Neural Fingerprint (nfp)

Peter St. John

Apr 27, 2022

CONTENTS

NFP can be installed with pip into a suitable python environment with

```
pip install nfp
```

The only dependency that is not pip-installable is rdkit, which can be installed via conda-forge.

CHAPTER

ONE

TUTORIALS

1.1 Tokenizer usage examples

The `Tokenizer` class allows transforming arbitrary inputs into integer classes

[1]:
```
from nfp.preprocessing import Tokenizer
```

[2]:
```
tokenizer = Tokenizer()
tokenizer.train = True
```

The `0` and `1` classes are reserved for the `<MASK>` and missing labels, respectively.

[3]:
```
[tokenizer(item) for item in ['A', 'B', 'C', 'A']]
```

[3]:
```
[2, 3, 4, 2]
```

When train is set to `False`, unknown items are assigned the missing label

[4]:
```
tokenizer.train = False
[tokenizer(item) for item in ['A', 'D']]
```

[4]:
```
[2, 1]
```

The total number of seen classes is available from the `num_classes` property, useful to initializing embedding layer weights.

[5]:
```
tokenizer.num_classes
```

[5]:
```
4
```

CHAPTER

TWO

NFP

Exceptions

MissingDependencyException	

2.1 nfp.MissingDependencyException

exception MissingDependencyException

nfp.frameworks	
nfp.layers	Keras layers that handle the node, edge, and connectivity features returned by the preprocessor classes.
nfp.models	Tensorflow loss functions that accept masked true values (with np.nan)
nfp.preprocessing	Utilities for converting graphs into sets of tensors for machine learning models

2.2 nfp.frameworks

2.3 nfp.layers

Keras layers that handle the node, edge, and connectivity features returned by the preprocessor classes.

Functions

batched_segment_op	Flattens data and segment_ids containing a batch dimension for tf.math.segment* operations.

2.3.1 nfp.layers.batched_segment_op

batched_segment_op(*data*, *segment_ids*, *num_segments*, *data_mask=None*, *reduction='sum'*)

Flattens data and segment_ids containing a batch dimension for tf.math.segment* operations. Includes support for masking.

Parameters

- **data** – tensor of shape [B, L, F], where B is the batch size, L is the length, and F is a feature dimension
- **segment_ids** – tensor of shape [B, L] containing up to N segments
- **num_segments** – N, integer
- **data_mask** – boolean tensor of shape [B, L] masking the input data
- **reduction** – string for specific tf.math.unsorted_segment_* function

Classes

ConcatDense	Layer to combine the concatenation and two dense layers.
Gather	
RBFExpansion	Layer to calculate radial basis function 'embeddings' for a continuous input variable.
Reduce	
Tile	

2.3.2 nfp.layers.ConcatDense

class ConcatDense(**args*, ***kwargs*)

Bases: `tensorflow.keras.layers.Layer`

Layer to combine the concatenation and two dense layers. Just useful as a common operation in the graph layers

Methods

`build`
`call`
`compute_mask`

Parameters

- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *Any*

`__call__`(**args*, ***kwargs*)

Call self as a function.

Parameters

- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *Any*

2.3.3 nfp.layers.Gather

`class Gather`(**args*, ***kwargs*)

Bases: `tensorflow.keras.layers.Layer`

Methods

`call`

Parameters

- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *Any*

`__call__`(**args*, ***kwargs*)

Call self as a function.

Parameters

- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *Any*

2.3.4 nfp.layers.RBFExpansion

`class RBFExpansion`(**args*, ***kwargs*)

Bases: `tensorflow.keras.layers.Layer`

Layer to calculate radial basis function 'embeddings' for a continuous input variable. The width and location of each bin can be optionally trained. Essentially equivalent to a 1-hot embedding for a continuous variable.

Parameters

- **dimension** (*The total number of distance bins*) –
- **init_gap** (*The initial width of each gaussian distribution*) –

- **init_max_distance** (*the initial maximum value of the continuous variable*) –
- **trainable** (*Whether the centers and gap parameters should be added as trainable*) – NN parameters.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *Any*

Methods

build
call
compute_mask
get_config

__call__(**args*, ***kwargs*)

Call self as a function.

Parameters

- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *Any*

2.3.5 nfp.layers.Reduce

class Reduce(**args*, ***kwargs*)

Bases: tensorflow.keras.layers.Layer

Methods

call
compute_output_shape
get_config

Parameters

- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *Any*

__call__(**args*, ***kwargs*)

Call self as a function.

Parameters

- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *Any*

2.3.6 nfp.layers.Tile

class Tile(**args*, ***kwargs*)

Bases: tensorflow.keras.layers.Layer

Methods

call
compute_mask

Parameters

- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *Any*

__call__(**args*, ***kwargs*)

Call self as a function.

Parameters

- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *Any*

2.4 nfp.models

Tensorflow loss functions that accept masked true values (with np.nan)

nfp.models.losses	Keras loss functions will choke on NaN inputs, which we'll often have for unspecified inputs.

2.4.1 nfp.models.losses

Keras loss functions will choke on NaN inputs, which we'll often have for unspecified inputs. These are just a couple simple loss functions that mask NaN values in both the test and predicted tensors when computing the cost.

Functions

nfp.models.losses.masked_log_cosh

masked_log_cosh(*y_true*, *y_pred*)

nfp.models.losses.masked_mean_absolute_error

masked_mean_absolute_error(*y_true*, *y_pred*)

nfp.models.losses.masked_mean_squared_error

masked_mean_squared_error(*y_true*, *y_pred*)

2.5 nfp.preprocessing

Utilities for converting graphs into sets of tensors for machine learning models

2.5.1 nfp.preprocessing.crystal_preprocessor

Classes

PymatgenPreprocessor

nfp.preprocessing.crystal_preprocessor.PymatgenPreprocessor

class PymatgenPreprocessor(*radius=None*, *num_neighbors=12*, ***kwargs*)

Bases: *nfp.preprocessing.preprocessor.PreprocessorMultiGraph*

Methods

construct_feature_matrices	Deprecated since version 0.3.0.
create_nx_graph	crystal should be a pymatgen.core.Structure object.
from_json	Set's the class's data with attributes taken from the save file
get_connectivity	Get the graph connectivity from the networkx graph
get_edge_features	Given a list of edge features from the nx.Graph, processes and concatenates them to an array.
get_graph_features	Process the nx.graph features into a dictionary of arrays.
get_node_features	Given a list of node features from the nx.Graph, processes and concatenates them to an array.
site_features	
to_json	Serialize the classes's data to a json file

Attributes

output_signature	
padding_values	
site_classes	
tfrecord_features	Useful feature for storing preprocessed outputs in tfrecord files

create_nx_graph(*crystal*, ***kwargs*)

crystal should be a pymatgen.core.Structure object.

Return type networkx.classes.multidigraph.MultiDiGraph

`get_edge_features`(*edge_data*, *max_num_edges*)

Given a list of edge features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **edge_data** (*list*) – A list of edge data generated by *nx_graph.edges(data=True)*
- **max_num_edges** – If desired, this function should pad to a maximum number of edges passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all edges in the graph.

Return type Dict[str, np.ndarray]

`get_node_features`(*node_data*, *max_num_nodes*)

Given a list of node features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **node_data** (*list*) – A list of edge data generated by *nx_graph.nodes(data=True)*
- **max_num_nodes** – If desired, this function should pad to a maximum number of nodes passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all nodes in the graph.

Return type Dict[str, np.ndarray]

`get_graph_features`(*graph_data*)

Process the nx.graph features into a dictionary of arrays.

Parameters **graph_data** (*dict*) – A dictionary of graph data generated by *nx_graph.graph*

Returns a dictionary of features for the graph

Return type Dict[str, np.ndarray]

`property tfrecord_features: Dict[str, tensorflow.io.FixedLenFeature]`

Useful feature for storing preprocessed outputs in tfrecord files

`__call__`(*structure*, **args*, *train=False*, *max_num_nodes=None*, *max_num_edges=None*, ***kwargs*)

Convert an input graph structure into a featurized set of node, edge, and graph-level features.

Parameters

- **structure** (*Any*) – An input graph structure (i.e., molecule, crystal, etc.)
- **train** (*bool*) – A training flag passed to *Tokenizer* member attributes
- **max_num_nodes** (*Optional[int]*) – A size attribute passed to *get_node_features*, defaults to the number of nodes in the current graph
- **max_num_edges** (*Optional[int]*) – A size attribute passed to *get_edge_features*, defaults to the number of edges in the current graph
- **kwargs** – Additional features or parameters passed to *construct_nx_graph*

Returns A dictionary of key, array pairs as a single sample.

Return type Dict[str, np.ndarray]

construct_feature_matrices(**args*, *train=False*, ***kwargs*)

Deprecated since version 0.3.0: *construct_feature_matrices* will be removed in 0.4.0, use *__call__* instead

Return type *Dict*[str, numpy.ndarray]

from_json(*filename*)

Set's the class's data with attributes taken from the save file

Parameters **filename** (*str*) –

Return type None

static get_connectivity(*graph*, *max_num_edges*)

Get the graph connectivity from the networkx graph

Parameters

- **graph** (*networkx.classes.digraph.DiGraph*) – The input graph
- **max_num_edges** (*int*) – len(graph.edges), or the specified maximum number of graph edges

Returns A dictionary of with the single 'connectivity' key, containing an (n,2) array of (node_index, node_index) pairs indicating the start and end nodes for each edge.

Return type Dict[str, np.ndarray]

to_json(*filename*)

Serialize the classes's data to a json file

Parameters **filename** (*str*) –

Return type None

2.5.2 nfp.preprocessing.features

Functions

atom_features_v1	Return an integer hash representing the atom type
atom_features_v2	
bond_features_v1	Return an integer hash representing the bond type.
bond_features_v2	
bond_features_v3	
bond_features_wbo	
get_ring_size	

nfp.preprocessing.features.atom_features_v1

atom_features_v1(*atom*)

Return an integer hash representing the atom type

nfp.preprocessing.features.atom_features_v2

atom_features_v2(*atom*)

nfp.preprocessing.features.bond_features_v1

bond_features_v1(*bond*, ***kwargs*)

Return an integer hash representing the bond type.

flipped [bool] Only valid for 'v3' version, whether to swap the begin and end atom types

nfp.preprocessing.features.bond_features_v2

bond_features_v2(*bond*, ***kwargs*)

nfp.preprocessing.features.bond_features_v3

bond_features_v3(*bond*, *flipped=False*)

nfp.preprocessing.features.bond_features_wbo

bond_features_wbo(*start_atom*, *end_atom*, *bondatoms*)

nfp.preprocessing.features.get_ring_size

get_ring_size(*obj*, *max_size=12*)

2.5.3 nfp.preprocessing.mol_preprocessor

Classes

BondIndexPreprocessor

MolPreprocessor

SmilesBondIndexPreprocessor

SmilesPreprocessor

nfp.preprocessing.mol_preprocessor.BondIndexPreprocessor

class BondIndexPreprocessor(*atom_features=None*, *bond_features=None*, ***kwargs*)

Bases: *nfp.preprocessing.mol_preprocessor.MolPreprocessor*

Methods

construct_feature_matrices	Deprecated since version 0.3.0.
create_nx_graph	Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.
from_json	Set's the class's data with attributes taken from the save file
get_connectivity	Get the graph connectivity from the networkx graph
get_edge_features	Given a list of edge features from the nx.Graph, processes and concatenates them to an array.
get_graph_features	Process the nx.graph features into a dictionary of arrays.
get_node_features	Given a list of node features from the nx.Graph, processes and concatenates them to an array.
to_json	Serialize the classes's data to a json file

Attributes

atom_classes	The number of atom types found (includes the 0 null-atom type)
bond_classes	The number of bond types found (includes the 0 null-bond type)
output_signature	
padding_values	Defaults to zero for each output
tfrecord_features	For loading preprocessed inputs from a tf records file

Parameters

- **atom_features** (*Optional[Callable[[rdkit.Chem.Atom], Hashable]]*) –
- **bond_features** (*Optional[Callable[[rdkit.Chem.Bond], Hashable]]*) –

Return type None

get_edge_features(*edge_data*, *max_num_edges*)

Given a list of edge features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **edge_data** (*list*) – A list of edge data generated by *nx_graph.edges(data=True)*
- **max_num_edges** – If desired, this function should pad to a maximum number of edges passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all edges in the graph.

Return type Dict[str, np.ndarray]

`__call__`(*structure*, **args*, *train=False*, *max_num_nodes=None*, *max_num_edges=None*, ***kwargs*)

Convert an input graph structure into a featurized set of node, edge, and graph-level features.

Parameters

- **structure** (*Any*) – An input graph structure (i.e., molecule, crystal, etc.)
- **train** (*bool*) – A training flag passed to *Tokenizer* member attributes
- **max_num_nodes** (*Optional[int]*) – A size attribute passed to *get_node_features*, defaults to the number of nodes in the current graph
- **max_num_edges** (*Optional[int]*) – A size attribute passed to *get_edge_features*, defaults to the number of edges in the current graph
- **kwargs** – Additional features or parameters passed to *construct_nx_graph*

Returns A dictionary of key, array pairs as a single sample.

Return type Dict[str, np.ndarray]

`property atom_classes: int`

The number of atom types found (includes the 0 null-atom type)

`property bond_classes: int`

The number of bond types found (includes the 0 null-bond type)

`construct_feature_matrices`(**args*, *train=False*, ***kwargs*)

Deprecated since version 0.3.0: *construct_feature_matrices* will be removed in 0.4.0, use *__call__* instead

Return type *Dict*[str, numpy.ndarray]

`create_nx_graph`(*mol*, ***kwargs*)

Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.

Parameters

- **structure** – Any input graph object
- **kwargs** – keyword arguments passed from *__call__*, useful for specifying additional features in addition to the graph object.
- **mol** (*rdkit.Chem.Mol*) –

Returns A networkx graph with the node, edge, and graph features set

Return type nx.DiGraph

`from_json`(*filename*)

Set's the class's data with attributes taken from the save file

Parameters **filename** (*str*) –

Return type None

static get_connectivity(*graph*, *max_num_edges*)

Get the graph connectivity from the networkx graph

Parameters

- **graph** (*networkx.classes.digraph.DiGraph*) – The input graph
- **max_num_edges** (*int*) – len(graph.edges), or the specified maximum number of graph edges

Returns A dictionary of with the single 'connectivity' key, containing an (n,2) array of (node_index, node_index) pairs indicating the start and end nodes for each edge.

Return type Dict[str, np.ndarray]

get_graph_features(*graph_data*)

Process the nx.graph features into a dictionary of arrays.

Parameters **graph_data** (*dict*) – A dictionary of graph data generated by *nx_graph.graph*

Returns a dictionary of features for the graph

Return type Dict[str, np.ndarray]

get_node_features(*node_data*, *max_num_nodes*)

Given a list of node features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **node_data** (*list*) – A list of edge data generated by *nx_graph.nodes(data=True)*
- **max_num_nodes** – If desired, this function should pad to a maximum number of nodes passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all nodes in the graph.

Return type Dict[str, np.ndarray]

property padding_values: Dict[str, tensorflow.constant]

Defaults to zero for each output

property tfrecord_features: Dict[str, tensorflow.io.FixedLenFeature]

For loading preprocessed inputs from a tf records file

to_json(*filename*)

Serialize the classes's data to a json file

Parameters **filename** (*str*) –

Return type None

nfp.preprocessing.mol_preprocessor.MolPreprocessor

class MolPreprocessor(*atom_features=None*, *bond_features=None*, ***kwargs*)

Bases: *nfp.preprocessing.preprocessor.Preprocessor*

Methods

construct_feature_matrices	Deprecated since version 0.3.0.
create_nx_graph	Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.
from_json	Set's the class's data with attributes taken from the save file
get_connectivity	Get the graph connectivity from the networkx graph
get_edge_features	Given a list of edge features from the nx.Graph, processes and concatenates them to an array.
get_graph_features	Process the nx.graph features into a dictionary of arrays.
get_node_features	Given a list of node features from the nx.Graph, processes and concatenates them to an array.
to_json	Serialize the classes's data to a json file

Attributes

atom_classes	The number of atom types found (includes the 0 null-atom type)
bond_classes	The number of bond types found (includes the 0 null-bond type)
output_signature	
padding_values	Defaults to zero for each output
tfrecord_features	For loading preprocessed inputs from a tf records file

Parameters

- **atom_features** (*Optional[Callable[[rdkit.Chem.Atom], Hashable]]*) –
- **bond_features** (*Optional[Callable[[rdkit.Chem.Bond], Hashable]]*) –

Return type None

create_nx_graph(*mol*, ***kwargs*)

Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.

Parameters

- **structure** – Any input graph object
- **kwargs** – keyword arguments passed from *__call__*, useful for specifying additional features in addition to the graph object.
- **mol** (*rdkit.Chem.Mol*) –

Returns A networkx graph with the node, edge, and graph features set

Return type nx.DiGraph

get_edge_features(*edge_data*, *max_num_edges*)

Given a list of edge features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **edge_data** (*list*) – A list of edge data generated by *nx_graph.edges(data=True)*
- **max_num_edges** – If desired, this function should pad to a maximum number of edges passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all edges in the graph.

Return type Dict[str, np.ndarray]

get_node_features(*node_data*, *max_num_nodes*)

Given a list of node features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **node_data** (*list*) – A list of edge data generated by *nx_graph.nodes(data=True)*
- **max_num_nodes** – If desired, this function should pad to a maximum number of nodes passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all nodes in the graph.

Return type Dict[str, np.ndarray]

get_graph_features(*graph_data*)

Process the nx.graph features into a dictionary of arrays.

Parameters **graph_data** (*dict*) – A dictionary of graph data generated by *nx_graph.graph*

Returns a dictionary of features for the graph

Return type Dict[str, np.ndarray]

property atom_classes: int

The number of atom types found (includes the 0 null-atom type)

property bond_classes: int

The number of bond types found (includes the 0 null-bond type)

property padding_values: Dict[str, tensorflow.constant]

Defaults to zero for each output

property tfrecord_features: Dict[str, tensorflow.io.FixedLenFeature]

For loading preprocessed inputs from a tf records file

__call__(*structure*, **args*, *train=False*, *max_num_nodes=None*, *max_num_edges=None*, ***kwargs*)

Convert an input graph structure into a featurized set of node, edge, and graph-level features.

Parameters

- **structure** (*Any*) – An input graph structure (i.e., molecule, crystal, etc.)
- **train** (*bool*) – A training flag passed to *Tokenizer* member attributes
- **max_num_nodes** (*Optional[int]*) – A size attribute passed to *get_node_features*, defaults to the number of nodes in the current graph

- **max_num_edges** (*Optional[int]*) – A size attribute passed to *get_edge_features*, defaults to the number of edges in the current graph
- **kwargs** – Additional features or parameters passed to *construct_nx_graph*

Returns A dictionary of key, array pairs as a single sample.

Return type Dict[str, np.ndarray]

construct_feature_matrices(**args*, *train=False*, ***kwargs*)

Deprecated since version 0.3.0: *construct_feature_matrices* will be removed in 0.4.0, use *__call__* instead

Return type *Dict*[str, numpy.ndarray]

from_json(*filename*)

Set's the class's data with attributes taken from the save file

Parameters **filename** (*str*) –

Return type None

static get_connectivity(*graph*, *max_num_edges*)

Get the graph connectivity from the networkx graph

Parameters

- **graph** (*networkx.classes.digraph.DiGraph*) – The input graph
- **max_num_edges** (*int*) – len(graph.edges), or the specified maximum number of graph edges

Returns A dictionary of with the single 'connectivity' key, containing an (n,2) array of (node_index, node_index) pairs indicating the start and end nodes for each edge.

Return type Dict[str, np.ndarray]

to_json(*filename*)

Serialize the classes's data to a json file

Parameters **filename** (*str*) –

Return type None

nfp.preprocessing.mol_preprocessor.SmilesBondIndexPreprocessor

class SmilesBondIndexPreprocessor(**args*, *explicit_hs=True*, ***kwargs*)

Bases: *nfp.preprocessing.mol_preprocessor.SmilesPreprocessor*, *nfp.preprocessing.mol_preprocessor.BondIndexPreprocessor*

Methods

construct_feature_matrices	Deprecated since version 0.3.0.
create_nx_graph	Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.
from_json	Set's the class's data with attributes taken from the save file
get_connectivity	Get the graph connectivity from the networkx graph
get_edge_features	Given a list of edge features from the nx.Graph, processes and concatenates them to an array.
get_graph_features	Process the nx.graph features into a dictionary of arrays.
get_node_features	Given a list of node features from the nx.Graph, processes and concatenates them to an array.
to_json	Serialize the classes's data to a json file

Attributes

atom_classes	The number of atom types found (includes the 0 null-atom type)
bond_classes	The number of bond types found (includes the 0 null-bond type)
output_signature	
padding_values	Defaults to zero for each output
tfrecord_features	For loading preprocessed inputs from a tf records file

Parameters **explicit_hs** (*bool*) –

__call__(*structure*, **args*, *train=False*, *max_num_nodes=None*, *max_num_edges=None*, ***kwargs*)

Convert an input graph structure into a featurized set of node, edge, and graph-level features.

Parameters

- **structure** (*Any*) – An input graph structure (i.e., molecule, crystal, etc.)
- **train** (*bool*) – A training flag passed to *Tokenizer* member attributes
- **max_num_nodes** (*Optional[int]*) – A size attribute passed to *get_node_features*, defaults to the number of nodes in the current graph
- **max_num_edges** (*Optional[int]*) – A size attribute passed to *get_edge_features*, defaults to the number of edges in the current graph
- **kwargs** – Additional features or parameters passed to *construct_nx_graph*

Returns A dictionary of key, array pairs as a single sample.

Return type Dict[str, np.ndarray]

property atom_classes: int

The number of atom types found (includes the 0 null-atom type)

property bond_classes: int

The number of bond types found (includes the 0 null-bond type)

construct_feature_matrices(*args, train=False, **kwargs)

Deprecated since version 0.3.0: *construct_feature_matrices* will be removed in 0.4.0, use *__call__* instead

Return type *Dict*[str, numpy.ndarray]

create_nx_graph(*smiles*, **args*, ***kwargs*)

Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.

Parameters

- **structure** – Any input graph object
- **kwargs** – keyword arguments passed from *__call__*, useful for specifying additional features in addition to the graph object.
- **smiles** (*str*) –

Returns A networkx graph with the node, edge, and graph features set

Return type nx.DiGraph

from_json(*filename*)

Set's the class's data with attributes taken from the save file

Parameters **filename** (*str*) –

Return type None

static get_connectivity(*graph*, *max_num_edges*)

Get the graph connectivity from the networkx graph

Parameters

- **graph** (*networkx.classes.digraph.DiGraph*) – The input graph
- **max_num_edges** (*int*) – len(graph.edges), or the specified maximum number of graph edges

Returns A dictionary of with the single 'connectivity' key, containing an (n,2) array of (node_index, node_index) pairs indicating the start and end nodes for each edge.

Return type Dict[str, np.ndarray]

get_edge_features(*edge_data*, *max_num_edges*)

Given a list of edge features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **edge_data** (*list*) – A list of edge data generated by *nx_graph.edges(data=True)*
- **max_num_edges** – If desired, this function should pad to a maximum number of edges passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all edges in the graph.

Return type Dict[str, np.ndarray]

get_graph_features(*graph_data*)

Process the nx.graph features into a dictionary of arrays.

Parameters **graph_data** (*dict*) – A dictionary of graph data generated by *nx_graph.graph*

Returns a dictionary of features for the graph

Return type Dict[str, np.ndarray]

get_node_features(*node_data*, *max_num_nodes*)

Given a list of node features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **node_data** (*list*) – A list of edge data generated by *nx_graph.nodes(data=True)*
- **max_num_nodes** – If desired, this function should pad to a maximum number of nodes passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all nodes in the graph.

Return type Dict[str, np.ndarray]

property padding_values: Dict[str, tensorflow.constant]

Defaults to zero for each output

property tfrecord_features: Dict[str, tensorflow.io.FixedLenFeature]

For loading preprocessed inputs from a tf records file

to_json(*filename*)

Serialize the classes's data to a json file

Parameters **filename** (*str*) –

Return type None

nfp.preprocessing.mol_preprocessor.SmilesPreprocessor

class SmilesPreprocessor(**args*, *explicit_hs=True*, ***kwargs*)

Bases: *nfp.preprocessing.mol_preprocessor.MolPreprocessor*

Methods

construct_feature_matrices	Deprecated since version 0.3.0.
create_nx_graph	Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.
from_json	Set's the class's data with attributes taken from the save file
get_connectivity	Get the graph connectivity from the networkx graph
get_edge_features	Given a list of edge features from the nx.Graph, processes and concatenates them to an array.
get_graph_features	Process the nx.graph features into a dictionary of arrays.
get_node_features	Given a list of node features from the nx.Graph, processes and concatenates them to an array.
to_json	Serialize the classes's data to a json file

Attributes

atom_classes	The number of atom types found (includes the 0 null-atom type)
bond_classes	The number of bond types found (includes the 0 null-bond type)
output_signature	
padding_values	Defaults to zero for each output
tfrecord_features	For loading preprocessed inputs from a tf records file

Parameters **explicit_hs** (*bool*) –

create_nx_graph(*smiles*, **args*, ***kwargs*)

Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.

Parameters

- **structure** – Any input graph object
- **kwargs** – keyword arguments passed from *__call__*, useful for specifying additional features in addition to the graph object.
- **smiles** (*str*) –

Returns A networkx graph with the node, edge, and graph features set

Return type nx.DiGraph

__call__(*structure*, **args*, *train=False*, *max_num_nodes=None*, *max_num_edges=None*, ***kwargs*)

Convert an input graph structure into a featurized set of node, edge, and graph-level features.

Parameters

- **structure** (*Any*) – An input graph structure (i.e., molecule, crystal, etc.)
- **train** (*bool*) – A training flag passed to *Tokenizer* member attributes
- **max_num_nodes** (*Optional[int]*) – A size attribute passed to *get_node_features*, defaults to the number of nodes in the current graph
- **max_num_edges** (*Optional[int]*) – A size attribute passed to *get_edge_features*, defaults to the number of edges in the current graph
- **kwargs** – Additional features or parameters passed to *construct_nx_graph*

Returns A dictionary of key, array pairs as a single sample.

Return type Dict[str, np.ndarray]

property atom_classes: int

The number of atom types found (includes the 0 null-atom type)

property bond_classes: int

The number of bond types found (includes the 0 null-bond type)

construct_feature_matrices(**args*, *train=False*, ***kwargs*)

Deprecated since version 0.3.0: *construct_feature_matrices* will be removed in 0.4.0, use *__call__* instead

Return type *Dict*[str, numpy.ndarray]

from_json(*filename*)

Set's the class's data with attributes taken from the save file

Parameters **filename** (*str*) –

Return type None

static get_connectivity(*graph*, *max_num_edges*)

Get the graph connectivity from the networkx graph

Parameters

- **graph** (*networkx.classes.digraph.DiGraph*) – The input graph
- **max_num_edges** (*int*) – len(graph.edges), or the specified maximum number of graph edges

Returns A dictionary of with the single 'connectivity' key, containing an (n,2) array of (node_index, node_index) pairs indicating the start and end nodes for each edge.

Return type Dict[str, np.ndarray]

get_edge_features(*edge_data*, *max_num_edges*)

Given a list of edge features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **edge_data** (*list*) – A list of edge data generated by *nx_graph.edges(data=True)*
- **max_num_edges** – If desired, this function should pad to a maximum number of edges passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all edges in the graph.

Return type Dict[str, np.ndarray]

get_graph_features(*graph_data*)

Process the nx.graph features into a dictionary of arrays.

Parameters **graph_data** (*dict*) – A dictionary of graph data generated by *nx_graph.graph*

Returns a dictionary of features for the graph

Return type Dict[str, np.ndarray]

get_node_features(*node_data*, *max_num_nodes*)

Given a list of node features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **node_data** (*list*) – A list of edge data generated by *nx_graph.nodes(data=True)*
- **max_num_nodes** – If desired, this function should pad to a maximum number of nodes passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all nodes in the graph.

Return type Dict[str, np.ndarray]

property padding_values: Dict[str, tensorflow.constant]

Defaults to zero for each output

property tfrecord_features: Dict[str, tensorflow.io.FixedLenFeature]

For loading preprocessed inputs from a tf records file

to_json(*filename*)

Serialize the classes's data to a json file

Parameters **filename** (*str*) –

Return type None

2.5.4 nfp.preprocessing.preprocessor

Functions

load_from_json	Function to set member attributes from json data recursively.

nfp.preprocessing.preprocessor.load_from_json

load_from_json(*obj*, *data*)

Function to set member attributes from json data recursively.

Parameters

- **obj** (*Any*) – the class to initialize
- **data** (*Dict*) – a dictionary of potentially nested attribute - value pairs

Returns The object, with attributes set to those from the data file.

Return type Any

Classes

Preprocessor	A base class for graph preprocessing
PreprocessorMultiGraph	Class to handle graphs with parallel edges and self-loops

nfp.preprocessing.preprocessor.Preprocessor

class Preprocessor(*output_dtype='int32'*)

Bases: abc.ABC

A base class for graph preprocessing

Parameters **output_dtype** (*str*) – A parameter used in child classes for determining the datatype of the returned arrays

Methods

construct_feature_matrices	Deprecated since version 0.3.0.
create_nx_graph	Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.
from_json	Set's the class's data with attributes taken from the save file
get_connectivity	Get the graph connectivity from the networkx graph
get_edge_features	Given a list of edge features from the nx.Graph, processes and concatenates them to an array.
get_graph_features	Process the nx.graph features into a dictionary of arrays.
get_node_features	Given a list of node features from the nx.Graph, processes and concatenates them to an array.
to_json	Serialize the classes's data to a json file

Attributes

output_signature	
padding_values	
tfrecord_features	Useful feature for storing preprocessed outputs in tfrecord files

abstract create_nx_graph(*structure*, **args*, ***kwargs*)

Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.

Parameters

- **structure** (*Any*) – Any input graph object

- **kwargs** – keyword arguments passed from *__call__*, useful for specifying additional features in addition to the graph object.

Returns A networkx graph with the node, edge, and graph features set

Return type nx.DiGraph

abstract get_edge_features(*edge_data*, *max_num_edges*)

Given a list of edge features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **edge_data** (*list*) – A list of edge data generated by *nx_graph.edges(data=True)*
- **max_num_edges** (*int*) – If desired, this function should pad to a maximum number of edges passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all edges in the graph.

Return type Dict[str, np.ndarray]

abstract get_node_features(*node_data*, *max_num_nodes*)

Given a list of node features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **node_data** (*list*) – A list of edge data generated by *nx_graph.nodes(data=True)*
- **max_num_nodes** (*int*) – If desired, this function should pad to a maximum number of nodes passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all nodes in the graph.

Return type Dict[str, np.ndarray]

abstract get_graph_features(*graph_data*)

Process the nx.graph features into a dictionary of arrays.

Parameters **graph_data** (*dict*) – A dictionary of graph data generated by *nx_graph.graph*

Returns a dictionary of features for the graph

Return type Dict[str, np.ndarray]

static get_connectivity(*graph*, *max_num_edges*)

Get the graph connectivity from the networkx graph

Parameters

- **graph** (*networkx.classes.digraph.DiGraph*) – The input graph
- **max_num_edges** (*int*) – len(graph.edges), or the specified maximum number of graph edges

Returns A dictionary of with the single 'connectivity' key, containing an (n,2) array of (node_index, node_index) pairs indicating the start and end nodes for each edge.

Return type Dict[str, np.ndarray]

__call__(*structure*, **args*, *train=False*, *max_num_nodes=None*, *max_num_edges=None*, ***kwargs*)

Convert an input graph structure into a featurized set of node, edge, and graph-level features.

Parameters

- **structure** (*Any*) – An input graph structure (i.e., molecule, crystal, etc.)
- **train** (*bool*) – A training flag passed to *Tokenizer* member attributes
- **max_num_nodes** (*Optional[int]*) – A size attribute passed to *get_node_features*, defaults to the number of nodes in the current graph
- **max_num_edges** (*Optional[int]*) – A size attribute passed to *get_edge_features*, defaults to the number of edges in the current graph
- **kwargs** – Additional features or parameters passed to *construct_nx_graph*

Returns A dictionary of key, array pairs as a single sample.

Return type Dict[str, np.ndarray]

construct_feature_matrices(**args*, *train=False*, ***kwargs*)

Deprecated since version 0.3.0: *construct_feature_matrices* will be removed in 0.4.0, use *__call__* instead

Return type *Dict*[str, numpy.ndarray]

to_json(*filename*)

Serialize the classes's data to a json file

Parameters **filename** (*str*) –

Return type None

from_json(*filename*)

Set's the class's data with attributes taken from the save file

Parameters **filename** (*str*) –

Return type None

property tfrecord_features: Dict[str, tensorflow.io.FixedLenFeature]

Useful feature for storing preprocessed outputs in tfrecord files

nfp.preprocessing.preprocessor.PreprocessorMultiGraph

class PreprocessorMultiGraph(*output_dtype='int32'*)

Bases: *nfp.preprocessing.preprocessor.Preprocessor*, abc.ABC

Class to handle graphs with parallel edges and self-loops

Methods

construct_feature_matrices	Deprecated since version 0.3.0.
create_nx_graph	Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.
from_json	Set's the class's data with attributes taken from the save file
get_connectivity	Get the graph connectivity from the networkx graph
get_edge_features	Given a list of edge features from the nx.Graph, processes and concatenates them to an array.
get_graph_features	Process the nx.graph features into a dictionary of arrays.
get_node_features	Given a list of node features from the nx.Graph, processes and concatenates them to an array.
to_json	Serialize the classes's data to a json file

Attributes

output_signature	
padding_values	
tfrecord_features	Useful feature for storing preprocessed outputs in tfrecord files

Parameters **output_dtype** (*str*) –

abstract create_nx_graph(*structure*, ***kwargs*)

Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.

Parameters

- **structure** (*Any*) – Any input graph object
- **kwargs** – keyword arguments passed from *__call__*, useful for specifying additional features in addition to the graph object.

Returns A networkx graph with the node, edge, and graph features set

Return type nx.DiGraph

static get_connectivity(*graph*, *max_num_edges*)

Get the graph connectivity from the networkx graph

Parameters

- **graph** (*networkx.classes.digraph.DiGraph*) – The input graph
- **max_num_edges** (*int*) – len(graph.edges), or the specified maximum number of graph edges

Returns A dictionary of with the single 'connectivity' key, containing an (n,2) array of (node_index, node_index) pairs indicating the start and end nodes for each edge.

Return type Dict[str, np.ndarray]

__call__(*structure*, **args*, *train=False*, *max_num_nodes=None*, *max_num_edges=None*, ***kwargs*)

Convert an input graph structure into a featurized set of node, edge, and graph-level features.

Parameters

- **structure** (*Any*) – An input graph structure (i.e., molecule, crystal, etc.)
- **train** (*bool*) – A training flag passed to *Tokenizer* member attributes
- **max_num_nodes** (*Optional[int]*) – A size attribute passed to *get_node_features*, defaults to the number of nodes in the current graph
- **max_num_edges** (*Optional[int]*) – A size attribute passed to *get_edge_features*, defaults to the number of edges in the current graph
- **kwargs** – Additional features or parameters passed to *construct_nx_graph*

Returns A dictionary of key, array pairs as a single sample.

Return type Dict[str, np.ndarray]

construct_feature_matrices(**args*, *train=False*, ***kwargs*)

Deprecated since version 0.3.0: *construct_feature_matrices* will be removed in 0.4.0, use *__call__* instead

Return type *Dict*[str, numpy.ndarray]

from_json(*filename*)

Set's the class's data with attributes taken from the save file

Parameters **filename** (*str*) –

Return type None

abstract get_edge_features(*edge_data*, *max_num_edges*)

Given a list of edge features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **edge_data** (*list*) – A list of edge data generated by *nx_graph.edges(data=True)*
- **max_num_edges** (*int*) – If desired, this function should pad to a maximum number of edges passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all edges in the graph.

Return type Dict[str, np.ndarray]

abstract get_graph_features(*graph_data*)

Process the nx.graph features into a dictionary of arrays.

Parameters **graph_data** (*dict*) – A dictionary of graph data generated by *nx_graph.graph*

Returns a dictionary of features for the graph

Return type Dict[str, np.ndarray]

abstract get_node_features(*node_data*, *max_num_nodes*)

Given a list of node features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **node_data** (*list*) – A list of edge data generated by *nx_graph.nodes(data=True)*
- **max_num_nodes** (*int*) – If desired, this function should pad to a maximum number of nodes passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all nodes in the graph.

Return type Dict[str, np.ndarray]

property tfrecord_features: Dict[str, tensorflow.io.FixedLenFeature]

Useful feature for storing preprocessed outputs in tfrecord files

to_json(*filename*)

Serialize the classes's data to a json file

Parameters **filename** (*str*) –

Return type None

2.5.5 nfp.preprocessing.tfrecord

Functions

serialize_value	

nfp.preprocessing.tfrecord.serialize_value

serialize_value(*value*)

2.5.6 nfp.preprocessing.tokenizer

Classes

Tokenizer	A class to turn arbitrary inputs into integer classes.

nfp.preprocessing.tokenizer.Tokenizer

class Tokenizer

Bases: object

A class to turn arbitrary inputs into integer classes.

Methods

__call__(*item*)

Check to see if the Tokenizer has seen *item* before, and if so, return the integer class associated with it. Otherwise, if we're training, create a new integer class, otherwise return the 'unknown' class.

2.5.7 nfp.preprocessing.xtb_preprocessor

Classes

xTBPreprocessor	
xTBSmilesPreprocessor	

nfp.preprocessing.xtb_preprocessor.xTBPreprocessor

class xTBPreprocessor(**args*, *explicit_hs=True*, *xtb_atom_features=None*, *xtb_bond_features=None*, *xtb_mol_features=None*, *cutoff=0.3*, ***kwargs*)

Bases: *nfp.preprocessing.mol_preprocessor.MolPreprocessor*

Methods

construct_feature_matrices	Deprecated since version 0.3.0.
create_nx_graph	Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.
from_json	Set's the class's data with attributes taken from the save file
get_connectivity	Get the graph connectivity from the networkx graph
get_edge_features	Given a list of edge features from the nx.Graph, processes and concatenates them to an array.
get_graph_features	Process the nx.graph features into a dictionary of arrays.
get_node_features	Given a list of node features from the nx.Graph, processes and concatenates them to an array.
to_json	Serialize the classes's data to a json file

Attributes

atom_classes	The number of atom types found (includes the 0 null-atom type)
bond_classes	The number of bond types found (includes the 0 null-bond type)
output_signature	
padding_values	Defaults to zero for each output
tfrecord_features	For loading preprocessed inputs from a tf records file

Parameters

- **explicit_hs** (*bool*) –
- **xtb_atom_features** (*Optional[List[str]]*) –
- **xtb_bond_features** (*Optional[List[str]]*) –
- **xtb_mol_features** (*Optional[List[str]]*) –
- **cutoff** (*float*) –

create_nx_graph(*mol*, *jsonfile*, ***kwargs*)

Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.

Parameters

- **structure** – Any input graph object
- **kwargs** – keyword arguments passed from *__call__*, useful for specifying additional features in addition to the graph object.
- **mol** (*rdkit.Chem.Mol*) –
- **jsonfile** (*str*) –

Returns A networkx graph with the node, edge, and graph features set

Return type nx.DiGraph

get_edge_features(*edge_data*, *max_num_edges*)

Given a list of edge features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **edge_data** (*list*) – A list of edge data generated by *nx_graph.edges(data=True)*
- **max_num_edges** – If desired, this function should pad to a maximum number of edges passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all edges in the graph.

Return type Dict[str, np.ndarray]

get_node_features(*node_data*, *max_num_nodes*)

Given a list of node features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **node_data** (*list*) – A list of edge data generated by *nx_graph.nodes(data=True)*
- **max_num_nodes** (*int*) – If desired, this function should pad to a maximum number of nodes passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all nodes in the graph.

Return type Dict[str, np.ndarray]

get_graph_features(*graph_data*)

Process the nx.graph features into a dictionary of arrays.

Parameters **graph_data** (*dict*) – A dictionary of graph data generated by *nx_graph.graph*

Returns a dictionary of features for the graph

Return type Dict[str, np.ndarray]

__call__(*structure*, **args*, *train=False*, *max_num_nodes=None*, *max_num_edges=None*, ***kwargs*)

Convert an input graph structure into a featurized set of node, edge, and graph-level features.

Parameters

- **structure** (*Any*) – An input graph structure (i.e., molecule, crystal, etc.)
- **train** (*bool*) – A training flag passed to *Tokenizer* member attributes
- **max_num_nodes** (*Optional[int]*) – A size attribute passed to *get_node_features*, defaults to the number of nodes in the current graph
- **max_num_edges** (*Optional[int]*) – A size attribute passed to *get_edge_features*, defaults to the number of edges in the current graph
- **kwargs** – Additional features or parameters passed to *construct_nx_graph*

Returns A dictionary of key, array pairs as a single sample.

Return type Dict[str, np.ndarray]

property atom_classes: int

The number of atom types found (includes the 0 null-atom type)

property bond_classes: int

The number of bond types found (includes the 0 null-bond type)

construct_feature_matrices(**args*, *train=False*, ***kwargs*)

Deprecated since version 0.3.0: *construct_feature_matrices* will be removed in 0.4.0, use *__call__* instead

Return type *Dict*[str, numpy.ndarray]

from_json(*filename*)

Set's the class's data with attributes taken from the save file

Parameters **filename** (*str*) –

Return type None

static get_connectivity(*graph*, *max_num_edges*)

Get the graph connectivity from the networkx graph

Parameters

- **graph** (*networkx.classes.digraph.DiGraph*) – The input graph

- **max_num_edges** (*int*) – len(graph.edges), or the specified maximum number of graph edges

Returns A dictionary of with the single 'connectivity' key, containing an (n,2) array of (node_index, node_index) pairs indicating the start and end nodes for each edge.

Return type Dict[str, np.ndarray]

property padding_values: Dict[str, tensorflow.constant]

Defaults to zero for each output

to_json(*filename*)

Serialize the classes's data to a json file

Parameters **filename** (*str*) –

Return type None

property tfrecord_features: Dict[str, tensorflow.io.FixedLenFeature]

For loading preprocessed inputs from a tf records file

nfp.preprocessing.xtb_preprocessor.xTBSmilesPreprocessor

class xTBSmilesPreprocessor(**args*, *explicit_hs=True*, ***kwargs*)

Bases: *nfp.preprocessing.mol_preprocessor.SmilesPreprocessor*, *nfp.preprocessing.xtb_preprocessor.xTBPreprocessor*

Methods

construct_feature_matrices	Deprecated since version 0.3.0.
create_nx_graph	Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.
from_json	Set's the class's data with attributes taken from the save file
get_connectivity	Get the graph connectivity from the networkx graph
get_edge_features	Given a list of edge features from the nx.Graph, processes and concatenates them to an array.
get_graph_features	Process the nx.graph features into a dictionary of arrays.
get_node_features	Given a list of node features from the nx.Graph, processes and concatenates them to an array.
to_json	Serialize the classes's data to a json file

Attributes

atom_classes	The number of atom types found (includes the 0 null-atom type)
bond_classes	The number of bond types found (includes the 0 null-bond type)
output_signature	
padding_values	Defaults to zero for each output
tfrecord_features	For loading preprocessed inputs from a tf records file

Parameters **explicit_hs** (*bool*) –

__call__(*structure*, **args*, *train=False*, *max_num_nodes=None*, *max_num_edges=None*, ***kwargs*)

Convert an input graph structure into a featurized set of node, edge, and graph-level features.

Parameters

- **structure** (*Any*) – An input graph structure (i.e., molecule, crystal, etc.)
- **train** (*bool*) – A training flag passed to *Tokenizer* member attributes
- **max_num_nodes** (*Optional[int]*) – A size attribute passed to *get_node_features*, defaults to the number of nodes in the current graph
- **max_num_edges** (*Optional[int]*) – A size attribute passed to *get_edge_features*, defaults to the number of edges in the current graph
- **kwargs** – Additional features or parameters passed to *construct_nx_graph*

Returns A dictionary of key, array pairs as a single sample.

Return type Dict[str, np.ndarray]

property atom_classes: int

The number of atom types found (includes the 0 null-atom type)

property bond_classes: int

The number of bond types found (includes the 0 null-bond type)

construct_feature_matrices(**args*, *train=False*, ***kwargs*)

Deprecated since version 0.3.0: *construct_feature_matrices* will be removed in 0.4.0, use *__call__* instead

Return type *Dict*[str, numpy.ndarray]

create_nx_graph(*smiles*, **args*, ***kwargs*)

Given an input structure object, convert it to a networkx digraph with node, edge, and graph features assigned.

Parameters

- **structure** – Any input graph object
- **kwargs** – keyword arguments passed from *__call__*, useful for specifying additional features in addition to the graph object.
- **smiles** (*str*) –

Returns A networkx graph with the node, edge, and graph features set

Return type nx.DiGraph

from_json(*filename*)

Set's the class's data with attributes taken from the save file

Parameters **filename** (*str*) –

Return type None

static get_connectivity(*graph*, *max_num_edges*)

Get the graph connectivity from the networkx graph

Parameters

- **graph** (*networkx.classes.digraph.DiGraph*) – The input graph
- **max_num_edges** (*int*) – len(graph.edges), or the specified maximum number of graph edges

Returns A dictionary of with the single 'connectivity' key, containing an (n,2) array of (node_index, node_index) pairs indicating the start and end nodes for each edge.

Return type Dict[str, np.ndarray]

get_edge_features(*edge_data*, *max_num_edges*)

Given a list of edge features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **edge_data** (*list*) – A list of edge data generated by *nx_graph.edges(data=True)*
- **max_num_edges** – If desired, this function should pad to a maximum number of edges passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all edges in the graph.

Return type Dict[str, np.ndarray]

get_graph_features(*graph_data*)

Process the nx.graph features into a dictionary of arrays.

Parameters **graph_data** (*dict*) – A dictionary of graph data generated by *nx_graph.graph*

Returns a dictionary of features for the graph

Return type Dict[str, np.ndarray]

get_node_features(*node_data*, *max_num_nodes*)

Given a list of node features from the nx.Graph, processes and concatenates them to an array.

Parameters

- **node_data** (*list*) – A list of edge data generated by *nx_graph.nodes(data=True)*
- **max_num_nodes** (*int*) – If desired, this function should pad to a maximum number of nodes passed from the *__call__* function.

Returns a dictionary of feature, array pairs, where array contains features for all nodes in the graph.

Return type Dict[str, np.ndarray]

property padding_values: Dict[str, tensorflow.constant]

Defaults to zero for each output

property tfrecord_features: Dict[str, tensorflow.io.FixedLenFeature]

For loading preprocessed inputs from a tf records file

to_json(*filename*)

Serialize the classes's data to a json file

Parameters **filename** (*str*) –

Return type None

CHAPTER

THREE

NEURAL FINGERPRINT (NFP), TF.KERAS LAYERS FOR MOLECULAR GRAPHS

A collection of tensorflow.keras layers, losses, and preprocessors for building graph neural networks on molecules.

PYTHON MODULE INDEX

n

INDEX

Symbols

A

B

C

F

G

L

M