GraphDot

Release 0.8.1

Dec 08, 2021

Contents

GraphDot is a GPU-accelerated Python library that carries out graph dot product operations to compute graph similarity. Currently, the library implements the Marginalized Graph Kernel algorithm, which uses a random walk process to compare subtree patterns and thus defining a generalized graph convolution process. The library can operate on undirected graphs, either weighted or unweighted, that contain arbitrary nodal and edge labels and attributes. It implements state-of-the-art GPU acceleration algorithms and supports versatile customization through just-in-time code generation and compilation.

CHAPTER 1

Features

- Compares graph with different number of nodes and/or edges.
- Allows user to define arbitrary attributes and custom similarity functions on individual nodes and edges.
- Fast, memory-efficient GPU algorithms for CUDA.
- Compatible with major graph libraries such as NetworkX and graphviz.
- Interoperable with scikit-learn.
- Built-in specialization for chemistry and material science applications.

CHAPTER 2

Contents

2.1 Installation

2.1.1 Prerequisites

GraphDot requires a CUDA Toolkit installation for carrying out GPU computations. To install it, following the instructions on https://developer.nvidia.com/cuda-toolkit.

2.1.2 Installation using pip

GraphDot can be installed from PyPI as simple as:

```
pip install graphdot
```

2.1.3 Install from source

For Ubuntu/Fedora/macOS

```
git clone https://gitlab.com/yhtang/graphdot
cd graphdot
pip3 install -r requirements/common.txt
python3 setup.py install
```

2.2 Examples

2.2.1 Unweighted, Node-Labeled Graph

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
'''An example of similarity comparison between node-labeled but unweighted
graphs using the marginalized graph kernel.'''
import numpy as np
import networkx as nx
from graphdot import Graph
from graphdot.kernel.marginalized import MarginalizedGraphKernel
from graphdot.microkernel import (
    TensorProduct,
    SquareExponential,
    KroneckerDelta,
    Constant
)

# {1.0, 1} -- {2.0, 1}
g1 = nx.Graph()
g1.add_node(0, radius=1.0, category=1)
g1.add_node(1, radius=2.0, category=1)
g1.add_edge(0, 1)

# {1.0, 1} -- {2.0, 1} -- {1.0, 2}
g2 = nx.Graph()
g2.add_node(0, radius=1.0, category=1)
g2.add_node(1, radius=2.0, category=1)
g2.add_node(2, radius=1.0, category=2)
g2.add_edge(0, 1)
g2.add_edge(1, 2)

# {1.0, 1} -- {2.0, 1}
#      \          /
#       {1.0, 2}
g3 = nx.Graph()
g3.add_node(0, radius=1.0, category=1)
g3.add_node(1, radius=2.0, category=1)
g3.add_node(2, radius=1.0, category=2)
g3.add_edge(0, 1)
g3.add_edge(0, 2)
g3.add_edge(1, 2)

# define node and edge kernelets
knode = TensorProduct(radius=SquareExponential(0.5),
                      category=KroneckerDelta(0.5))

kedge = Constant(1.0)

# compose the marginalized graph kernel and compute pairwise similarity
mlgk = MarginalizedGraphKernel(knode, kedge, q=0.05)

R = mlgk([Graph.from_networkx(g) for g in [g1, g2, g3]])

# normalize the similarity matrix
```

(continues on next page)

(continued from previous page)

```
d = np.diag(R)**-0.5
K = np.diag(d).dot(R).dot(np.diag(d))

print(K)
```

Exptected output:

```
[[1.         0.59784909 0.45851714]
 [0.59784909 1.         0.7339707 ]
 [0.45851714 0.7339707  1.        ]]
```

2.2.2 Unweighted, Unlabeled Graph

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
'''An example of similarity comparison between unlabeled and unweighted graphs
using the marginalized graph kernel.

Note: it is known that all unlabeled and unweighted graphs are identical under
the similarity metric defined by the marginalized graph kernel. This example
merely illustrates the usage of the package.'''
import numpy as np
import networkx as nx
from graphdot import Graph
from graphdot.kernel.marginalized import MarginalizedGraphKernel
from graphdot.microkernel import Constant

# 0 -- 1
g1 = nx.Graph()
g1.add_node(0)
g1.add_node(1)
g1.add_edge(0, 1)

# 0 -- 1 -- 2
g2 = nx.Graph()
g2.add_node(0)
g2.add_node(1)
g2.add_node(2)
g2.add_edge(0, 1)
g2.add_edge(1, 2)

# 0 --- 1
#  \  /
#   2
g3 = nx.Graph()
g3.add_node(0)
g3.add_node(1)
g3.add_node(2)
g3.add_edge(0, 1)
g3.add_edge(0, 2)
g3.add_edge(1, 2)

# define trivial node and edge kernelets
knode = Constant(1.0)
kedge = Constant(1.0)
```

(continues on next page)

(continued from previous page)

```

# compose the marginalized graph kernel and compute pairwise similarity
mlgk = MarginalizedGraphKernel(knode, kedge, q=0.05)

R = mlgk([Graph.from_networkx(g) for g in [g1, g2, g3]])

# normalize the similarity matrix
d = np.diag(R)**-0.5
K = np.diag(d).dot(R).dot(np.diag(d))

# all entries should be approximately 1 plus round-off error
print(K)
```

Exptected output:

```
[[1.         0.99999963 1.00000005]
 [0.99999963 1.         0.99999958]
 [1.00000005 0.99999958 1.        ]]
```

2.2.3 Weighted, Node-Labeled Graph

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
'''An example of similarity comparison between node-labeled and edge-weighted
graphs using the marginalized graph kernel.'''
import numpy as np
import networkx as nx
from graphdot import Graph
from graphdot.kernel.marginalized import MarginalizedGraphKernel
from graphdot.microkernel import (
    TensorProduct,
    SquareExponential,
    KroneckerDelta,
    Constant
)

# {1.0, 1} --[1.0]-- {2.0, 1}
g1 = nx.Graph()
g1.add_node(0, radius=1.0, category=1)
g1.add_node(1, radius=2.0, category=1)
g1.add_edge(0, 1, w=1.0)

# {1.0, 1} --[1.0]-- {2.0, 1} --[2.0]-- {1.0, 2}
g2 = nx.Graph()
g2.add_node(0, radius=1.0, category=1)
g2.add_node(1, radius=2.0, category=1)
g2.add_node(2, radius=1.0, category=2)
g2.add_edge(0, 1, w=1.0)
g2.add_edge(1, 2, w=2.0)

# {1.0, 1} --[1.0]-- {2.0, 1}
#       \               /
#      [0.5]         [2.0]
#         \           /
#          {1.0, 2}
```

(continues on next page)

(continued from previous page)

```
g3 = nx.Graph()
g3.add_node(0, radius=1.0, category=1)
g3.add_node(1, radius=2.0, category=1)
g3.add_node(2, radius=1.0, category=2)
g3.add_edge(0, 1, w=1.0)
g3.add_edge(0, 2, w=0.5)
g3.add_edge(1, 2, w=2.0)

# define node and edge kernelets
knode = TensorProduct(radius=SquareExponential(0.5),
                      category=KroneckerDelta(0.5))

kedge = Constant(1.0)

# compose the marginalized graph kernel and compute pairwise similarity
mlgk = MarginalizedGraphKernel(knode, kedge, q=0.05)

R = mlgk([Graph.from_networkx(g, weight='w') for g in [g1, g2, g3]])

# normalize the similarity matrix
d = np.diag(R)**-0.5
K = np.diag(d).dot(R).dot(np.diag(d))

print(K)
```

Exptected output:

```
[[1.         0.48829836 0.47495925]
 [0.48829836 1.         0.8931447 ]
 [0.47495925 0.8931447  1.        ]]
```

2.2.4 Weighted, Node- and Edge-Labeled Graph

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
'''An example of similarity comparison between both node- and edge-labeled,
weighted graphs using the marginalized graph kernel.

This is the scenario that takes full advantage of the marginalized graph
kernel frame.
'''
import numpy as np
import networkx as nx
from graphdot import Graph
from graphdot.kernel.marginalized import MarginalizedGraphKernel
from graphdot.kernel.basekernel import TensorProduct
from graphdot.kernel.basekernel import SquareExponential
from graphdot.kernel.basekernel import KroneckerDelta

# {1.0, 1} --{1.5}[1.0]-- {2.0, 1}
g1 = nx.Graph()
g1.add_node(0, radius=1.0, category=1)
g1.add_node(1, radius=2.0, category=1)
g1.add_edge(0, 1, w=1.0, length=1.5)

```

(continues on next page)

(continued from previous page)

```
# {1.0, 1} --{1.5}[1.0]-- {2.0, 1} --{2.0}[2.0]-- {1.0, 2}
g2 = nx.Graph()
g2.add_node(0, radius=1.0, category=1)
g2.add_node(1, radius=2.0, category=1)
g2.add_node(2, radius=1.0, category=2)
g2.add_edge(0, 1, w=1.0, length=1.5)
g2.add_edge(1, 2, w=2.0, length=2.0)

# {1.0, 1} --{1.5}[1.0]-- {2.0, 1}
#     \                     /
#      \                   /
#   {2.0}[0.5]      1.0}[2.0]
#        \           /
#         \         /
#          {1.0, 2}
g3 = nx.Graph()
g3.add_node(0, radius=1.0, category=1)
g3.add_node(1, radius=2.0, category=1)
g3.add_node(2, radius=1.0, category=2)
g3.add_edge(0, 1, w=1.0, length=1.5)
g3.add_edge(0, 2, w=0.5, length=2.0)
g3.add_edge(1, 2, w=2.0, length=1.0)

# define node and edge kernelets
knode = TensorProduct(radius=SquareExponential(1.0),
                      category=KroneckerDelta(0.5))

kedge = TensorProduct(length=SquareExponential(1.0))

# compose the marginalized graph kernel and compute pairwise similarity
mlgk = MarginalizedGraphKernel(knode, kedge, q=0.05)

R = mlgk([Graph.from_networkx(g, weight='w') for g in [g1, g2, g3]])

# normalize the similarity matrix
d = np.diag(R)**-0.5
K = np.diag(d).dot(R).dot(np.diag(d))

print(K)
```

Exptected output:

```
[[1.         0.50970827 0.52433483]
 [0.50970827 1.         0.71869168]
 [0.52433483 0.71869168 1.        ]]
```

2.2.5 2D Molecular Graph

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
""" Similarity comparison between molecules whose formula are specified using
the OpenSMILES format.

Since a SMILES string is intrinsically a graph representation of a molecule,
it can be easily used by the marginalized graph kernel.
```

(continues on next page)

(continued from previous page)

```
"""
import numpy as np
import pandas as pd
from graphdot import Graph
from graphdot.kernel.marginalized import MarginalizedGraphKernel
from graphdot.microkernel import (
    TensorProduct,
    SquareExponential,
    KroneckerDelta
)

# build sample molecules
smiles_list = [
    'CC',  # ethane
    'CCO',  # acetic acid
    'CCN',  # ethylamine
    'C=C',  # ethene
    'CC=C',  # propene
    'CC=CC',  # 2-n-butene
]

# convert to molecular graphs
# nodes(atoms) has 'aromatic', 'charge', 'element', 'hcount' attributes
# edges(bonds) has the 'order' attribute
graphs = [Graph.from_smiles(smi) for smi in smiles_list]

# define node and edge kernelets
knode = TensorProduct(aromatic=KroneckerDelta(0.8),
                      charge=SquareExponential(1.0),
                      element=KroneckerDelta(0.5),
                      hcount=SquareExponential(1.0))

kedge = TensorProduct(order=KroneckerDelta(0.5))

# compose the marginalized graph kernel and compute pairwise similarity
kernel = MarginalizedGraphKernel(knode, kedge, q=0.05)

R = kernel(graphs)

# normalize the similarity matrix and then print
d = np.diag(R)**-0.5
K = np.diag(d).dot(R).dot(np.diag(d))

print(pd.DataFrame(K, columns=smiles_list, index=smiles_list))
```

Exptected output:

```
              CC       CCO       CCN       C=C      CC=C     CC=CC
CC      1.000000  0.301240  0.320140  0.081422  0.168009  0.184527
CCO     0.301240  1.000000  0.688769  0.242019  0.533106  0.565582
CCN     0.320140  0.688769  1.000000  0.234430  0.473560  0.484795
C=C     0.081422  0.242019  0.234430  1.000000  0.361879  0.246465
CC=C    0.168009  0.533106  0.473560  0.361879  1.000000  0.827114
CC=CC   0.184527  0.565582  0.484795  0.246465  0.827114  1.000000
```

2.2.6 3D Molecular Graph

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
""" Similarity comparison between molecular configurations in 3D.

The molecules are first converted to molecular graphs using an 'adjacency rule'
as described in Tang & de Jong https://doi.org/10.1063/1.5078640, then computed
using the marginalized graph kernel.
"""
import numpy as np
import pandas as pd
from ase.build import molecule, bulk
from graphdot import Graph
from graphdot.kernel.molecular import Tang2019MolecularKernel

# build sample molecules
small_title = ['H2O', 'HCl', 'NaCl']
bulk_title = ['NaCl-bulk', 'NaCl-bulk2']
bulk = [
    bulk('NaCl', 'rocksalt', a=5.64),
    bulk('NaCl', 'rocksalt', a=5.66),
]
molecules = [molecule(name) for name in small_title] + bulk

# convert to molecular graphs
graphs = [Graph.from_ase(m) for m in molecules]

# use pre-defined molecular kernel
kernel = Tang2019MolecularKernel(edge_length_scale=0.1)

R = kernel(graphs)

# normalize the similarity matrix
d = np.diag(R)**-0.5
K = np.diag(d).dot(R).dot(np.diag(d))

# note the difference between the NaCl variants
title = small_title + bulk_title
print(pd.DataFrame(K, columns=title, index=title))
```

Exptected output:

```
                H2O       HCl       NaCl  NaCl-bulk  NaCl-bulk2
H2O        1.000000  0.073903  0.031434   0.031434    0.031434
HCl        0.073903  1.000000  0.015842   0.015842    0.015841
NaCl       0.031434  0.015842  1.000000   0.023764    0.023764
NaCl-bulk  0.031434  0.015842  0.023764   1.000000    0.803760
NaCl-bulk2 0.031434  0.015841  0.023764   0.803760    1.000000
```

2.2.7 Graphs with Variable-Length Node and Edge Features

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
'''An example of similarity comparison between graphs whose node or edge labels
```

(continues on next page)

(continued from previous page)

```
are variable-length sequences rather than scalars using the marginalized graph
kernel.'''
import numpy as np
import networkx as nx
from graphdot import Graph
from graphdot.kernel.marginalized import MarginalizedGraphKernel
from graphdot.microkernel import (
    TensorProduct,
    Convolution,
    SquareExponential,
    KroneckerDelta
)

# The 'category' attribute on the nodes could have variable lengths.
# So does the 'spectra' attributes on the edges.
g1 = nx.Graph()
g1.add_node(0, category=(1, 2), symbol=1)
g1.add_node(1, category=(2,), symbol=2)
g1.add_edge(0, 1, w=1.0, spectra=[0.5, 0.2])

g2 = nx.Graph()
g2.add_node(0, category=(1, 3), symbol=1)
g2.add_node(1, category=(2, 3, 5), symbol=2)
g2.add_node(2, category=(1,), symbol=1)
g2.add_edge(0, 1, w=2.0, spectra=[0.1, 0.9, 1.5])
g2.add_edge(0, 2, w=0.5, spectra=[0.4])
g2.add_edge(1, 2, w=0.5, spectra=[0.3, 0.6])

# Define node and edge base kernels using the R-convolution framework
# Reference: Haussler, David. Convolution kernels on discrete structures. 1999.
knode = TensorProduct(
    symbol=KroneckerDelta(0.5),
    category=Convolution(
        KroneckerDelta(0.5)
    )
)

kedge = TensorProduct(
    spectra=Convolution(
        SquareExponential(0.3)
    )
)

# compose the marginalized graph kernel and compute pairwise similarity
mlgk = MarginalizedGraphKernel(knode, kedge, q=0.05)

R = mlgk([Graph.from_networkx(g, weight='w') for g in [g1, g2]])

# normalize the similarity matrix
d = np.diag(R)**-0.5
K = np.diag(d).dot(R).dot(np.diag(d))

print(K)
```

Exptected output:

```
[[1.         0.33337701]
 [0.33337701 1.        ]]
```

2.3 Quick Start Tutorial

2.3.1 Marginalized Graph kernel

This quick-start guide here assumes that the reader is already familiar with the marginalized graph kernel algorithm [Kashima, H., Tsuda, K., & Inokuchi, A. (2003). Marginalized kernels between labeled graphs. *In Proceedings of the 20th international conference on machine learning (ICML-03)* (pp. 321-328).]. Otherwise, please refer to *A Short Tutorial on the Marginalized Graph Kernel*.

Graphs can be imported from a variety of formats, such as a networkx undirected Graph, an ASE atoms collection, a pymatgen structure or molecule, a SMILES string.

To compare the overall similarity between two graphs, the user needs to supply two base kernels: one for comparing individual nodes, and another one for comparing individual edges. The base kernels can be picked from a library of prebuilt kernels as defined in the graphdot.kernel.basekernel module:

`graphdot.kernel.basekernel.`**`Constant`**(*c*, *c_bounds='fixed'*)

Creates a no-op microkernel that returns a constant value, i.e. $k_{\mathrm{c}}(\cdot,\cdot) \equiv constant$. This kernel is often mutliplied with other microkernels as an adjustable weight.

Parameters **`c`** (*`float > 0`*) – The constant value.

`graphdot.kernel.basekernel.`**`KroneckerDelta`**(*h*, *h_bounds=(0.001, 1)*)

Creates a Kronecker delta microkernel that returns either 1 or h depending on whether two features compare equal, i.e. $k_\delta(i,j) = \begin{cases} 1, i = j \\ h, otherwise \end{cases}$.

Parameters

- **`h`** (*`float in (0, 1)`*) – The value of the microkernel when two features do not compare equal.
- **`h_bounds`** (*`tuple or "fixed"`*) – If tuple, contains the lower and upper bounds that *h* is allowed to vary during hyperparameter optimization. If "fixed", the hyperparameter will not be optimized during training.

`graphdot.kernel.basekernel.`**`SquareExponential`**(**args*, ***kwargs*)

A square exponential kernel smoothly transitions from 1 to 0 as the distance between two vectors increases from zero to infinity, i.e. $k_{\mathrm{se}}(\mathbf{x},\mathbf{y}) = \exp(-\frac{1}{2}\frac{\|\mathbf{x}-\mathbf{y}\|^2}{\sigma^2})$

Parameters

- **`length_scale`** (*`float32`*) – Determines how quickly should the kernel decay to zero. The kernel has a value of approx. 0.606 at one length scale, 0.135 at two length scales, and 0.011 at three length scales.
- **`length_scale_bounds`** (*`tuple or "fixed"`*) – Lower and upper bounds of *length_scale* with respect to hyperparameter optimization. If "fixed", the hyperparameter will not be optimized during training.

`graphdot.kernel.basekernel.`**`TensorProduct`**(***kw_kernels*)

Alias of *Composite('*', **kw_kernels)*. $k_\otimes(X,Y) = \prod_{a\in\text{features}} k_a(X_a, Y_a)$

2.4 A Short Tutorial on the Marginalized Graph Kernel

In the framework of the marginalized graph kernel algorithm [Kashima, H., Tsuda, K., & Inokuchi, A. (2003). Marginalized kernels between labeled graphs. *In Proceedings of the 20th international conference on machine learning (ICML-03)* (pp. 321-328).], the similarity between two graphs is computed as the expectation of individual similarities of pairs of paths generated from a random walk process.

2.5 API Reference

2.5.1 graphdot.graph package

GraphDot's native graph container class

This module defines the class `Graph` that are used to store graphs across this library, and provides conversion and importing methods from popular graph formats.

class graphdot.graph.**Graph**(*nodes*, *edges*, *title=''*)

Bases: object

This is the class that stores a graph in GraphDot.

Parameters

- **nodes** (*dataframe*) – each row represent a node
- **edges** (*dataframe*) – each row represent an edge
- **title** (*str*) – a unique identifier of the graph

adjacency_matrix

Get the adjacency matrix of the graph as a sparse matrix.

Returns **adjacency_matrix** – The adjacency matrix, either weighted or unweighted depending on the original graph.

Return type sparse matrix

cookie

copy(*deep=False*)

Make a copy of an existing graph.

Parameters **deep** (*boolean*) – If deep=True, then real copies will be made for the node and edge dataframes as well as other user-specified attributes. Otherwise, only references to the dataframe columns and user-specified attributes will be inserted into the new graph.

Returns **g** – A new graph.

Return type Graph

classmethod from_ase(*atoms*, *adjacency='default'*, *use_charge=False*, *use_pbc=True*)

Convert from ASE atoms to molecular graph

Parameters

- **atoms** (*ASE Atoms object*) – A molecule as represented by a collection of atoms in 3D space.
- **usb_pbc** (*boolean or list of 3 booleans*) – Whether to use the periodic boundary condition as specified in the atoms object to create edges between atoms.

- **adjacency** (*'default' or object*) – A functor that implements the rule for making edges between atoms.

Returns a molecular graph where atoms become nodes while edges resemble short-range interatomic interactions.

Return type graphdot.Graph

classmethod from_networkx(*graph*, *weight=None*)
Convert from NetworkX `Graph`

Parameters

- **graph** (a NetworkX `Graph` instance) – an undirected graph with homogeneous node and edge features, i.e. carrying same features.
- **weight** (*str*) – name of the attribute that encode edge weights

Returns the converted graph

Return type *graphdot.graph.Graph*

classmethod from_pymatgen(*molecule*, *use_pbc=True*, *adjacency='default'*)
Convert from pymatgen molecule to molecular graph

Parameters

- **molecule** (*pymatgen Molecule object*) – A molecule as represented by a collection of atoms in 3D space.
- **usb_pbc** (*boolean or list of 3 booleans*) – Whether to use the periodic boundary condition as specified in the atoms object to create edges between atoms.
- **adjacency** (*'default' or object*) – A functor that implements the rule for making edges between atoms.

Returns A molecular graph where atoms become nodes while edges resemble short-range interatomic interactions.

Return type graphdot.Graph

classmethod from_rdkit(*mol*, *title=None*, *bond_type='order'*, *set_ring_list=True*, *set_ring_stereo=True*)
Convert a RDKit molecule to a graph

Parameters

- **bond_type** (*'order' or 'type'*) – If 'order', an edge attribute 'order' will be populated with numeric values such as 1 for single bonds, 2 for double bonds, and 1.5 for aromatic bonds. If 'type', an attribute 'type' will be populated with `rdkit.Chem.BondType` values.
- **set_ring_list** (*bool*) – if True, a nodal attribute 'ring_list' will be used to store a list of the size of the rings that the atom participates in.
- **set_ring_stereo** (*bool*) – If True, an edge attribute 'ring_stereo' will be used to store the E-Z stereo configuration of substitutes at the end of each bond along a ring.

Returns A graph where nodes represent atoms and edges represent bonds. Each node and edge carries an array of features as inferred from the chemical structure of the molecule.

Return type graphdot.Graph

classmethod from_smiles(*smiles*)
DEPRECATED and replaced by from_rdkit.

static has_unified_types(*graphs*)
Check if all graphs have the same set of nodal/edge features.

laplacian
Get the graph Laplacian as a sparse matrix.

Returns laplacian – The laplacian matrix, either weighted or unweighted depending on the original graph.

Return type sparse matrix

permute(*perm*, *inplace=False*)
Rearrange the node indices of a graph by a permutation array.

Parameters

- **perm** (*sequence*) – Array of permuted node indices
- **inplace** (*boolean*) – Whether to reorder the nodes in-place or to create a new graph.

Returns permuted_graph – The original graph object (inplace=True) or a new one (inplace=False) with the nodes permuted.

Return type Graph

to_networkx()
Convert the graph to a NetworkX `Graph` and copy the node and edge attributes.

classmethod unify_datatype(*graphs*, *inplace=False*)
Ensure that each attribute has the same data type across graphs.

Parameters

- **graphs** (*list*) – A list of graphs that have the same set of node and edge features. The types for each attribute will then be chosen to be the smallest scalar type that can safely hold all the values as found across the graphs.
- **inplace** (*bool*) – Whether or not to modify the graph features in-place.

Returns If inplace is True, the graphs will be modified in-place and nothing will be returned. Otherwise, a new list of graphs with type-unified features will be returned.

Return type None or list

Subpackages

graphdot.graph.adjacency package

class graphdot.graph.adjacency.**Gaussian**
Bases: `object`

__call__(*d*, *length_scale*)
Call self as a function.

cutoff(*length_scale*)

class graphdot.graph.adjacency.**Tent**(*ord*)
Bases: `object`

__call__(*d*, *length_scale*)
Call self as a function.

cutoff(*length_scale*)

class graphdot.graph.adjacency.**AtomicAdjacency**(*shape='tent1'*, *length_scale='vdw_radius'*, *zoom=1.0*)

Bases: object

Converts interatomic distances into edge weights using the equation $a(i,j) = w(\frac{\|\mathbf{r}_{ij}\|}{\sigma_{ij}})$, where w is a weight function that generally decays with distance, and σ_{ij} is a length scale parameter betweens atom i and j and loosely corresponds to the typically distance of interatomic interactions between the atoms.

Parameters

- **shape** (*str or callable*) – If string, must match one of the following patterns:
  - tent[n]: e.g. tent1, teng2, etc. *Tent*.
  - gaussian: *Gaussian*.
  - compactbell[a,b]: e.g. compactbell4,2, CompactBell.
- **length_scale** (*str*) – The atomic property to be used to determine the range and strength of edges to be constructed between pairs of atoms. The strength will generally fall to zero at roughly a distance 3 times the length scale. Possible values are:
  - **atomic_radius**
  - atomic_radius_rahm
  - **vdw_radius** (default)
  - vdw_radius_bondi
  - vdw_radius_truhlar
  - vdw_radius_rt
  - vdw_radius_batsanov
  - vdw_radius_dreiding
  - vdw_radius_uff
  - vdw_radius_mm3
  - vdw_radius_alvarez
  - covalent_radius_cordero
  - covalent_radius_pyykko
  - covalent_radius_bragg
  - covalent_radius_pyykko_double
  - covalent_radius_pyykko_triple
  - metallic_radius
  - metallic_radius_c12
- **zoom** (*float*) – A zooming factor to be multiplied with the length scales to extend the range of interactions.

__call__(*n1*, *n2*, *r*)

compute adjacency between atoms

Parameters

- **n1** (*int*) – Atomic number of the element
- **n2** (*int*) – Same as n1

- **r** (*float*) – Distance between the two atoms

Returns A non-negative weight

Return type float

cutoff(*elements*)

Submodules

graphdot.graph.adjacency.atomic module

class graphdot.graph.adjacency.atomic.**AtomicAdjacency**(*shape='tent1'*, *length_scale='vdw_radius'*, *zoom=1.0*)

Bases: object

Converts interatomic distances into edge weights using the equation $a(i,j) = w(\frac{\|\mathbf{r}_{ij}\|}{\sigma_{ij}})$, where w is a weight function that generally decays with distance, and σ_{ij} is a length scale parameter betweens atom i and j and loosely corresponds to the typically distance of interatomic interactions between the atoms.

Parameters

- **shape** (*str or callable*) – If string, must match one of the following patterns:
  - tent[n]: e.g. tent1, teng2, etc. Tent.
  - gaussian: Gaussian.
  - compactbell[a,b]: e.g. compactbell4,2, CompactBell.
- **length_scale** (*str*) – The atomic property to be used to determine the range and strength of edges to be constructed between pairs of atoms. The strength will generally fall to zero at roughly a distance 3 times the length scale. Possible values are:
  - **atomic_radius**
  - atomic_radius_rahm
  - **vdw_radius** (default)
  - vdw_radius_bondi
  - vdw_radius_truhlar
  - vdw_radius_rt
  - vdw_radius_batsanov
  - vdw_radius_dreiding
  - vdw_radius_uff
  - vdw_radius_mm3
  - vdw_radius_alvarez
  - covalent_radius_cordero
  - covalent_radius_pyykko
  - covalent_radius_bragg
  - covalent_radius_pyykko_double
  - covalent_radius_pyykko_triple

- metallic_radius
- metallic_radius_c12

- **zoom** (*float*) – A zooming factor to be multiplied with the length scales to extend the range of interactions.

__call__(*n1*, *n2*, *r*)
compute adjacency between atoms

Parameters

- **n1** (*int*) – Atomic number of the element
- **n2** (*int*) – Same as n1
- **r** (*float*) – Distance between the two atoms

Returns A non-negative weight

Return type float

cutoff(*elements*)

graphdot.graph.adjacency.atomic.**copying_lru_cache**(**args*, ***kwargs*)

graphdot.graph.adjacency.atomic.**get_length_scales**(**args*, ***kwargs*)

graphdot.graph.adjacency.atomic.**get_ptable**(**args*, ***kwargs*)

graphdot.graph.adjacency.euclidean module

Convert spatial distances between nodes to edge adjacencies

class graphdot.graph.adjacency.euclidean.**CompactBell**(*a*, *b*)
Bases: object

__call__(*d*, *length_scale*)
Call self as a function.

cutoff(*length_scale*)

class graphdot.graph.adjacency.euclidean.**Gaussian**
Bases: object

__call__(*d*, *length_scale*)
Call self as a function.

cutoff(*length_scale*)

class graphdot.graph.adjacency.euclidean.**Tent**(*ord*)
Bases: object

__call__(*d*, *length_scale*)
Call self as a function.

cutoff(*length_scale*)

graphdot.graph.reorder package

graphdot.graph.reorder.**rcm**(*g*)
Compute the reverse Cuthill-Mckee permutation of a graph. Note that the method does NOT modify the graph, but rather just returns a permutation vector that can be used by Graph.permute to achieve the actual reordering.

Parameters **g** (*Graph*) – The graph to be reordered.

Returns **perm** – Array of permuted node indices.

Return type numpy.ndarray

graphdot.graph.reorder.**pbr**(*g*, *partitioner=None*)

Compute a partition-based permutation of a graph that minimizes the number of cross-tile messages. Note that the method does NOT modify the graph, but rather just returns a permutation vector that can be used by Graph.permute to achieve the actual reordering.

Parameters **g** (*graphdot.Graph*) – The graph to be reordered.

Returns **perm** – Array of permuted node indices.

Return type numpy.ndarray

Subpackages

graphdot.graph.reorder.pbr package

graphdot.graph.reorder.pbr.**pbr**(*g*, *partitioner=None*)

Compute a partition-based permutation of a graph that minimizes the number of cross-tile messages. Note that the method does NOT modify the graph, but rather just returns a permutation vector that can be used by Graph.permute to achieve the actual reordering.

Parameters **g** (*graphdot.Graph*) – The graph to be reordered.

Returns **perm** – Array of permuted node indices.

Return type numpy.ndarray

Submodules

graphdot.graph.reorder.pbr.colnet_hypergraph module

class graphdot.graph.reorder.pbr.colnet_hypergraph.**ColnetHygr**(*unitVertexWeights=False*)

Bases: *graphdot.graph.reorder.pbr.hypergraph.Hygr*

Standard colnet hypergraph from matrix market file, or any other means. Supports symmetric and non-symmetric matrices.

createFromPairs(*row_ids*, *col_ids*, *nr*, *nc*)

graphdot.graph.reorder.pbr.config module

graphdot.graph.reorder.pbr.config.**to_ini**(*config=None*)

Converts a dictionary-based configuration set to an .ini file so that it can be loaded by Kahypar's C extension module.

Parameters **config** (*dict*) – Key-value pairs as appears in https://github.com/kahypar/kahypar/tree/master/config.

graphdot.graph.reorder.pbr.hypergraph module

class graphdot.graph.reorder.pbr.hypergraph.**Hygr**(*unitVertexWeights=False*)
Bases: object

A generic hypergraph format. Uses 0-based indexing.

graphdot.graph.reorder.pbr.mnom module

class graphdot.graph.reorder.pbr.mnom.**PbrMnom**(*tilesize=8*, *mnc=100*, *addMsgNets=True*, *config=None*)
Bases: object

Partitioning-based reordering.

Parameters

- **tilesize** (*int, default=8*) – Size of the tile to be used for partitioning
- **mnc** (*int, default=100*) – Message net cost. The higher the value, the more aggressive the will try to minimize the number of nonempty tiles.
- **addMsgNets** (*bool, default=True*) – Whether to add message nets to minimize the number of nonempty tiles. Should be True in most cases.

__call__(*row_ids*, *col_ids*, *nrow*, *ncol*)
Reorder a graph (as represented by a symmetric sparse matrix) using PBR and return the permutation array.

Parameters

- **row_ids** (*sequence*) – Row indices of the non-zero elements.
- **col_ids** (*sequence*) – Column indices of the non-zero elements.
- **nrow** (*int*) – Number of rows
- **ncol** (*int*) – Number of columns

Returns **perm** – Array of permuted row/column indices.

Return type ndarray

partition_hygr(*h*)

Submodules

graphdot.graph.reorder.rcm module

graphdot.graph.reorder.rcm.**rcm**(*g*)
Compute the reverse Cuthill-Mckee permutation of a graph. Note that the method does NOT modify the graph, but rather just returns a permutation vector that can be used by Graph.permute to achieve the actual reordering.

Parameters **g** (*Graph*) – The graph to be reordered.

Returns **perm** – Array of permuted node indices.

Return type numpy.ndarray

2.5.2 graphdot.kernel package

class graphdot.kernel.**Tang2019MolecularKernel**(*stopping_probability=0.01*, *starting_probability=1.0*, *element_prior=0.2*, *edge_length_scale=0.05*, ***kwargs*)

Bases: object

A margianlized graph kernel for **3D molecular structures** as in: Tang, Y. H., & de Jong, W. A. (2019). Prediction of atomization energy using graph kernel and active learning. *The Journal of chemical physics*, 150(4), 044107. The kernel can be directly used together with Graph.from_ase() to operate on molecular structures.

Parameters

- **stopping_probability** (*float in (0, 1)*) – The probability for the random walk to stop during each step.
- **starting_probability** (*float*) – The probability for the random walk to start from any node. See the *p* kwarg of *graphdot.kernel.marginalized.MarginalizedGraphKernel*
- **element_prior** (*float in (0, 1)*) – The baseline similarity between distinct elements — an element always have a similarity 1 to itself.
- **edge_length_scale** (*float in (0, inf)*) – length scale of the Gaussian kernel on edge length. A rule of thumb is that the similarity decays smoothly from 1 to nearly 0 around three times of the length scale.

__call__(*X*, *Y=None*, ***kwargs*)
Same call signature as *graphdot.kernel.marginalized.MarginalizedGraphKernel.__call__()*

bounds

clone_with_theta(*theta*)

diag(*X*, ***kwargs*)
Same call signature as *graphdot.kernel.marginalized.MarginalizedGraphKernel.diag()*

hyperparameter_bounds

hyperparameters

theta

class graphdot.kernel.**KernelOverMetric**(*distance*, *expr*, *x*, ***hyperparameters*)

Bases: object

__call__(*X*, *Y=None*, *eval_gradient=False*)
Call self as a function.

bounds

clone_with_theta(*theta=None*)

diag(*X*)

get_params()

hyperparameters

theta

class graphdot.kernel.**MarginalizedGraphKernel**(*node_kernel*, *edge_kernel*, *p=1.0*, *q=0.01*, *q_bounds=(0.0001, 0.9999)*, *eps=0.01*, *ftol=1e-08*, *gtol=1e-06*, *dtype=<class 'float'>*, *backend='auto'*)

Bases: `object`

Implements the random walk-based graph similarity kernel as proposed in: Kashima, H., Tsuda, K., & Inokuchi, A. (2003). Marginalized kernels between labeled graphs. *In Proceedings of the 20th international conference on machine learning (ICML-03)* (pp. 321-328).

Parameters

- **node_kernel** (*microkernel*) – A kernelet that computes the similarity between individual nodes
- **edge_kernel** (*microkernel*) – A kernelet that computes the similarity between individual edge
- **p** (positive number (default=1.0) or `StartingProbability`) – The starting probability of the random walk on each node. Must be either a positive number or a concrete subclass instance of `StartingProbability`.
- **q** (*float in (0, 1)*) – The probability for the random walk to stop during each step.
- **q_bounds** (*pair of floats*) – The lower and upper bound that the stopping probability can vary during hyperparameter optimization.
- **eps** (*float*) – The step size used for finite difference approximation of the gradient. Only used for nodal matrices (`nodal=True`).
- **dtype** (*numpy dtype*) – The data type of the kernel matrix to be returned.
- **backend** (*'auto' or 'cuda' or an instance of*) –

:param graphdot.kernel.marginalized.Backend.: The computing engine that solves the marginalized graph ke
generalized Laplacian equation.

__call__(*X*, *Y=None*, *eval_gradient=False*, *nodal=False*, *lmin=0*, *timing=False*)
Compute pairwise similarity matrix between graphs

Parameters

- **X** (*list of N graphs*) – The graphs must all have same node and edge attributes.
- **Y** (*None or list of M graphs*) – The graphs must all have same node and edge attributes.
- **eval_gradient** (*Boolean*) – If True, computes the gradient of the kernel matrix with respect to hyperparameters and return it alongside the kernel matrix.
- **nodal** (*bool*) – If True, return node-wise similarities; otherwise, return graphwise similarities.
- **lmin** (*0 or 1*) – Number of steps to skip in each random walk path before similarity is computed. (lmin + 1) corresponds to the starting value of l in the summation of Eq. 1 in Tang & de Jong, 2019 https://doi.org/10.1063/1.5078640 (or the first unnumbered equation in Section 3.3 of Kashima, Tsuda, and Inokuchi, 2003).

Returns

- **kernel_matrix** (*ndarray*) – if Y is None, return a square matrix containing pairwise similarities between the graphs in X; otherwise, returns a matrix containing similarities across graphs in X and Y.

- **gradient** (*ndarray*) – The gradient of the kernel matrix with respect to kernel hyperparameters. Only returned if eval_gradient is True.

active_theta_mask

bounds
The logarithms of a reshaped X-by-2 array of kernel hyperparameter bounds, excluing those declared as 'fixed' or those with equal lower and upper bounds.

clone_with_theta(*theta*)

diag(*X*, *eval_gradient=False*, *nodal=False*, *lmin=0*, *active_theta_only=True*, *timing=False*)
Compute the self-similarities for a list of graphs

Parameters

- **X** (*list of N graphs*) – The graphs must all have same node attributes and edge attributes.
- **eval_gradient** (*Boolean*) – If True, computes the gradient of the kernel matrix with respect to hyperparameters and return it alongside the kernel matrix.
- **nodal** (*bool*) – If True, returns a vector containing nodal self similarties; if False, returns a vector containing graphs' overall self similarities; if 'block', return a list of square matrices which forms a block-diagonal matrix, where each diagonal block represents the pairwise nodal similarities within a graph.
- **lmin** (*0 or 1*) – Number of steps to skip in each random walk path before similarity is computed. (lmin + 1) corresponds to the starting value of l in the summation of Eq. 1 in Tang & de Jong, 2019 https://doi.org/10.1063/1.5078640 (or the first unnumbered equation in Section 3.3 of Kashima, Tsuda, and Inokuchi, 2003).
- **active_theta_only** (*bool*) – Whether or not to return only gradients with regard to the non-fixed hyperparameters.

Returns

- **diagonal** (*numpy.array or list of np.array(s)*) – If nodal=True, returns a vector containing nodal self similarties; if nodal=False, returns a vector containing graphs' overall self similarities; if nodal = 'block', return a list of square matrices, each being a pairwise nodal similarity matrix within a graph.
- *gradient* – The gradient of the kernel matrix with respect to kernel hyperparameters. Only returned if eval_gradient is True.

flat_hyperparameters

hyperparameter_bounds

hyperparameters
A hierarchical representation of all the kernel hyperparameters.

is_stationary()

n_dims
Number of hyperparameters including both optimizable and fixed ones.

requires_vector_input

theta
The logarithms of a flattened array of kernel hyperparameters, excluing those declared as 'fixed' or those with equal lower and upper bounds.

trait_t
alias of `Traits`

classmethod traits(*diagonal=False*, *symmetric=False*, *nodal=False*, *lmin=0*, *eval_gradient=False*)

Subpackages

graphdot.kernel.marginalized package

class graphdot.kernel.marginalized.**MarginalizedGraphKernel**(*node_kernel*, *edge_kernel*, *p=1.0*, *q=0.01*, *q_bounds=(0.0001*, *0.9999)*, *eps=0.01*, *ftol=1e-08*, *gtol=1e-06*, *dtype=<class 'float'>*, *backend='auto'*)

Bases: `object`

Implements the random walk-based graph similarity kernel as proposed in: Kashima, H., Tsuda, K., & Inokuchi, A. (2003). Marginalized kernels between labeled graphs. *In Proceedings of the 20th international conference on machine learning (ICML-03)* (pp. 321-328).

Parameters

- **node_kernel** (*microkernel*) – A kernelet that computes the similarity between individual nodes
- **edge_kernel** (*microkernel*) – A kernelet that computes the similarity between individual edge
- **p** (positive number (default=1.0) or `StartingProbability`) – The starting probability of the random walk on each node. Must be either a positive number or a concrete subclass instance of `StartingProbability`.
- **q** (*float in (0, 1)*) – The probability for the random walk to stop during each step.
- **q_bounds** (*pair of floats*) – The lower and upper bound that the stopping probability can vary during hyperparameter optimization.
- **eps** (*float*) – The step size used for finite difference approximation of the gradient. Only used for nodal matrices (`nodal=True`).
- **dtype** (*numpy dtype*) – The data type of the kernel matrix to be returned.
- **backend** (*'auto' or 'cuda' or an instance of*) –

:param graphdot.kernel.marginalized.Backend.: The computing engine that solves the marginalized graph ke
generalized Laplacian equation.

__call__(*X*, *Y=None*, *eval_gradient=False*, *nodal=False*, *lmin=0*, *timing=False*)
Compute pairwise similarity matrix between graphs

Parameters

- **X** (*list of N graphs*) – The graphs must all have same node and edge attributes.

- **Y** (*None or list of M graphs*) – The graphs must all have same node and edge attributes.
- **eval_gradient** (*Boolean*) – If True, computes the gradient of the kernel matrix with respect to hyperparameters and return it alongside the kernel matrix.
- **nodal** (*bool*) – If True, return node-wise similarities; otherwise, return graphwise similarities.
- **lmin** (*0 or 1*) – Number of steps to skip in each random walk path before similarity is computed. (lmin + 1) corresponds to the starting value of l in the summation of Eq. 1 in Tang & de Jong, 2019 https://doi.org/10.1063/1.5078640 (or the first unnumbered equation in Section 3.3 of Kashima, Tsuda, and Inokuchi, 2003).

Returns

- **kernel_matrix** (*ndarray*) – if Y is None, return a square matrix containing pairwise similarities between the graphs in X; otherwise, returns a matrix containing similarities across graphs in X and Y.
- **gradient** (*ndarray*) – The gradient of the kernel matrix with respect to kernel hyperparameters. Only returned if eval_gradient is True.

active_theta_mask

bounds

The logarithms of a reshaped X-by-2 array of kernel hyperparameter bounds, excluing those declared as 'fixed' or those with equal lower and upper bounds.

clone_with_theta(*theta*)

diag(*X*, *eval_gradient=False*, *nodal=False*, *lmin=0*, *active_theta_only=True*, *timing=False*)

Compute the self-similarities for a list of graphs

Parameters

- **X** (*list of N graphs*) – The graphs must all have same node attributes and edge attributes.
- **eval_gradient** (*Boolean*) – If True, computes the gradient of the kernel matrix with respect to hyperparameters and return it alongside the kernel matrix.
- **nodal** (*bool*) – If True, returns a vector containing nodal self similarties; if False, returns a vector containing graphs' overall self similarities; if 'block', return a list of square matrices which forms a block-diagonal matrix, where each diagonal block represents the pairwise nodal similarities within a graph.
- **lmin** (*0 or 1*) – Number of steps to skip in each random walk path before similarity is computed. (lmin + 1) corresponds to the starting value of l in the summation of Eq. 1 in Tang & de Jong, 2019 https://doi.org/10.1063/1.5078640 (or the first unnumbered equation in Section 3.3 of Kashima, Tsuda, and Inokuchi, 2003).
- **active_theta_only** (*bool*) – Whether or not to return only gradients with regard to the non-fixed hyperparameters.

Returns

- **diagonal** (*numpy.array or list of np.array(s)*) – If nodal=True, returns a vector containing nodal self similarties; if nodal=False, returns a vector containing graphs' overall self similarities; if nodal = 'block', return a list of square matrices, each being a pairwise nodal similarity matrix within a graph.

- *gradient* – The gradient of the kernel matrix with respect to kernel hyperparameters. Only returned if eval_gradient is True.

flat_hyperparameters

hyperparameter_bounds

hyperparameters
A hierarchical representation of all the kernel hyperparameters.

is_stationary()

n_dims
Number of hyperparameters including both optimizable and fixed ones.

requires_vector_input

theta
The logarithms of a flattened array of kernel hyperparameters, excluing those declared as 'fixed' or those with equal lower and upper bounds.

trait_t
alias of `Traits`

classmethod traits(*diagonal=False*, *symmetric=False*, *nodal=False*, *lmin=0*, *eval_gradient=False*)

Submodules

graphdot.kernel.marginalized.basekernel module

graphdot.kernel.marginalized.starting_probability module

class graphdot.kernel.marginalized.starting_probability.**StartingProbability**
Bases: `abc.ABC`

Assigns non-negative starting probabilities to each node of a graph. Note that such a notion of starting probability can be safely generalize so that the probabilies does not have to sum to 1.

__call__(*nodes*)
Takes in a dataframe of nodes and returns an array of probabilities.

Parameters **nodes** (*DataFrame*) – Each node corresponds to a row in the data frame.

Returns

- **p** (*numpy.ndarray*) – The starting probabilities on each node.
- **d_p** (*numpy.ndarray*) – The gradient of the starting probabilities with respect to the hyperparameters as a matrix where each row corresponds to one hyperparameter.

bounds
The log-scale bounds of the hyperparameters as a 2D array.

gen_expr()
Returns the C++ expression for calculating the starting probability and its partial derivatives.

theta
The log-scale hyperparameters of the starting probability distribution as an ndarray.

Submodules

graphdot.kernel.basekernel module

graphdot.kernel.fix module

class graphdot.kernel.fix.**Exponentiation**(*kernel*, *xi=1.0*, *xi_bounds=(0.1, 20.0)*)

Bases: object

Raises a kernel to some exponentiation. $k_{\text{exponentiation}}(x, y) = k(x, y)^{\xi}$.

Parameters

- **kernel** (*object*) – The graph kernel to be exponentiated.
- **xi** (*float*) – The exponent to be raises.
- **xi_bounds** ((*float, float*)) – The range of the exponents to be searched during hyperparameter optimization.

__call__(*X*, *Y=None*, *eval_gradient=False*, ***options*)

Normalized outcome of **:py:'self.kernel(X, Y, eval_gradient, **options)'**.

Parameters that of the graph kernel object. (*Inherits*) –

Returns

Return type Inherits that of the graph kernel object.

bounds

clone_with_theta(*theta*)

diag(*X*, ***options*)

Normalized outcome of **:py:'self.kernel.diag(X, **options)'**.

Parameters that of the graph kernel object. (*Inherits*) –

Returns

Return type Inherits that of the graph kernel object.

hyperparameter_bounds

hyperparameters

theta

class graphdot.kernel.fix.**Normalization**(*kernel*)

Bases: object

Normalizes a kernel using the cosine of angle formula: $k_{\text{normalized}}(x, y) = \frac{k(x,y)}{\sqrt{k(x,x)k(y,y)}}$.

Parameters **kernel** (*object*) – The kernel to be normalized.

__call__(*X*, *Y=None*, *eval_gradient=False*, ***options*)

Normalized outcome of **:py:'self.kernel(X, Y, eval_gradient, **options)'**.

Parameters that of the graph kernel object. (*Inherits*) –

Returns

Return type Inherits that of the graph kernel object.

bounds

clone_with_theta(*theta*)

diag(*X*, *eval_gradient=False*, ***options*)
Normalized outcome of **:py:'self.kernel.diag(X, **options)'**.

Parameters that of the graph kernel object. (*Inherits*) –

Returns

Return type Inherits that of the graph kernel object.

hyperparameter_bounds

hyperparameters

theta

graphdot.kernel.molecular module

class graphdot.kernel.molecular.**Tang2019MolecularKernel**(*stopping_probability=0.01*, *starting_probability=1.0*, *element_prior=0.2*, *edge_length_scale=0.05*, ***kwargs*)

Bases: object

A margianlized graph kernel for **3D molecular structures** as in: Tang, Y. H., & de Jong, W. A. (2019). Prediction of atomization energy using graph kernel and active learning. *The Journal of chemical physics*, 150(4), 044107. The kernel can be directly used together with Graph.from_ase() to operate on molecular structures.

Parameters

- **stopping_probability** (*float in (0, 1)*) – The probability for the random walk to stop during each step.
- **starting_probability** (*float*) – The probability for the random walk to start from any node. See the *p* kwarg of *graphdot.kernel.marginalized.MarginalizedGraphKernel*
- **element_prior** (*float in (0, 1)*) – The baseline similarity between distinct elements — an element always have a similarity 1 to itself.
- **edge_length_scale** (*float in (0, inf)*) – length scale of the Gaussian kernel on edge length. A rule of thumb is that the similarity decays smoothly from 1 to nearly 0 around three times of the length scale.

__call__(*X*, *Y=None*, ***kwargs*)
Same call signature as *graphdot.kernel.marginalized.MarginalizedGraphKernel.__call__()*

bounds

clone_with_theta(*theta*)

diag(*X*, ***kwargs*)
Same call signature as *graphdot.kernel.marginalized.MarginalizedGraphKernel.diag()*

hyperparameter_bounds

hyperparameters

theta

graphdot.kernel.rbf module

class graphdot.kernel.rbf.**RBFKernel**(*expr*, *x*, ***hyperparameters*)
Bases: object

__call__(*X*, *Y=None*)
Call self as a function.

diag(*X*)

get_params()

gradient(*X*)

theta

2.5.3 graphdot.metric package

class graphdot.metric.**MaxiMin**(**args*, ***kwargs*)
Bases: graphdot.kernel.marginalized._kernel.MarginalizedGraphKernel

The maximin graph distance is a variant of the Hausdorff distance. Given the nodal similarity measure defined on individual nodes by the marginalized graph kernel, the maximin distance is the greatest of all the kernel-induced distances from a node in one graph to the closest node in the other graph. Two graphs are close in the maximin distance if every node of either graph is close to some node of the other graph.

Parameters

- **args** (*arguments*) – Inherits from *graphdot.kernel.marginalized.MarginalizedGraphKernel*.
- **kwargs** (*keyword arguments*) – Inherits from *graphdot.kernel.marginalized.MarginalizedGraphKernel*.

__call__(*X*, *Y=None*, *eval_gradient=False*, *lmin=0*, *return_hotspot=False*, *timing=False*)
Computes the distance matrix and optionally its gradient with respect to hyperparameters.

Parameters

- **X** (*list of graphs*) – The first dataset to be compared.
- **Y** (*list of graphs or None*) – The second dataset to be compared. If None, X will be compared with itself.
- **eval_gradient** (*bool*) – If True, returns the gradient of the weight matrix alongside the matrix itself.
- **lmin** (*0 or 1*) – Number of steps to skip in each random walk path before similarity is computed. (lmin + 1) corresponds to the starting value of l in the summation of Eq. 1 in Tang & de Jong, 2019 https://doi.org/10.1063/1.5078640 (or the first unnumbered equation in Section 3.3 of Kashima, Tsuda, and Inokuchi, 2003).
- **return_hotspot** (*bool*) – Whether or not to return the indices of the node pairs that determines the maximin distance between the graphs. Generally, these hotspots represent the location of the largest difference between the graphs.
- **options** (*keyword arguments*) – Additional arguments to be passed to the underlying kernel.

Returns

- **distance** (*2D matrix*) – A distance matrix between the data points.

- **hotspot** (*a pair of 2D integer matrices*) – Indices of the hotspot node pairs between the graphs. Only returned if the `return_hotspot` argument is True.
- **gradient** (*3D tensor*) – A tensor where the i-th frontal slide [:, :, i] contain the partial derivative of the distance matrix with respect to the i-th hyperparameter. Only returned if the `eval_gradient` argument is True.

class graphdot.metric.**KernelInducedDistance**(*kernel*, *kernel_options={}*)

Bases: `object`

The kernel-induced distance is defined by **:py:math:'d(x, y) = \sqrt{\frac{1}{2}(k(x, x) + k(y, y)) - k(x, y)}'**.

Parameters

- **kernel** (*callable*) – A positive semidefinite kernel such as one from *graphdot.kernel*.
- **kernel_options** (*dict*) – Additional arguments to be forwarded to the kernel.

__call__(*X*, *Y=None*, *eval_gradient=False*)

Computes the distance matrix and optionally its gradient with respect to hyperparameters.

Parameters

- **X** (*list of graphs*) – The first dataset to be compared.
- **Y** (*list of graphs or None*) – The second dataset to be compared. If None, X will be compared with itself.
- **eval_gradient** (*bool*) – If True, returns the gradient of the weight matrix alongside the matrix itself.

Returns

- **distance** (*2D matrix*) – A distance matrix between the data points.
- **gradient** (*3D tensor*) – A tensor where the i-th frontal slide [:, :, i] contain the partial derivative of the distance matrix with respect to the i-th hyperparameter. Only returned if the `eval_gradient` argument is True.

bounds

clone_with_theta(*theta=None*)

hyperparameters

theta

Subpackages

graphdot.metric.maximin package

class graphdot.metric.maximin.**MaxiMin**(**args*, ***kwargs*)

Bases: `graphdot.kernel.marginalized._kernel.MarginalizedGraphKernel`

The maximin graph distance is a variant of the Hausdorff distance. Given the nodal similarity measure defined on individual nodes by the marginalized graph kernel, the maximin distance is the greatest of all the kernel-induced distances from a node in one graph to the closest node in the other graph. Two graphs are close in the maximin distance if every node of either graph is close to some node of the other graph.

Parameters

- **args** (*arguments*) – Inherits from *graphdot.kernel.marginalized.MarginalizedGraphKernel*.
- **kwargs** (*keyword arguments*) – Inherits from *graphdot.kernel.marginalized.MarginalizedGraphKernel*.

__call__(*X*, *Y=None*, *eval_gradient=False*, *lmin=0*, *return_hotspot=False*, *timing=False*)

Computes the distance matrix and optionally its gradient with respect to hyperparameters.

Parameters

- **X** (*list of graphs*) – The first dataset to be compared.
- **Y** (*list of graphs or None*) – The second dataset to be compared. If None, X will be compared with itself.
- **eval_gradient** (*bool*) – If True, returns the gradient of the weight matrix alongside the matrix itself.
- **lmin** (*0 or 1*) – Number of steps to skip in each random walk path before similarity is computed. (lmin + 1) corresponds to the starting value of l in the summation of Eq. 1 in Tang & de Jong, 2019 https://doi.org/10.1063/1.5078640 (or the first unnumbered equation in Section 3.3 of Kashima, Tsuda, and Inokuchi, 2003).
- **return_hotspot** (*bool*) – Whether or not to return the indices of the node pairs that determines the maximin distance between the graphs. Generally, these hotspots represent the location of the largest difference between the graphs.
- **options** (*keyword arguments*) – Additional arguments to be passed to the underlying kernel.

Returns

- **distance** (*2D matrix*) – A distance matrix between the data points.
- **hotspot** (*a pair of 2D integer matrices*) – Indices of the hotspot node pairs between the graphs. Only returned if the `return_hotspot` argument is True.
- **gradient** (*3D tensor*) – A tensor where the i-th frontal slide [:, :, i] contain the partial derivative of the distance matrix with respect to the i-th hyperparameter. Only returned if the `eval_gradient` argument is True.

2.5.4 graphdot.microkernel package

Microkernels are positive-semidefinite functions between individual nodes and edges of graphs.

class graphdot.microkernel.**MicroKernel**

Bases: abc.ABC

The abstract base class for all microkernels.

__add__(*k*)

Implements the additive kernel composition semantics, i.e. expression `k1 + k2` creates $k_+(a,b) = k_1(a,b) + k_2(a,b)$

__call__(*i*, *j*, *jac=False*)

Evaluates the kernel.

Parameters

- **j** (*i,*) – Inputs to the kernel.

- **jac** (*Boolean*) – Whether or not to return the gradient of the kernel with respect to kernel hyperparameters alongside the kernel value.

Returns

- **k_ij** (*scalar*) – The value of the kernel as evaluated on i and j.
- **jacobian** (*1D ndarray*) – The gradient of the kernel with regard to hyperparameters.

__mul__(*k*)
Implements the multiplicative kernel composition semantics, i.e. expression `k1 * k2` creates $k_\times(a,b) = k_1(a,b) \times k_2(a,b)$

bounds
A list of 2-tuples for the lower and upper bounds of each kernel hyperparameter.

static from_sympy(*name*, *desc*, *expr*, *vars*, **hyperparameter_specs*, *minmax=(0, 1)*)
Create a pairwise kernel class from a SymPy expression.

Parameters

- **name** (*str*) – The name of the kernel. Must be a valid Python identifier.
- **desc** (*str*) – A human-readable description of the kernel. Will be used to build the docstring of the returned kernel class.
- **expr** (*str or SymPy expression*) – Expression of the kernel in SymPy format.
- **vars** (*2-tuple of str or SymPy symbols*) – The input variables of the kernel as shown up in the expression. A kernel must have exactly 2 input variables. All other symbols that show up in its expression should be regarded as hyperparameters.
- **hyperparameter_specs** (*list of hyperparameter specifications in one of*) –
- **formats below** (*the*) –

  symbol,
  (symbol,),
  (symbol, dtype),
  (symbol, dtype, description),
  (symbol, dtype, lower_bound, upper_bound),
  (symbol, dtype, lower_bound, upper_bound, description),

  If a default set of lower and upper bounds are not defined here, then it must be specified explicitly during kernel object creation, using arguments as specified in the kernel class's docstring.

gen_expr(*x*, *y*, *theta_scope=''*)
Generate the C++ expression for evaluating the kernel and its partial derivatives.

Parameters

- **y** (*x,*) – Name of the input variables.
- **theta_scope** (*str*) – The scope in which the hyperparameters is located.

Returns

- **expr** (*str*) – A C++ expression that evaluates the kernel.
- **jac_expr** (*list of strs*) – C++ expressions that evaluate the derivative of the kernel.

minmax
A 2-tuple of the minimum and maximum values that the kernel could take.

name
Name of the kernel.

normalized
A normalized version of the original kernel using the dot product formula: **:py:math:'k^\mathrm{normalized}(i, j) = \frac{k(i, j)}{\sqrt{k(i, i) k(j, j)}}'**.

theta
A tuple of all the kernel hyperparameters.

class graphdot.microkernel.**Product**
Bases: graphdot.microkernel.product.Product

dtype = dtype([], align=True)

state

graphdot.microkernel.**Constant**(*c*, *c_bounds='fixed'*)
Creates a no-op microkernel that returns a constant value, i.e. $k_c(\cdot, \cdot) \equiv constant$. This kernel is often mutliplied with other microkernels as an adjustable weight.

Parameters **c** (*float > 0*) – The constant value.

graphdot.microkernel.**KroneckerDelta**(*h*, *h_bounds=(0.001, 1)*)
Creates a Kronecker delta microkernel that returns either 1 or h depending on whether two features compare equal, i.e. $k_\delta(i, j) = \begin{cases} 1, i = j \\ h, otherwise \end{cases}$.

Parameters

- **h** (*float in (0, 1)*) – The value of the microkernel when two features do not compare equal.
- **h_bounds** (*tuple or "fixed"*) – If tuple, contains the lower and upper bounds that *h* is allowed to vary during hyperparameter optimization. If "fixed", the hyperparameter will not be optimized during training.

graphdot.microkernel.**SquareExponential**
alias of graphdot.microkernel._base._from_sympy.<locals>.uKernel

graphdot.microkernel.**RationalQuadratic**
alias of graphdot.microkernel._base._from_sympy.<locals>.uKernel

graphdot.microkernel.**Normalize**(*kernel: graphdot.microkernel._base.MicroKernel*)
Normalize the value range of a microkernel to [0, 1] using the cosine of angle formula: $k_{normalized}(x, y) = \frac{k(x,y)}{\sqrt{k(x,x)k(y,y)}}$.

Parameters **kernel** – The microkernel to be normalized.

graphdot.microkernel.**Composite**(*oper*, ***kw_kernels*)
Creates a microkernel on multiple features, which uses a reduction operator to combine the outputs of multiple microkernels on individual features. $k_{\text{composite}}(X, Y; \text{op}) = k_{a_1}(X_{a_1}, Y_{a_1}) \,\text{op}\, k_{a_2}(X_{a_2}, Y_{a_2}) \,\text{op}\, \ldots$

Parameters

- **oper** (*str*) – A reduction operator. Due to positive definiteness requirements, the available options are currently limited to '+', '*'.
- **kw_kernels** (*dict of attribute=kernel pairs*) – The kernels can be any microkernels and their compositions as defined in this module, while features should be strings that represent valid Python/C++ identifiers.

graphdot.microkernel.**TensorProduct**(***kw_kernels*)
Alias of *Composite('*', **kw_kernels)*. $k_{\otimes}(X, Y) = \prod_{a \in \text{features}} k_a(X_a, Y_a)$

graphdot.microkernel.**Additive**(***kw_kernels*)
Alias of *Composite('+', **kw_kernels)*. $k_{\oplus}(X, Y) = \sum_{a \in \text{features}} k_a(X_a, Y_a)$

graphdot.microkernel.**Convolution**(*kernel: graphdot.microkernel._base.MicroKernel*, *mean=True*)
Creates a convolutional microkernel, which averages evaluations of a base microkernel between pairs of elements of two variable-length feature sequences.

Parameters

- **kernel** (MicroKernel) – The base kernel can be any elementary or composite microkernels, while the attribute to be convolved should be sequences.
- **mean** (*bool*) – If True, return the arithmetic mean of the kernel evaluations, i.e. $k_{conv}(X, Y) = \frac{\sum_{x \in X} \sum_{y \in Y} k_{base}(x,y)}{|X||Y|}$. Otherwise, return the sum of the kernel evaluations, i.e. $k_{conv}(X, Y) = \sum_{x \in X} \sum_{y \in Y} k_{base}(x, y)$. Thus, this serves as a means of normalization beyonds the dot product based one.

graphdot.microkernel.**DotProduct**()
Creates a dot product microkernel, which computes the inner product between two vector-valued features.

Parameters **kernel does not have any tunable hyperparameters.** (*This*) –

Submodules

graphdot.microkernel.additive module

graphdot.microkernel.additive.**Additive**(***kw_kernels*)
Alias of *Composite('+', **kw_kernels)*. $k_{\oplus}(X, Y) = \sum_{a \in \text{features}} k_a(X_a, Y_a)$

graphdot.microkernel.composite module

graphdot.microkernel.composite.**Composite**(*oper*, ***kw_kernels*)
Creates a microkernel on multiple features, which uses a reduction operator to combine the outputs of multiple microkernels on individual features. $k_{\text{composite}}(X, Y; \text{op}) = k_{a_1}(X_{a_1}, Y_{a_1}) \,\text{op}\, k_{a_2}(X_{a_2}, Y_{a_2}) \,\text{op}\, \ldots$

Parameters

- **oper** (*str*) – A reduction operator. Due to positive definiteness requirements, the available options are currently limited to '+', '*'.
- **kw_kernels** (*dict of attribute=kernel pairs*) – The kernels can be any microkernels and their compositions as defined in this module, while features should be strings that represent valid Python/C++ identifiers.

graphdot.microkernel.convolution module

graphdot.microkernel.convolution.**Convolution**(*kernel: graphdot.microkernel._base.MicroKernel*, *mean=True*)
Creates a convolutional microkernel, which averages evaluations of a base microkernel between pairs of elements of two variable-length feature sequences.

Parameters

- **kernel** (MicroKernel) – The base kernel can be any elementary or composite microkernels, while the attribute to be convolved should be sequences.
- **mean** (*bool*) – If True, return the arithmetic mean of the kernel evaluations, i.e. $k_{conv}(X,Y) = \frac{\sum_{x\in X}\sum_{y\in Y} k_{base}(x,y)}{|X||Y|}$. Otherwise, return the sum of the kernel evaluations, i.e. $k_{conv}(X,Y) = \sum_{x\in X}\sum_{y\in Y} k_{base}(x,y)$. Thus, this serves as a means of normalization beyonds the dot product based one.

graphdot.microkernel.dotproduct module

graphdot.microkernel.dotproduct.**DotProduct**()
Creates a dot product microkernel, which computes the inner product between two vector-valued features.

Parameters **kernel does not have any tunable hyperparameters.** (*This*) –

graphdot.microkernel.kronecker_delta module

graphdot.microkernel.kronecker_delta.**KroneckerDelta**(*h*, *h_bounds=(0.001, 1)*)
Creates a Kronecker delta microkernel that returns either 1 or h depending on whether two features compare equal, i.e. $k_\delta(i,j) = \begin{cases} 1, i = j \\ h, otherwise \end{cases}$.

Parameters

- **h** (*float in (0, 1)*) – The value of the microkernel when two features do not compare equal.
- **h_bounds** (*tuple or "fixed"*) – If tuple, contains the lower and upper bounds that *h* is allowed to vary during hyperparameter optimization. If "fixed", the hyperparameter will not be optimized during training.

graphdot.microkernel.product module

graphdot.microkernel.rational_quadratic module

graphdot.microkernel.square_exponential module

graphdot.microkernel.tensor_product module

graphdot.microkernel.tensor_product.**TensorProduct**(***kw_kernels*)
Alias of *Composite('*', **kw_kernels)*. $k_\otimes(X,Y) = \prod_{a\in\text{features}} k_a(X_a, Y_a)$

2.5.5 graphdot.model package

Subpackages

graphdot.model.active_learning package

class graphdot.model.active_learning.**HierarchicalDrafter**(*selector*, *k=2*, *a=2*, *leaf_ratio='auto'*)
Bases: object

Hierarhically select representative samples from a large dataset where a direct algorithm can be prohibitively expensive.

Parameters

- **selector** (*callable*) – A selection algorithm that can pick a given number of samples from a dataset to maximize a certain acquisition function.
- **k** (*int > 1*) – The branching factor of the search hierarchy.
- **a** (*float in (1, k]*) – The multiplier to the number of samples that each level need to generate during hierarchical screening. For example, if n samples are wanted in the end, then the immediate next level should forward at least m * n samples for the last level drafter to choose from.
- **leaf_ratio** (*float in (0, 1)*) – If ratio berween output and input samples is greater than it, stop further division and carry out selection using the given selector.

__call__(*X*, *n*, *random_state=None*, *verbose=False*)

Find a n-sample subset of X that attempts to maximize a certain diversity criterion.

Parameters

- **X** (*feature matrix or list of objects*) – Input dataset.
- **n** (*int*) – The size of the subset to be chosen.
- **random_state** (int or **:py:'np.random.Generator'**) – The seed to the random number generator (RNG), or the RNG itself. If None, the default RNG in numpy will be used.

Returns **chosen** – A sorted list of indices of the samples that are chosen.

Return type list

class graphdot.model.active_learning.**DeterminantMaximizer**(*kernel*, *kernel_options=None*)

Bases: object

Select a subset of a dataset such that the determinant of the kernel matrix of the selected samples are as large as possible. In other words, the objective here is to ensure that the samples are as linearly independent as possible in a reproducible kernel Hilbert space (RKHS).

Parameters

- **kernel** (*callable or 'precomputed'*) – A symmetric positive semidefinite function implemented via the __call__ semantics. Alternatively, if the value is 'precomputed', a square kernel matrix will be expected as an argument to **:py:'__call__'**.
- **kernel_options** (*dict*) – Additional arguments to be passed into the kernel.

__call__(*X*, *n*)

Find a n-sample subset of X that attempts to maximize the diversity and return the indices of the samples.

Parameters

- **X** (*feature matrix or list of objects*) – Input dataset.
- **n** (*int*) – Number of samples to be chosen.

Returns **chosen** – Indices of the samples that are chosen.

Return type list

class graphdot.model.active_learning.**VarianceMinimizer**(*kernel*, *alpha=1e-06*, *kernel_options=None*)

Bases: object

Select a subset of a dataset such that the Gaussian process posterior variance, i.e. the Nystrom residual norm, of the kernel matrix of the UNSELECTED samples are as small as possible. In other words, the objective here is to ensure that the chosen samples can effectively span the vector space as occupied by the entire dataset in a reproducible kernel Hilbert space (RKHS).

Parameters

- **kernel** (*callable or 'precomputed'*) – A symmetric positive semidefinite function implemented via the __call__ semantics. Alternatively, if the value is 'precomputed', a square kernel matrix will be expected as an argument to **:py:'__call__'**.
- **alpha** (*float, default=1e-7*) – A small value added to the diagonal elements of the kernel matrix in order to regularize the variance calculations.
- **kernel_options** (*dict*) – Additional arguments to be passed into the kernel.

__call__ (*X*, *n*)

Find a n-sample subset of X that attempts to maximize the diversity and return the indices of the samples.

Parameters

- **X** (*feature matrix or list of objects*) – Input dataset.
- **n** (*int*) – Number of samples to be chosen.

Returns **chosen** – Indices of the samples that are chosen.

Return type list

Submodules

graphdot.model.active_learning.determinant_maximizer module

class graphdot.model.active_learning.determinant_maximizer.**DeterminantMaximizer**(*kernel*, *kernel_options=No*

Bases: object

Select a subset of a dataset such that the determinant of the kernel matrix of the selected samples are as large as possible. In other words, the objective here is to ensure that the samples are as linearly independent as possible in a reproducible kernel Hilbert space (RKHS).

Parameters

- **kernel** (*callable or 'precomputed'*) – A symmetric positive semidefinite function implemented via the __call__ semantics. Alternatively, if the value is 'precomputed', a square kernel matrix will be expected as an argument to **:py:'__call__'**.
- **kernel_options** (*dict*) – Additional arguments to be passed into the kernel.

__call__ (*X*, *n*)

Find a n-sample subset of X that attempts to maximize the diversity and return the indices of the samples.

Parameters

- **X** (*feature matrix or list of objects*) – Input dataset.
- **n** (*int*) – Number of samples to be chosen.

Returns **chosen** – Indices of the samples that are chosen.

Return type list

graphdot.model.active_learning.hierarchical_drafter module

class graphdot.model.active_learning.hierarchical_drafter.**HierarchicalDrafter**(*selector*, *k=2*, *a=2*, *leaf_ratio='auto'*)

Bases: object

Hierarhically select representative samples from a large dataset where a direct algorithm can be prohibitively expensive.

Parameters

- **selector** (*callable*) – A selection algorithm that can pick a given number of samples from a dataset to maximize a certain acquisition function.
- **k** (*int > 1*) – The branching factor of the search hierarchy.
- **a** (*float in (1, k]*) – The multiplier to the number of samples that each level need to generate during hierarchical screening. For example, if n samples are wanted in the end, then the immediate next level should forward at least m * n samples for the last level drafter to choose from.
- **leaf_ratio** (*float in (0, 1)*) – If ratio berween output and input samples is greater than it, stop further division and carry out selection using the given selector.

__call__(*X*, *n*, *random_state=None*, *verbose=False*)

Find a n-sample subset of X that attempts to maximize a certain diversity criterion.

Parameters

- **X** (*feature matrix or list of objects*) – Input dataset.
- **n** (*int*) – The size of the subset to be chosen.
- **random_state** (int or **:py:'np.random.Generator'**) – The seed to the random number generator (RNG), or the RNG itself. If None, the default RNG in numpy will be used.

Returns **chosen** – A sorted list of indices of the samples that are chosen.

Return type list

graphdot.model.active_learning.variance_minimizer module

class graphdot.model.active_learning.variance_minimizer.**VarianceMinimizer**(*kernel*, *alpha=1e-06*, *kernel_options=None*)

Bases: object

Select a subset of a dataset such that the Gaussian process posterior variance, i.e. the Nystrom residual norm, of the kernel matrix of the UNSELECTED samples are as small as possible. In other words, the objective here is to ensure that the chosen samples can effectively span the vector space as occupied by the entire dataset in a reproducible kernel Hilbert space (RKHS).

Parameters

- **kernel** (*callable or 'precomputed'*) – A symmetric positive semidefinite function implemented via the __call__ semantics. Alternatively, if the value is 'precomputed', a square kernel matrix will be expected as an argument to **:py:'__call__'**.

- **alpha** (*float, default=1e-7*) – A small value added to the diagonal elements of the kernel matrix in order to regularize the variance calculations.
- **kernel_options** (*dict*) – Additional arguments to be passed into the kernel.

__call__(*X*, *n*)
Find a n-sample subset of X that attempts to maximize the diversity and return the indices of the samples.

Parameters

- **X** (*feature matrix or list of objects*) – Input dataset.
- **n** (*int*) – Number of samples to be chosen.

Returns **chosen** – Indices of the samples that are chosen.

Return type list

graphdot.model.gaussian_field package

class graphdot.model.gaussian_field.**GaussianFieldRegressor**(*weight*, *optimizer=None*, *smoothing=0.001*)
Bases: object

Semi-supervised learning and prediction of missing labels of continuous value on a graph. Reference: Zhu, Ghahramani, Lafferty. ICML 2003

Parameters

- **weight** (*callable or 'precomputed'*) – A function that implements a weight function that converts distance matrices to weight matrices. The value of a weight function should generally decay with distance. If weight is 'precomputed', then the result returned by *metric* will be directly used as weight.
- **optimizer** (*one of (str, True, None, callable)*) – A string or callable that represents one of the optimizers usable in the scipy.optimize.minimize method. if None, no hyperparameter optimization will be carried out in fitting. If True, the optimizer will default to L-BFGS-B.
- **smoothing** (*float in [0, 1)*) – Controls the strength of regularization via the smoothing of the transition matrix.

average_label_entropy(*X*, *y*, *theta=None*, *eval_gradient=False*, *verbose=False*)
Evaluate the average label entropy of the Gaussian field model on a dataset.

Parameters

- **X** (*2D array or list of objects*) – Feature vectors or other generic representations of input data.
- **y** (*1D array*) – Label of each data point. Values of None or NaN indicates missing labels that will be filled in by the model.
- **theta** (*1D array*) – Hyperparameters for the weight class.
- **eval_gradients** – Whether or not to evaluate the gradient of the average label entropy with respect to weight hyperparameters.
- **verbose** (*bool*) – If true, print out some additional information as a markdown table.

Returns

- **average_label_entropy** (*float*) – The average label entropy of the Gaussian field prediction on the unlabeled nodes.
- **grad** (*1D array*) – Gradient with respect to the hyperparameters.

fit (*X*, *y*, *loss='loocv2'*, *tol=1e-05*, *repeat=1*, *theta_jitter=1.0*, *verbose=False*)
Train the Gaussian field model.

Parameters

- **X** (*2D array or list of objects*) – Feature vectors or other generic representations of input data.
- **y** (*1D array*) – Label of each data point. Values of None or NaN indicates missing labels that will be filled in by the model.
- **loss** (*str*) – The loss function to be used to optimizing the hyperparameters. Options are:

  – 'ale' or 'average-label-entropy': average label entropy. Only

  works if the labels are 0/1 binary. - 'loocv1' or 'loocv2': the leave-one-out cross validation of the labeled samples as measured in L1/L2 norm.
- **tol** (*float*) – Tolerance for termination.
- **repeat** (*int*) – Repeat the hyperparameter optimization by the specified number of times and return the best result.
- **theta_jitter** (*float*) – Standard deviation of the random noise added to the initial logscale hyperparameters across repeated optimization runs.

Returns **self** – returns an instance of self.

Return type *GaussianFieldRegressor*

fit_predict (*X*, *y*, *loss='average-label-entropy'*, *tol=1e-05*, *repeat=1*, *theta_jitter=1.0*, *return_influence=False*, *verbose=False*)
Train the Gaussian field model and make predictions for the unlabeled nodes.

Parameters

- **X** (*2D array or list of objects*) – Feature vectors or other generic representations of input data.
- **y** (*1D array*) – Label of each data point. Values of None or NaN indicates missing labels that will be filled in by the model.
- **loss** (*str*) – The loss function to be used to optimizing the hyperparameters. Options are:

  – 'ale' or 'average-label-entropy': average label entropy. Only

  works if the labels are 0/1 binary. - 'loocv1' or 'loocv2': the leave-one-out cross validation of the labeled samples as measured in L1/L2 norm.
- **tol** (*float*) – Tolerance for termination.
- **repeat** (*int*) – Repeat the hyperparameter optimization by the specified number of times and return the best result.
- **theta_jitter** (*float*) – Standard deviation of the random noise added to the initial logscale hyperparameters across repeated optimization runs.
- **return_influence** (*bool*) – If True, also returns the contributions of each labeled sample to each predicted label as an 'influence matrix'.

Returns

- **z** (*1D array*) – Node labels with missing ones filled in by prediction.
- **influence_matrix** (*2D array*) – Contributions of each labeled sample to each predicted label. Only returned if `return_influence` is True.

loocv_error(*X*, *y*, *p=2*, *theta=None*, *eval_gradient=False*, *verbose=False*)
Evaluate the leave-one-out cross validation error and gradient.

Parameters

- **X** (*2D array or list of objects*) – Feature vectors or other generic representations of input data.
- **y** (*1D array*) – Label of each data point. Values of None or NaN indicates missing labels that will be filled in by the model.
- **p** (*float > 1*) – The order of the p-norm for LOOCV error.
- **theta** (*1D array*) – Hyperparameters for the weight class.
- **eval_gradients** – Whether or not to evaluate the gradient of the average label entropy with respect to weight hyperparameters.
- **verbose** (*bool*) – If true, print out some additional information as a markdown table.

Returns

- **err** (*1D array*) – LOOCV Error
- **grad** (*1D array*) – Gradient with respect to the hyperparameters.

loocv_error_1(*X*, *y*, ***kwargs*)
Leave-one-out cross validation error measured in L1 norm. Equivalent to **:py:method:‘loocv_error(X, y, p=1, **kwargs)‘**.

loocv_error_2(*X*, *y*, ***kwargs*)
Leave-one-out cross validation error measured in L2 norm. Equivalent to **:py:method:‘loocv_error(X, y, p=2, **kwargs)‘**.

predict(*X*, *y*, *return_influence=False*)
Make predictions for the unlabeled elements in y.

Parameters

- **X** (*2D array or list of objects*) – Feature vectors or other generic representations of input data.
- **y** (*1D array*) – Label of each data point. Values of None or NaN indicates missing labels that will be filled in by the model.
- **return_influence** (*bool*) – If True, also returns the contributions of each labeled sample to each predicted label as an ‘influence matrix’.

Returns

- **z** (*1D array*) – Node labels with missing ones filled in by prediction.
- **influence_matrix** (*2D array*) – Contributions of each labeled sample to each predicted label. Only returned if `return_influence` is True.

class graphdot.model.gaussian_field.**Weight**
Bases: abc.ABC

__call__(*X*, *Y=None*, *eval_gradient=False*)
Computes the weight matrix and optionally its gradient with respect to hyperparameters.

Parameters

- **X** (*list*) – The first dataset to be compared.
- **Y** (*list or None*) – The second dataset to be compared. If None, X will be compared with itself.
- **eval_gradient** (*bool*) – If True, returns the gradient of the weight matrix alongside the matrix itself.

Returns

- **weight_matrix** (*2D ndarray*) – A weight matrix between the datasets.
- **weight_matrix_gradients** (*3D ndarray*) – A tensor where the i-th frontal slide [:, :, i] contain the partial derivative of the weight matrix with respect to the i-th hyperparameter.

bounds
The log-scale bounds of the hyperparameters as a 2D array.

clone_with_theta(*theta*)

theta
An ndarray of all the hyperparameters in log scale.

class graphdot.model.gaussian_field.**RBFOverDistance**(*metric*, *sigma*, *sigma_bounds=(0.001, 1000.0)*, *mopts={}*)
Bases: *graphdot.model.gaussian_field.weight.Weight*

Set weights by applying an RBF onto a distance matrix.

Parameters

- **metric** (*callable*) – An object that implements a distance metric.
- **sigma** (*float*) – The log scale hyperparameter for the RBF Kernel.
- **sigma_bounds** (*float*) – The bounds for sigma.
- **sticky_cache** (*bool*) – Whether or not to save the distance matrix upon first evaluation of the weights. This could speedup hyperparameter optimization if the underlying distance matrix remains unchanged during the process.

__call__(*X*, *Y=None*, *eval_gradient=False*)

Parameters **eval_gradient** (*bool*) – If true, also return the gradient of the weights with respect to the **log-scale** hyperparameters.

bounds
The log-scale bounds of the hyperparameters as a 2D array.

theta
An ndarray of all the hyperparameters in log scale.

class graphdot.model.gaussian_field.**RBFOverFixedDistance**(*D*, *sigma*, *sigma_bounds=(0.001, 1000.0)*, *sticky_cache=False*)
Bases: *graphdot.model.gaussian_field.weight.Weight*

Set weights by applying an (optimizable) RBF onto a fixed distance matrix.

Parameters

- **metric** (*callable*) – An object that implements a distance metric.
- **sigma** (*float*) – The log scale hyperparameter for the RBF Kernel.
- **sigma_bounds** (*float*) – The bounds for sigma.

__call__ (*X*, *Y=None*, *eval_gradient=False*)

Parameters **eval_gradient** (*bool*) – If true, also return the gradient of the weights with respect to the **log-scale** hyperparameters.

bounds
The log-scale bounds of the hyperparameters as a 2D array.

theta
An ndarray of all the hyperparameters in log scale.

Submodules

graphdot.model.gaussian_field.gfr module

class graphdot.model.gaussian_field.gfr.**GaussianFieldRegressor**(*weight*, *optimizer=None*, *smoothing=0.001*)

Bases: object

Semi-supervised learning and prediction of missing labels of continuous value on a graph. Reference: Zhu, Ghahramani, Lafferty. ICML 2003

Parameters

- **weight** (*callable or 'precomputed'*) – A function that implements a weight function that converts distance matrices to weight matrices. The value of a weight function should generally decay with distance. If weight is 'precomputed', then the result returned by *metric* will be directly used as weight.
- **optimizer** (*one of (str, True, None, callable)*) – A string or callable that represents one of the optimizers usable in the scipy.optimize.minimize method. if None, no hyperparameter optimization will be carried out in fitting. If True, the optimizer will default to L-BFGS-B.
- **smoothing** (*float in [0, 1)*) – Controls the strength of regularization via the smoothing of the transition matrix.

average_label_entropy (*X*, *y*, *theta=None*, *eval_gradient=False*, *verbose=False*)
Evaluate the average label entropy of the Gaussian field model on a dataset.

Parameters

- **X** (*2D array or list of objects*) – Feature vectors or other generic representations of input data.
- **y** (*1D array*) – Label of each data point. Values of None or NaN indicates missing labels that will be filled in by the model.
- **theta** (*1D array*) – Hyperparameters for the weight class.
- **eval_gradients** – Whether or not to evaluate the gradient of the average label entropy with respect to weight hyperparameters.

- **verbose** (*bool*) – If true, print out some additional information as a markdown table.

Returns

- **average_label_entropy** (*float*) – The average label entropy of the Gaussian field prediction on the unlabeled nodes.
- **grad** (*1D array*) – Gradient with respect to the hyperparameters.

fit (*X*, *y*, *loss='loocv2'*, *tol=1e-05*, *repeat=1*, *theta_jitter=1.0*, *verbose=False*)

Train the Gaussian field model.

Parameters

- **X** (*2D array or list of objects*) – Feature vectors or other generic representations of input data.
- **y** (*1D array*) – Label of each data point. Values of None or NaN indicates missing labels that will be filled in by the model.
- **loss** (*str*) – The loss function to be used to optimizing the hyperparameters. Options are:

  – 'ale' or 'average-label-entropy': average label entropy. Only

  works if the labels are 0/1 binary. - 'loocv1' or 'loocv2': the leave-one-out cross validation of the labeled samples as measured in L1/L2 norm.
- **tol** (*float*) – Tolerance for termination.
- **repeat** (*int*) – Repeat the hyperparameter optimization by the specified number of times and return the best result.
- **theta_jitter** (*float*) – Standard deviation of the random noise added to the initial logscale hyperparameters across repeated optimization runs.

Returns **self** – returns an instance of self.

Return type *GaussianFieldRegressor*

fit_predict (*X*, *y*, *loss='average-label-entropy'*, *tol=1e-05*, *repeat=1*, *theta_jitter=1.0*, *return_influence=False*, *verbose=False*)

Train the Gaussian field model and make predictions for the unlabeled nodes.

Parameters

- **X** (*2D array or list of objects*) – Feature vectors or other generic representations of input data.
- **y** (*1D array*) – Label of each data point. Values of None or NaN indicates missing labels that will be filled in by the model.
- **loss** (*str*) – The loss function to be used to optimizing the hyperparameters. Options are:

  – 'ale' or 'average-label-entropy': average label entropy. Only

  works if the labels are 0/1 binary. - 'loocv1' or 'loocv2': the leave-one-out cross validation of the labeled samples as measured in L1/L2 norm.
- **tol** (*float*) – Tolerance for termination.
- **repeat** (*int*) – Repeat the hyperparameter optimization by the specified number of times and return the best result.
- **theta_jitter** (*float*) – Standard deviation of the random noise added to the initial logscale hyperparameters across repeated optimization runs.

- **return_influence** (*bool*) – If True, also returns the contributions of each labeled sample to each predicted label as an 'influence matrix'.

Returns

- **z** (*1D array*) – Node labels with missing ones filled in by prediction.
- **influence_matrix** (*2D array*) – Contributions of each labeled sample to each predicted label. Only returned if `return_influence` is True.

loocv_error (*X*, *y*, *p=2*, *theta=None*, *eval_gradient=False*, *verbose=False*)
Evaluate the leave-one-out cross validation error and gradient.

Parameters

- **X** (*2D array or list of objects*) – Feature vectors or other generic representations of input data.
- **y** (*1D array*) – Label of each data point. Values of None or NaN indicates missing labels that will be filled in by the model.
- **p** (*float > 1*) – The order of the p-norm for LOOCV error.
- **theta** (*1D array*) – Hyperparameters for the weight class.
- **eval_gradients** – Whether or not to evaluate the gradient of the average label entropy with respect to weight hyperparameters.
- **verbose** (*bool*) – If true, print out some additional information as a markdown table.

Returns

- **err** (*1D array*) – LOOCV Error
- **grad** (*1D array*) – Gradient with respect to the hyperparameters.

loocv_error_1 (*X*, *y*, ***kwargs*)
Leave-one-out cross validation error measured in L1 norm. Equivalent to **:py:method:'loocv_error(X, y, p=1, **kwargs)'**.

loocv_error_2 (*X*, *y*, ***kwargs*)
Leave-one-out cross validation error measured in L2 norm. Equivalent to **:py:method:'loocv_error(X, y, p=2, **kwargs)'**.

predict (*X*, *y*, *return_influence=False*)
Make predictions for the unlabeled elements in y.

Parameters

- **X** (*2D array or list of objects*) – Feature vectors or other generic representations of input data.
- **y** (*1D array*) – Label of each data point. Values of None or NaN indicates missing labels that will be filled in by the model.
- **return_influence** (*bool*) – If True, also returns the contributions of each labeled sample to each predicted label as an 'influence matrix'.

Returns

- **z** (*1D array*) – Node labels with missing ones filled in by prediction.
- **influence_matrix** (*2D array*) – Contributions of each labeled sample to each predicted label. Only returned if `return_influence` is True.

graphdot.model.gaussian_field.weight module

class graphdot.model.gaussian_field.weight.**RBFOverDistance**(*metric*, *sigma*, *sigma_bounds=(0.001, 1000.0)*, *mopts={}*)

Bases: *graphdot.model.gaussian_field.weight.Weight*

Set weights by applying an RBF onto a distance matrix.

Parameters

- **metric** (*callable*) – An object that implements a distance metric.
- **sigma** (*float*) – The log scale hyperparameter for the RBF Kernel.
- **sigma_bounds** (*float*) – The bounds for sigma.
- **sticky_cache** (*bool*) – Whether or not to save the distance matrix upon first evaluation of the weights. This could speedup hyperparameter optimization if the underlying distance matrix remains unchanged during the process.

__call__(*X*, *Y=None*, *eval_gradient=False*)

Parameters **eval_gradient** (*bool*) – If true, also return the gradient of the weights with respect to the **log-scale** hyperparameters.

bounds
The log-scale bounds of the hyperparameters as a 2D array.

theta
An ndarray of all the hyperparameters in log scale.

class graphdot.model.gaussian_field.weight.**RBFOverFixedDistance**(*D*, *sigma*, *sigma_bounds=(0.001, 1000.0)*, *sticky_cache=False*)

Bases: *graphdot.model.gaussian_field.weight.Weight*

Set weights by applying an (optimizable) RBF onto a fixed distance matrix.

Parameters

- **metric** (*callable*) – An object that implements a distance metric.
- **sigma** (*float*) – The log scale hyperparameter for the RBF Kernel.
- **sigma_bounds** (*float*) – The bounds for sigma.

__call__(*X*, *Y=None*, *eval_gradient=False*)

Parameters **eval_gradient** (*bool*) – If true, also return the gradient of the weights with respect to the **log-scale** hyperparameters.

bounds
The log-scale bounds of the hyperparameters as a 2D array.

theta
An ndarray of all the hyperparameters in log scale.

class graphdot.model.gaussian_field.weight.**Weight**

Bases: abc.ABC

__call__(*X*, *Y=None*, *eval_gradient=False*)
Computes the weight matrix and optionally its gradient with respect to hyperparameters.

Parameters

- **X** (*list*) – The first dataset to be compared.
- **Y** (*list or None*) – The second dataset to be compared. If None, X will be compared with itself.
- **eval_gradient** (*bool*) – If True, returns the gradient of the weight matrix alongside the matrix itself.

Returns

- **weight_matrix** (*2D ndarray*) – A weight matrix between the datasets.
- **weight_matrix_gradients** (*3D ndarray*) – A tensor where the i-th frontal slide [:, :, i] contain the partial derivative of the weight matrix with respect to the i-th hyperparameter.

bounds
The log-scale bounds of the hyperparameters as a 2D array.

clone_with_theta(*theta*)

theta
An ndarray of all the hyperparameters in log scale.

graphdot.model.gaussian_process package

class graphdot.model.gaussian_process.**GaussianProcessRegressor**(*kernel*, *alpha=1e-08*, *beta=1e-08*, *optimizer=None*, *normalize_y=False*, *regularization='+'*, *kernel_options={}*)

Bases: *graphdot.model.gaussian_process.base.GaussianProcessRegressorBase*

Gaussian process regression (GPR).

Parameters

- **kernel** (*kernel instance*) – The covariance function of the GP.
- **alpha** (*float > 0*) – Value added to the diagonal of the kernel matrix during fitting. Larger values correspond to increased noise level in the observations. A practical usage of this parameter is to prevent potential numerical stability issues during fitting, and ensures that the kernel matrix is always positive definite in the precense of duplicate entries and/or round-off error.
- **beta** (*float > 0*) – Cutoff value on the singular values for the spectral pseudoinverse computation, which serves as a backup mechanism to invert the kernel matrix in case if it is singular.
- **optimizer** (*one of (str, True, None, callable)*) – A string or callable that represents one of the optimizers usable in the scipy.optimize.minimize method. If None, no hyperparameter optimization will be carried out in fitting. If True, the optimizer will default to L-BFGS-B.

- **normalize_y** (*boolean*) – Whether to normalize the target values y so that the mean and variance become 0 and 1, respectively. Recommended for cases where zero-mean, unit-variance kernels are used. The normalization will be reversed when the GP predictions are returned.
- **regularization** (*'+' or 'additive' or '*' or 'multiplicative'*) – Determines the method of regularization. If '+' or 'additive', `alpha` is added to the diagonals of the kernel matrix. If '*' or 'multiplicative', a factor of `1 + alpha` will be multiplied with each diagonal element.
- **kernel_options** (*dict, optional*) – A dictionary of additional options to be passed along when applying the kernel to data.

fit (*X*, *y*, *loss='likelihood'*, *tol=1e-05*, *repeat=1*, *theta_jitter=1.0*, *verbose=False*)

Train a GPR model. If the *optimizer* argument was set while initializing the GPR object, the hyperparameters of the kernel will be optimized using the specified loss function.

Parameters

- **X** (*list of objects or feature vectors.*) – Input values of the training data.
- **y** (*1D array*) – Output/target values of the training data.
- **loss** (*'likelihood' or 'loocv'*) – The loss function to be minimzed during training. Could be either 'likelihood' (negative log-likelihood) or 'loocv' (mean-square leave-one-out cross validation error).
- **tol** (*float*) – Tolerance for termination.
- **repeat** (*int*) – Repeat the hyperparameter optimization by the specified number of times and return the best result.
- **theta_jitter** (*float*) – Standard deviation of the random noise added to the initial logscale hyperparameters across repeated optimization runs.
- **verbose** (*bool*) – Whether or not to print out the optimization progress and outcome.

Returns **self** – returns an instance of self.

Return type *GaussianProcessRegressor*

fit_loocv (*X*, *y*, ***options*)

Alias of **:py:'fit_loocv(X, y, loss='loocv', **options)'**.

log_marginal_likelihood (*theta=None*, *X=None*, *y=None*, *eval_gradient=False*, *clone_kernel=True*, *verbose=False*)

Returns the log-marginal likelihood of a given set of log-scale hyperparameters.

Parameters

- **theta** (*array-like*) – Kernel hyperparameters for which the log-marginal likelihood is to be evaluated. If None, the current hyperparameters will be used.
- **X** (*list of objects or feature vectors.*) – Input values of the training data. If None, *self.X* will be used.
- **y** (*1D array*) – Output/target values of the training data. If None, *self.y* will be used.
- **eval_gradient** (*boolean*) – If True, the gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta will be returned alongside.
- **clone_kernel** (*boolean*) – If True, the kernel is copied so that probing with theta does not alter the trained kernel. If False, the kernel hyperparameters will be modified in-place.

- **verbose** (*boolean*) – If True, the log-likelihood value and its components will be printed to the screen.

Returns

- **log_likelihood** (*float*) – Log-marginal likelihood of theta for training data.
- **log_likelihood_gradient** (*1D array*) – Gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta. Only returned when eval_gradient is True.

predict (*Z*, *return_std=False*, *return_cov=False*)
Predict using the trained GPR model.

Parameters

- **Z** (*list of objects or feature vectors.*) – Input values of the unknown data.
- **return_std** (*boolean*) – If True, the standard-deviations of the predictions at the query points are returned along with the mean.
- **return_cov** (*boolean*) – If True, the covariance of the predictions at the query points are returned along with the mean.

Returns

- **ymean** (*1D array*) – Mean of the predictive distribution at query points.
- **std** (*1D array*) – Standard deviation of the predictive distribution at query points.
- **cov** (*2D matrix*) – Covariance of the predictive distribution at query points.

predict_loocv (*Z*, *z*, *return_std=False*)
Compute the leave-one-out cross validation prediction of the given data.

Parameters

- **Z** (*list of objects or feature vectors.*) – Input values of the unknown data.
- **z** (*1D array*) – Target values of the training data.
- **return_std** (*boolean*) – If True, the standard-deviations of the predictions at the query points are returned along with the mean.

Returns

- **ymean** (*1D array*) – Leave-one-out mean of the predictive distribution at query points.
- **std** (*1D array*) – Leave-one-out standard deviation of the predictive distribution at query points.

squared_loocv_error (*theta=None*, *X=None*, *y=None*, *eval_gradient=False*, *clone_kernel=True*, *verbose=False*)
Returns the squared LOOCV error of a given set of log-scale hyperparameters.

Parameters

- **theta** (*array-like*) – Kernel hyperparameters for which the log-marginal likelihood is to be evaluated. If None, the current hyperparameters will be used.
- **X** (*list of objects or feature vectors.*) – Input values of the training data. If None, *self.X* will be used.
- **y** (*1D array*) – Output/target values of the training data. If None, *self.y* will be used.

- **eval_gradient** (*boolean*) – If True, the gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta will be returned alongside.
- **clone_kernel** (*boolean*) – If True, the kernel is copied so that probing with theta does not alter the trained kernel. If False, the kernel hyperparameters will be modified in-place.
- **verbose** (*boolean*) – If True, the log-likelihood value and its components will be printed to the screen.

Returns

- **squared_error** (*float*) – Squared LOOCV error of theta for training data.
- **squared_error_gradient** (*1D array*) – Gradient of the Squared LOOCV error with respect to the kernel hyperparameters at position theta. Only returned when eval_gradient is True.

class graphdot.model.gaussian_process.**LowRankApproximateGPR**(*kernel*, *alpha=1e-07*, *beta=1e-07*, *optimizer=None*, *normalize_y=False*, *regularization='+'*, *kernel_options={}*)

Bases: *graphdot.model.gaussian_process.base.GaussianProcessRegressorBase*

Accelerated Gaussian process regression (GPR) using the Nystrom low-rank approximation.

Parameters

- **kernel** (*kernel instance*) – The covariance function of the GP.
- **alpha** (*float > 0*) – Value added to the diagonal of the core matrix during fitting. Larger values correspond to increased noise level in the observations. A practical usage of this parameter is to prevent potential numerical stability issues during fitting, and ensures that the core matrix is always positive definite in the precense of duplicate entries and/or round-off error.
- **beta** (*float > 0*) – Cutoff value on the singular values for the spectral pseudoinverse of the low-rank kernel matrix.
- **optimizer** (*one of (str, True, None, callable)*) – A string or callable that represents one of the optimizers usable in the scipy.optimize.minimize method. If None, no hyperparameter optimization will be carried out in fitting. If True, the optimizer will default to L-BFGS-B.
- **normalize_y** (*boolean*) – Whether to normalize the target values y so that the mean and variance become 0 and 1, respectively. Recommended for cases where zero-mean, unit-variance kernels are used. The normalization will be reversed when the GP predictions are returned.
- **regularization** (*'+' or 'additive' or '*' or 'multiplicative'*) – Determines the method of regularization. If '+' or 'additive', `alpha` is added to the diagonals of the kernel matrix. If '*' or 'multiplicative', a factor of `1 + alpha` will be multiplied with each diagonal element.
- **kernel_options** (*dict, optional*) – A dictionary of additional options to be passed along when applying the kernel to data.

C

The core sample set for constructing the subspace for low-rank approximation.

fit(*C*, *X*, *y*, *loss='likelihood'*, *tol=1e-05*, *repeat=1*, *theta_jitter=1.0*, *verbose=False*)

Train a low-rank approximate GPR model. If the *optimizer* argument was set while initializing the GPR object, the hyperparameters of the kernel will be optimized using the specified loss function.

Parameters

- **C** (*list of objects or feature vectors.*) – The core set that defines the subspace of low-rank approximation.
- **X** (*list of objects or feature vectors.*) – Input values of the training data.
- **y** (*1D array*) – Output/target values of the training data.
- **loss** (*'likelihood' or 'loocv'*) – The loss function to be minimzed during training. Could be either 'likelihood' (negative log-likelihood) or 'loocv' (mean-square leave-one-out cross validation error).
- **tol** (*float*) – Tolerance for termination.
- **repeat** (*int*) – Repeat the hyperparameter optimization by the specified number of times and return the best result.
- **theta_jitter** (*float*) – Standard deviation of the random noise added to the initial logscale hyperparameters across repeated optimization runs.
- **verbose** (*bool*) – Whether or not to print out the optimization progress and outcome.

Returns **self** – returns an instance of self.

Return type *LowRankApproximateGPR*

log_marginal_likelihood(*theta=None*, *C=None*, *X=None*, *y=None*, *eval_gradient=False*, *clone_kernel=True*, *verbose=False*)

Returns the log-marginal likelihood of a given set of log-scale hyperparameters.

Parameters

- **theta** (*array-like*) – Kernel hyperparameters for which the log-marginal likelihood is to be evaluated. If None, the current hyperparameters will be used.
- **C** (*list of objects or feature vectors.*) – The core set that defines the subspace of low-rank approximation. If None, *self.C* will be used.
- **X** (*list of objects or feature vectors.*) – Input values of the training data. If None, *self.X* will be used.
- **y** (*1D array*) – Output/target values of the training data. If None, *self.y* will be used.
- **eval_gradient** (*boolean*) – If True, the gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta will be returned alongside.
- **clone_kernel** (*boolean*) – If True, the kernel is copied so that probing with theta does not alter the trained kernel. If False, the kernel hyperparameters will be modified in-place.
- **verbose** (*boolean*) – If True, the log-likelihood value and its components will be printed to the screen.

Returns

- **log_likelihood** (*float*) – Log-marginal likelihood of theta for training data.
- **log_likelihood_gradient** (*1D array*) – Gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta. Only returned when eval_gradient is True.

predict(*Z*, *return_std=False*, *return_cov=False*)
Predict using the trained GPR model.

Parameters

- **Z** (*list of objects or feature vectors.*) – Input values of the unknown data.
- **return_std** (*boolean*) – If True, the standard-deviations of the predictions at the query points are returned along with the mean.
- **return_cov** (*boolean*) – If True, the covariance of the predictions at the query points are returned along with the mean.

Returns

- **ymean** (*1D array*) – Mean of the predictive distribution at query points.
- **std** (*1D array*) – Standard deviation of the predictive distribution at query points.
- **cov** (*2D matrix*) – Covariance of the predictive distribution at query points.

predict_loocv(*Z*, *z*, *return_std=False*, *method='auto'*)
Compute the leave-one-out cross validation prediction of the given data.

Parameters

- **Z** (*list of objects or feature vectors.*) – Input values of the unknown data.
- **z** (*1D array*) – Target values of the training data.
- **return_std** (*boolean*) – If True, the standard-deviations of the predictions at the query points are returned along with the mean.
- **method** (*'auto' or 'ridge-like' or 'gpr-like'*) – Selects the algorithm used for fast evaluation of the leave-one-out cross validation without expliciting training one model per sample. 'ridge-like' seems to be more stable with a smaller core size (that is not rank-deficit), while 'gpr-like' seems to be more stable with a larger core size. By default, the option is 'auto' and the function will choose a method based on an analysis on the eigenspectrum of the dataset.

Returns

- **ymean** (*1D array*) – Leave-one-out mean of the predictive distribution at query points.
- **std** (*1D array*) – Leave-one-out standard deviation of the predictive distribution at query points.

class graphdot.model.gaussian_process.**GPROutlierDetector**(*kernel*, *sigma_bounds=(0.0001*, *inf)*, *beta=1e-08*, *optimizer=True*, *normalize_y=False*, *kernel_options={}*)
Bases: *graphdot.model.gaussian_process.base.GaussianProcessRegressorBase*

Gaussian process regression (GPR) with noise/outlier detection via maximum likelihood estimation.

Parameters

- **kernel** (*kernel instance*) – The covariance function of the GP.
- **sigma_bounds** (*a tuple of two floats*) – As Value added to the diagonal of the kernel matrix during fitting. The 2-tuple will be regarded as the lower and upper bounds of

the values added to each diagonal element, which will be optimized individually by training. Larger values correspond to increased noise level in the observations. A practical usage of this parameter is to prevent potential numerical stability issues during fitting, and ensures that the kernel matrix is always positive definite in the precense of duplicate entries and/or round-off error.

- **beta** (*float > 0*) – Cutoff value on the singular values for the spectral pseudoinverse computation, which serves as a backup mechanism to invert the kernel matrix in case if it is singular.
- **optimizer** (*one of (str, True, None, callable)*) – A string or callable that represents one of the optimizers usable in the scipy.optimize.minimize method. If None, no hyperparameter optimization will be carried out in fitting. If True, the optimizer will default to L-BFGS-B.
- **normalize_y** (*boolean*) – Whether to normalize the target values y so that the mean and variance become 0 and 1, respectively. Recommended for cases where zero-mean, unit-variance kernels are used. The normalization will be reversed when the GP predictions are returned.
- **kernel_options** (*dict, optional*) – A dictionary of additional options to be passed along when applying the kernel to data.

fit (*X*, *y*, *w*, *udist=None*, *tol=0.0001*, *repeat=1*, *theta_jitter=1.0*, *verbose=False*)

Train a GPR model. If the *optimizer* argument was set while initializing the GPR object, the hyperparameters of the kernel will be optimized using the specified loss function.

Parameters

- **X** (*list of objects or feature vectors.*) – Input values of the training data.
- **y** (*1D array*) – Output/target values of the training data.
- **w** (*float*) – The strength of L1 penalty on the noise terms.
- **udist** (*callable*) – A random number generator for the initial guesses of the uncertainties. A lognormal distribution will be used by default if the argument is None.
- **tol** (*float*) – Tolerance for termination.
- **repeat** (*int*) – Repeat the hyperparameter optimization by the specified number of times and return the best result.
- **theta_jitter** (*float*) – Standard deviation of the random noise added to the initial logscale hyperparameters across repeated optimization runs.
- **verbose** (*bool*) – Whether or not to print out the optimization progress and outcome.

Returns **self** – returns an instance of self.

Return type *GaussianProcessRegressor*

log_marginal_likelihood (*theta_ext*, *X=None*, *y=None*, *eval_gradient=False*, *clone_kernel=True*, *verbose=False*)

Returns the log-marginal likelihood of a given set of log-scale hyperparameters.

Parameters

- **theta_ext** (*array-like*) – Kernel hyperparameters and per-sample noise prior for which the log-marginal likelihood is to be evaluated. If None, the current hyperparameters will be used.

- **X** (*list of objects or feature vectors.*) – Input values of the training data. If None, *self.X* will be used.
- **y** (*1D array*) – Output/target values of the training data. If None, *self.y* will be used.
- **eval_gradient** (*boolean*) – If True, the gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta will be returned alongside.
- **clone_kernel** (*boolean*) – If True, the kernel is copied so that probing with theta does not alter the trained kernel. If False, the kernel hyperparameters will be modified in-place.
- **verbose** (*boolean*) – If True, the log-likelihood value and its components will be printed to the screen.

Returns

- **log_likelihood** (*float*) – Log-marginal likelihood of theta for training data.
- **log_likelihood_gradient** (*1D array*) – Gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta. Only returned when eval_gradient is True.

predict (*Z*, *return_std=False*, *return_cov=False*)

Predict using the trained GPR model.

Parameters

- **Z** (*list of objects or feature vectors.*) – Input values of the unknown data.
- **return_std** (*boolean*) – If True, the standard-deviations of the predictions at the query points are returned along with the mean.
- **return_cov** (*boolean*) – If True, the covariance of the predictions at the query points are returned along with the mean.

Returns

- **ymean** (*1D array*) – Mean of the predictive distribution at query points.
- **std** (*1D array*) – Standard deviation of the predictive distribution at query points.
- **cov** (*2D matrix*) – Covariance of the predictive distribution at query points.

y_uncertainty

The learned uncertainty magnitude of each training sample.

Submodules

graphdot.model.gaussian_process.base module

class graphdot.model.gaussian_process.base.**GaussianProcessRegressorBase**(*kernel*, *normalize_y*, *regularization*, *kernel_options*)

Bases: object

Base class for all Gaussian process regression (GPR) models.

X
The input values of the training set.

load(*path*, *filename='model.pkl'*)
Load a stored GaussianProcessRegressor model from a pickle file.

Parameters

- **path** (*str*) – The directory where the model is saved.
- **filename** (*str*) – The file name for the saved model.

static mask(*iterable*)

save(*path*, *filename='model.pkl'*, *overwrite=False*)
Save the trained GaussianProcessRegressor with the associated data as a pickle.

Parameters

- **path** (*str*) – The directory to store the saved model.
- **filename** (*str*) – The file name for the saved model.
- **overwrite** (*bool*) – If True, a pre-existing file will be overwritten. Otherwise, a runtime error will be raised.

y
The output/target values of the training set.

graphdot.model.gaussian_process.gpr module

class graphdot.model.gaussian_process.gpr.**GaussianProcessRegressor**(*kernel*, *alpha=1e-08*, *beta=1e-08*, *optimizer=None*, *normalize_y=False*, *regularization='+'*, *kernel_options={}*)

Bases: *graphdot.model.gaussian_process.base.GaussianProcessRegressorBase*

Gaussian process regression (GPR).

Parameters

- **kernel** (*kernel instance*) – The covariance function of the GP.
- **alpha** (*float > 0*) – Value added to the diagonal of the kernel matrix during fitting. Larger values correspond to increased noise level in the observations. A practical usage of this parameter is to prevent potential numerical stability issues during fitting, and ensures that the kernel matrix is always positive definite in the precense of duplicate entries and/or round-off error.
- **beta** (*float > 0*) – Cutoff value on the singular values for the spectral pseudoinverse computation, which serves as a backup mechanism to invert the kernel matrix in case if it is singular.
- **optimizer** (*one of (str, True, None, callable)*) – A string or callable that represents one of the optimizers usable in the scipy.optimize.minimize method. If None, no hyperparameter optimization will be carried out in fitting. If True, the optimizer will default to L-BFGS-B.
- **normalize_y** (*boolean*) – Whether to normalize the target values y so that the mean and variance become 0 and 1, respectively. Recommended for cases where zero-mean, unit-variance kernels are used. The normalization will be reversed when the GP predictions are returned.
- **regularization** (*'+' or 'additive' or '*' or 'multiplicative'*) – Determines the method of regularization. If '+' or 'additive', `alpha` is added to the diagonals of the kernel matrix. If '*' or 'multiplicative', a factor of `1 + alpha` will be multiplied with each diagonal element.
- **kernel_options** (*dict, optional*) – A dictionary of additional options to be passed along when applying the kernel to data.

fit(*X*, *y*, *loss='likelihood'*, *tol=1e-05*, *repeat=1*, *theta_jitter=1.0*, *verbose=False*)

Train a GPR model. If the *optimizer* argument was set while initializing the GPR object, the hyperparameters of the kernel will be optimized using the specified loss function.

Parameters

- **X** (*list of objects or feature vectors.*) – Input values of the training data.

- **y** (*1D array*) – Output/target values of the training data.
- **loss** (*'likelihood' or 'loocv'*) – The loss function to be minimzed during training. Could be either 'likelihood' (negative log-likelihood) or 'loocv' (mean-square leave-one-out cross validation error).
- **tol** (*float*) – Tolerance for termination.
- **repeat** (*int*) – Repeat the hyperparameter optimization by the specified number of times and return the best result.
- **theta_jitter** (*float*) – Standard deviation of the random noise added to the initial logscale hyperparameters across repeated optimization runs.
- **verbose** (*bool*) – Whether or not to print out the optimization progress and outcome.

Returns **self** – returns an instance of self.

Return type *GaussianProcessRegressor*

fit_loocv(*X*, *y*, ***options*)

Alias of **:py:'fit_loocv(X, y, loss='loocv', **options)'**.

log_marginal_likelihood(*theta=None*, *X=None*, *y=None*, *eval_gradient=False*, *clone_kernel=True*, *verbose=False*)

Returns the log-marginal likelihood of a given set of log-scale hyperparameters.

Parameters

- **theta** (*array-like*) – Kernel hyperparameters for which the log-marginal likelihood is to be evaluated. If None, the current hyperparameters will be used.
- **X** (*list of objects or feature vectors.*) – Input values of the training data. If None, *self.X* will be used.
- **y** (*1D array*) – Output/target values of the training data. If None, *self.y* will be used.
- **eval_gradient** (*boolean*) – If True, the gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta will be returned alongside.
- **clone_kernel** (*boolean*) – If True, the kernel is copied so that probing with theta does not alter the trained kernel. If False, the kernel hyperparameters will be modified in-place.
- **verbose** (*boolean*) – If True, the log-likelihood value and its components will be printed to the screen.

Returns

- **log_likelihood** (*float*) – Log-marginal likelihood of theta for training data.
- **log_likelihood_gradient** (*1D array*) – Gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta. Only returned when eval_gradient is True.

predict(*Z*, *return_std=False*, *return_cov=False*)

Predict using the trained GPR model.

Parameters

- **Z** (*list of objects or feature vectors.*) – Input values of the unknown data.
- **return_std** (*boolean*) – If True, the standard-deviations of the predictions at the query points are returned along with the mean.

- **return_cov** (*boolean*) – If True, the covariance of the predictions at the query points are returned along with the mean.

Returns

- **ymean** (*1D array*) – Mean of the predictive distribution at query points.
- **std** (*1D array*) – Standard deviation of the predictive distribution at query points.
- **cov** (*2D matrix*) – Covariance of the predictive distribution at query points.

predict_loocv(*Z*, *z*, *return_std=False*)

Compute the leave-one-out cross validation prediction of the given data.

Parameters

- **Z** (*list of objects or feature vectors.*) – Input values of the unknown data.
- **z** (*1D array*) – Target values of the training data.
- **return_std** (*boolean*) – If True, the standard-deviations of the predictions at the query points are returned along with the mean.

Returns

- **ymean** (*1D array*) – Leave-one-out mean of the predictive distribution at query points.
- **std** (*1D array*) – Leave-one-out standard deviation of the predictive distribution at query points.

squared_loocv_error(*theta=None*, *X=None*, *y=None*, *eval_gradient=False*, *clone_kernel=True*, *verbose=False*)

Returns the squared LOOCV error of a given set of log-scale hyperparameters.

Parameters

- **theta** (*array-like*) – Kernel hyperparameters for which the log-marginal likelihood is to be evaluated. If None, the current hyperparameters will be used.
- **X** (*list of objects or feature vectors.*) – Input values of the training data. If None, *self.X* will be used.
- **y** (*1D array*) – Output/target values of the training data. If None, *self.y* will be used.
- **eval_gradient** (*boolean*) – If True, the gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta will be returned alongside.
- **clone_kernel** (*boolean*) – If True, the kernel is copied so that probing with theta does not alter the trained kernel. If False, the kernel hyperparameters will be modified in-place.
- **verbose** (*boolean*) – If True, the log-likelihood value and its components will be printed to the screen.

Returns

- **squared_error** (*float*) – Squared LOOCV error of theta for training data.
- **squared_error_gradient** (*1D array*) – Gradient of the Squared LOOCV error with respect to the kernel hyperparameters at position theta. Only returned when eval_gradient is True.

graphdot.model.gaussian_process.nystrom module

class graphdot.model.gaussian_process.nystrom.**LowRankApproximateGPR**(*kernel*, *alpha=1e-07*, *beta=1e-07*, *optimizer=None*, *normalize_y=False*, *regularization='+'*, *kernel_options={}*)

Bases: *graphdot.model.gaussian_process.base.GaussianProcessRegressorBase*

Accelerated Gaussian process regression (GPR) using the Nystrom low-rank approximation.

Parameters

- **kernel** (*kernel instance*) – The covariance function of the GP.
- **alpha** (*float > 0*) – Value added to the diagonal of the core matrix during fitting. Larger values correspond to increased noise level in the observations. A practical usage of this parameter is to prevent potential numerical stability issues during fitting, and ensures that the core matrix is always positive definite in the precense of duplicate entries and/or round-off error.
- **beta** (*float > 0*) – Cutoff value on the singular values for the spectral pseudoinverse of the low-rank kernel matrix.
- **optimizer** (*one of (str, True, None, callable)*) – A string or callable that represents one of the optimizers usable in the scipy.optimize.minimize method. If None, no hyperparameter optimization will be carried out in fitting. If True, the optimizer will default to L-BFGS-B.
- **normalize_y** (*boolean*) – Whether to normalize the target values y so that the mean and variance become 0 and 1, respectively. Recommended for cases where zero-mean, unit-variance kernels are used. The normalization will be reversed when the GP predictions are returned.
- **regularization** (*'+' or 'additive' or '*' or 'multiplicative'*) – Determines the method of regularization. If '+' or 'additive', `alpha` is added to the diagonals of the kernel matrix. If '*' or 'multiplicative', a factor of `1 + alpha` will be multiplied with each diagonal element.
- **kernel_options** (*dict, optional*) – A dictionary of additional options to be passed along when applying the kernel to data.

C

The core sample set for constructing the subspace for low-rank approximation.

fit(*C*, *X*, *y*, *loss='likelihood'*, *tol=1e-05*, *repeat=1*, *theta_jitter=1.0*, *verbose=False*)

Train a low-rank approximate GPR model. If the *optimizer* argument was set while initializing the GPR object, the hyperparameters of the kernel will be optimized using the specified loss function.

Parameters

- **C** (*list of objects or feature vectors.*) – The core set that defines the subspace of low-rank approximation.
- **X** (*list of objects or feature vectors.*) – Input values of the training data.
- **y** (*1D array*) – Output/target values of the training data.
- **loss** (*'likelihood' or 'loocv'*) – The loss function to be minimzed during training. Could be either 'likelihood' (negative log-likelihood) or 'loocv' (mean-square leave-one-out cross validation error).
- **tol** (*float*) – Tolerance for termination.
- **repeat** (*int*) – Repeat the hyperparameter optimization by the specified number of times and return the best result.
- **theta_jitter** (*float*) – Standard deviation of the random noise added to the initial logscale hyperparameters across repeated optimization runs.
- **verbose** (*bool*) – Whether or not to print out the optimization progress and outcome.

Returns **self** – returns an instance of self.

Return type *LowRankApproximateGPR*

log_marginal_likelihood(*theta=None*, *C=None*, *X=None*, *y=None*, *eval_gradient=False*, *clone_kernel=True*, *verbose=False*)

Returns the log-marginal likelihood of a given set of log-scale hyperparameters.

Parameters

- **theta** (*array-like*) – Kernel hyperparameters for which the log-marginal likelihood is to be evaluated. If None, the current hyperparameters will be used.
- **C** (*list of objects or feature vectors.*) – The core set that defines the subspace of low-rank approximation. If None, *self.C* will be used.
- **X** (*list of objects or feature vectors.*) – Input values of the training data. If None, *self.X* will be used.
- **y** (*1D array*) – Output/target values of the training data. If None, *self.y* will be used.
- **eval_gradient** (*boolean*) – If True, the gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta will be returned alongside.
- **clone_kernel** (*boolean*) – If True, the kernel is copied so that probing with theta does not alter the trained kernel. If False, the kernel hyperparameters will be modified in-place.
- **verbose** (*boolean*) – If True, the log-likelihood value and its components will be printed to the screen.

Returns

- **log_likelihood** (*float*) – Log-marginal likelihood of theta for training data.
- **log_likelihood_gradient** (*1D array*) – Gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta. Only returned when eval_gradient is True.

predict(*Z*, *return_std=False*, *return_cov=False*)

Predict using the trained GPR model.

Parameters

- **Z** (*list of objects or feature vectors.*) – Input values of the unknown data.
- **return_std** (*boolean*) – If True, the standard-deviations of the predictions at the query points are returned along with the mean.
- **return_cov** (*boolean*) – If True, the covariance of the predictions at the query points are returned along with the mean.

Returns

- **ymean** (*1D array*) – Mean of the predictive distribution at query points.
- **std** (*1D array*) – Standard deviation of the predictive distribution at query points.
- **cov** (*2D matrix*) – Covariance of the predictive distribution at query points.

predict_loocv(*Z*, *z*, *return_std=False*, *method='auto'*)

Compute the leave-one-out cross validation prediction of the given data.

Parameters

- **Z** (*list of objects or feature vectors.*) – Input values of the unknown data.
- **z** (*1D array*) – Target values of the training data.
- **return_std** (*boolean*) – If True, the standard-deviations of the predictions at the query points are returned along with the mean.
- **method** (*'auto' or 'ridge-like' or 'gpr-like'*) – Selects the algorithm used for fast evaluation of the leave-one-out cross validation without expliciting training one model per sample. 'ridge-like' seems to be more stable with a smaller core size (that is not rank-deficit), while 'gpr-like' seems to be more stable with a larger core size. By default, the option is 'auto' and the function will choose a method based on an analysis on the eigenspectrum of the dataset.

Returns

- **ymean** (*1D array*) – Leave-one-out mean of the predictive distribution at query points.
- **std** (*1D array*) – Leave-one-out standard deviation of the predictive distribution at query points.

graphdot.model.gaussian_process.outlier_detector module

class graphdot.model.gaussian_process.outlier_detector.**GPROutlierDetector**(*kernel*, *sigma_bounds=(0.0001, inf)*, *beta=1e-08*, *optimizer=True*, *normalize_y=False*, *kernel_options={}*)

Bases: *graphdot.model.gaussian_process.base.GaussianProcessRegressorBase*

Gaussian process regression (GPR) with noise/outlier detection via maximum likelihood estimation.

Parameters

- **kernel** (*kernel instance*) – The covariance function of the GP.
- **sigma_bounds** (*a tuple of two floats*) – As Value added to the diagonal of the kernel matrix during fitting. The 2-tuple will be regarded as the lower and upper bounds of the values added to each diagonal element, which will be optimized individually by training. Larger values correspond to increased noise level in the observations. A practical usage of this parameter is to prevent potential numerical stability issues during fitting, and ensures that the kernel matrix is always positive definite in the precense of duplicate entries and/or round-off error.
- **beta** (*float > 0*) – Cutoff value on the singular values for the spectral pseudoinverse computation, which serves as a backup mechanism to invert the kernel matrix in case if it is singular.
- **optimizer** (*one of (str, True, None, callable)*) – A string or callable that represents one of the optimizers usable in the scipy.optimize.minimize method. If None, no hyperparameter optimization will be carried out in fitting. If True, the optimizer will default to L-BFGS-B.
- **normalize_y** (*boolean*) – Whether to normalize the target values y so that the mean and variance become 0 and 1, respectively. Recommended for cases where zero-mean, unit-variance kernels are used. The normalization will be reversed when the GP predictions are returned.
- **kernel_options** (*dict, optional*) – A dictionary of additional options to be passed along when applying the kernel to data.

fit (*X*, *y*, *w*, *udist=None*, *tol=0.0001*, *repeat=1*, *theta_jitter=1.0*, *verbose=False*)

Train a GPR model. If the *optimizer* argument was set while initializing the GPR object, the hyperparameters of the kernel will be optimized using the specified loss function.

Parameters

- **X** (*list of objects or feature vectors.*) – Input values of the training data.
- **y** (*1D array*) – Output/target values of the training data.
- **w** (*float*) – The strength of L1 penalty on the noise terms.
- **udist** (*callable*) – A random number generator for the initial guesses of the uncertainties. A lognormal distribution will be used by default if the argument is None.
- **tol** (*float*) – Tolerance for termination.
- **repeat** (*int*) – Repeat the hyperparameter optimization by the specified number of times and return the best result.
- **theta_jitter** (*float*) – Standard deviation of the random noise added to the initial logscale hyperparameters across repeated optimization runs.
- **verbose** (*bool*) – Whether or not to print out the optimization progress and outcome.

Returns **self** – returns an instance of self.

Return type *GaussianProcessRegressor*

log_marginal_likelihood (*theta_ext*, *X=None*, *y=None*, *eval_gradient=False*, *clone_kernel=True*, *verbose=False*)

Returns the log-marginal likelihood of a given set of log-scale hyperparameters.

Parameters

- **theta_ext** (*array-like*) – Kernel hyperparameters and per-sample noise prior for which the log-marginal likelihood is to be evaluated. If None, the current hyperparameters will be used.
- **X** (*list of objects or feature vectors.*) – Input values of the training data. If None, *self.X* will be used.
- **y** (*1D array*) – Output/target values of the training data. If None, *self.y* will be used.
- **eval_gradient** (*boolean*) – If True, the gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta will be returned alongside.
- **clone_kernel** (*boolean*) – If True, the kernel is copied so that probing with theta does not alter the trained kernel. If False, the kernel hyperparameters will be modified in-place.
- **verbose** (*boolean*) – If True, the log-likelihood value and its components will be printed to the screen.

Returns

- **log_likelihood** (*float*) – Log-marginal likelihood of theta for training data.
- **log_likelihood_gradient** (*1D array*) – Gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta. Only returned when eval_gradient is True.

predict(*Z*, *return_std=False*, *return_cov=False*)

Predict using the trained GPR model.

Parameters

- **Z** (*list of objects or feature vectors.*) – Input values of the unknown data.
- **return_std** (*boolean*) – If True, the standard-deviations of the predictions at the query points are returned along with the mean.
- **return_cov** (*boolean*) – If True, the covariance of the predictions at the query points are returned along with the mean.

Returns

- **ymean** (*1D array*) – Mean of the predictive distribution at query points.
- **std** (*1D array*) – Standard deviation of the predictive distribution at query points.
- **cov** (*2D matrix*) – Covariance of the predictive distribution at query points.

y_uncertainty

The learned uncertainty magnitude of each training sample.

graphdot.model.tree_search package

class graphdot.model.tree_search.**MCTSGraphTransformer**(*rewriter*, *surrogate*, *exploration_bias=1.0*, *precision=0.01*)

Bases: object

A varient of Monte Carlo tree search for optimization and root-finding in a space of graphs.

Parameters

- **rewriter** (*callable*) – A callable that implements the `Rewriter` abstract class.
- **surrogate** (*object*) – A predictor used to calculate the target property of a given graph.
- **exploration_bias** (*float*) – Tunes the preference of the MCTS model between exploitation and exploration of the search space.
- **precision** (*float*) – Target precision of MCTS search outcome.

seek(*g0*, *target*, *maxiter=500*, *return_tree=False*, *random_state=None*)

Transforms an initial graph into one with a specific desired target property value.

Parameters

- **g0** (*object*) – A graph to start the tree search with.
- **target** (*float*) – Target property value of the desired graph.
- **maxiter** (*int*) – Maximum number of MCTS iterations to perform.
- **return_tree** (*Boolean*) – Whether or not to return the search tree in its original form or as a flattened dataframe.
- **random_state** (int or **:py:'np.random.Generator'**) – The seed to the random number generator (RNG), or the RNG itself. If None, the default RNG in numpy will be used.

Returns **tree** – If *return_tree* is True, a hierarchical dataframe representing the search tree will be returned; otherwise, a flattened dataframe will be returned.

Return type DataFrame

class graphdot.model.tree_search.**LookAheadSequenceRewriter**(*n=1*, *b=3*, *min_edits=1*, *max_edits=5*, *p_insert=1*, *p_mutate=1*, *p_delete=1*, *random_state=None*)

Bases: `graphdot.model.tree_search._rewriter.AbstractRewriter`

A sequence rewriter that performs contextual updates to a symbol sequence using the n-gram preceding the location of modification. It can carry out three types of operations:

- Insertion: insert an symbol at a random location. The symbol inserted should be probabilistically determined by up to **n** items in front of it unless when there are less than n symbols in the front, or when there is no matching n-gram in the training set. In that case, the longest matching k-gram (k < n) is used.
- Mutation: replace an symbol by a random one. This is context-sensitive.
- Deletion: remove an symbol at random from a sequence. This is context-insensitive.

Parameters

- **n** (*int*) – The maximum number of items to look ahead for contextual rewrites.
- **b** (*int*) – The branching factor, i.e. the number of new sequences to create from each input sequence.
- **min_edits** (*int*) – The minimum number of edits made to create a new sequence.
- **max_edits** (*int*) – The maximum number of edits made to create a new sequence.
- **p** (*list of three numbers*) – The relative frequencies of insertation, mutation, and deletion operations.

- **random_state** (*np.random.Generator or int*) – Initial state for the internal RNG.

class Payload(***kwargs*)
Bases: object

__call__(*s*)
Generate b offspring sequences, each being rewritten at least min_edits and at most max_edits times.

Parameters **s** (*sequence*) – The sequence to be rewritten.

Returns **T** – A collection of unique offspring sequences

Return type list of sequences

fit(*X*)
Learn the n-gram distribution from the given dataset.

Parameters **X** (*list of sequences*) – The training set.

tree
A tree-representation of the 1- to n-gram distributions of the training set.

Submodules

graphdot.model.tree_search.graph_transformer module

class graphdot.model.tree_search.graph_transformer.**MCTSGraphTransformer**(*rewriter*, *surrogate*, *exploration_bias=1.0*, *precision=0.01*)
Bases: object

A varient of Monte Carlo tree search for optimization and root-finding in a space of graphs.

Parameters

- **rewriter** (*callable*) – A callable that implements the Rewriter abstract class.
- **surrogate** (*object*) – A predictor used to calculate the target property of a given graph.
- **exploration_bias** (*float*) – Tunes the preference of the MCTS model between exploitation and exploration of the search space.
- **precision** (*float*) – Target precision of MCTS search outcome.

seek(*g0*, *target*, *maxiter=500*, *return_tree=False*, *random_state=None*)
Transforms an initial graph into one with a specific desired target property value.

Parameters

- **g0** (*object*) – A graph to start the tree search with.
- **target** (*float*) – Target property value of the desired graph.

- **maxiter** (*int*) – Maximum number of MCTS iterations to perform.
- **return_tree** (*Boolean*) – Whether or not to return the search tree in its original form or as a flattened dataframe.
- **random_state** (int or **:py:'np.random.Generator'**) – The seed to the random number generator (RNG), or the RNG itself. If None, the default RNG in numpy will be used.

Returns **tree** – If *return_tree* is True, a hierarchical dataframe representing the search tree will be returned; otherwise, a flattened dataframe will be returned.

Return type DataFrame

2.5.6 graphdot.experimental package

Subpackages

graphdot.experimental.alterantive_mgk package

class graphdot.experimental.alterantive_mgk.**AltMarginalizedGraphKernel**(**args*, ***kwargs*)

Bases: graphdot.kernel.marginalized._kernel.MarginalizedGraphKernel

__call__(*X*, *ij*, *lmin=0*, *timing=False*)

Compute a list of pairwise similarities between graphs.

Parameters

- **X** (*list of N graphs*) – The graphs must all have same node and edge attributes.
- **ij** (*list of pairs of ints*) – Pair indices for which the graph kernel is to be evaluated.
- **lmin** (*0 or 1*) – Number of steps to skip in each random walk path before similarity is computed. (lmin + 1) corresponds to the starting value of l in the summation of Eq. 1 in Tang & de Jong, 2019 https://doi.org/10.1063/1.5078640 (or the first unnumbered equation in Section 3.3 of Kashima, Tsuda, and Inokuchi, 2003).

Returns **gramian** – A vector with the same length as ij

Return type ndarray

trait_t

alias of Traits

classmethod traits(*lmin=0*)

graphdot.experimental.metric package

class graphdot.experimental.metric.**M3**(*use_charge=False*, *adjacency='default'*, *q=0.01*, *element_delta=0.2*, *bond_eps=0.02*, *charge_eps=0.2*)

Bases: object

The Marginalized MiniMax (M3) metric between molecules

__call__(*atoms1*, *atoms2*)

Call self as a function.

Submodules

graphdot.experimental.metric.m3 module

class graphdot.experimental.metric.m3.**M3**(*use_charge=False*, *adjacency='default'*, *q=0.01*, *element_delta=0.2*, *bond_eps=0.02*, *charge_eps=0.2*)

Bases: `object`

The Marginalized MiniMax (M3) metric between molecules

__call__(*atoms1*, *atoms2*)
Call self as a function.

2.6 How to contribute

GraphDot is an open-source project released under a BSD license. Everyone is welcome to contribute.

The project is hosted on GitLab. For questions, suggestions and bug reports, please take advantage of the issue tracking system. In addition, contributions are very welcomed and could be submitted as merge requests.

2.6.1 Submitting a bug report or a feature request

Please feel free to open an issue should you run into a bug or wish a feature could be implemented.

When submitting an issue, please try to follow the guidelines below:

- Include a minimal reproducible example of the issue for bug reports.
- Provide a mock code snippt for feature suggestions.
- Provide a full traceback when an exception is raised.
- Please include your operating system type and version number, as well as your Python, graphdot, numpy, and pycuda versions. This information can be found by running:

```
import platform; print(platform.platform())
import sys; print('Python', sys.version)
import graphdot; print('GraphDot', graphdot.__version__)
import numpy; print('NumPy', numpy.__version__)
import pycuda; print('PyCUDA', pycuda.VERSION)
import sympy; print('SymPy', sympy.__version__)
```

2.6.2 Contributing Code

The most recommended way to contribute to GraphDot is to fork the main repository, then submit a "merge request" following the procedure below:

1. Fork the project repository.
2. Clone your own fork to local disk via `git clone`
3. *Setting up the development environment*
4. Create a branch for development via `git checkout -b feature/<feature-name> master` (replace `feature-name` with the actual name of the feature).

5. Make changes on the feature branch
6. Test the changes with *Quality assurance measures*.
7. Push the completed feature to your own fork, then create a merge request.

2.6.3 Development Guide

Setting up the development environment

A recipe is provided in the project's Makefile for creating a Python virtual environment containing all dependencies for development:

```
make setup
```

Alternatively, a virtual environment can be set up manually by:

```
virtualenv venv
source venv/bin/activate
pip install -r requirements/common.txt
pip install -r requirements/tests.txt
pip install -r requirements/docs.txt
```

Python code style

Variable, function, file, module, and package names should use all lower case letters with underscores being word separators. For example, use `module_name.py` instead of `Module-Name.py`, and `some_function()` instead of `SomeFunction()`. Class names should use the Pascal case. For example, use `ClassName` instead of `class_name`.

In addition, the PEP8 style guide should be followed. An (incomplete) summary of the style guide is:

- 4 spaces per indentation level
- 79 characters at most per line
- 2 blank lines around top-level functions and class definitions
- 1 blank line around method definitions inside a class
- 1 module per import line, imports always at top of file
- UTF-8 source file encoding
- 0 space on the inner side of parentheses, e.g. use `(x + y)` instead of `( x + y )`.
- 1 space on both sides of binary operators (including assignments), e.g. use `x = y * 3 + z` instead of `x=y*3+z`.
- 0 space around keyword argument assignments, e.g. use `f(x=1)` instead of `f(x = 1)`.

Comformance to the style guide can be checked via *Code style check*.

C++ code style

A configuration file for clang-format has been defined in the root directory of the repository. It can be used to format C++ files using:

```
clang-format -i files
```

Quality assurance measures

Unit tests

```
make test
```

Or alternatively

```
tox -e py37  # or py35, py36 etc.
```

Code style check

```
make lint
```

Coverage test

```
make test-coverage
```

Or alternatively

```
tox -e coverage
```

Coverage reports are stored in the `htmlcov` directory.

Performance Benchmark

```
tox -e benchmark
```

CHAPTER 3

Citation

Like the package? Please cite:

- Tang, Yu-Hang, and Wibe A. de Jong. “Prediction of atomization energy using graph kernel and active learning.” The Journal of chemical physics 150, no. 4 (2019): 044107.
- Tang, Yu-Hang, Oguz Selvitopi, Doru Thom Popovici, and Aydın Buluç. “A High-Throughput Solver for Marginalized Graph Kernels on GPU.” In 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 728-738. IEEE, 2020.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

CHAPTER 5

Contributors

Yu-Hang “Maxin” Tang, Oguz Selvitopi, Doru Popovici, Yin-Jia Zhang

CHAPTER 6

Copyright

CHAPTER 7

Funding Acknowledgment

This work was supported by the Luis W. Alvarez Postdoctoral Fellowship at Lawrence Berkeley National Laboratory. This work is also supported in part by the Applied Mathematics program of the DOE Office of Advanced Scientific Computing Research under Contract No. DE-AC02-05CH11231, and in part by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

Python Module Index

g

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

P

R

S

T

U

V

W

X

Y