MolML Documentation

Release 0.9.0

Chris Collins

Jul 31, 2019

Contents:

CHAPTER 1

molml package

1.1 Submodules

1.1.1 molml.atom module

A module to compute atom based representations.

This module contains a variety of methods to extract features from molecules based on the atoms in the molecule. This means that every molecule will result in an array of values (n_atoms, n_features).

class molml.atom.**Shell**(*input_type='list'*, *n_jobs=1*, *depth=1*, *use_coordination=False*, *add_unknown=False*)

Bases: *molml.base.SetMergeMixin*, *molml.base.BaseFeature*

A feature that counts the number of elements in a distance shell from the starting atom. This is similar to the features developed in Qu et. al. with the exception that it is atom-based rather than bond-based.

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

depth [int, default=1] The length of the atom chains to generate for connections

use_coordination [boolean, default=False] Specifies whether or not to use the coordination number of the atoms (C1 vs C2 vs C3 vs C4).

add_unknown [boolean, default=False] Specifies whether or not to include an extra UNKNOWN count in the feature vector.

References

Qu, X.; Latino, D. A.; Aires-de Sousa, J. A Big Data Approach to the Ultra-fast Prediction of DFT-calculated Bond Energies. J. Cheminf. 2013, 5, 34.

Attributes

_elements [tuple] All the elements/types that are in the fit molecules.

ATTRIBUTES = ('_elements',)

LABELS = (('get_shell_labels', '_elements'),)

get_shell_labels(*self*, *elements*)

class molml.atom.**LocalEncodedBond**(*input_type='list'*, *n_jobs=1*, *segments=100*, *smoothing='norm'*, *start=0.2*, *end=6.0*, *slope=20.0*, *min_depth=0*, *max_depth=0*, *spacing='linear'*, *form=1*, *add_unknown=False*, *use_comb_idxs=False*)

Bases: *molml.base.FormMixin*, *molml.base.SetMergeMixin*, *molml.base.EncodedFeature*

A smoothed histogram of atomic distances.

This is a method to generallize the idea of bond counting. Instead of seeing bonds as a discrete count that is thresholded at a given length, they are seen as general distance histograms. This is supplemented with smoothing functions. This is a slight modification of the EncodedBond to use with atoms.

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

segments [int, default=100] The number of bins/segments to use when generating the histogram.

smoothing [string or callable, default='norm'] A string or callable to use to smooth the histogram values. If a callable is given, it must take just a single argument that is a float. For a list of supported default functions look at SMOOTHING_FUNCTIONS.

start [float, default=0.2] The starting point for the histgram sampling in angstroms.

end [float, default=6.0] The ending point for the histogram sampling in angstroms.

slope [float, default=20.] A parameter to tune the smoothing values. This is applied as a multiplication before calling the smoothing function.

min_depth [int, default=0] A parameter to set the minimum geodesic distance to include in the interactions. A value of np.inf signifies including only intermolecular interactions.

max_depth [int, default=0] A parameter to set the maximum geodesic distance to include in the interactions. A value of 0 signifies that all interactions are included.

spacing [string or callable, default='linear'] The histogram interval spacing type. Must be one of ("linear", "inverse", or "log"). Linear spacing is normal spacing. Inverse takes and evaluates the distances as 1/r and the start and end points are 1/x. For log spacing, the distances are evaluated as numpy.log(r) and the start and end points are numpy.log(x). If the value is callable, then it should take a float or vector of floats and return a similar mapping like the other methods.

form [int, default=1] The histogram splitting style to use. This value changes the scaling of this method to be O(E) or O(1) for 1 or 0 respectively (where E is the number of elements).

add_unknown [boolean, default=False] Specifies whether or not to include an extra UNKNOWN count in the feature vector.

use_comb_idxs [bool, default=False] Whether or not to use all combinations of indices when doing the subselection. If this is false, a middle out scheme will be used.

Attributes

_elements [tuple] A tuple of all the elements in the fit molecules.

```
ATTRIBUTES = ('_elements',)
```

```
LABELS = (('get_encoded_labels', '_elements'),)
```

class molml.atom.**LocalEncodedAngle**(*input_type='list'*, *n_jobs=1*, *segments=100*, *smoothing='norm'*, *slope=20.0*, *min_depth=0*, *max_depth=0*, *r_cut=6.0*, *form=2*, *add_unknown=False*)

Bases: *molml.base.FormMixin*, *molml.base.SetMergeMixin*, *molml.base.EncodedFeature*

A smoothed histogram of atomic angles.

This method is similar to EncodedBond but for angles in molecules. This is done by enumerating triplets of atoms and computing the angle between them. The bins are thing smoothed with smoothing functions. This is a slight modification of the EncodedAngle to work with single atoms at a time. This sets the vertex of the angle to be the atom being examined.

Note: The angles used are 0 to π.

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

segments [int, default=100] The number of bins/segments to use when generating the histogram.

smoothing [string or callable, default='norm'] A string or callable to use to smooth the histogram values. If a callable is given, it must take just a single argument that is a float. For a list of supported default functions look at SMOOTHING_FUNCTIONS.

slope [float, default=20.] A parameter to tune the smoothing values. This is applied as a multiplication before calling the smoothing function.

min_depth [int, default=0] A parameter to set the minimum geodesic distance to include in the interactions. A value of np.inf signifies including only intermolecular interactions.

max_depth [int, default=0] A parameter to set the maximum geodesic distance to include in the interactions. A value of 0 signifies that all interactions are included.

form [int, default=2] The histogram splitting style to use. This value changes the scaling of this method to be O(E^2), O(E), or O(1) for 2, 1, or 0 respectively (where E is the number of elements).

add_unknown [boolean, default=False] Specifies whether or not to include an extra UNKNOWN count in the feature vector.

use_comb_idxs [bool, default=False] Whether or not to use all combinations of indices when doing the subselection. If this is false, a middle out scheme will be used.

Attributes

_pairs [tuple] A tuple of all the element pairs in the fit molecules.

ATTRIBUTES = ('_pairs',)

LABELS = (('get_encoded_labels', '_pairs'),)

f_c(*self*, *R*)

class molml.atom.**LocalCoulombMatrix**(*input_type='list'*, *n_jobs=1*, *max_occupancy=4*, *r_cut=10.0*, *alpha=6*, *use_reduced=False*, *use_decay=False*)

Bases: *molml.base.BaseFeature*

An implementation of the Coulomb Matrix where only the local atom environment is used by using a cutoff radius.

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

max_occupancy [int, default=4] The maximum number of atoms to be included the in local environment.

r_cut [float, default=6] The maximum distance allowed for atoms to be considered local to the "central atom".

alpha [number, default=6] Some value to exponentiate the distance in the coulomb matrix.

use_reduced [bool, default=False] This setting uses only the first row of the local coulomb matrix and the diagonal. This reduces the feature from scaling as O(max_occupancy ** 2) to just O(max_occupancy).

use_decay [bool, default=False] This setting defines an extra decay for the values as they get futher away from the "central atom". This is to alleviate issues the arise as atoms enter or leave the cutoff radius.

$$M_{ij} = \begin{cases} \frac{Z_{p_i} Z_{p_j}}{(\|R_{p_1} - R_{p_i}\|_2 + \|R_{p_1} - R_{p_j}\|_2 + \|R_{p_i} - R_{p_j}\|_2)^{\alpha}}, & i \neq j \\ 0.5 Z_{p_i}^{2.4} & i = j \end{cases}$$

References

Barker, J.; Bulin, J.; Hamaekers, J.; Mathias, S. Localized Coulomb Descriptors for the Gaussian Approximation Potential. arXiv 1611.05126

ATTRIBUTES = None

LABELS = (('get_local_coulomb_labels', None),)

fit(*self*, *X*, *y=None*)
No fitting is required because it is defined by the parameters.

get_local_coulomb_labels(*self*)

class molml.atom.**BehlerParrinello**(*input_type='list'*, *n_jobs=1*, *r_cut=6.0*, *r_s=1.0*, *eta=1.0*, *lambda_=1.0*, *zeta=1.0*)

Bases: *molml.base.SetMergeMixin*, *molml.base.BaseFeature*

An implementation of the descriptors used in Behler-Parrinello Neural Networks.

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

r_cut [float, default=6.] The maximum distance allowed for atoms to be considered local to the "central atom".

r_s [float, default=1.0] An offset parameter for computing gaussian values between pairwise distances.

eta [float, default=1.0] A decay parameter for the gaussian distances.

lambda_ [float, default=1.0] This value sets the orientation of the cosine function for the angles. It should only take values in {-1., 1.}.

zeta [float, default=1.0] A decay parameter for the angular terms.

References

Behler, J; Parrinello, M. Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces. Phys. Rev. Lett. 98, 146401.

Attributes

_elements [tuple] A set of all the elements in the molecules.

_element_pairs [tuple] A set of all the element pairs in the molecules.

ATTRIBUTES = ('_elements', '_element_pairs')

LABELS = ('_elements', ('get_chain_labels', '_element_pairs'))

calculate_Theta(*self*, *R_vecs*)

Compute the angular term for all triples of atoms.

$$\Theta_{ijk} = (R_{ij}.R_{ik})/(|R_{ij}||R_{ik}|)$$

Right now this is a fairly naive implementation so this could be optimized quite a bit.

Parameters

R_vecs [array, shape=(N_atoms, 3)] An array of the Cartesian coordinates of all the atoms

Returns

Theta [array, shape=(N_atoms, N_atoms, N_atoms)] The angular term for all the atoms given.

f_c(*self*, *R*)

g_1 (*self*, *R*, *elements*)
A radial symmetry function.

$$G_i^1 = \sum_{j \neq i} \exp(-\eta(R_{ij} - R_s)^2) f_c(R_{ij})$$

Parameters

R [array, shape=(N_atoms, N_atoms)] A distance matrix for all the atoms (scipy.spatial.cdist)

Returns

total [array, shape=(N_atoms, N_elements)] The atom-wise g_1 evaluations.

g_2 (*self*, *Theta*, *R*, *elements*)
An angular symmetry function.

$$G_i^2 = 2^{1-\zeta} \sum_{i,k \neq i} (1 + \lambda \cos(\Theta_{ijk}))^\zeta \exp(-\eta(R_{ij}^2 + R_{ik}^2 + R_{jk}^2)) f_c(R_{ij}) f_c(R_{ik}) f_c(R_{jk})$$

This function needs to be optimized.

Parameters

Theta [array, shape=(N_atoms, N_atoms, N_atoms)] An array of triplet angles.

R [array, shape=(N_atoms, N_atoms)] A distance matrix for all the atoms (scipy.spatial.cdist).

elements [list] A list of all the elements in the molecule.

Returns

total [array, shape=(N_atoms, len(self._element_pairs))] The atom-wise g_2 evaluations.

get_chain_labels (*self*, *chains*)

1.1.2 molml.base module

A collection of all the base transformer constructions.

This module is a collection of all the base classes and mixins for use with the other transformers.

class molml.base.**BaseFeature**(*input_type='list'*, *n_jobs=1*)
Bases: object

A base class for all the features.

Parameters

input_type [str, list of str, or callable, default='list'] Specifies the format the input values will be (must be one of 'list', 'filename', a list of strings, or a callable). If it is a list of strings, the strings tell the order of (and if they are included) the different molecule attributes (coords, elements, numbers, connections). If a callable is given, then it is assumed to return a LazyValues object.

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

check_fit (*self*)
Check if the transformer has been fit

Raises

ValueError The transformer has not been fit.

convert_input(*self*, *X*)
Convert the input (as specified in self.input_type) to a usable form.

Parameters

X [list or string (depends on the instance value of input_type)] An object that stores the data for a single molecule. See the Notes for more details.

Returns

values [Object] An object that allows the lazy evaluation of different properties

Raises

ValueError If the input_type given is not allowed.

Notes

If input_type is 'list', then it must be an iterable of (elements, coodinates pairs) for each molecule. Where the elements are an iterable of the form (ele1, ele2, ..., elen) and coordinates are an iterable of the form [(x1, y1, z1), (x2, y2, z2), ..., (xn, yn, zn)]. This allows allows for connections to be included. This is a dictionary where the keys are the indices of the atoms and the values are dictonaries with the key being another index and the value is the bond order (one of '1', 'Ar', '2', or '3'). Example for methane:

```
{
    0: {1: "1", 2: "1", 3: "1", 4: "1"},
    1: {0: "1"},
    2: {0: "1"},
    3: {0: "1"},
    4: {0: "1"},
}
```

If input_type is 'filename', then it must be an iterable of paths/filenames for each molecule. Currently, the supported formats are: xyz, mol2, and a simple xyz format (.out).

If input_type is a list, then they will be treated as labels to each of the arguments passed in via a tuple. For example, input_type="list" can be reproduced with ["elements", "coords"] or ["elements", "coords", "connections"].

If input_type is a callable, then it is assumed that the callable returns a LazyValues object.

fit(*self*, *X*, *y=None*)
Fit the model.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to fit.

Returns

self [object] Returns the instance itself.

fit_transform(*self*, *X*, *y=None*)
A naive default implementation of fitting and transforming.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to fit and then transform

Returns

array [array, shape=(n_samples, n_features)] The transformed features

classmethod get_citation(*self*)

get_labels(*self*)
Get the labels for the features in the transformer

Returns

values [tuple] All of the labels of the resulting features.

get_params(*self*)
Get a dictonary of all the feature parameters.

Returns

params [dict] A dictonary of all the feature parameters.

map(*self*, *f*, *seq*)
Parallel implementation of map.

Parameters

f [callable] A function to map to all the values in 'seq'

seq [iterable] An iterable of values to process with 'f'

Returns

results [list, shape=[len(seq)]] The evaluated values

reduce(*self*, *f*, *seq*)
Parallel implementation of reduce.

This changes the problem from being O(n) steps to O(lg n)

Parameters

f [callable] A function to use to reduce the values of 'seq'

seq [iterable] An iterable of values to process

Returns

results [object] A single reduced object based on 'seq' and 'f'

save_json(*self*, *f*)
Save the model data in a json file

Parameters

f [str or file descriptor] The path to save the data or a file descriptor to save it to.

set_params(*self*, ***kwargs*)
Set the feature parameter values.

Parameters

kwargs [kwargs] Key value pairs to set for the feature parameters. Keys that are not valid parameters will be ignored.

slugify(*self*)
Convert an instance to a simple string.

Returns

string [str] The slug string

to_json(*self*)
Return model data as a json compatible dict

This will recursively convert other transformer objects as well.

Returns

data [dict] The json data

transform(*self*, *X*, *y=None*)
Framework for a potentially parallel transform.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to transform

Returns

array [array, shape=(n_samples, n_features)] The transformed features

class molml.base.**EncodedFeature**(*input_type='list'*, *n_jobs=1*, *segments=100*, *smoothing='norm'*, *slope=20.0*, *start=0.2*, *end=6.0*, *spacing='linear'*)
Bases: *molml.base.BaseFeature*

This is a generalized class to handle all kinds of encoding feature representations. These approaches seem to be a fairly general way of making lists of scalar values more effective to use in machine learning models. Essentially, it can be viewed as kernel smoothed histograms over the values of interest.

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

segments [int, default=100] The number of bins/segments to use when generating the histogram. Empirically, it has been found that values beyond 50-100 have little benefit.

smoothing [string or callable, default='norm'] A string or callable to use to smooth the histogram values. If a callable is given, it must take just a single argument that is a float (or vector of floats). For a list of supported default functions look at SMOOTHING_FUNCTIONS.

start [float, default=0.2] The starting point for the histgram sampling in angstroms.

end [float, default=6.0] The ending point for the histogram sampling in angstroms.

slope [float, default=20.] A parameter to tune the smoothing values. This is applied as a multiplication before calling the smoothing function.

spacing [string or callable, default='linear'] The histogram interval spacing type. Must be one of ("linear", "inverse", or "log"). Linear spacing is normal spacing. Inverse takes and evaluates the distances as 1/r and the start and end points are 1/x. For log spacing, the distances are evaluated as numpy.log(r) and the start and end points are numpy.log(x). If the value is callable, then it should take a float or vector of floats and return a similar mapping like the other methods.

References

Collins, C.; Gordon, G.; von Lilienfeld, O. A.; Yaron, D. Constant Size Molecular Descriptors For Use With Machine Learning. arXiv:1701.06649

encode_values(*self*, *iterator*, *lengths*, *saved_lengths=0*)

Encodes an iterable of values into a uniform length array. These values can then be indexed to allow binning them in different sections of the array. After the values are processed, the array can by flattened down to a desired number of axes.

Parameters

iterator [iterable]

The collection of values to encode. Each item in the iterable must contain values for (idx, value, scaling). Where idx is a tuple of integer values indicating which encoding bucket the values go in, value is the value to encode, and scaling is a factor that gets multiplied by the final encoded subvector before getting added to the total (This is mostly used to mask values and scale their influence with distance. If idx is None, then the value will be skipped.

lengths [tuple of ints] The number of encoding axes to create. In terms of EncodedBonds, this would be the number of element pairs.

saved_lengths [ints] The number of axis components to retain. The order that they get saved is the same order that is given in lengths. For example, when doing atom encodings, this should be 1 to retain the atom axis.

Returns

vector [array] The final concatenated vector of all the subvectors. This will have a shape of lengths[:saved_lengths] + product(lengths[saved_lengths:]) * segments).

get_encoded_labels(*self*, *groups*)

Parameters

groups [list] A list of all the groups.

Returns

labels [list] A list of all the feature labels.

get_group_order(*self*, *groups*)

Parameters

groups [list] A list of all the groups.

Returns

value_order [list] A list of all groups in order.

class molml.base.**FormMixin**(*form=1*, *add_unknown=False*, *use_comb_idxs=False*, **args*, ***kwargs*)

Bases: object

A simple mixin for handling form transformations

This mixin handles all how index mapping is done when going from higher dimensional attributes to lower dimensional ones. By default, this mixin uses the first value in ATTRIBUTES as the basis for the index mapping.

use_comb_idxs [bool, default=False] Whether or not to use all combinations of indices when doing the subselection. If this is false, a middle out scheme will be used.

get_group_order(*self*, *groups*)

Parameters

groups [list] A list of all the groups. This is ignored.

Returns

value_order [list] A list of all groups in order.

get_idx_map(*self*)
Lazily load the idx_map.

Returns

idx_map [IndexMap] The IndexMap object for this form and add_unknown setting.

transform(*self*, *X*, *y=None*)
Framework for a potentially parallel transform.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to transform

Returns

array [array, shape=(n_samples, n_features)] The transformed features

class molml.base.**InputTypeMixin**
Bases: object

A simple mixin to to check input_types if there are multiples.

This mixin adds a method to check if a transformer parameter does not have the same input_type as the parent object.

check_transformer(*self*, *transformer*)
Check a transformer.

Parameters

transformer [BaseFeature] A transformer object.

Raises

ValueError If the input_type pairing given is not allowed.

class molml.base.**SetMergeMixin**
Bases: object

A simple mixin that will merge sets.

This mixin replaces all the duplicate code that just merges sets when doing the parallel fits. For this to work, it requires that the subclasses define *ATTRIBUTES*.

fit(*self*, *X*, *y=None*)
Fit the model.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to fit.

Returns

self [object] Returns the instance itself.

1.1.3 molml.constants module

1.1.4 molml.crystal module

A module to compute molecule based representations.

This module contains a variety of methods to extract features from molecules based on the entire molecule. All of the methods included here will produce one vector per molecule input.

class molml.crystal.**GenerallizedCrystal**(*input_type=None*, *n_jobs=1*, *transformer=None*, *radius=None*, *units=None*)

Bases: *molml.base.InputTypeMixin*, *molml.base.BaseFeature*

A wrapper around other features to facilitate faking crystals.

This is done by a brute force expansion of atoms in the molecules based on a given unit cell. This is highly inefficient, but it does set a baseline.

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

transformer [BaseFeature, default=None] The transformer that will be used once the atoms have been expanded into the crystal.

radius [float, default=None] The cutoff radius for including unit cells in angstroms.

units [list or int, default=None] The number of unit cells to include for each axis (if this is an int, then it is the same for all).

References

Faber, F.; Lindmaa, A; von Lilienfeld, O. A.; Armiento, R. Crystal Structure Representations for Machine Learning Models of Formation Energies. arXiv:1503.07406

ATTRIBUTES = None

LABELS = None

convert_input(*self*, *X*)

Convert the input (as specified in self.input_type) to a usable form.

Parameters

X [list or string (depends on the instance value of input_type)] An object that stores the data for a single molecule. See the Notes for more details.

Returns

values [Object] An object that allows the lazy evaluation of different properties

Raises

ValueError If the input_type given is not allowed.

Notes

If input_type is 'list', then it must be an iterable of (elements, coodinates pairs) for each molecule. Where the elements are an iterable of the form (ele1, ele2, ..., elen) and coordinates are an iterable of the form [(x1, y1, z1), (x2, y2, z2), ..., (xn, yn, zn)]. This allows allows for connections to be included. This is a dictionary where the keys are the indices of the atoms and the values are dictonaries with the key being another index and the value is the bond order (one of '1', 'Ar', '2', or '3'). Example for methane:

```
{
    0: {1: "1", 2: "1", 3: "1", 4: "1"},
    1: {0: "1"},
    2: {0: "1"},
    3: {0: "1"},
    4: {0: "1"},
}
```

If input_type is 'filename', then it must be an iterable of paths/filenames for each molecule. Currently, the supported formats are: xyz, mol2, and a simple xyz format (.out).

If input_type is a list, then they will be treated as labels to each of the arguments passed in via a tuple. For example, input_type="list" can be reproduced with ["elements", "coords"] or ["elements", "coords", "connections"].

If input_type is a callable, then it is assumed that the callable returns a LazyValues object.

fit(*self*, *X*, *y=None*)
Fit the model.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to fit.

Returns

self [object] Returns the instance itself.

fit_transform(*self*, *X*, *y=None*)
A naive default implementation of fitting and transforming.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to fit and then transform

Returns

array [array, shape=(n_samples, n_features)] The transformed features

transform(*self*, *X*, *y=None*)
Framework for a potentially parallel transform.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to transform

Returns

array [array, shape=(n_samples, n_features)] The transformed features

class molml.crystal.**EwaldSumMatrix**(*input_type='list'*, *n_jobs=1*, *L_max=10*, *G_max=10*, *sort=False*, *eigen=False*)
Bases: *molml.molecule.CoulombMatrix*

In this construction, we use a similar form to the Ewald sum of breaking the interaction into three parts and adding them together.

The interaction between two atoms is defined as follows

$$x_{ij} = x_{ij}^{(r)} + x_{ij}^{(m)} + x_{ij}^{0}.$$

The components are defined as follows

$$x_{ij}^{(r)} = Z_i Z_j \sum_L \frac{\text{erfc}(\alpha \|r_i - r_j + L\|_2)}{\|r_i - r_j + L\|_2}$$

$$x_{ij}^{(m)} = \frac{Z_i Z_j}{\pi V} \sum_G \frac{e^{-\|G\|_2^2/(2\alpha)^2}}{\|G\|_2^2} \cos(G \cdot (r_i - r_j))$$

$$x_{ij}^0 = -(Z_i^2 + Z_j^2)\frac{\alpha}{\sqrt{\pi}} - (Z_i + Z_j)^2 \frac{\pi}{2V\alpha^2}$$

$$x_{ii} = -Z_i^2 \frac{\alpha}{\sqrt{\pi}} - Z_i^2 \frac{\pi}{2V\alpha^2}$$

$$\alpha = \sqrt{\pi}\left(\frac{0.01M}{V}\right)^{1/6}$$

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

sort [bool, default=False] Specifies whether or not to sort the coulomb matrix based on the sum of the rows (same as L1 norm).

eigen [bool, default=False] Specifies whether or not to use the eigen spectrum of the coulomb matrix rather than the matrix itself. This changes the scaling to be linear in the number of atoms.

References

Faber, F.; Lindmaa, A; von Lilienfeld, O. A.; Armiento, R. Crystal Structure Representations for Machine Learning Models of Formation Energies. arXiv:1503.07406

Attributes

_max_size [int] The size of the largest molecule in the fit molecules by number of atoms.

ATTRIBUTES = ('_max_size',)

LABELS = None

class molml.crystal.**SineMatrix**(*input_type='list'*, *n_jobs=1*, *sort=False*, *eigen=False*)
Bases: *molml.molecule.CoulombMatrix*

A molecular descriptor based on Coulomb interactions.

This is a feature that uses a Coulomb-like interaction between all atoms in the molecule to generate a matrix that is then vectorized.

$$C_{ij} = \begin{cases} Z_i Z_j \Phi(r_i, r_j) & i \neq j \\ 0.5 Z_i^{2.4} & i = j \end{cases}$$

Where $\Phi(r_i, r_j)$

$$\|B \cdot \sum_{k=x,y,z} \hat{e}_k \sin^2\left[\pi \hat{e}_k B^{-1} \cdot (r_i - r_j)\right]\|_2^{-1}$$

and B is a matrix of the lattice basis vectors.

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

sort [bool, default=False] Specifies whether or not to sort the coulomb matrix based on the sum of the rows (same as L1 norm).

eigen [bool, default=False] Specifies whether or not to use the eigen spectrum of the coulomb matrix rather than the matrix itself. This changes the scaling to be linear in the number of atoms.

References

Faber, F.; Lindmaa, A; von Lilienfeld, O. A.; Armiento, R. Crystal Structure Representations for Machine Learning Models of Formation Energies. arXiv:1503.07406

Attributes

_max_size [int] The size of the largest molecule in the fit molecules by number of atoms.

```
ATTRIBUTES = ('_max_size',)
```

```
LABELS = None
```

1.1.5 molml.features module

1.1.6 molml.fragment module

A module to compute fragment based representations.

This module contains a variety of methods to extract features from molecules based on defined fragments in the molecule. This means that every molecule will result in an array of values (n_fragments, n_features). Note: If atom-wise features are used, then this would extend to be (n_fragments, n_atoms, n_features).

class molml.fragment.**FragmentMap**(*input_type='filename'*, *n_jobs=1*, *transformer=None*, *filename_to_label='basename'*, *label_to_filename='.'*, *)*)

Bases: *molml.base.BaseFeature*

Extract information based on features from fragments.

This is like if there were *n* features that were extracted from the molecule of interest, and each of these *n* features corresponded to their own feature vectors. These fragments are then used together as a single representation. The output of these fragment vectors is in the same order that they are given.

For example,

FragmentMap().fit_transform([['A', 'B'], ['C, 'A'], ['B', 'C'])

would produce arrays like

[[f_A, f_B], [f_C, f_A], [f_B, f_C]]

for a final shape of (3, 2, n_features).

Parameters

input_type [str, default='filename'] Specifies the format the input values will be (must be one of 'label' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specific amount, and numbers less than 1 will use the number of cores the computer has.

transformer [BaseFeature, default=None] Some feature extractor that takes inputs and converts them to a numpy array of data. This should convert the fragment fragments into some vector representation to use. Because the information given to this class is at the label/filename level, the transformer must be able to work with the filenames directly. Either using the standard 'filename' *input_type*, or using a user-defined function.

filename_to_label [callable or str, default='basename'] The function to use to convert labels into filenames. The function should take a single str argument and return a label to use for that filename. The conversion between labels and filenames is not really required, but may allow for simpler bookkeeping outside this class. There are some predefined functions available in cls.LABEL_FUNCTIONS ('identity', 'basename') as recommendations for what to use.

label_to_filename [callable or list of str, default=('.',)] A function to convert labels into filenames to pass to the transformer. The function should take a single str argument and return a valid path. If a valid path does not exist, this should raise a ValueError. If this is a list, then it will be interpreted as a list of paths to search for files. Specifically, these are used in globs of the form os.path.join(dir_name, label + '.*'). Note: This will only use the first file that is found matching that label. The directories will be searched in the order given.

Attributes

_x_fragments [dict, str->numpy.array] Dictionary mapping label strings to their corresponding feature vectors.

ATTRIBUTES = ('_x_fragments',)

LABELS = (('get_mapping_labels', None),)

LABEL_FUNCTIONS = {'basename': <function <lambda>>, 'identity': <function <lambda>>}

convert_input(*self*, *X*)

Convert the input (as specified in self.input_type) to a usable form.

Parameters

X [list or string (depends on the instance value of input_type)] An object that stores the data for a single molecule. See the Notes for more details.

Returns

values [Object] An object that allows the lazy evaluation of different properties

Raises

ValueError If the input_type given is not allowed.

Notes

If input_type is 'list', then it must be an iterable of (elements, coodinates pairs) for each molecule. Where the elements are an iterable of the form (ele1, ele2, ..., elen) and coordinates are an iterable of the form [(x1, y1, z1), (x2, y2, z2), ..., (xn, yn, zn)]. This allows allows for connections to be included. This is a dictionary where the keys are the indices of the atoms and the values are dictonaries with the key being another index and the value is the bond order (one of '1', 'Ar', '2', or '3'). Example for methane:

```
{
    0: {1: "1", 2: "1", 3: "1", 4: "1"},
    1: {0: "1"},
    2: {0: "1"},
    3: {0: "1"},
    4: {0: "1"},
}
```

If input_type is 'filename', then it must be an iterable of paths/filenames for each molecule. Currently, the supported formats are: xyz, mol2, and a simple xyz format (.out).

If input_type is a list, then they will be treated as labels to each of the arguments passed in via a tuple. For example, input_type="list" can be reproduced with ["elements", "coords"] or ["elements", "coords", "connections"].

If input_type is a callable, then it is assumed that the callable returns a LazyValues object.

fit (*self*, *X*, *y=None*)
Fit the model.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to fit.

Returns

self [object] Returns the instance itself.

get_mapping_labels (*self*)

1.1.7 molml.io module

A collection of functions for loading molecule data from different file types.

Note: Functions in this file should be agnostic to the elements/numbers. This should be deferred to the LazyValues object.

molml.io.**read_cry_data** (*path*)
Read a cry file and extract the molecule's geometry.

The format should be as follows:

```
U_xx U_xy U_xz
U_yx U_yy U_yz
U_zx U_zy U_zz
energy (or comment, this is ignored for now)
ele0 x0 y0 z0
ele1 x1 y1 z1
...
elen xn yn zn
```

Where the U matrix is made of the unit cell basis vectors as column vectors.

Parameters

path [str] A path to a file to read

Returns

val [LazyValues] An object storing all the data

molml.io.**read_file_data**(*path*)
Determine the file type and call the correct parser.

The accepted file types are .out and .xyz files.

Parameters

path [str] A path to a file to read

Returns

elements [list] All the elements in the molecule.

numbers [list] All the atomic numbers in the molecule.

coords [numpy.array, shape=(n_atoms, 3)] The atomic coordinates of the molecule.

molml.io.**read_mol2_data**(*path*)
Read a mol2 file and extract the molecule's geometry.

Roughly, the file format is something like:

```
@<TRIPOS>MOLECULE
...
@<TRIPOS>ATOM
 1 ele0id x0 y0 z0 ele0.type 1 MOL charge0
 2 ele1id x1 y1 z1 ele1.type 1 MOL charge1
...
@<TRIPOS>BOND
...
```

Parameters

path [str] A path to a file to read

Returns

val [LazyValues] An object storing all the data

molml.io.**read_out_data**(*path*)
Read an out and extract the molecule's geometry.

The file should be in the format:

```
ele0 x0 y0 z0
ele1 x1 y1 z1
...
```

Parameters

path [str] A path to a file to read

Returns

val [LazyValues] An object storing all the data

molml.io.**read_xyz_data**(*path*)
Read an xyz file and extract the molecule's geometry.

The file should be in the format:

```
num_atoms
comment
ele0 x0 y0 z0
ele1 x1 y1 z1
...
```

Parameters

path [str] A path to a file to read

Returns

val [LazyValues] An object storing all the data

1.1.8 molml.kernel module

A module to compute kernel based representations.

The methods in this module are intended to be used directly as kernels for kernel methods (e.g. SVMs or KRR). This results in features that are dependent on the number of molecules used to fit the transformers. These should then give single vectors that have length n_fit_molecules.

class molml.kernel.**AtomKernel**(*input_type=None*, *n_jobs=1*, *gamma=1e-07*, *transformer=None*, *same_element=True*, *kernel='rbf'*)

Bases: *molml.base.InputTypeMixin*, *molml.base.BaseFeature*

Computes a kernel between molecules using atom similarity.

This kernel comes with the benefit that because it is atom-wise, it stays size consistent. So, for extensive properties this should properly scale with the size of the molecule compared to other kernel methods.

Parameters

input_type [string, default=None] Specifies the format the input values will be (must be one of 'list' or 'filename'). Note: This input type depends on the value from transformer. See Below for more details. If this value is None, then it will take the value from transformer, or if there is no transformer then it will default to 'list'. If a value is given and it does not match the value given for the transformer, then this will raise a ValueError.

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

gamma [float, default=1e-7] The hyperparameter to use for the width of the RBF or Laplace kernels

transformer [BaseFeature, default=None] The transformer to use to convert molecules to atom-wise features. If this is not given, then it is assumed that the features have already been created and will be passed directly to fit/transform. Note: if no transformer is given, then the assumed input type is going to be a a list of (numbers, features) pairs. Where numbers is an iterable of the atomic numbers, and features is a numpy array of the features (shape=(n_atoms, n_features)).

same_element [bool, default=True] Require that the atom-atom similarity only be computed if the two atoms are the same element.

kernel [string or callable, default="rbf"] The kernel function to use when computing the atom-atom interactions. There possible string options are the keys of KERNELS. If a callable object is given, then it must take two arrays and return the pairwise kernel metric between them.

Raises

ValueError If the input_type of the transformer and the input_type keyword given do not match.

References

Barker, J.; Bulin, J.; Hamaekers, J. LC-GAP: Localized Coulomb Descriptors for the Gaussian Approximation Potential. 2016

Attributes

_features [numpy.array, shape=(n_mols, (n_atoms, n_features))] A numpy array of numpy arrays (that may be different lengths) that stores all of the atom features for the training molecules.

_numbers [numpy.array, shape=(n_mols, (n_atoms))] A numpy array of numpy arrays (that may be different lengths) that stores all the atomic numbers for the training atoms.

ATTRIBUTES = ('_features', '_numbers')

LABELS = None

compute_kernel(*self*, *b_feats*, *b_nums*, *symmetric=False*)

Compute a kernel between molecules based on atom features.

Parameters

b_feats [list of numpy.array, shape=(n_molecules_b,)]

Each array is of shape (n_atoms, n_features), where n_atoms is for that particular molecule.

b_nums [list of lists, shape=(n_molecules_b,)] Contains all the atom elements for each molecule in group b

symmetric [bool, default=True] Whether or not the kernel is symmetric. This is just to cut the computational cost in half. This is mainly an optimization when computing the (train, train) kernel.

Returns

kernel [numpy.array, shape=(n_molecules_b, n_molecules_fit)] The kernel matrix between the two sets of molecules

fit(*self*, *X*, *y=None*)

Fit the model.

If there is no self.transformer, then this assumes that the input is a list of (features, numbers) pairs where features is a numpy array of features (shape=(n_atoms, n_features)), and numbers is a list of atomic numbers in the molecule.

Otherwise, it directly passes these values to the transformer to compute the features, and extracts all the atomic numbers.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to fit.

Returns

self [object] Returns the instance itself.

fit_transform(*self*, *X*, *y=None*)
A slightly cheaper way of fitting and then transforming.

This benefit comes from the resulting kernel matrix being symmetric. Meaning, that only half of it has to be computed.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to transform

Returns

kernel [array, shape=(n_samples, n_samples)] The resulting kernel matrix

transform(*self*, *X*, *y=None*)
Transform features/molecules into a kernel matrix.

If there is no self.transformer, then this assumes that the input is a list of (features, numbers) pairs where features is a numpy array of features (shape=(n_atoms, n_features)), and numbers is a list of atomic numbers in the molecule.

Otherwise, it directly passes these values to the transformer to compute the features, and extracts all the atomic numbers.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to transform

Returns

kernel [array, shape=(n_samples, n_samples_fit)] The resulting kernel matrix

Raises

ValueError If the transformer has not been fit.

1.1.9 molml.molecule module

A module to compute molecule based representations.

This module contains a variety of methods to extract features from molecules based on the entire molecule. All of the methods included here will produce one vector per molecule input.

class molml.molecule.**Connectivity**(*input_type='list'*, *n_jobs=1*, *depth=1*, *use_bond_order=False*, *use_coordination=False*, *add_unknown=False*, *do_tfidf=False*)
Bases: *molml.base.SetMergeMixin*, *molml.base.BaseFeature*

A collection of feature types based on the connectivity of atoms.

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

depth [int, default=1] The length of the atom chains to generate for connections

use_bond_order [boolean, default=False] Specifies whether or not to use bond order information (C-C versus C=C). Note: for depth=1, this option does nothing.

use_coordination [boolean, default=False] Specifies whether or not to use the coordination number of the atoms (C1 vs C2 vs C3 vs C4).

add_unknown [boolean, default=False] Specifies whether or not to include an extra UNKNOWN count in the feature vector.

do_tfidf [boolean, default=False] Apply weighting to counts based on their inverse document (molecule) frequency.

References

Collins, C.; Gordon, G.; von Lilienfeld, O. A.; Yaron, D. Constant Size Molecular Descriptors For Use With Machine Learning. arXiv:1701.06649

Attributes

_base_groups [tuple, tuples] All the chains that are in the fit molecules.

```
ATTRIBUTES = ('_base_groups', '_idf_values')
```

```
LABELS = (('get_chain_labels', '_base_groups'),)
```

fit(*self*, *X*, *y=None*)
Fit the model.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to fit.

Returns

self [object] Returns the instance itself.

get_chain_labels(*self*, *chains*)

class molml.molecule.**ConnectivityTree**(*input_type='list'*, *n_jobs=1*, *depth=1*, *use_bond_order=False*, *use_coordination=False*, *preserve_paths=False*, *use_parent_element=True*, *add_unknown=False*, *do_tfidf=False*)

Bases: *molml.molecule.Connectivity*

A collection of feature types based on a connectivity tree of atoms.

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

depth [int, default=1] The length of the atom trees to generate for connections

use_bond_order [boolean, default=False] Specifies whether or not to use bond order information (C-C versus C=C). Note: for depth=1, this option does nothing.

use_coordination [boolean, default=False] Specifies whether or not to use the coordination number of the atoms (C1 vs C2 vs C3 vs C4).

preserve_paths [boolean, default=False] Include the local index to the parent node in each tuple. This helps to differentiate elements at the same depth, but with different parents. Note: for depth<3, this option does nothing.

use_parent_element [boolean, default=True] Include the parent nodes element type. This helps to differentiate elements with different parent elements, but not to the same extreme as *preserve_paths*. Note: this does nothing if *use_bond_order* is set as they are redundant.

add_unknown [boolean, default=False] Specifies whether or not to include an extra UNKNOWN count in the feature vector.

do_tfidf [boolean, default=False] Apply weighting to counts based on their inverse document (molecule) frequency.

Attributes

_base_groups [tuple, tuples] All the trees that are in the fit molecules.

ATTRIBUTES = ('_base_groups', '_idf_values')

LABELS = (('get_tree_labels', '_base_groups'),)

get_tree_labels(*self*, *trees*)

class molml.molecule.**Autocorrelation**(*input_type='list'*, *n_jobs=1*, *depths=(0, 1, 2, 3)*, *properties=None*)

Bases: *molml.base.BaseFeature*

A molecular descriptor based on Autocorrelation functions for properties.

This is a compact (only depends on the number of properties used and the number of depths) molecule representation that uses the graph distance between atoms to extract information.

$$V_d = \sum_i \sum_j P_i P_j \delta(d_{ij}, d)$$

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

depths [list/tuple, default=(0, 1, 2, 3)] A list of depths to use for computing the autocorrelations functions.

properties [list/tuple, default=None] A list/tuple of properties to use. Each of these properties should be defined for a single atom in the molecule. Each property can be either a function (that takes in a LazyValues function and returns a vector the with one element per atom) or it can be a one of the following strings ('Z', 'EN', 'CN', 'I', 'R'). Each of these keys corresponds to the atomic number, the electronegativity, coordination number, the identity function (always returns 1), and the covalent radius. If this value is None, then all the predefined properties will be used.

References

Janet, J. P. and Kulik, H. J. Resolving Transition Metal Chemical Space: Feature Selection for Machine Learning and Structure-Property Relationships. J. Phys. Chem. A 2017, 121, 8939-8954

ATTRIBUTES = None

FUNCTIONS = {'CN': <function <lambda>>, 'EN': <function <lambda>>, 'I': <function <lam

LABELS = ('_labels',)

fit (*self*, *X*, *y=None*)
No fitting is required because it is defined by the parameters.

class molml.molecule.**EncodedAngle**(*input_type='list'*, *n_jobs=1*, *segments=40*, *smoothing='norm'*, *slope=20.0*, *min_depth=0*, *max_depth=0*, *form=3*, *r_cut=6.0*, *add_unknown=False*, *use_comb_idxs=False*)

Bases: *molml.base.FormMixin*, *molml.base.SetMergeMixin*, *molml.base.EncodedFeature*

A smoothed histogram of atomic angles.

This method is similar to EncodedBond but for angles in molecules. This is done by enumerating all triplets of atoms and computing the angle between them. The bins are then smoothed with smoothing functions. Note: The angles used are 0 to pi.

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

segments [int, default=40] The number of bins/segments to use when generating the histogram. Empirically, it has been found that there is no benefit to having more than 40-50 segments.

smoothing [string or callable, default='norm'] A string or callable to use to smooth the histogram values. If a callable is given, it must take just a single argument that is a float (or vector of floats). For a list of supported default functions look at SMOOTHING_FUNCTIONS.

slope [float, default=20.] A parameter to tune the smoothing values. This is applied as a multiplication before calling the smoothing function.

min_depth [int, default=0] A parameter to set the minimum geodesic distance to include in the interactions. A value of np.inf signifies including only intermolecular interactions.

max_depth [int, default=0] A parameter to set the maximum geodesic distance to include in the interactions. A value of 0 signifies that all interactions are included.

form [int, default=3] The histogram splitting style to use. This changes the scaling of this method to be O(E^3), O(E^2), O(E), or O(1) for 3, 2, 1, or 0 respectively (where E is the number of elements).

r_cut [float, default=6.] The maximum distance allowed for atoms to be considered local to the "central atom".

add_unknown [boolean, default=False] Specifies whether or not to include an extra UNKNOWN count in the feature vector.

use_comb_idxs [bool, default=False] Whether or not to use all combinations of indices when doing the subselection. If this is false, a middle out scheme will be used.

Attributes

_groups [tuple, tuples] A tuple of all the groups (element chains) in the fit molecules.

ATTRIBUTES = ('_groups',)

LABELS = (('get_encoded_labels', '_groups'),)

f_c(*self*, *R*)

class molml.molecule.**EncodedBond**(*input_type='list'*, *n_jobs=1*, *segments=100*, *smoothing='norm'*, *start=0.2*, *end=6.0*, *slope=20.0*, *min_depth=0*, *max_depth=0*, *spacing='linear'*, *form=2*, *add_unknown=False*, *use_comb_idxs=False*)

Bases: *molml.base.FormMixin*, *molml.base.SetMergeMixin*, *molml.base.EncodedFeature*

A smoothed histogram of atomic distances.

This is a method to generallize the idea of bond counting. Instead of seeing bonds as a discrete count that is thresholded at a given length, they are seen as general distance histograms. This is supplemented with smoothing functions.

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

segments [int, default=100] The number of bins/segments to use when generating the histogram. Empirically, it has been found that values beyond 50-100 have little benefit.

smoothing [string or callable, default='norm'] A string or callable to use to smooth the histogram values. If a callable is given, it must take just a single argument that is a float (or vector of floats). For a list of supported default functions look at SMOOTHING_FUNCTIONS.

start [float, default=0.2] The starting point for the histgram sampling in angstroms.

end [float, default=6.0] The ending point for the histogram sampling in angstroms.

slope [float, default=20.] A parameter to tune the smoothing values. This is applied as a multiplication before calling the smoothing function.

min_depth [int, default=0] A parameter to set the minimum geodesic distance to include in the interactions. A value of np.inf signifies including only intermolecular interactions.

max_depth [int, default=0] A parameter to set the maximum geodesic distance to include in the interactions. A value of 0 signifies that all interactions are included.

spacing [string or callable, default='linear'] The histogram interval spacing type. Must be one of ("linear", "inverse", or "log"). Linear spacing is normal spacing. Inverse takes and evaluates the distances as 1/r and the start and end points are 1/x. For log spacing, the distances are evaluated as numpy.log(r) and the start and end points are numpy.log(x). If the value is callable, then it should take a float or vector of floats and return a similar mapping like the other methods.

form [int, default=2] The histogram splitting style to use. This changes the scaling of this method to be O(E^2), O(E), or O(1) for 2, 1, or 0 respectively (where E is the number of elements).

add_unknown [boolean, default=False] Specifies whether or not to include an extra UNKNOWN count in the feature vector.

use_comb_idxs [bool, default=False] Whether or not to use all combinations of indices when doing the subselection. If this is false, a middle out scheme will be used.

References

Collins, C.; Gordon, G.; von Lilienfeld, O. A.; Yaron, D. Constant Size Molecular Descriptors For Use With Machine Learning. arXiv:1701.06649

Attributes

_element_pairs [tuple, tuples] A tuple of all the element pairs in the fit molecules.

```
ATTRIBUTES = ('_element_pairs',)
```

```
LABELS = (('get_encoded_labels', '_element_pairs'),)
```

class molml.molecule.**CoulombMatrix**(*input_type='list'*, *n_jobs=1*, *sort=False*, *eigen=False*, *drop_values=False*, *only_lower_triangle=False*)

Bases: *molml.base.BaseFeature*

A molecular descriptor based on Coulomb interactions.

This is a feature that uses a Coulomb-like interaction between all atoms in the molecule to generate a matrix that is then vectorized.

$$C_{ij} = \begin{cases} \frac{Z_i Z_j}{\|r_i - r_j\|} & i \neq j \\ 0.5 Z_i^{2.4} & i = j \end{cases}$$

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

sort [bool, default=False] Specifies whether or not to sort the coulomb matrix based on the sum of the rows (same as L1 norm).

eigen [bool, default=False] Specifies whether or not to use the eigen spectrum of the coulomb matrix rather than the matrix itself. This changes the scaling to be linear in the number of atoms.

drop_values [bool, default=False] Specifies whether or not to drop the atoms from molecules larger than the training set. If this value is set to False, and the molecule is too large to transform, the transform will throw a ValueError. If it is set to True, then it will truncate the molecule to only include the first _max_size atoms of the molecule.

only_lower_triangle [bool, default=False] Specifies whether or not to only use the lower triangle of the Coulomb Matrix. This cuts the dimensionality in half by removing the duplicate values in the upper triangle of the matrix. This does nothing if *eigen* is set.

References

Rupp, M.; Tkatchenko, A.; Muller, K.-R.; von Lilienfeld, O. A. Fast and Accurate Modeling of Molecular Atomization Energies with Machine Learning. Phys. Rev. Lett. 2012, 108, 058301.

Hansen, K.; Montavon, G.; Biegler, F.; Fazli, S.; Rupp, M.; Scheffler, M.; von Lilienfeld, O. A.; Tkatchenko, A.; Muller, K.-R. Assessment and Validation of Machine Learning Methods for Predicting Molecular Atomization Energies. J. Chem. Theory Comput. 2013, 9, 3404-3419.

Attributes

_max_size [int] The size of the largest molecule in the fit molecules by number of atoms.

ATTRIBUTES = ('_max_size',)

LABELS = (('get_coulomb_labels', '_max_size'),)

fit(*self*, *X*, *y=None*)
Fit the model.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to fit.

Returns

self [object] Returns the instance itself.

get_coulomb_labels(*self*, *max_size*)

class molml.molecule.**BagOfBonds**(*input_type='list'*, *n_jobs=1*, *drop_values=False*, *add_atoms=False*)
Bases: *molml.base.BaseFeature*

A molecular descriptor that groups interactions from the Coulomb Matrix.

This feature starts the same as the Coulomb Matrix, and then interaction terms of the same element pair are grouped together and then sorted before they are vectorized.

Parameters

input_type [string, default='list'] Specifies the format the input values will be (must be one of 'list' or 'filename').

n_jobs [int, default=1] Specifies the number of processes to create when generating the features. Positive numbers specify a specifc amount, and numbers less than 1 will use the number of cores the computer has.

drop_values [bool, default=False] Specifies whether or not to drop interactions if there are more than was seen in the training set. If this value is set to False, and the molecule is too large to transform, it will throw a ValueError. If it is set to True, then it will truncate that particular bag to only include the largest _bag_sizes[ele1, ele2] of the molecule.

add_atoms [bool, default=False] Adds the diagonal of the Coulomb Matrix to the bags.

References

Hansen, K.; Biegler, F.; Ramakrishnan, R.; Pronobis, W.; von Lilienfeld, O. A.; Muller, K.-R.; Tkatchenko, A. Machine Learning Predictions of Molecular Properties: Accurate Many-body Potentials and Nonlocality in Chemical Space. J. Phys. Chem. Lett. 2015, 6, 2326-2331.

Attributes

_bag_sizes [dict, element pair->int] A dictonary mapping element pairs to the maximum size of that element pair block in all the fit molecules.

ATTRIBUTES = ('_bag_sizes',)

LABELS = (('get_bob_labels', '_bag_sizes'),)

fit(*self*, *X*, *y=None*)
Fit the model.

Parameters

X [list, shape=(n_samples,)] A list of objects to use to fit.

Returns

self [object] Returns the instance itself.

get_bob_labels(*self*, *bag_sizes*)

1.1.10 molml.utils module

A collection of assorted utility functions.

class molml.utils.**IndexMap**(*values*, *depth*, *add_unknown=False*, *use_comb_idxs=False*)
Bases: object

An object to handle dynamic mapping of groups to indices.

The intention of the class is to allow for dynamic subselection from lists to give new mapping groups.

For example, with the a group of values that have length three might be reduced as follows:

('A', 'B', 'C') -> ('A', 'C')

using some predefined index selection. Then, this new reduced value is used to find the index for this new value just like a dict that maps the new shorter values to an int index.

This class also allows handling of groups that are not in the map.

Parameters

values [list of tuples] A collection of values overwhich the mapping will be done.

depth [int] The number of values that are retained in the subselection.

add_unknown [bool, default=False] Whether or not to allocate an UNKNOWN index.

use_comb_idxs [bool, default=False] Whether or not to use all combinations of indices when doing the subselection. If this is False, it will use the old style of trying to select indices from the center of the value outward.

get_idx_iter(*self*, *key*, *other=None*)

static get_index_mapping(*values*, *depth*, *idx_groups*)
Determine the ordering and mapping of feature groups.

Parameters

values [list] A list of possible values.

depth [int] The number of elements to use from each values value.

idx_groups [list of list of int] A list of list of indices to select.

Returns

mapping [dict(key)->int] A dict that gives the mapping index for a given key.

get_value_order(*self*)

is_valid(*self*, *values*)

class molml.utils.**LazyValues**(*connections=None*, *coords=None*, *numbers=None*, *elements=None*, *unit_cell=None*)
Bases: object

An object to store molecule graph properties in a lazy fashion.

This object allows only needing to compute different molecule graph properties if they are needed. The prime example of this being the computation of connections.

Parameters

connections [dict, key->list of keys, default=None] A dictionary edge table with all the bidirectional connections.

numbers [array-like, shape=(n_atoms,), default=None] The atomic numbers of all the atoms.

coords [array-like, shape=(n_atoms, 3), default=None] The xyz coordinates of all the atoms (in angstroms).

elements [array-like, shape=(n_atoms,), default=None] The element symbols of all the atoms.

unit_cell [array-like, shape=(3, 3), default=None] An array of unit cell basis vectors, where the vectors are columns.

Attributes

connections [dict, key->list of keys] A dictionary edge table with all the bidirectional connections. If the initialized value for this was None, then this will be computed from the coords and numbers/elements.

numbers [array, shape=(n_atoms,)] The atomic numbers of all the atoms. If the initialized value for this was None, then this will be computed from the elements.

coords [array, shape=(n_atoms, 3)] The xyz coordinates of all the atoms (in angstroms).

elements [array, shape=(n_atoms,)] The element symbols of all the atoms. If the initialized value for this was None, then this will be computed from the numbers.

unit_cell [array, shape=(3, 3)] An array of unit cell basis vectors, where the vectors are columns.

connections

coords

elements

fill_in_crystal(*self*, *radius=None*, *units=None*)

Duplicate the atoms to form a crystal.

Parameters

radius [float, default=None] Specifies the radius of unit cell points to include

units [list or int, default=None] Specifies the number of unit cells to include on each axis. These will all be equal if it is an int.

Raises

ValueError If radius and units are either both None, or if both are not None.

numbers

unit_cell

molml.utils.**cosine_decay**(*R*, *r_cut=6.0*)

Compute all the cutoff distances.

The cutoff is defined as

$$f_{R_c}(R_{ij}) = \begin{cases} 0.5(\cos(\frac{\pi R_{ij}}{R_c}) + 1), & R_{ij} \leq R_c \\ 0, & otherwise \end{cases}$$

Parameters

R [array, shape=(N_atoms, N_atoms)] A distance matrix for all the atoms (scipy.spatial.cdist)

r_cut [float, default=6.] The maximum distance allowed for atoms to be considered local to the "central atom".

Returns

values [array, shape=(N_atoms, N_atoms)] The new distance matrix with the cutoff function applied

molml.utils.**deslugify**(*string*)

Convert a string to a feature name and its parameters.

Parameters

string [str] The slug string to extract values from.

Returns

name [str]

The name of the class corresponding to the string.

final_params [dict] A dictionary of the feature parameters.

molml.utils.**get_angles**(*coords*)

Get the angles between all triples of coords.

The resulting values are $[0, \pi]$ and all invalid values are NaNs.

Parameters

coords [numpy.array, shape=(n_atoms, n_dim)] An array of all the coordinates.

Returns

res [numpy.array, shape=(n_atoms, n_atoms, n_atoms)] An array the angles of all triples.

molml.utils.**get_bond_type**(*element1*, *element2*, *dist*)

Get the bond type between two elements based on their distance.

If there is no bond, return None.

Parameters

element1 [str] The element of the first atom

element2 [str] The element of the second atom

dist [float] The distance between the two atoms

Returns

——-

key [str] The type of the bond

molml.utils.**get_connections**(*elements1*, *coords1*, *elements2=None*, *coords2=None*)

Return a dictionary edge list

If two sets of elements and coordinates are given, then they will be treated as two disjoint sets of atoms.

Each value is is a tuple of the index of the connecting atom and the bond order as a string. Where the bond order is one of ['1', 'Ar', '2', '3'].

Note: If two sets are given, this returns only the connections from the first set to the second. This is in contrast to returning connections from both directions.

Parameters

elements1 [list] All the elements in set 1.

coords1 [array, shape=(n_atoms, 3)] The coordinates of the atoms in set 1.

elements2 [list, default=None] All the elements in set 2.

coords2 [array, shape=(n_atoms, 3), default=None] The coordinates of the atoms in set 2.

Returns

connections [dict, int->dict] Contains all atoms that are connected to each atom and bond type.

molml.utils.**get_coulomb_matrix**(*numbers*, *coords*, *alpha=1*, *use_decay=False*)

Return the coulomb matrix for the given coords and numbers.

$$C_{ij} = \begin{cases} \frac{Z_i Z_j}{\|r_i - r_j\|^\alpha} & i \neq j \\ \frac{1}{2} Z_i^{2.4} & i = j \end{cases}$$

Parameters

numbers [array-like, shape=(n_atoms,)] The atomic numbers of all the atoms

coords [array-like, shape=(n_atoms, 3)] The xyz coordinates of all the atoms (in angstroms)

alpha [number, default=6] Some value to exponentiate the distance in the coulomb matrix.

use_decay [bool, default=False] This setting defines an extra decay for the values as they get futher away from the "central atom". This is to alleviate issues the arise as atoms enter or leave the cutoff radius.

Returns

top [array, shape=(n_atoms, n_atoms)] The coulomb matrix

molml.utils.**get_depth_threshold_mask_connections**(*connections*, *min_depth=0*, *max_depth=<Mock name='mock.inf' id='139849510067088'>*)

Get the depth threshold mask from connections.

Parameters

connections [dict, index->list of indices] A dictionary that contains lists of all connected atoms.

min_depth [int, default=0] The minimum depth to allow in the masking

max_depth [int, default=numpy.inf] The maximum depth to allow in the masking

Returns

mask [numpy.array, shape=(len(connections), len(connections))] A mask of all the atoms that are less than or equal to *max_depth* away.

molml.utils.**get_dict_func_getter**(*d*, *label=''*)

molml.utils.**get_element_pairs**(*elements*)

Extract all the element pairs in a molecule.

Parameters

elements [list] All the elements in the molecule

Returns

value [list] All the element pairs in the molecule

molml.utils.**get_graph_distance**(*connections*)

Compute the graph distance between all pairs of atoms using Floyd-Warshall

Parameters

connections [dict, index->list of indices] A dictionary that contains lists of all connected atoms.

Returns

dist [numpy.array, shape=(len(connections), len(connections))] The graph distance between all pairs of atoms

molml.utils.**get_smoothing_function**(*key*)

molml.utils.**get_spacing_function**(*key*)

molml.utils.**lerp_smooth**(*x*)

molml.utils.**load_json**(*f*)
Load the model data from a json file

Parameters

f [str or file descriptor] The path to save the data or a file descriptor to save it to.

Returns

obj [Transformer] The transformer object.

molml.utils.**multi_beta**(*f*)

molml.utils.**needs_reversal**(*chain*)
Determine if the chain needs to be reversed.

This is to set the chains such that they are in a canonical ordering

Parameters

chain [tuple] A tuple of elements to treat as a chain

Returns

needs_flip [bool] Whether or not the chain needs to be reversed

molml.utils.**sort_chain**(*chain*)
Sort a chain from the inside out.

Parameters

chain [tuple] A tuple of elements to treat as a chain

Returns

chain [tuple] The sorted chain

1.2 Module contents

1.2.1 MolML

An interface between molecules and machine learning

MolML is a python module to use to map molecules into representations that are usable with machine learning. This is done using an API similar to scikit-learn to keep things simple and straightforward. For documentation, look at the docstrings.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

G

I

L

M

N

R

S

T

U