

5.1.2 Submodules

5.1.3 oddt.datasets module

Datasets wrapped in convenient models

```python
class oddt.datasets.CASF(home):
    Bases: object


    Parameters

    home: string  Path to CASF dataset main directory

    Methods

    precomputed_score([scoring_function])  Load precomputed results of scoring power test for various scoring functions.

    precomputed_screening([scoring_function, ...])  Load precomputed results of screening power test for various scoring functions

    precomputed_score(scoring_function=None)

        Load precomputed results of scoring power test for various scoring functions.

        Parameters

        scoring_function: string (default=None)  Name of the scoring function to get results If None, all results are returned.

    precomputed_screening(scoring_function=None, cluster_id=None)

        Load precomputed results of screening power test for various scoring functions

        Parameters

        scoring_function: string (default=None)  Name of the scoring function to get results If None, all results are returned

        cluster_id: int (default=None)  Number of the protein cluster to get results If None, all results are returned

class oddt.datasets.dude(home):
    Bases: object

    A wrapper for DUD-E (A Database of Useful Decoys: Enhanced) http://dude.docking.org/

    Parameters

    home [str]  Path to files from dud-e

class oddt.datasets.pdbbind(home, version=None, default_set=None, opt=None):
    Bases: object

    Attributes

    activities
```

Module checks interactions between two molecules and creates interaction fingerprints.

\[ \text{oddt.fingerprints.} \text{ECFP} \left( \text{mol, depth=2, size=4096, count_bits=True, sparse=True, use_pharm_features=False} \right) \]

Extended connectivity fingerprints (ECFP) with an option to include atom features (FCPF). Depth of a fingerprint is counted as bond-steps, thus the depth for ECFP2 = 1, ECPF4 = 2, ECFP6 = 3, etc.


**Parameters**

- **mol** [oddt.toolkit.Molecule object] Input molecule for the FP calculations
- **depth** [int (default = 2)] The depth of the fingerprint, i.e. the number of bonds in Morgan algorithm. Note: For ECFP2: depth = 1, ECFP4: depth = 2, etc.
- **size** [int (default = 4096)] Final size of fingerprint to which it is folded.
- **count_bits** [bool (default = True)] Should the bits be counted or unique. In dense representation it translates to integer array (count_bits=True) or boolean array if False.
- **sparse** [bool (default= True)] Should fingerprints be dense (contain all bits) or sparse (just the on bits).
- **use_pharm_features** [bool (default=False)] Switch to use pharmacophoric features as atom representation instead of explicit atomic numbers etc.

**Returns**

- **fingerprint** [numpy array] Calculated FP of fixed size (dense) or on bits indices (sparse). Dtype is either integer or boolean.

\[ \text{oddt.fingerprints.} \text{InteractionFingerprint} \left( \text{ligand, protein, strict=True} \right) \]

Interaction fingerprint accomplished by converting the molecular interaction of ligand-protein into bit array according to the residue of choice and the interaction. For every residue (One row = one residue) there are eight bits which represent eight type of interactions:

- (Column 0) hydrophobic contacts
- (Column 1) aromatic face to face
- (Column 2) aromatic edge to face
- (Column 3) hydrogen bond (protein as hydrogen bond donor)
- (Column 4) hydrogen bond (protein as hydrogen bond acceptor)
- (Column 5) salt bridges (protein positively charged)
- (Column 6) salt bridges (protein negatively charged)
- (Column 7) salt bridges (ionic bond with metal ion)

**Parameters**
**ligand, protein** [oddt.toolkit.Molecule object] Molecules, which are analysed in order to find interactions.

**strict** [bool (default = True)] If False, do not include condition, which informs whether atoms form ‘strict’ H-bond (pass all angular cutoffs).

**Returns**

**InteractionFingerprint** [numpy array] Vector of calculated IFP (size = no residues * 8 type of interaction)

```python
oddt.fingerprints.PLEC(ligand, protein, depth_ligand=2, depth_protein=4, distance_cutoff=4.5, size=16384, count_bits=True, sparse=True, ignore_hoh=True, bits_info=None)
```

Protein ligand extended connectivity fingerprint. For every pair of atoms in contact, compute ECFP and then hash every single, corresponding depth.

**Parameters**

**ligand, protein** [oddt.toolkit.Molecule object] Molecules, which are analysed in order to find interactions.

**depth_ligand, depth_protein** [int (default = (2, 4))] The depth of the fingerprint, i.e. the number of bonds in Morgan algorithm. Note: For ECFP2: depth = 1, ECFP4: depth = 2, etc.

**size**: int (default = 16384) SPLIF is folded to given size.

**distance_cutoff**: float (default=4.5) Cutoff distance for close contacts.

**sparse** [bool (default = True)] Should fingerprints be dense (contain all bits) or sparse (just the on bits).

**count_bits** [bool (default = True)] Should the bits be counted or unique. In dense representation it translates to integer array (count_bits=True) or boolean array if False.

**ignore_hoh** [bool (default = True)] Should the water molecules be ignored. This is based on the name of the residue (‘HOH’).

**bits_info** [dict or None (default = None)] If dictionary is provided it is filled with information about bit contents. Root atom index and depth is provided for both ligand and protein. Dictionary is modified in-place.

**Returns**

**PLEC** [numpy array] fp (size = atoms in contacts * max(depth_protein, depth_ligand))

```python
oddt.fingerprints.SPLIF(ligand, protein, depth=1, size=4096, distance_cutoff=4.5)
```


**Parameters**

**ligand, protein** [oddt.toolkit.Molecule object] Molecules, which are analysed in order to find interactions.

**depth** [int (default = 1)] The depth of the fingerprint, i.e. the number of bonds in Morgan algorithm. Note: For ECFP2: depth = 1, ECFP4: depth = 2, etc.

**size**: int (default = 4096) SPLIF is folded to given size.

**distance_cutoff**: float (default=4.5) Cutoff distance for close contacts.

**Returns**
**SPLIF** [numpy array] Calculated SPLIF.shape = (no. of atoms, ). Every row consists of three elements:
- row[0] = index of hashed atoms
- row[1].shape = (7, 3) -> ligand's atom coords and 6 his neighbor's
- row[2].shape = (7, 3) -> protein's atom coords and 6 his neighbor's

oddt.fingerprints.SimpleInteractionFingerprint(*ligand, protein, strict=True*)

Based on [http://dx.doi.org/10.1016/j.csbj.2014.05.004](http://dx.doi.org/10.1016/j.csbj.2014.05.004). Every IFP consists of 8 bits per amino acid (One row = one amino acid) and present eight type of interaction:

- (Column 0) hydrophobic contacts
- (Column 1) aromatic face to face
- (Column 2) aromatic edge to face
- (Column 3) hydrogen bond (protein as hydrogen bond donor)
- (Column 4) hydrogen bond (protein as hydrogen bond acceptor)
- (Column 5) salt bridges (protein positively charged)
- (Column 6) salt bridges (protein negatively charged)
- (Column 7) salt bridges (ionic bond with metal ion)


**Parameters**

- **ligand, protein** [oddt.toolkit.Molecule object] Molecules, which are analysed in order to find interactions.
- **strict** [bool (default = True)] If False, do not include condition, which informs whether atoms form ‘strict’ H-bond (pass all angular cutoffs).

**Returns**

- **InteractionFingerprint** [numpy array] Vector of calculated IFP (size = 168)

oddt.fingerprints.dice(*a, b, sparse=False*)

Calculates the Dice coefficient, the ratio of the bits in common to the arithmetic mean of the number of ‘on’ bits in the two fingerprints. Supports integer and boolean fingerprints.

**Parameters**

- **a, b** [numpy array] Interaction fingerprints, which are compared in order to determine similarity.
- **sparse** [bool (default=False)] Type of FPs to use. Defaults to dense form.

**Returns**

- **score** [float] Similarity between a, b.

oddt.fingerprints.similarity_SPLIF(*reference, query, rmsd_cutoff=1.0*)


**Parameters**

- **reference, query**: numpy.array SPLIFs, which are compared in order to determine similarity.
- **rmsd_cutoff** [int (default = 1)] Specific threshold for which, bits are considered as fully matching.
Returns

**SimilarityScore** [float] Similarity between given fingerprints.

**oddt.fingerprints.tanimoto** *(a, b, sparse=False)*

Tanimoto coefficient, supports boolean fingerprints. Integer fingerprints are casted to boolean.

**Parameters**

- **a, b** [numpy array] Interaction fingerprints, which are compared in order to determine similarity.
- **sparse** [bool (default=False)] Type of FPs to use. Defaults to dense form.

**Returns**

- **score** [float] Similarity between a, b.

### 5.1.5 oddt.interactions module

Module calculates interactions between two molecules (protein-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.acceptor_metal** *(mol1, mol2, 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
- **tolerance** [int, (default=30)] Range (+/- tolerance) from perfect direction defined by atoms hybridization 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.close_contacts** *(x, y, cutoff, x_column='coords', y_column='coords', cut-off_low=0.0)*

Returns pairs of atoms which are within close contact distance cutoff. The cutoff is semi-inclusive, i.e (cut-off_low, 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)

cutoff_low [float (default=0.)] Lower bound of contacts to find (exclusive). Zero by default. .. versionadded:: 0.6

Returns

x_, y_ [atom_dict-type numpy array] Aligned pairs of atoms in close contact for further processing.

oddt.interactions.halogenbond_acceptor_halogen (mol1, mol2, 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

tolerance [int, (default=30)] Range (+/- tolerance) from perfect direction defined by atoms hybridization 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.halogenbonds (mol1, mol2, cutoff=4, tolerance=30)
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

tolerance [int, (default=30)] Range (+/- tolerance) from perfect direction defined by atoms hybridization 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.hbond_acceptor_donor (mol1, mol2, cutoff=3.5, tolerance=30, donor_exact=False)
Returns pairs of acceptor-donor atoms, which meet H-bond criteria

Parameters
**oddt.interactions.hbonds** *(mol1, mol2, cutoff=3.5, tolerance=30, mol1_exact=False, mol2_exact=False)*  
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
- **tolerance** [int, (default=30)] Range (+/- tolerance) from perfect direction defined by acceptor/donor hybridization in which H-bonds are considered as strict.
- **mol1_exact, mol2_exact** [bool] Use exact protonation states for donors, i.e. require Hs on donor. By default ODDT implies some tautomeric structures as protonated, even if there is no H on specific atom.

**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.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, cation_exact=False)*
Returns pairs of ring-cation atoms, which meet pi-cation criteria

**Parameters**
moll, 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.

cation_exact [bool] Requires interacting atoms to have non-zero formal charge.

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.pi_metal (moll, mol2, cutoff=5, tolerance=30)

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

Parameters

moll, 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’.

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

Returns pairs of rings, which meet pi stacking criteria

Parameters

moll, 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**

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.
- **cation_exact, anion_exact** [bool] Requires interacting atoms to have non-zero formal charge.

**Returns**

- **plus, minus** [atom_dict-type numpy array] Aligned arrays of atoms forming salt bridge, firstly plus, secondly minus.

**oddt.interactions.salt_bridges**

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.
- **cation_exact, anion_exact** [bool] Requires interacting atoms to have non-zero formal charge.

**Returns**

- **mol1_atoms, mol2_atoms** [atom_dict-type numpy array] Aligned arrays of atoms forming salt bridges.

---

**5.1.6 oddt.metrics module**

Metrics for estimating performance of drug discovery methods implemented in ODDT.

**oddt.metrics.auc**

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()`. For an alternative way to summarize a precision-recall curve, see `average_precision_score()`.

**Parameters**

- **x** [ndarray of shape (n,)] x coordinates. These must be either monotonic increasing or monotonic decreasing.
- **y** [ndarray of shape, (n,)] y coordinates.

**Returns**

- **auc** [float]

**See also:**

- `roc_auc_score` Compute the area under the ROC curve.
- `average_precision_score` Compute average precision from prediction scores.
- `precision_recall_curve` Compute precision-recall pairs for different probability thresholds.
Examples

```python
>>> 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
```

```
oodd.metrics.bedroc(y_true, y_score, alpha=20.0, pos_label=None)

Computes Boltzmann-Enhanced Discrimination of Receiver Operating Characteristic [1]. This function assumes that results are already sorted and samples with best predictions are first.

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
alpha: float Alpha. 1/Alpha should be proportional to the percentage in EF.
pos_label: int Positive label of samples (if other than 1)

Returns

bedroc_score [float] Boltzmann-Enhanced Discrimination of Receiver Operating Characteristic

References

[1]
```
oodd.metrics.enrichment_factor(y_true, y_score, percentage=1, pos_label=None, kind='fold')

Computes enrichment factor for given percentage, i.e. EF_1% is enrichment factor for first percent of given samples. This function assumes that results are already sorted and samples with best predictions are first.

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)
kind: 'fold' or 'percentage' (default='fold') Two kinds of enrichment factor: fold and percentage. Fold shows the increase over random distribution (1 is random, the higher EF the better enrichment). Percentage returns the fraction of positive labels within the top x% of dataset.

Returns

ef [float] Enrichment Factor for given percentage in range 0:1
```
oodd.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.rie**(*y_true*, *y_score*, *alpha=20*, *pos_label=None*)

Computes Robust Initial Enhancement [1]. This function assumes that results are already sorted and samples with best predictions are first.

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

* **alpha** float Alpha. 1/Alpha should be proportional to the percentage in EF.

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

Returns:

**rie_score** [float] Robust Initial Enhancement

**References**

[1]

**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).

**oddt.metrics.roc**(*y_true*, *y_score*, *, *pos_label=None*, *sample_weight=None*, *drop_intermediate=True*)

Compute Receiver operating characteristic (ROC).

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

Read more in the User Guide.

Parameters:

* **y_true** [ndarray of shape (n_samples,)] True binary labels. If labels are not either {-1, 1} or {0, 1}, then pos_label should be explicitly given.

* **y_score** [ndarray of shape (n_samples,)] Target scores, can either be probability estimates of the positive class, confidence values, or non-thresholded measure of decisions (as returned by “decision_function” on some classifiers).

* **pos_label** [int or str, default=None] The label of the positive class. When pos_label=None, if y_true is in {-1, 1} or {0, 1}, pos_label is set to 1, otherwise an error will be raised.

* **sample_weight** [array-like of shape (n_samples,), default=None] Sample weights.
**drop_intermediate** [bool, default=True] Whether to drop some suboptimal thresholds which would not appear on a plotted ROC curve. This is useful in order to create lighter ROC curves.

New in version 0.17: parameter `drop_intermediate`.

**Returns**

- **fpr** [ndarray of shape (>2,)] Increasing false positive rates such that element $i$ is the false positive rate of predictions with score $\geq$ `thresholds[i]`.
- **tpr** [ndarray of shape (>2,)] Increasing true positive rates such that element $i$ is the true positive rate of predictions with score $\geq$ `thresholds[i]`.
- **thresholds** [ndarray of 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:**

- `plot_roc_curve` Plot Receiver operating characteristic (ROC) curve.
- `RocCurveDisplay` ROC Curve visualization.
- `det_curve` Compute error rates for different probability thresholds.
- `roc_auc_score` Compute the area under the ROC curve.

**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**

[1], [2]

**Examples**

```python
>>> 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. , 0.5, 0.5, 1. ])
>>> tpr
array([0. , 0.5, 0.5, 1. , 1. ])
>>> thresholds
array([1.8 , 0.8 , 0.4 , 0.35, 0.1 ])```

oddt.metrics.roc_auc(y_true, y_score, pos_label=None, ascending_score=True)

Computes ROC AUC score

**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, 1] Positive label of samples (if other than 1)

**ascending_score** [bool, True] Indicates if your score is ascending. Ascending score increases with decreasing activity. In other words it ascends on ranking list (where actives are on top).

**Returns**

**roc_auc** [float] ROC AUC in range 0:1

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

Computes area under semi-log ROC.

**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, 1] Positive label of samples (if other than 1)

**ascending_score** [bool, True] Indicates if your score is ascending. Ascending score increases with decreasing activity. In other words it ascends on ranking list (where actives are on top).

**log_min** [float, 0.001] Minimum value for estimating AUC. Lower values will be clipped for numerical stability.

**log_max** [float, 1.0] Maximum value for estimating AUC. Higher values will be ignored.

**Returns**

**auc** [float] semi-log ROC AUC

### 5.1.7 oddt.pandas module

### 5.1.8 oddt.shape module

```python
oddt.shape.common_usr(molecule, ctd=None, cst=None, fct=None, ftf=None, atoms_type=None)
```

Function used in USR and USRCAT function

**Parameters**

**molecule** [oddt.toolkit.Molecule] Molecule to compute USR shape descriptor

**ctd** [numpy array or None, default = None] Coordinates of the molecular centroid If ‘None’, the point is calculated

**cst** [numpy array or None, default = None] Coordinates of the closest atom to the molecular centroid If ‘None’, the point is calculated

**fct** [numpy array or None, default = None] Coordinates of the farthest atom to the molecular centroid If ‘None’, the point is calculated

**ftf** [numpy array or None, default = None] Coordinates of the farthest atom to the farthest atom to the molecular centroid If ‘None’, the point is calculated

**atoms_type** [str or None, default None] Type of atoms to be selected from atom_dict If ‘None’, all atoms are used to calculate shape descriptor
Returns

```python
shape_descriptor  [numpy array, shape = (12)] Array describing shape of molecule
```

```python
oddt.shape.electroshape(mol)
```


Aside from spatial coordinates, atoms' charges are also used as the fourth dimension to describe shape of the molecule.

Parameters

```python
mol  [oddt.toolkit.Molecule] Molecule to compute Electroshape descriptor
```

Returns

```python
shape_descriptor  [numpy array, shape = (15)] Array describing shape of molecule
```

```python
oddt.shape.usr(molecule)
```


Parameters

```python
molecule  [oddt.toolkit.Molecule] Molecule to compute USR shape descriptor
```

Returns

```python
shape_descriptor  [numpy array, shape = (12)] Array describing shape of molecule
```

```python
oddt.shape.usr_cat(molecule)
```


Parameters

```python
molecule  [oddt.toolkit.Molecule] Molecule to compute USRCAT shape descriptor
```

Returns

```python
shape_descriptor  [numpy array, shape = (60)] Array describing shape of molecule
```

```python
oddt.shape.usr_similarity(mol1_shape, mol2_shape, ow=1.0, hw=1.0, rw=1.0, aw=1.0, dw=1.0)
```

Computes similarity between molecules

Parameters

```python
mol1_shape  [numpy array] USR shape descriptor
```

```python
mol2_shape  [numpy array] USR shape descriptor
```

```python
ow  [float (default = 1.0)] Scaling factor for all atoms Only used for USRCAT, ignored for other types
```

```python
hw  [float (default = 1.0)] Scaling factor for hydrophobic atoms Only used for USRCAT, ignored for other types
```

```python
rw  [float (default = 1.0)] Scaling factor for aromatic atoms Only used for USRCAT, ignored for other types
```

```python
aw  [float (default = 1.0)] Scaling factor for acceptors Only used for USRCAT, ignored for other types
```
\[ dw \] [float (default = 1.)] Scaling factor for donors Only used for USRCAT, ignored for other types

Returns

\[ similarity \] [float from 0 to 1] Similarity between shapes of molecules, 1 indicates identical molecules

5.1.9 oddt.spatial module

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

\texttt{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

\texttt{oddt.spatial.angle_2v}(v1, v2)

Returns an angle between two vecors. 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

\texttt{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

\texttt{oddt.spatial.distance}(x, y)

Computes distance between each pair of points from x and y.

Parameters

\[ x \] [numpy arrays, shape = \[n_x, 3\]] Array of poinds in 3D

\[ y \] [numpy arrays, shape = \[n_y, 3\]] Array of poinds in 3D

Returns

\[ dist\_matrix \] [numpy arrays, shape = \[n_x, n_y\]] Distance matrix

\texttt{oddt.spatial.rmsd}(ref, mol, ignore_h=True, method=None, normalize=False)

Computes root mean square deviation (RMSD) between two molecules (including or excluding Hydrogens). No symmetry checks are performed.
Parameters

ref [oddt.toolkit.Molecule object] Reference molecule for the RMSD calculation

col [oddt.toolkit.Molecule object] Query molecule for RMSD calculation

ignore_h [bool (default=False)] Flag indicating to ignore Hydrogen atoms while performing RMSD calculation. This toggle works only with ‘hungarian’ method and without sorting (method=None).

method [str (default=None)] The method to be used for atom assignment between ref and mol. None means that direct matching is applied, which is the default behavior. Available methods:

- canonize - match heavy atoms using canonical ordering (it forces ignoring H’s)
- hungarian - minimize RMSD using Hungarian algorithm
- min_symmetry - makes multiple molecule-molecule matches and finds minimal RMSD (the slowest). Hydrogens are ignored.

normalize [bool (default=False)] Normalize RMSD by square root of rot. bonds

Returns

rmsd [float] RMSD between two molecules

oddt.spatial.rotate (coords, alpha, beta, gamma)

Rotate coords by certain angle in X, Y, Z. Angles are specified in radians.

Parameters

coords [numpy arrays, shape = [n_points, 3]] Coordinates in 3-dimensional space.

alpha, beta, gamma: float Angles to rotate the coordinates along X, Y and Z axis. Angles are specified in radians.

Returns

new_coords [numpy arrays, shape = [n_points, 3]] Rotated coordinates in 3-dimensional space.

5.1.10 oddt.surface module

This module generates and does computation with molecular surfaces.

oddt.surface.find_surface_residues (molecule, max_dist=None, scaling=1.0)

Finds residues close to the molecular surface using generate_surface_marching_cubes. Ignores hydrogens and waters present in the molecule.

Parameters


max_dist [array_like, numeric or None (default = None)] Maximum distance from the surface where residues would still be considered close. If None, compares distances to radii of respective atoms.

scaling [float (default = 1.0)] Expands the grid in which computation is done by generate_surface_marching_cubes by a factor of scaling. Results in a more accurate representation of the surface, and therefore more accurate computation of distances but increases computation time.

Returns
atom_dict [numpy array] An atom_dict containing only the surface residues from the original molecule.

oddt.surface.generate_surface_marching_cubes(molecule, remove_hoh=False, scaling=1.0, probe_radius=1.4)

Generates a molecular surface mesh using the marching_cubes method from scikit-image. Ignores hydrogens present in the molecule.

Parameters

- **molecule** [oddt.toolkit.Molecule object] Molecule for which the surface will be generated.
- **remove_hoh** [bool (default = False)] If True, remove waters from the molecule before generating the surface. Requires molecule.protein to be set to True.
- **scaling** [float (default = 1.0)] Expands the grid in which computation is done by a factor of scaling. Results in a more accurate representation of the surface, but increases computation time.
- **probe_radius** [float (default = 1.4)] Radius of a ball used to patch up holes inside the molecule resulting from some molecular distances being larger (usually in protein). Basically reduces the surface to one accessible by other molecules of radius smaller than probe_radius.

Returns

- **verts** [numpy array] Spatial coordinates for mesh vertices.
- **faces** [numpy array] Faces are defined by referencing vertices from verts.

5.1.11 oddt.utils module

Common utilities for ODDT

oddt.utils.check_molecule(mol, force_protein=False, force_coords=False, non_zero_atoms=False)

Universal validator of molecule objects. Usage of positional arguments is allowed only for molecule object, otherwise it is prohibited (i.e. the order of arguments will change). Desired properties of molecule are validated based on specified arguments. By default only the object type is checked. In case of discrepancy to the specification a ValueError is raised with appropriate message.

New in version 0.6.

Parameters

- **mol**: oddt.toolkit.Molecule object Object to verify.
- **force_protein**: bool (default=False) Force the molecule to be marked as protein (mol.protein).
- **force_coords**: bool (default=False) Force the molecule to have non-zero coordinates.
- **non_zero_atoms**: bool (default=False) Check if molecule has at least one atom.

oddt.utils.chunker(iterable, chunksize=100)

Generate chunks from a generator object. If iterable is passed which is not a generator it will be converted to one with iter().

New in version 0.6.

oddt.utils.compose_iter(iterable, funcs)

Chain functions and apply them to iterable, by exhausting the iterable. Functions are executed in the order from funcs.

New in version 0.6.
oddt.utils.is_molecule(obj)
    Check whether an object is an oddt.toolkits.{rdk,ob}.Molecule instance.
    New in version 0.6.

oddt.utils.is_openbabel_molecule(obj)
    Check whether an object is an oddt.toolkits.ob.Molecule instance.
    New in version 0.6.

oddt.utils.is_rdkit_molecule(obj)
    Check whether an object is an oddt.toolkits.rdk.Molecule instance.
    New in version 0.6.

oddt.utils.method_caller(obj, methodname, *args, **kwargs)
    Helper function to workaround Python 2 pickle limitations to parallelize methods and generator objects.

5.1.12 oddt.virtualscreening module

ODDT pipeline framework for virtual screening

class oddt.virtualscreening.virtualscreening(n_cpu=-1, verbose=False, chunk_size=100)
    Bases: object

    Virtual Screening pipeline stack

    Parameters

    n_cpu: int (default=-1) The number of parallel processors to use
    verbose: bool (default=False) Verbosity flag for some methods

    Methods

    apply_filter(expression[, soft_fail])
        Filtering method, can use raw expressions (strings to be evaled in if statement, can use oddt.toolkit.Molecule methods, eg. mol.molwt < 500) Currently supported presets:
        • Lipinski Rule of 5 (‘ro5’ or ‘l5’)
        • Fragment Rule of 3 (‘ro3’)
        • PAINS filter (‘pains’).
    dock(engine, protein, *args, **kwargs)
        Docking procedure.
    fetch()
        A method to exhaust the pipeline.
    load_ligands(fmt, ligands_file, **kwargs)
        Loads file with ligands.
    score(function[, protein])
        Scoring procedure compatible with any scoring function implemented in ODDT and other pickled SFs which are subclasses of oddt.scoring.scorer.
    similarity(method, query[, cutoff, protein])
        Similarity filter. Supported structural methods:
    write(fmt, filename[, csv_filename])
        Outputs molecules to a file
    write_csv(csv_filename[, fields, keep_pipe])
        Outputs molecules to a csv file

    apply_filter(expression, soft_fail=0)
        Filtering method, can use raw expressions (strings to be evaled in if statement, can use oddt.toolkit.Molecule methods, eg. mol.molwt < 500) Currently supported presets:
        • Lipinski Rule of 5 (‘ro5’ or ‘l5’)
- Fragment Rule of 3 (‘ro3’)
- PAINS filter (‘pains’)

**Parameters**

*expression: string or list of strings*  Expresion(s) to be used while filtering.

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

### dock (engine, protein, *args, **kwargs)
Docking procedure.

**Parameters**

*engine: string*  Which docking engine to use.

**Notes**

Additional parameters are passed directly to the engine. Following docking engines are supported:

1. Audodock Vina (‘engine="autodock_vina"’), see oddt.docking.autodock_vina.

### fetch()
A method to exhaust the pipeline. Itself it is lazy (a generator)

### load_ligands (fmt, ligands_file, **kwargs)
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=None, *args, **kwargs)
Scoring procedure compatible with any scoring function implemented in ODDT and other pickled SFs which are subclasses of oddt.scoring.scorer.

**Parameters**

*function: string*  Which scoring function to use.

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

**Notes**

Additional parameters are passed directly to the scoring function.

### similarity (method, query, cutoff=0.9, protein=None)
Similarity filter. Supported structural methods:

- ift: interaction fingerprints
- sift: simple interaction fingerprints
- usr: Ultrafast Shape recognition
- usr_cat: Ultrafast Shape recognition, Credo Atom Types
- electroshape: Electroshape, an USR method including partial charges

---

5.1. oddt package 73
**Parameters**

- **method**: string  
  Similarity method used to compare molecules. Available methods: *ifp* - interaction fingerprint (requires a receptor) *sifp* - simple interaction fingerprint (requires a receptor) *usr* - Ultrafast Shape Recognition *usr_cat* - USR, with CREDO atom types *electroshape* - Electroshape, USR with moments representing partial charge

- **query**: oddt.toolkit.Molecule or list of oddt.toolkit.Molecule  
  Query molecules to compare the pipeline to.

- **cutoff**: float  
  Similarity cutoff for filtering molecules. Any similarity lower than it will be filtered out.

- **protein**: oddt.toolkit.Molecule (default = None)  
  Protein for underlying method. By default it’s empty, but structural fingerprints need one.

```python
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

```python
write_csv(csv_filename, fields=None, keep_pipe=False, **kwargs)
```

Outputs molecules to a csv file

- **Parameters**

  - **csv_filename**: string  
    Optional path to a CSV file

  - **fields**: list (default None)  
    List of fields to save in CSV file

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

### 5.1.13 Module contents

**Open Drug Discovery Toolkit**

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

**Attributes**

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

To be announced.
CHAPTER
SEVEN

DOCUMENTATION INDICES AND TABLES

• genindex
• modindex
• search


oddt, 74
oddt.datasets, 55
oddt.docking, 19
oddt.docking.AutodockVina, 15
oddt.docking.internal, 17
oddt.fingerprints, 56
oddt.interactions, 59
oddt.metrics, 63
oddt.scoring, 32
oddt.scoring.descriptors, 22
oddt.scoring.descriptors.binana, 21
oddt.scoring.functions, 27
oddt.scoring.functions.NNScore, 23
oddt.scoring.functions.PLECscore, 24
oddt.scoring.functions.RFScore, 26
oddt.scoring.models, 32
oddt.scoring.models.classifiers, 30
oddt.scoring.models.regressors, 31
oddt.shape, 67
oddt.spatial, 69
oddt.surface, 70
oddt.toolkits, 55
oddt.toolkits.common, 40
oddt.toolkits.extras, 40
oddt.toolkits.extras.rdkit, 39
oddt.toolkits.extras.rdkit.fixer, 35
oddt.toolkits.ob, 41
oddt.toolkits.rdk, 47
oddt.utils, 71
oddt.virtualscreening, 72
INDEX

A
acceptor_metal() (in module oddt.interactions), 59
activities() (oddt.datasets.pdbbind property), 56
AddAtomsError, 35
addh() (oddt.toolkits.ob.Molecule method), 43
addh() (oddt.toolkits.rdk.Molecule method), 49
AddMissingAtoms() (in module oddt.toolkits.extras.rdkit.fixer), 35
angle() (in module oddt.spatial), 69
angle_2v() (in module oddt.spatial), 69
apply_filter() (oddt.virtualscreening.virtualscreening method), 72
Atom (class in oddt.toolkits.ob), 41
Atom (class in oddt.toolkits.rdk), 47
atom_dict() (oddt.toolkits.ob.Molecule property), 43
atom_dict() (oddt.toolkits.rdk.Molecule property), 49
atomicnum() (oddt.toolkits.rdk.Atom property), 47
AtomListToSubMol() (in module oddt.toolkits.extras.rdkit), 39
atoms() (oddt.toolkits.ob.Bond property), 42
atoms() (oddt.toolkits.ob.Molecule property), 43
atoms() (oddt.toolkits.ob.Residue property), 45
atoms() (oddt.toolkits.rdk.Bond property), 48
atoms() (oddt.toolkits.rdk.Molecule property), 49
atoms() (oddt.toolkits.rdk.Residue property), 52
AtomStack (class in oddt.toolkits.ob), 42
AtomStack (class in oddt.toolkits.rdk), 47
auc() (in module oddt.metrics), 63
autodock_vina (class in oddt.docking), 19
autodock_vina (class in oddt.docking.AutodockVina), 15
autodock_vina_descriptor (class in oddt.scoring.descriptors), 22
bonds() (oddt.toolkits.ob.Atom property), 41
bonds() (oddt.toolkits.ob.Molecule property), 43
bonds() (oddt.toolkits.rdk.Atom property), 47
bonds() (oddt.toolkits.rdk.Molecule property), 49
BondStack (class in oddt.toolkits.ob), 42
BondStack (class in oddt.toolkits.rdk), 48
build() (oddt.scoring.descriptors.autodock_vina_descriptor method), 22
build() (oddt.scoring.descriptors.binana.binana_descriptor method), 21
build() (oddt.scoring.descriptors.close_contacts_descriptor method), 22
build() (oddt.scoring.descriptors.fingerprints method), 23
build() (oddt.scoring.descriptors.oddt_vina_descriptor method), 23
build() (oddt.scoring.ensemble_descriptor method), 32
C
calccharges() (oddt.toolkits.ob.Molecule method), 43
calccharges() (oddt.toolkits.rdk.Molecule method), 49
calcdesc() (oddt.toolkits.rdk.Molecule method), 49
calcfp() (oddt.toolkits.rdk.Molecule method), 50
canonic_order() (oddt.toolkits.ob.Molecule property), 43
canonic_order() (oddt.toolkits.rdk.Molecule property), 50
canonize_ring_path() (in module oddt.toolkits.common), 40
CASF (class in oddt.datasets), 55
chain() (oddt.toolkits.ob.Residue property), 45
chain() (oddt.toolkits.rdk.Residue property), 52
change_dihedral() (in module oddt.docking.internal), 17
charges() (oddt.toolkits.ob.Molecule property), 43
charges() (oddt.toolkits.rdk.Molecule property), 50
check_molecule() (in module oddt.utils), 71
chunker() (in module oddt.utils), 71
clean() (oddt.docking.autodock_vina method), 19
PathFromAtomList () (in module oddt.toolkits.extras.rdkit), 40
pddbbind (class in oddt.datasets), 55
PDBQTAtomLines () (in module oddt.toolkits.extras.rdkit), 40
pi_cation () (in module oddt.interactions), 61
pi_metal () (in module oddt.interactions), 62
pi_stacking () (in module oddt.interactions), 62
PLEC () (in module oddt.docking.autodock_vina), 57
PLECsore (class in oddt.interactions), 27
PLECsore () (class in oddt.interactions.PLECscore), 24
pls (in module oddt.scoring.models.regressors), 31
precomputed_score () (oddt.datasets.CASF method), 55
precomputed_screening () (oddt.datasets.CASF method), 55
predict () (oddt.scoring.ensemble_model method), 33
predict () (oddt.scoring.scorer method), 33
predict_ligand () (oddt.docking.autodock_vina method), 20
predict_ligand () (oddt.docking.autodock_vina method), 16
predict_ligands () (oddt.docking.autodock_vina method), 20
predict_ligands () (oddt.docking.AutodockVina.autodock_vina method), 16
predict_ligands () (oddt.scoring.scorer method), 34
PreparedComplexes () (in module oddt.toolkits.extras.rdkit.fixer), 37
PreparedBMol () (in module oddt.toolkits.extras.rdkit.fixer), 37
PreparedBResidue () (in module oddt.toolkits.extras.rdkit.fixer), 38
protein () (oddt.toolkits.ob.Molecule property), 44
protein () (oddt.toolkits.rdk.Molecule property), 50

random_roc_log_auc () (in module oddt.metrics), 64
randomforest (in module oddt.scoring.models.classifiers), 30
randomforest (in module oddt.scoring.models.regressors), 31
raw () (oddt.toolkits.ob.Fingerprint property), 42
raw () (oddt.toolkits.rdk.Fingerprint property), 48
readfile () (in module oddt.toolkits.ob), 46
readfile () (in module oddt.toolkits.rdk), 54
readstring () (in module oddt.toolkits.rdk), 54
ReadTemplates () (in module oddt.toolkits.extras.rdkit.fixer), 38
removeh () (oddt.toolkits.ob.Molecule method), 44
removeh () (oddt.toolkits.rdk.Molecule method), 50
res_dict () (oddt.toolkits.ob.Molecule property), 44
res_dict () (oddt.toolkits.rdk.Molecule property), 50
Residue (class in oddt.toolkits.ob), 45
Residue (class in oddt.toolkits.rdk), 52
residue () (oddt.toolkits.ob.Atom property), 42
residues () (oddt.toolkits.ob.Molecule property), 44
residues () (oddt.toolkits.rdk.Molecule property), 50
ResidueStack (class in oddt.toolkits.ob), 45
ResidueStack (class in oddt.toolkits.rdk), 53
rfscore (class in oddt.scoring.functions), 29
rfscore (class in oddt.scoring.functions.RFScore), 26
rie () (in module oddt.metrics), 65
ring_dict () (oddt.toolkits.ob.Molecule property), 44
ring_dict () (oddt.toolkits.rdk.Molecule property), 50
rmsd () (in module oddt.spatial), 69
rmse () (in module oddt.metrics), 65
roc () (in module oddt.metrics), 65
roc_auc () (in module oddt.metrics), 66
roc_log_auc () (in module oddt.metrics), 67

S
salt_bridge_plus_minus () (in module oddt.interactions), 62
salt_bridges () (in module oddt.interactions), 63
SanitizeError, 38
save () (oddt.scoring.scorer method), 34
score () (oddt.docking.AutodockVina.autodock_vina method), 20
score () (oddt.docking.AutodockVina.autodock_vina method), 16
score () (oddt.docking.internal.vina_docking method), 18
score () (oddt.scoring.ensemble_model method), 33
score () (oddt.scoring.scorer method), 34
score () (oddt.virtualscreening.virtualscreening method), 73
score_inter () (oddt.docking.internal.vina_docking method), 18
score_intra () (oddt.docking.internal.vina_docking method), 18
score_total () (oddt.docking.internal.vina_docking method), 18
scorer (class in oddt.scoring), 33
set_box () (oddt.docking.internal.vina_docking method), 18
set_coords () (oddt.docking.internal.vina_docking method), 18
set_ligand () (oddt.docking.internal.vina_docking method), 18
set_protein () (oddt.docking.autodock_vina method), 20