CatLearn Documentation

SUNCAT

May 17, 2019

User Guide:

CatLearn provides utilities for building and testing atomistic machine learning models for surface science and catalysis.

Note: This is part of the SUNCAT centers code base for understanding materials for catalytic applications. Other code is hosted on the center's Github repository.

CatLearn provides an environment to facilitate utilization of machine learning within the field of materials science and catalysis. Workflows are typically expected to utilize the Atomic Simulation Environment (ASE), or NetworkX graphs. Through close coupling with these codes, CatLearn can generate numerous embeddings for atomic systems. As well as generating a useful feature space for numerous problems, CatLearn has functions for model optimization. Further, Gaussian processes (GP) regression machine learning routines are implemented with additional functionality over standard implementations such as that in scikit-learn. A more detailed explanation of how to utilize the code can be found in the Tutorials folder.

To featurize ASE atoms objects, the following lines of code can be used:

```
import ase
from ase.cluster.cubic import FaceCenteredCubic

from catlearn.featurize.setup import FeatureGenerator

# First generate an atoms object.
surfaces = [(1, 0, 0), (1, 1, 0), (1, 1, 1)]
layers = [6, 9, 5]
lc = 3.61000
atoms = FaceCenteredCubic('Cu', surfaces, layers, latticeconstant=lc)

# Then generate some features.
generator = FeatureGenerator(nprocs=1)
features = generator.return_vec([atoms], [generator.eigenspectrum_vec,
                                          generator.composition_vec])
```

In the most basic form, it is possible to set up a GP model and make some predictions using the following lines of code:

```
import numpy as np
from catlearn.regression import GaussianProcess

# Define some input data.
train_features = np.arange(200).reshape(50, 4)
target = np.random.random_sample((50,))
test_features = np.arange(100).reshape(25, 4)

# Setup the kernel.
kernel = [{'type': 'gaussian', 'width': 0.5}]

# Train the GP model.
gp = GaussianProcess(kernel_list=kernel, regularization=1e-3,
                     train_fp=train_features, train_target=target,
```

(continues on next page)

(continued from previous page)

```
                    optimize_hyperparameters=True)

# Get the predictions.
prediction = gp.predict(test_fp=test_features)
```

There is much functionality in CatLearn to assist in handling atom data and building optimal models. This includes:

- API to other codes:
  - Atomic simulation environment API
  - Magpie API
  - NetworkX API
- Fingerprint generators:
  - Bulk systems
  - Support/slab systems
  - Discrete systems
- Preprocessing routines:
  - Data cleaning
  - Feature elimination
  - Feature engineering
  - Feature extraction
  - Feature scaling
- Regression methods:
  - Regularized ridge regression
  - Gaussian processes regression
- Cross-validation:
  - K-fold cv
  - Ensemble k-fold cv
- Optimize:
  - Machine Learning Accelerated Nudged Elastic Band ML-NEB
- General utilities:
  - K-means clustering
  - Neighborlist generators
  - Penalty functions
  - SQLite db storage

CHAPTER 1

Installation

A number of different methods can be used to run the CatLearn code.

1.1 Requirements

- ase
- h5py
- networkx
- numpy
- pandas
- scikit-learn
- scipy
- tqdm

1.2 Installation using pip

The easiest way to install CatLearn is with:

```
$ pip install catlearn
```

This will automatically install the code as well as the dependencies.

1.3 Installation from source

To get the most up-to-date development version of the code, you can clone the git repository to a local directory with:

```
$ git clone https://github.com/SUNCAT-Center/CatLearn.git
```

And then put the <install_dir>/ into your $PYTHONPATH environment variable. If you are using Windows, there is some advice on how to do that here.

Be sure to install dependencies in with:

```
$ pip install -r requirements.txt
```

CHAPTER 2

Changelog

CHAPTER 3

Version 0.6.1 (April 2019)

- Fixed compatibility issue with MLNEB and GPAW
- Various bugfixes

CHAPTER 4

Version 0.6.0 (January 2019)

- Added ML-MIN algorithm for energy minimization.
- Added ML-NEB algorithm for transition state search.
- Changed input format for kernels in the GP.

CHAPTER 5

Version 0.5.0 (October 2018)

- Restructure of fingerprint module
- Pandas DataFrame getter in FeatureGenerator
- CatMAP API using ASE database.
- New active learning module.
- Small fixes in adsorbate fingerprinter.

CHAPTER 6

Version 0.4.4 (August 2018)

- Major modifications to adsorbates fingerprinter
- Bag of site neighbor coordinations numbers implemented.
- Bag of connections implemented for adsorbate systems.
- General bag of connections implemented.
- Data cleaning function now return a dictionary with 'index' of clean features.
- New clean function to discard features with excessive skewness.
- New adsorbate-chalcogenide fingerprint generator.
- Enhancements to automatic identification of adsorbate, site.
- Generalized coordination number for site.
- Formal charges utility.
- New sum electronegativity over bonds fingerprinter.

CHAPTER 7

Version 0.4.3 (May 2018)

- `ConvolutedFingerprintGenerator` added for bulk and molecules.
- Dropped support for Python3.4 as it appears to start causing problems.

CHAPTER 8

Version 0.4.2 (May 2018)

- Genetic algorithm feature selection can parallelize over population within each generation.
- Default fingerprinter function sets accessible using `catlearn.fingerprint.setup.default_fingerprinters`
- New surrogate model utility
- New utility for evaluating cutoff radii for connectivity based fingerprinting.
- `default_catlearn_radius` improved.

CHAPTER 9

Version 0.4.1 (April 2018)

- AtoML renamed to CatLearn and moved to Github.
- Adsorbate fingerprinting again parallelizable.
- Adsorbate fingerprinting use atoms.tags to get layers if present.
- Adsorbate fingerprinting relies on connectivity matrix before neighborlist.
- New bond-electronegativity centered fingerprints for adsorbates.
- Fixed a bug that caused the negative log marginal likelihood to be attached to the gp class.
- Small speed improvement for initialize and updates to `GaussianProcess`.

CHAPTER 10

Version 0.4.0 (April 2018)

- Added `autogen_info` function for list of atoms objects representing adsorbates.
  - This can auto-generate all atomic group information and attach it to `atoms.info`.
  - Parallelized fingerprinting is not yet supported for output from `autogen_info`.
- Added `database_to_list` for import of atoms objects from ase.db with formatted metadata.
- Added function to translate a connection matrix to a formatted neighborlist dict.
- `periodic_table_data.list_mendeleev_params` now returns a numpy array.
- Magpie api added, allows for Voronoi and prototype feature generation.
- A genetic algorithm added for feature optimization.
- Parallelism updated to be compatable with Python2.
- Added in better neighborlist generation.
  - Updated wrapper for ase neighborlist.
  - Updated CatLearn neighborlist generator.
  - Defaults cutoffs changed to `atomic_radius` plus a relative tolerance.
- Added basic NetworkX api.
- Added some general functions to clean data and build a GP.
- Added a test for dependencies. Will raise a warning in the CI if things get out of date.
- Added a custom docker image for the tests. This is compiled in the `setup/` directory in root.
- Modified uncertainty output. The user can ask for the uncertainty with and without adding noise parameter (regularization).
- Clean up some bits of code, fix some bugs.

CHAPTER 11

Version 0.3.1 (February 2018)

- Added a parallel version of the greedy feature selection. **Python3 only!**
- Updated the k-fold cross-validation function to handle features and targets explicitly.
- Added some basic read/write functionality to the k-fold CV.
- A number of minor bugs have been fixed.

CHAPTER 12

Version 0.3.0 (February 2018)

- Update the fingerprint generator functions so there is now a `FeatureGenerator` class that wraps round all type specific generators.
- Feature generation can now be performed in parallel, setting `nprocs` variable in the `FeatureGenerator` class. **Python3 only!**
- Add better handling when passing variable length/composition data objects to the feature generators.
- More acquisition functions added.
- Penalty functions added.
- Started adding a general api for ASE.
- Added some more test and changed the way test are called/handled.
- A number of minor bugs have been fixed.

CHAPTER 13

Version 0.2.1 (February 2018)

- Update functions to compile features allowing for variable length of atoms objects.
- Added some tutorials for hierarchy cross-validation and prediction on organic molecules.

CHAPTER 14

Version 0.2.0 (January 2018)

- Gradients added to hyperparameter optimization.
- More features added to the adsorbate fingerprint generator.
- Acquisition function structure updated. Added new functions.
- Add some standardized input/output functions to save and load models.
- The kernel setup has been made more modular.
- Better test coverage, the tests have also been optimized for speed.
- Better CI configuration. The new method is much faster and more flexible.
- Added Dockerfile and appropriate documentation in the README and CONTRIBUTING guidelines.
- A number of minor bugs have been fixed.

CHAPTER 15

Version 0.1.0 (December 2017)

- The first stable version of the code base!
- For those that used the precious development version, there are many big changes in the way the code is structured. Most scripts will need to be rewritten.
- A number of minor bugs have been fixed.

CHAPTER 16

Contributing

16.1 General

There are some general coding conventions that the CatLearn repository adheres to. These include the following:

- Code should support Python 2.7, 3.4 and higher.
- Code should adhere to the pep8 and pyflakes style guides.
- Tests are run using TravisCI and coverage tracked using Coveralls.
- When new functions are added, tests should be written and added to the CI script.
- Documentation is hosted on Read the Docs at http://catlearn.readthedocs.io.
- Should use NumPy style docstrings.

16.2 Git Setup

We adhere to the git workflow described here, if you are considering contributing, please familiarize yourself with this. It is a bad idea to develop directly on the on the main CatLearn repository. Instead, fork a version into your own namespace on Github with the following:

- Fork the repository and then clone it to your local machine.

```
$ git clone https://github.com/SUNCAT-Center/CatLearn.git
```

- Add and track upstream to the local copy.

```
$ git remote add upstream https://github.com/SUNCAT-Center/CatLearn.git
```

All development can then be performed on the fork and a merge request opened into the upstream when appropriate. It is normally best to open merge requests as soon as possible, as it will allow everyone to see what is being worked on and comment on any potential issues.

16.3 Development

The following workflow is recommended when adding some new functionality:

- Before starting any new work, always sync with the upstream version.

```
$ git fetch upstream
$ git checkout master
$ git merge upstream/master --ff-only
```

- It is a good idea to keep the remote repository up to date.

```
$ git push origin master
```

- Start a new branch to do work on.

```
$ git checkout -b branch-name
```

- Once a file has been changed/created, add it to the staging area.

```
$ git add file-name
```

- Now commit it to the local repository and push it to the remote.

```
$ git commit -m 'some descriptive message'
$ git push --set-upstream origin branch-name
```

- When the desired changes have been made on your fork of the repository, open up a merge request on Github.

16.4 Environment

It is highly recommended to use `pipenv` for handling dependencies and the virtual environment, more information can be found here. Once installed, go to the root directory of CatLearn and use:

```
$ pipenv shell
```

From here it is possible to install and upgrade all the dependencies:

```
$ pipenv install --dev
$ pipenv update
```

There are a number of packages that may be important for the development cycle, these are installed with the `--dev` flag. There are then two ways to install additional dependencies required for new functionality, etc:

```
$ pipenv install package
$ pipenv install --dev package
```

The first command will install the package as a dependency for everyone using the code, e.g. people who install CatLearn with `pip` would be expected to also install this dependency. The second line will only install a package for developers. This workflow can even be used to keep the `requirements.txt` file up-to-date:

```
$ pipenv lock -r > requirements.txt
```

When complete, use `exit` to quit the virtualenv.

16.5 Docker

A docker image is included in the repository. It is sometimes easier to develop within a controlled environment such as this. In particular, it is possible for other developers to attain the same environment. To run CatLearn in the docker container, use the following commands:

```
$ docker build -t catlearn .
$ docker run -it catlearn bash
```

This will load up the CatLearn directory. To check that everything is working correctly simply run the following:

```
$ python2 test/test_suite.py
$ python3 test/test_suite.py
```

This will run the `test_suite.py` script with python version 2 and 3, respectively. If one version of python is preferred over the other, it is possible to create an alias as normal with:

```
$ alias python=python3
```

Use ctrl+d to exit.

To make changes to this, it is possible to simply edit the `Dockerfile`. To list the images available on the local system, use the following:

```
$ docker images
$ docker inspect REPOSITORY
```

It is a good idea to remove old images. This can be performed using the following lines:

```
$ docker rm $(docker ps -q -f status=exited)
$ docker rmi $(docker images -q -f "dangling=true")
```

16.6 Testing

When writing new code, please add some tests to ensure functionality doesn't break over time. We look at test coverage when merge requests are opened and will expect that coverage does not decrease due to large portions of new code not being tested. In CatLearn we just use the built-in unittest framework.

When commits are made, the CI will also automatically test if dependencies are up to date. This test is allowed to fail and will simply return a warning if a module in `requirements.txt` is out of date. This shouldn't be of concern and is mostly in place for us to keep track of changes in other code bases that could cause problems.

If changes are being made that change some core functionality, please run the `tutorials/test_notebooks.py` script. In general, the tutorials involve more demanding computations and thus are not run with the CI. The `test_notebooks.py` script will run through the various tutorials and make sure that they do not fail.

16.7 Tutorials

Where appropriate please consider adding some tutorials for new functionality. It would be great if they were written in jupyter notebook form, allowing for some detailed discussion of what is going on in the code.

CHAPTER 17

catlearn.api

17.1 catlearn.api.ase_atoms_api

Functions that interface ase with CatLearn.

catlearn.api.ase_atoms_api.**database_to_list**(*fname*, *selection=None*)
Return a list of atoms objects imported from an ase database.

Parameters

- **fname** (*str*) – path/filename of ase database.
- **selection** (*list*) – search filters to limit the import.

catlearn.api.ase_atoms_api.**extend_atoms_class**(*atoms*)
A wrapper to add extra functionality to ase atoms objects.

Parameters **atoms** (*class*) – An ase atoms object.

catlearn.api.ase_atoms_api.**get_features**(*self*)
Function to read feature vector from ase atoms object.

This function provides a uniform way in which to return a feature vector from an atoms object.

Parameters **self** (*class*) – An ase atoms object to attach feature vector to.

Returns **fp** – The feature vector attached to the atoms object.

Return type array

catlearn.api.ase_atoms_api.**get_graph**(*self*)
Function to read networkx graph from ase atoms object.

This function provides a uniform way in which to return a graph object from an atoms object.

Parameters **self** (*class*) – An ase atoms object to attach feature vector to.

Returns **graph** – The networkx graph object attached to the atoms object.

Return type object

catlearn.api.ase_atoms_api.**get_neighborlist**(*self*)
Function to read neighborlist from ase atoms object.

This function provides a uniform way in which to return a neighborlist from an atoms object.

Parameters **self** (*class*) – An ase atoms object to attach feature vector to.

Returns **neighborlist** – The neighbor list attached to the atoms object.

Return type dict

catlearn.api.ase_atoms_api.**images_connectivity**(*images*, *check_cn_max=False*)
Return a list of atoms objects imported from an ase database.

Parameters

- **fname** (*str*) – path/filename of ase database.
- **selection** (*list*) – search filters to limit the import.

catlearn.api.ase_atoms_api.**images_pair_distances**(*images*, *mic=True*)
Return a list of atoms objects imported from an ase database.

Parameters

- **fname** (*str*) – path/filename of ase database.
- **selection** (*list*) – search filters to limit the import.

catlearn.api.ase_atoms_api.**set_features**(*self*, *fp*)
Function to write feature vector to ase atoms object.

This function provides a uniform way in which to attach a feature vector to an atoms object. Can be used in conjunction with the *get_features* function.

Parameters

- **self** (*class*) – An ase atoms object to attach feature vector to.
- **fp** (*array*) – The feature vector to attach.

catlearn.api.ase_atoms_api.**set_graph**(*self*, *graph*)
Function to write networkx graph to ase atoms object.

This function provides a uniform way in which to attach a graph object to an atoms object. Can be used in conjunction with the *ase_to_networkx* function.

Parameters

- **self** (*class*) – An ase atoms object to attach feature vector to.
- **graph** (*object*) – The networkx graph object to attach.

catlearn.api.ase_atoms_api.**set_neighborlist**(*self*, *neighborlist*)
Function to write neighborlist to ase atoms object.

This function provides a uniform way in which to attach a neighbor list to an atoms object. Can be used in conjunction with the *get_neighborlist* function.

Parameters

- **self** (*class*) – An ase atoms object to attach feature vector to.
- **neighborlist** (*dict*) – The neighbor list dict to attach.

17.2 catlearn.api.ase_data_setup

Data generation functions to interact with ASE atoms objects.

catlearn.api.ase_data_setup.**get_train**(*atoms*, *key*, *size=None*, *taken=None*)
Return a training dataset.

Parameters

- **atoms** (*list*) – A list of ASE atoms objects.
- **size** (*int*) – Size of training dataset.
- **taken** (*list*) – List of candidates that have been used in unique dataset.
- **key** (*string*) – Property on which to base the predictions stored in the atoms object as atoms.info['key_value_pairs'][key].

catlearn.api.ase_data_setup.**get_unique**(*atoms*, *size*, *key*)
Return a unique test dataset.

Parameters

- **atoms** (*list*) – A list of ASE atoms objects.
- **size** (*int*) – Size of unique dataset to be returned.
- **key** (*string*) – Property on which to base the predictions stored in the atoms object as atoms.info['key_value_pairs'][key].

17.3 catlearn.api.networkx_graph_api

API to convert from ASE and NetworkX.

catlearn.api.networkx_graph_api.**ase_to_networkx**(*atoms*, *cutoffs=None*)
Make the NetworkX graph form ASE atoms object.

The graph is dependent on the generation of the neighborlist. Currently this is handled by the version implemented in ASE.

Parameters

- **atoms** (*object*) – An ASE atoms object.
- **cutoffs** (*list*) – A list of distance paramteres for each atom.

Returns **atoms_graph** – A networkx graph object.

Return type object

catlearn.api.networkx_graph_api.**matrix_to_nl**(*matrix*)
Returns a neighborlist as a dictionary. :param matrix: symmetric connection matrix. :type matrix: numpy array

Returns **nl** – neighborlist.

Return type dict

catlearn.api.networkx_graph_api.**networkx_to_adjacency**(*graph*)
Simple wrapper for graph to adjacency matrix.

Parameters **graph** (*object*) – The networkx graph object.

Returns **matrix** – The numpy adjacency matrix.

Return type array

CHAPTER 18

catlearn.cross_validation

18.1 catlearn.cross_validation.hierarchy_cv

Cross validation routines to work with feature database.

class catlearn.cross_validation.hierarchy_cv.**Hierarchy**(*file_name*, *db_name*, *table='FingerVector'*, *file_format='pickle'*)

Bases: object

Class to form hierarchy crossvalidation setup.

This class is used to cross-validate with respect to data size. The initial dataset is split in two and subsequent datasets split further until a minimum size is reached. Predictions are made on all subsets of data giving averaged error and certainty at each data size.

get_subset_data(*index_split*, *indicies*, *split=None*)
Make array with training data according to index.

Parameters

- **index_split** (*array*) – Array with the index data.
- **indicies** (*array*) – Index used to generate data.

globalscaledata(*index_split*)
Make an array with all data.

Parameters **index_split** (*array*) – Array with the index data.

load_split()
Function to load the split from file.

split_index(*min_split*, *max_split=None*, *all_index=None*)
Function to split up the db index to form subsets of data.

Parameters

- **min_split** (*int*) – Minimum size of a data subset.

- **max_split** (*int*) – Maximum size of a data subset.
- **all_index** (*list*) – List of indices in the feature database.

split_predict(*index_split*, *predict*, ***kwargs*)
Function to make predictions looping over all subsets of data.

Parameters

- **index_split** (*dict*) – All data for the split.
- **predict** (*function*) – The prediction function. Must return dict with 'result' in it.

Returns

- **result** (*list*) – A list of averaged errors for each subset of data.
- **size** (*list*) – A list of data sizes corresponding to the errors list.

todb(*features*, *targets*)
Function to convert numpy arrays to basic db.

transform_output(*data*)
Function to compile results in a format for plotting average error.

Parameters **data** (*dict*) – The dictionary output from the split_predict function.

Returns

- **size** (*list*) – A list of the data sizes used in the CV.
- **error** (*list*) – A list of the mean errors at each data size.

18.2 catlearn.cross_validation.k_fold_cv

Setup k-fold array split for cross validation.

catlearn.cross_validation.k_fold_cv.**k_fold**(*features*, *targets=None*, *nsplit=3*, *fix_size=None*)
Routine to split feature matrix and return sublists.

Parameters

- **features** (*array*) – An n, d feature array.
- **targets** (*list*) – A list to target values.
- **nsplit** (*int*) – The number of bins that data should be devided into.
- **fix_size** (*int*) – Define a fixed sample size, e.g. nsplit=5 fix_size=100, generates 5 x 100 data split. Default is None, all available data is divided nsplit times.

Returns

- **features** (*list*) – A list of feature arrays of length nsplit.
- **targets** (*list*) – A list of targets lists of length nsplit.

catlearn.cross_validation.k_fold_cv.**read_split**(*fname*, *fformat='pickle'*)
Function to read the k-fold split from file.

Parameters

- **fname** (*str*) – The name of the read file.
- **fformat** (*str*) – File format to read from. Can be json or pickle, default is pickle.

Returns

- **features** (*list*) – A list of feature arrays of length nsplit.
- **targets** (*list*) – A list of targets lists of length nsplit.

catlearn.cross_validation.k_fold_cv.**write_split**(*features*, *targets*, *fname*, *fformat='pickle'*)

Function to write the k-fild split to file.

Parameters

- **features** (*array*) – An n, d feature array.
- **targets** (*list*) – A list to target values.
- **fname** (*str*) – The name of the write file.
- **fformat** (*str*) – File format to write to. Can be json or pickle, default is pickle.

Cross validation functions.

CHAPTER 19

catlearn.featurize package

19.1 Submodules

19.2 catlearn.featurize.adsorbate_prep module

This function constructs a dictionary with abinitio_energies.

Input: fname (str) path/filename of ase.db file selection (list) ase.db selection

catlearn.featurize.adsorbate_prep.**ads_index**(*atoms*)
Returns a list of indices of atoms belonging to the adsorbate. These are defined as atoms that are not belonging to the slab.

Parameters **atoms** (*ase atoms object*) – The atoms object must have the key 'ads_atoms' in atoms.subsets:

- **'slab_atoms'** [list] indices of atoms belonging to the adsorbate

catlearn.featurize.adsorbate_prep.**attach_cations**(*atoms*, *anion_number=8*)
Attaches list of cation and anion atomic indices.

Parameters

- **atoms** (*object*) – ase.Atoms object.
- **anion_number** (*int*) – Atomic number of the anion of this chalcogenide.

catlearn.featurize.adsorbate_prep.**auto_layers**(*atoms*, *miller=(0, 0, 1)*)
Returns two arrays describing which layer each atom belongs to and the distance between the layers and origo. Assumes the tolerance corresponds to the average atomic radii of the slab.

Parameters **atoms** (*object*) – The atoms object must have the following keys in atoms.subsets:

'slab_atoms' [list] indices of atoms belonging to the slab

catlearn.featurize.adsorbate_prep.**autogen_info**(*images*)
Return a list of atoms objects with atomic group information attached to atoms.subsets. This information is needed by some functions in the AdsorbateFingerprintGenerator.

Parameters **images** (*list*) – list of atoms objects representing adsorbates on slabs. No further information is required in atoms.subsets.

catlearn.featurize.adsorbate_prep.**catalysis_hub_to_info**(*images*)

catlearn.featurize.adsorbate_prep.**check_reconstructions**(*image_pairs*)
Return a list of database ids, for adsorbate/slab structures, which has a reconstructed slab with respect to the reference slab.

Parameters **image_pairs** (*list*) – List of tuples containing pairs of ASE atoms objects. The first element in each tuple must represent an adsorbate*slab structure and the second element must represent a slab.

catlearn.featurize.adsorbate_prep.**compare_slab_connectivity**(*atoms*, *reference_atoms*)
Return a boolean for whether an adsorbate has caused a slab to reconstruct and change it's connectivity.

Parameters

- **atoms** (*object*) – ASE atoms object with connectivity and 'slab_atoms' subsets attached. This represents an adsorbate*slab structure.
- **reference_atoms** (*object*) – ASE atoms object with connectivity and 'slab_atoms' subsets attached. This represents a slab structure.

Returns **identical** – Are the connectivities within the slabs identical or not.

Return type boolean

catlearn.featurize.adsorbate_prep.**connectivity2ads_index**(*atoms*, *species*)
Return the indexes of atoms from the global list of adsorbate symbols.

Parameters

- **atoms** (*object*) – ASE atoms object with connectivity attached. This represents an adsorbate*slab structure.
- **species** (*str*) – chemical formula of the adsorbate.

catlearn.featurize.adsorbate_prep.**connectivity_termination**(*atoms*)
Return lists bulk, term and subsurf containing atom indices belonging to those subsets of a surface atoms object. This function relies on the connectivity of the atoms.

Parameters **atoms** (*object*) – atoms.connectivity should be a connectivity matrix. The atoms object must have the following keys in atoms.subsets:

'slab_atoms' [list] indices of atoms belonging to the slab

catlearn.featurize.adsorbate_prep.**constraints_termination**(*atoms*)
Return lists bulk, term and subsurf containing atom indices belonging to those subsets of a surface atoms object. This function relies on the connectivity of the atoms and it assumes that bulk atoms are those that have are constrained in the first constraint.

Parameters **atoms** (*object*) – atoms.connectivity should be a connectivity matrix. The atoms object must have the following keys in atoms.subsets:

'slab_atoms' [list] indices of atoms belonging to the slab.

catlearn.featurize.adsorbate_prep.**detect_adsorbate**(*atoms*)
Return a list of indices of atoms belonging to an adsorbate.

Parameters **atoms** (*object*) – An ase atoms object.

catlearn.featurize.adsorbate_prep.**detect_termination**(*atoms*)

Returns three lists, the first containing indices of bulk atoms and the second containing indices of atoms in the second outermost layer, and the last denotes atoms in the outermost layer or termination or the slab.

Parameters **atoms** (*object.*) – The atoms object must have the following keys in atoms.subsets:

'slab_atoms' [list] indices of atoms belonging to the slab

catlearn.featurize.adsorbate_prep.**formula2ads_index**(*atoms*, *species*)

Return the indexes of atoms, which have symbols matching the chemical formula of the adsorbate. This function will not work for adsorbates containing the same elements as the slab.

Parameters

- **atoms** (*ase atoms object.*) – atoms.info must be a dictionary containing the key 'key_value_pairs', which is expected to contain CatMAP standard adsorbate structure key value pairs. See the ase db to catmap module in catmap. the key value pair 'species' must be the chemical formula of the adsorbate.
- **species** (*str*) – chemical formula of the adsorbate.

catlearn.featurize.adsorbate_prep.**info2primary_index**(*atoms*)

Returns lists identifying the nearest neighbors of the adsorbate atoms.

Parameters **atoms** (*ase atoms object.*) – The atoms object must have the following keys in atoms.subsets:

'ads_atoms' [list] indices of atoms belonging to the adsorbate

'slab_atoms' [list] indices of atoms belonging to the slab

catlearn.featurize.adsorbate_prep.**last2ads_index**(*atoms*, *species*)

Return the indexes of the last n atoms in the atoms object, where n is the length of the composition of the adsorbate species. This function will work on atoms objects, where the slab was set up first, and the adsorbate was added after.

Parameters

- **atoms** (*ase atoms object.*) – atoms.info must be a dictionary containing the key 'key_value_pairs', which is expected to contain CATMAP standard adsorbate structure key value pairs. See the ase db to catmap module in catmap. the key value pair 'species' must be the chemical formula of the adsorbate.
- **species** (*str*) – chemical formula of the adsorbate.

catlearn.featurize.adsorbate_prep.**layers2ads_index**(*atoms*, *species*)

Returns the indexes of atoms in layers exceeding the number of layers stored in the key value pair 'layers'.

Parameters

- **atoms** (*ase atoms object.*) – atoms.info must be a dictionary containing the key 'key_value_pairs', which is expected to contain CatMAP standard adsorbate structure key value pairs. See the ase db to catmap module in catmap. the key value pair 'species' must be the chemical formula of the adsorbate and 'layers' must be an integer.
- **species** (*str*) – chemical formula of the adsorbate.

catlearn.featurize.adsorbate_prep.**layers_termination**(*atoms*, *miller=(0*, *0*, *1)*)

Return lists bulk, term and subsurf containing atom indices belonging to those subsets of a surface atoms object. This function relies on ase.atoms.get_layers, default atomic radii, and a slab oriented in the xy plane, where the termination in the z+ direction is the surface.

Parameters **atoms** (*object*) – The atoms object must have the following keys in atoms.subsets:

'slab_atoms' [list] indices of atoms belonging to the slab.

catlearn.featurize.adsorbate_prep.**slab_index**(*atoms*)

Returns a list of indices of atoms belonging to the slab. These are defined as atoms that are not belonging to the adsorbate.

Parameters **atoms** (*ase atoms object*) – The atoms object must have the key 'ads_atoms' in atoms.subsets:

- **'ads_atoms'** [list] indices of atoms belonging to the adsorbate

catlearn.featurize.adsorbate_prep.**slab_positions2ads_index**(*atoms*, *slab*, *species*)

Return the indexes of adsorbate atoms identified by comparing positions to a reference slab structure.

Parameters **atoms** (*object*) –

catlearn.featurize.adsorbate_prep.**sym2ads_index**(*atoms*, *ads_syms*)

Return the indexes of atoms from the global list of adsorbate symbols.

Parameters **atoms** (*object*) – An ase atoms object.

catlearn.featurize.adsorbate_prep.**tags2ads_index**(*atoms*)

Return the indexes of atoms from the global list of adsorbate symbols.

Parameters **atoms** (*object*) – An ase atoms object. *atoms.tags* must label adsorbate atoms with 0 or negative numbers.

catlearn.featurize.adsorbate_prep.**tags_termination**(*atoms*)

Return lists bulk, term and subsurf containing atom indices belonging to those subsets of a surface atoms object. CatKit and ase.build contain functions that by default store this information in tags.

Parameters **atoms** (*object*) – the termination atoms should have tag=1 and subsequent layers should be tagged in increasing order.

catlearn.featurize.adsorbate_prep.**termination_info**(*images*)

Return a list of atoms objects with attached information about the slab termination, the slab second outermost layer and the bulk slab compositions.

Parameters **images** (*list*) – list of atoms objects representing adsorbates on slabs. The atoms objects must have the following keys in atoms.subsets:

- **'ads_atoms'** [list] indices of atoms belonging to the adsorbate
- **'slab_atoms'** [list] indices of atoms belonging to the slab

catlearn.featurize.adsorbate_prep.**z2ads_index**(*atoms*, *species*)

Returns the indexes of the n atoms with the highest position in the z direction, where n is the number of atoms in the chemical formula from the species key value pair.

Parameters

- **atoms** (*ase atoms object.*) – atoms.info must be a dictionary containing the key 'key_value_pairs', which is expected to contain CatMAP standard adsorbate structure key value pairs. See the ase db to catmap module in catmap. the key value pair 'species'.
- **species** (*str*) – chemical formula of the adsorbate.

19.3 catlearn.featurize.asap_wrapper module

19.4 catlearn.featurize.base module

Base class for the feature generators.

This is inherited by the other fingerprint generators and allows access to a number of useful and commonly used functions. Standard functionality that is implemented and applicable to more than one of the other classes should be put here.

class catlearn.featurize.base.**BaseGenerator**(***kwargs*)

Bases: object

Base class for feature generation.

get_all_distances(*candidate*)

Function to return the atomic distances.

Parameters **candidate** (*object*) – Target data object from which to get the atomic distances.

get_atomic_numbers(*candidate*)

Function to return the atomic numbers.

Parameters **candidate** (*object*) – Target data object from which to get the atomic numbers.

get_masses(*candidate*)

Function to return the atomic masses.

Parameters **candidate** (*object*) – Target data object from which to get the atomic masses.

get_neighborlist(*candidate*)

Function to return the neighborlist.

It will check to see if the neighbor list is stored in the data object. If not it will generate the neighborlist from scratch.

Parameters **candidate** (*object*) – Target data object from which to get the neighbor list.

get_positions(*candidate*)

Function to return the atomic coordinates.

Parameters **candidate** (*object*) – Target data object from which to get the atomic coordinates.

make_neighborlist(*candidate*, *neighbor_number=1*)

Function to generate the neighborlist.

Parameters

- **candidate** (*object*) – Target data object on which to generate neighbor list.
- **dx** (*dict*) – Buffer to calculate nearest neighbor pairs in dict format: dx = {atomic_number: buffer}.
- **neighbor_number** (*int*) – Neighbor shell.

catlearn.featurize.base.**check_labels**(*labels*, *result*, *atoms*)

Check that two lists have the same length. If not, print an informative error message containing a databse id if present.

Parameters

- **labels** (*list*) – A list of feature names.
- **result** (*list*) – A fingerprint.
- **atoms** (*object*) – A single atoms object.

19.5 catlearn.featurize.neighbor_matrix module

Functions to build a neighbor matrix feature representation.

catlearn.featurize.neighbor_matrix.**connection_dict**(*atoms*, *periodic=False*, *dx=0.2*, *neighbor_number=1*, *reuse_nl=False*)

Generate a dict of atom connections.

Parameters

- **atoms** (*object*) – Target ase atoms object on which to build the connections matrix.
- **periodic** (*boolean*) – Specify whether to use the periodic neighborlist generator. None periodic method is faster and used by default.
- **dx** (*float*) – Buffer to calculate nearest neighbor pairs.
- **neighbor_number** (*int*) – Neighbor shell.
- **reuse_nl** (*boolean*) – Whether to reuse a previously stored neighborlist if available.

catlearn.featurize.neighbor_matrix.**connection_matrix**(*atoms*, *periodic=False*, *dx=0.2*, *neighbor_number=1*, *reuse_nl=False*)

Generate a connections matrix from an atoms object.

Parameters

- **atoms** (*object*) – Target ase atoms object on which to build the connections matrix.
- **periodic** (*boolean*) – Specify whether to use the periodic neighborlist generator. None periodic method is faster and used by default.
- **dx** (*float*) – Buffer to calculate nearest neighbor pairs.
- **neighbor_number** (*int*) – Neighbor shell.
- **reuse_nl** (*boolean*) – Whether to reuse a previously stored neighborlist if available.

catlearn.featurize.neighbor_matrix.**neighbor_features**(*atoms*, *property=None*, *periodic=False*, *dx=0.2*, *neighbor_number=1*, *reuse_nl=False*)

Generate predefined features from atoms objects.

Parameters

- **atoms** (*object*) – The target ase atoms object.
- **property** (*list*) – List of the target properties from mendeleev.
- **periodic** (*boolean*) – Specify whether to use the periodic neighborlist generator. None periodic method is faster and used by default.
- **dx** (*float*) – Buffer to calculate nearest neighbor pairs.
- **neighbor_number** (*int*) – Neighbor shell.

- **reuse_nl** (*boolean*) – Whether to reuse a previously stored neighborlist if available.

catlearn.featurize.neighbor_matrix.**property_matrix**(*atoms*, *property*)
Generate a property matrix based on the atomic types.

Parameters

- **atoms** (*object*) – The target ase atoms opject.
- **property** (*str*) – The target property from mendeleev.

19.6 catlearn.featurize.periodic_table_data module

Function pulling atomic data for elements.

This is typically used in conjunction with other fiungerprint generators to combine general atomic data with more specific properties.

catlearn.featurize.periodic_table_data.**default_catlearn_radius**(*z*)
Return the default CatLearn covalent radius of element z.

Parameters **z** (*int*) – Atomic number.

catlearn.featurize.periodic_table_data.**get_mendeleev_params**(*atomic_number*, *params=None*)
Return a list of generic parameters about an atom.

Parameters

- **atomic_number** (*list or int*) – An atomic number.
- **extra_params** (*list of str*) – Extra Mendaleev parameters to be returned in the list. For a full list see here - https://goo.gl/G4eTvu

Returns **var** – All parameters of the element with specified atomic number.

Return type list

catlearn.featurize.periodic_table_data.**get_radius**(*z, params=['atomic_radius', 'covalent_radius_cordero']*)
Return a metric of atomic radius.

Parameters

- **z** (*int*) – Atomic number.
- **params** (*list*) – Atomic radius metrics in order of preference. The first successful value will be returned.

catlearn.featurize.periodic_table_data.**list_mendeleev_params**(*numbers*, *params=None*)
Return an n by p array, containing p parameters of n atoms.

Parameters

- **numbers** (*list*) – atomic numbers.
- **params** (*list*) – elemental parameters.

catlearn.featurize.periodic_table_data.**make_labels**(*params*, *prefix*, *suffix*)
Return a list of feature labels.

Parameters

- **params** (*list*) – Parameter keys.

- **prefix** (*str*) – Appended in front of each parameter key.
- **suffix** (*str*) – Appended to end of each parameter key.

Returns **labels**

Return type list

catlearn.featurize.periodic_table_data.**n_outer**(*econf*)

catlearn.featurize.periodic_table_data.**stat_mendeleev_params**(*composition*, *params=None*)

Return an n by p array, containing p parameters of n atoms and stoichiometry weigths associated with the unique elements in the formula.

Parameters

- **composition** (*str*) – chemical composition formula. Floats are accepted.
- **params** (*list*) – elemental parameters.

19.7 catlearn.featurize.setup module

Functions to setup fingerprint vectors.

class catlearn.featurize.setup.**FeatureGenerator**(*atom_types=None*, *atom_len=None*, *nprocs=1*, ***kwargs*)

Bases: *catlearn.fingerprint.adsorbate.AdsorbateFingerprintGenerator*, *catlearn.fingerprint.particle.ParticleFingerprintGenerator*, *catlearn.fingerprint.standard.StandardFingerprintGenerator*, *catlearn.fingerprint.graph.GraphFingerprintGenerator*, *catlearn.fingerprint.bulk.BulkFingerprintGenerator*, *catlearn.fingerprint.convoluted.ConvolutedFingerprintGenerator*, *catlearn.fingerprint.chalcogenide.ChalcogenideFingerprintGenerator*, catlearn.fingerprint.catapp.CatappFingerprintGenerator, *catlearn.fingerprint.molecule.AutoCorrelationFingerprintGenerator*

Feature generator class.

It is sometimes necessary to normalize the length of feature vectors when data is supplied with variable numbers of atoms or elemental types. If this is the case, use the *normalize_features* function.

In this class, there are functions to take a data object and return a feature vector. This is done with the *return_vec* function. The names of the descriptors in the feature vector can be accessed with the *return_names* function.

The class inherits the actual generator functions from the [NAME]FingerprintGenerator classes. Additional variables are passed as kwargs.

featurize_atomic_pairs(*candidates*)

Featurize pairs of atoms by their elements and pair distances, in order to optimize the bond classifier.

Parameters **candidates** (*list of atoms objects.*) –

Returns **data** – Data matrix.

Return type array

get_dataframe(*candidates*, *vec_names*)

Sequentially combine feature vectors. Padding handled automatically.

Parameters

- **candidates** (*list or dict*) – Atoms objects to construct fingerprints for.

- **vec_name** (*list*) – List of fingerprinting functions.

Returns **df** – Fingerprint dataframe with n rows and m columns (n, m) where n is the number of candidates and m is the summed number of features from all fingerprint classes supplied.

Return type DataFrame

normalize_features(*train_candidates*, *test_candidates=None*)
Function to attach feature data to class.

Currently the function attaches data on all elemental types present in the data as well as the maximum number of atoms in a data object.

Parameters

- **train_candidates** (*list*) – List of atoms objects.
- **test_candidates** (*list*) – List of atoms objects.

return_names(*vec_names*)
Function to return a list of feature names.

Parameters **vec_name** (*list*) – List of fingerprinting functions.

Returns **fingerprint_vector** – Name array.

Return type ndarray

return_vec(*candidates*, *vec_names*)
Sequentially combine feature vectors. Padding handled automatically.

Parameters

- **candidates** (*list or dict*) – Atoms objects to construct fingerprints for.
- **vec_name** (*list*) – List of fingerprinting functions.

Returns **vector** – Fingerprint array (n, m) where n is the number of candidates and m is the summed number of features from all fingerprint classes supplied.

Return type ndarray

catlearn.featurize.setup.**default_fingerprinters**(*generator*, *data_type*)
"Return a list of generators.

Parameters

- **generator** (*object*) – FeatureGenerator object
- **data_type** (*str*) – 'bulk', 'adsorbates' or 'fragment'

Returns **vec_name** – List of fingerprinting classes.

Return type list of / single vec class(es)

19.8 catlearn.featurize.slab_utilities module

catlearn.featurize.slab_utilities.**is_metal**(*chemical_symbol*)
Checks whether string is a metal elementary symbol.

Parameters **chemical_symbol** (*string*) – The element name.

Returns **metal** – Whether it's a metal.

Return type Boolean

catlearn.featurize.slab_utilities.**is_oxide**(*atoms*)
Checks whether atms object is an oxide.

Parameters **atoms** (*object*) – ASE atoms object.

Returns **oxide** – Whether it is likely an oxide.

Return type Boolean

catlearn.featurize.slab_utilities.**slab_layers**(*atoms*, *max_layers=20*, *tolerance=0.5*)
Return a number of layers given a slab.

Parameters

- **atoms** (*object*) – ASE atoms object.
- **max_layers** (*maximum number of layers expected.*) –
- **tolerance** (*convergence criterion for clustering*) –
- **on the pooled standard deviation of z-coordinates.** (*based*) –
- **Suggested** (*0.5 for oxides, 0.2 for metals.*) –

Returns

- **layer_avg_z** (*list*) – List of average z-values of all layers.
- **layer_atoms** (*list of list*) – Each sublist contains the atom indices of the atoms in that layer.

catlearn.featurize.slab_utilities.**stoichiometry**(*atoms*)
Return a number of layers given a slab.

Parameters **atoms** (*object*) – ASE atoms object.

Returns **num_dict** – First entry is total number of atoms. Then key = element and entry = number

Return type dictionary

19.9 Module contents

CHAPTER 20

catlearn.fingerprint package

20.1 Submodules

20.2 catlearn.fingerprint.adsorbate module

Slab adsorbate fingerprint functions for machine learning.

class catlearn.fingerprint.adsorbate.**AdsorbateFingerprintGenerator**(***kwargs*)

Bases: *catlearn.featurize.base.BaseGenerator*

ads_av(*atoms=None*)

Function that takes an atoms objects and returns a fingerprint vector with averages of the atomic properties of the adsorbate.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

ads_sum(*atoms=None*)

Function that takes an atoms objects and returns a fingerprint vector with averages of the atomic properties of the adsorbate.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

bag_atoms_ads(*atoms=None*)

Function that takes an atoms object and returns a fingerprint vector containing the count of each element in the adsorbate.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

bag_cn(*atoms*)

Count the number of neighbors of the site, which has a n number of neighbors. This is equivalent to a bag of coordination numbers over the site neighbors. These can be used in the "alpha parameters" linear model.

Please cite: Roling LT, Abild-Pedersen F. Structure-Sensitive Scaling Relations: Adsorption Energies from Surface Site Stability. ChemCatChem. 2018 Apr 9;10(7):1643-50.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

bag_cn_general(*atoms*)

Count the number of neighbors of the site, which has a n number of neighbors. This is equivalent to a bag of coordination numbers over the site neighbors. These can be used in the "alpha parameters" linear model for alloys.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

bag_edges_ads(*atoms*)

Returns bag of connections, counting only the bonds within the adsorbate.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

bag_edges_all(*atoms*)

Returns bag of connections, counting all bonds within the adsorbate and between adsorbate atoms and surface. If we assign an energy to each type of bond, considering first neighbors only, this fingerprint would work independently in a linear model. The length of the vector is atom_types * ads_atom_types.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

bag_edges_chemi(*atoms*)

Returns bag of connections, counting only the bonds within the adsorbate and the connections between adsorbate and surface.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

bulk(*atoms=None*)

Return a fingerprint vector with propeties averaged over the bulk atoms.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

count_chemisorbed_fragment(*atoms=None*)
Function that takes an atoms objects and returns a fingerprint vector containing the count over atom types, that are neighbors to the chemisorbing atom.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

ctime(*atoms=None*)
Return the contents of atoms.info['ctime'] as a feature.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

db_size(*atoms=None*)
Return a fingerprint containing the number of layers in the slab, the number of surface atoms in the unit cell and the adsorbate coverage.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

dbid(*atoms=None*)
Return the contents of atoms.info['id'] as a feature.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

delta_energy(*atoms=None*)
Return the contents of atoms.info['key_value_pairs']['delta_energy'] as a feature.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

en_difference_active(*atoms=None*)
Returns a list of electronegativity metrics, squared and summed over adsorbate bonds including those with the surface.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

en_difference_ads(*atoms=None*)
Returns a list of electronegativity metrics, squared and summed over bonds within the adsorbate atoms.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

en_difference_chemi(*atoms=None*)
Returns a list of electronegativity metrics, squared and summed over adsorbate-site bonds.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

generalized_cn(*atoms*)
Returns the averaged generalized coordination number over the site. Calle-Vallejo et al. Angew. Chem. Int. Ed. 2014, 53, 8316-8319.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

max_site(*atoms=None*)
Function that takes an atoms objects and returns a fingerprint vector with properties averaged over the surface metal atoms closest to an add atom.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

mean_chemisorbed_atoms(*atoms=None*)
Function that takes an atoms objects and returns a fingerprint vector containing properties of the closest add atom to a surface metal atom.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

mean_site(*atoms=None*)
Function that takes an atoms objects and returns a fingerprint vector with properties averaged over the surface metal atoms closest to an add atom.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

mean_surf_ligands(*atoms=None*)
Function that takes an atoms objects and returns a fingerprint vector containing the count of nearest neighbors and properties of the nearest neighbors.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

median_site(*atoms=None*)
Function that takes an atoms objects and returns a fingerprint vector with properties averaged over the surface metal atoms closest to an add atom.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

min_site(*atoms=None*)
Function that takes an atoms objects and returns a fingerprint vector with properties averaged over the surface metal atoms closest to an add atom.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

strain(*atoms=None*)
Return a fingerprint with the espected strain of the site atoms and the termination atoms.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

sum_site(*atoms=None*)
Function that takes an atoms objects and returns a fingerprint vector with properties summed over the surface metal atoms closest to an add atom.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

term(*atoms=None*)
Return a fingerprint vector with propeties averaged over the termination atoms.

Parameters **atoms** (*object*) –

20.3 catlearn.fingerprint.bulk module

Slab adsorbate fingerprint functions for machine learning.

class catlearn.fingerprint.bulk.**BulkFingerprintGenerator**(***kwargs*)
Bases: *catlearn.featurize.base.BaseGenerator*

bulk_average(*atoms=None*)
Return a fingerprint vector with propeties of the element name saved in the atoms.info['key_value_pairs']['bulk']

bulk_std(*atoms=None*)
Return a fingerprint vector with propeties of the element name saved in the atoms.info['key_value_pairs']['bulk']

bulk_summation(*atoms=None*)
Return a fingerprint vector with propeties of the element name saved in the atoms.info['key_value_pairs']['bulk']

xyz_id(*atoms=None*)

20.4 catlearn.fingerprint.chalcogenide module

Slab adsorbate fingerprint functions for machine learning.

class catlearn.fingerprint.chalcogenide.**ChalcogenideFingerprintGenerator**(***kwargs*)
Bases: *catlearn.featurize.base.BaseGenerator*

formal_charges(*atoms*)
Return a fingerprint based on formal charges.

Parameters **atoms** (*object*) –

max_cation(*atoms=None*)
Function that takes an atoms objects and returns a fingerprint vector with properties averaged over the surface metal atoms closest to an add atom.

Parameters **atoms** (*object*) –

mean_cation(*atoms=None*)
Function that takes an atoms objects and returns a fingerprint vector with properties averaged over the surface metal atoms closest to an add atom.

Parameters **atoms** (*object*) –

median_cation(*atoms=None*)
Function that takes an atoms objects and returns a fingerprint vector with properties averaged over the surface metal atoms closest to an add atom.

Parameters **atoms** (*object*) –

min_cation(*atoms=None*)
Function that takes an atoms objects and returns a fingerprint vector with properties averaged over the surface metal atoms closest to an add atom.

Parameters **atoms** (*object*) –

sum_cation(*atoms=None*)
Function that takes an atoms objects and returns a fingerprint vector with properties summed over the surface metal atoms closest to an add atom.

Parameters **atoms** (*object*) –

20.5 catlearn.fingerprint.convoluted module

Slab adsorbate convoluted fingerprint functions for machine learning.

class catlearn.fingerprint.convoluted.**ConvolutedFingerprintGenerator**(***kwargs*)
Bases: *catlearn.featurize.base.BaseGenerator*

conv_bulk(*atoms=None*)
Return a fingerprint vector with propeties convoluted over the bulk atoms.

Parameters **atoms** (*object*) – A single atoms object.

conv_term(*atoms=None*)
Return a fingerprint vector with propeties convoluted over the terminal atoms.

Parameters **atoms** (*object*) – A single atoms object.

catlearn.fingerprint.convoluted.**check_length**(*labels*, *result*, *atoms*)
Check that two lists have the same length.

If not, print an informative error message containing a databse id if present.

Parameters

- **labels** (*list*) – A list of feature names.

- **result** (*list*) – A fingerprint.
- **atoms** (*object*) – A single atoms object.

20.6 catlearn.fingerprint.graph module

Functions to build a neighbor matrix feature representation.

class catlearn.fingerprint.graph.**GraphFingerprintGenerator**(***kwargs*)
Bases: *catlearn.featurize.base.BaseGenerator*

Function to build a fingerprint vector based on an atoms object.

neighbor_mean_vec(*data*)
Transform neighborlist into a neighbor averaged feature vector.

Parameters **data** (*object*) – Target data object from which to generate features.

Returns **features** – A 1d numpy array of the feature vector.

Return type array

neighbor_sum_vec(*data*)
Transform neighborlist into a neighbor sum feature vector.

Parameters **data** (*object*) – Target data object from which to generate features.

Returns **features** – A 1d numpy array of the feature vector.

Return type array

20.7 catlearn.fingerprint.molecule module

Functions to build a gas phase molecule fingerprint.

class catlearn.fingerprint.molecule.**AutoCorrelationFingerprintGenerator**(***kwargs*)
Bases: *catlearn.featurize.base.BaseGenerator*

Class for constructing an autocorrelation fingerprint.

get_autocorrelation(*atoms*)
Return the autocorrelation fingerprint for a molecule.

20.8 catlearn.fingerprint.particle module

Nanoparticle fingerprint functions.

These functions will typically perform well at describing chemical ordering within alloyed nanoparticles. However, they may be applicable to other applications where bond counting or coordination numbers are important descriptors.

This class inherits from the catlearn.fingerprint.BaseGenerator function.

class catlearn.fingerprint.particle.**ParticleFingerprintGenerator**(***kwargs*)
Bases: *catlearn.featurize.base.BaseGenerator*

Function to build a fingerprint vector based on an atoms object.

bond_count_vec(*data*)
Bond counting with a distribution measure for coordination.

Parameters **data** (*object*) – Data object with atomic distances.

Returns **track_nnmat** – List with summed number of atoms with given coordination numbers.

Return type list

connections_vec(*data*)
Sum atoms with a certain number of connections.

distribution_vec(*data*)
Return atomic distribution measure.

nearestneighbour_vec(*data*)
Nearest neighbour average, Topics in Catalysis, 2014, 57, 33.

This is a slightly modified version of the code found in the *ase.ga* module.

Parameters **data** (*object*) – Data object with atomic numbers available.

Returns **nnlist** – Feature vector that will be n**2 where n is the number of atomic species passed to the class.

Return type list

rdf_vec(*data*)
Return list of partial rdfs for use as fingerprint vector.

20.9 catlearn.fingerprint.prototype module

Prototype fingerprint based on Magpie.

class catlearn.fingerprint.prototype.**PrototypeFingerprintGenerator**(*atoms*, *sites*, *system_name=''*, *target='id'*, *delete_temp=True*, *properties=[]*)

Bases: object

Function to build prototype fingerprint in pandas.DataFrame.

Based on a list of ase.atoms object.

generate()
Generate Prototype fingerprint and return all the fingerprint.

Returns **FP**

Return type pandas.Frame

generate_all()

run_proto()
Call Magpie to generate Prototype FP and write to proto_FP.csv.

update_str()

write_proto_input()
Write Prototype input for Magpie.

class catlearn.fingerprint.prototype.**PrototypeSites**(*site_dict=None*)
Bases: object

Prototype site objective for generating prototype input.

20.10 catlearn.fingerprint.standard module

Standard fingerprint functions.

These feature sets should perform relatively well on a variety of different systems. They are general descriptors based predominantly on the elemental properties and in some cases structure.

This class inherits from the catlearn.fingerprint.BaseGenerator function.

class catlearn.fingerprint.standard.**StandardFingerprintGenerator**(***kwargs*)
Bases: *catlearn.featurize.base.BaseGenerator*

Function to build a fingerprint vector based on an atoms object.

bag_edges(*atoms*)
Returns the bag of connections, defined as counting connections between types of elements pairs. We define the bag as a vector, e.g. return [Number of C-H connections, # C-C, # C-O, . . . , # M-X]

Parameters **atoms** (*object*) –

Returns **features**

Return type list

bag_edges_cn(*atoms*)
Returns the bag of connections folded with coordination numbers of the node atoms.

Parameters **atoms** (*object*) –

Returns **features**

Return type list

bag_element_cn(*atoms*)
Bag elements folded with coordination numbers, e.g. number of C with CN = 4, number of C with CN = 3, ect.

Parameters **atoms** (*object*) – ASE Atoms object.

Returns **features** – If None was passed, the elements are strings, naming the feature.

Return type list

bag_elements(*atoms*)
Returns the bag of elements, defined as counting occurence of elements in a given structure. This is mostly useful for subtracting atomization energies.

Parameters **atoms** (*object*) –

Returns **features**

Return type list

composition_vec(*data*)
Function to return a feature vector based on the composition.

Parameters **data** (*object*) – Data object with atomic numbers available.

Returns **features** – Vector containing a count of the different atomic types, e.g. for CH3OH the vector [1, 4, 1] would be returned.

Return type array

distance_vec(*data*)
Averaged distance between e.g. A-A atomic pairs.

Parameters **data** (*object*) – Data object with Cartesian coordinates and atomic numbers available.

Returns **features** – Vector of averaged distances between homoatomic atoms.

Return type ndarray

eigenspectrum_vec(*data*)
Sorted eigenspectrum of the Coulomb matrix.

Parameters **data** (*object*) – Data object with Cartesian coordinates and atomic numbers available.

Returns **features** – Sorted Eigen values of the coulomb matrix, n atoms is size.

Return type ndarray

element_mass_vec(*data*)
Function to return a vector based on mass parameter.

Parameters **data** (*object*) – Data object with atomic masses available.

Returns **features** – Vector of the summed mass.

Return type ndarray

element_parameter_vec(*data*)
Function to return a vector based on a defined paramter.

The vector is compiled based on the summed parameters for each elemental type as well as the sum for all atoms.

Parameters **data** (*object*) – Data object with atomic numbers available.

Returns **features** – An n + 1 array where n in the length of self.atom_types.

Return type array

20.11 catlearn.fingerprint.voro module

Voronoi fingerprint based on Magpie.

class catlearn.fingerprint.voro.**VoronoiFingerprintGenerator**(*atoms*, *delete_temp=True*)

Bases: object

Function to build voronoi fingerprint in pandas.DataFrame.

Based on a list of ase.atoms object.

generate()
Generate Voronoi fingerprint and return all the fingerprint.

Returns **FP**

Return type pandas.Frame

run_voro()
Call Magpie to generate Voronoi FP and write to voro_FP.csv.

write_voro_input()
Write Voronoi input for Magpie.

20.12 Module contents

CHAPTER 21

catlearn.ga

21.1 catlearn.ga.algorithm

The GeneticAlgorithm class methods.

class catlearn.ga.algorithm.**GeneticAlgorithm**(*fit_func*, *features*, *targets*, *population_size='auto'*, *population=None*, *operators=None*, *fitness_parameters=1*, *nsplit=2*, *accuracy=None*, *nprocs=1*, *dmax=None*)

Bases: object

Genetic algorithm for parameter optimization.

search(*steps*, *natural_selection=True*, *convergence_operator=None*, *repeat=5*, *verbose=False*, *writefile=None*)

Do the actual search.

Parameters

- **steps** (*int*) – Maximum number of steps to be taken.
- **natural_selection** (*bool*) – A flag that when set True will perform natural selection.
- **convergence_operator** (*object*) – The function to perform the convergence check. If None is passed then the *no_progress* function is used.
- **repeat** (*int*) – Number of repeat generations with no progress.
- **verbose** (*bool*) – If True, will print out the progress of the search. Default is False.
- **writefile** (*str*) – Name of a json file to save data too.

population

The current population.

Type list

fitness
The fitness for the current population.
Type list

21.2 catlearn.ga.convergence

Functions to check for convergence in the GA.

class catlearn.ga.convergence.**Convergence**
Bases: object

Class to check convergence.

no_progress(*fitness*, *repeat*)
Convergence based on a lack of any progress in the search.

Parameters

- **fitness** (*array*) – A List of fitnesses from the search.
- **repeat** (*int*) – Number of repeat generations with no progress.

Returns **converged** – True if convergence has been reached, False otherwise.

Return type bool

stagnation(*fitness*, *repeat*)
Convergence based on a stagnation of the population.

Parameters

- **fitness** (*array*) – A List of fitnesses from the search.
- **repeat** (*int*) – Number of repeat generations with no progress.

Returns **converged** – True if convergence has been reached, False otherwise.

Return type bool

21.3 catlearn.ga.initialize

Function to initialize a population.

catlearn.ga.initialize.**initialize_population**(*pop_size*, *dimension*, *dmax=None*)
Generate a random starting population.

Parameters

- **pop_size** (*int*) – Population size.
- **d_param** (*int*) – Dimension of parameters in model.

21.4 catlearn.ga.io

Functions to read and write GA data.

catlearn.ga.io.**read_data**(*writefile*)
Funtion to read population and fitness.

Parameters **writefile** (*str*) – Name of the JSON file to read.

Returns

- **population** (*array*) – The population saved from a previous search.
- **fitness** (*array*) – The fitness associated with the saved population.

21.5 catlearn.ga.mating

Cut and splice mating function.

catlearn.ga.mating.**cut_and_splice**(*parent_one*, *parent_two*, *index='random'*)
Perform cut_and_splice between two parents.

Parameters

- **parent_one** (*list*) – List of params for first parent.
- **parent_two** (*list*) – List of params for second parent.
- **index** (*str*) – Define how to choose size of each cut index.

Returns **offspring** – A new child candidate from the two parents.

Return type array

21.6 catlearn.ga.mutate

Define some mutation functions.

catlearn.ga.mutate.**probability_include**(*parent_one*)
A mutation that will include features with a certain probability.

Parameters **parent_one** (*list*) – List of params for first parent.

Returns **p1** – Mutated parameter list based on the parent parameters provided.

Return type list

catlearn.ga.mutate.**probability_remove**(*parent_one*)
A mutation that will remove features with a certain probability.

Parameters **parent_one** (*list*) – List of params for first parent.

Returns **p1** – Mutated parameter list based on the parent parameters provided.

Return type list

catlearn.ga.mutate.**random_permutation**(*parent_one*)
Perform a random permutation on a parameter index.

Parameters **parent_one** (*list*) – List of params for first parent.

Returns **p1** – Mutated parameter list based on the parent parameters provided.

Return type list

21.7 catlearn.ga.natural_selection

Functions to perform some natural selection.

catlearn.ga.natural_selection.**population_reduction**(*pop*, *fit*, *population_size*)
Method to reduce population size to constant.

Parameters

- **pop** (*list*) – Extended population.
- **fit** (*list*) – Extended fitness assignment.
- **population_size** (*int*) – The population size.
- **pareto** (*bool*) – Flag to specify whether search is for Pareto optimal set.

Returns

- **population** (*list*) – The population after natural selection.
- **fitness** (*list*) – The fitness for the current population.

catlearn.ga.natural_selection.**remove_duplicates**(*population*, *fitness*, *accuracy*)
Function to delete duplicate candidates based on fitness.

Parameters

- **population** (*array*) – The current population.
- **fitness** (*array*) – The fitness for the current population.
- **accuracy** (*int*) – Number of decimal places to include when finding unique.

Returns

- **population** (*list*) – The population after duplicates deleted.
- **fitness** (*list*) – The fitness for the population after duplicates deleted.

21.8 catlearn.ga.predictors

Some generic prediction functions.

catlearn.ga.predictors.**minimize_error**(*train_features*, *train_targets*, *test_features*, *test_targets*)
A generic fitness function.

This fitness function will minimize the cost function.

Parameters

- **train_features** (*array*) – The training features.
- **train_targets** (*array*) – The training targets.
- **test_features** (*array*) – The test feaatures.
- **test_targets** (*array*) – The test targets.

catlearn.ga.predictors.**minimize_error_descriptors**(*train_features*, *train_targets*, *test_features*, *test_targets*)
A generic fitness function.

This fitness function will minimize the cost function as well as the number of descriptors. This will provide a Pareto optimial set of solutions upon convergence.

Parameters

- **train_features** (*array*) – The training features.
- **train_targets** (*array*) – The training targets.
- **test_features** (*array*) – The test feaatures.
- **test_targets** (*array*) – The test targets.

catlearn.ga.predictors.**minimize_error_time**(*train_features*, *train_targets*, *test_features*, *test_targets*)

A generic fitness function.

This fitness function will minimize the cost function as well as the time to train the model. This will provide a Pareto optimial set of solutions upon convergence.

Parameters

- **train_features** (*array*) – The training features.
- **train_targets** (*array*) – The training targets.
- **test_features** (*array*) – The test feaatures.
- **test_targets** (*array*) – The test targets.

CHAPTER 22

catlearn.learning_curve

22.1 catlearn.learning_curve.data_process

Processing of data for HierarchyValidation.

class catlearn.learning_curve.data_process.**data_process**(*features*, *min_split*, *max_split*, *scale=True*, *normalization=True*, *ridge=True*, *loocv=True*, *batchfarm=False*)

Bases: object

Class to glue different function used for HierarchyValidation.

This class pick up data from HierarchyValidation. The data is then modified if requested with "feature_preprocess", and "predict". The data is then fitted with regression model for example with "ridge_regression". The error of the fit is then measured.

average_nested(*Y*, *X*)
Calculate statistics for predicition.

Parameters

- **data_size** (*list*) – Data_size for where the prediction were made.
- **p_error** (*list*) – Error for where the prediction were made.

get_statistic(*data_size*, *p_error*)
Generate statistics for predicition.

Parameters

- **data_size** (*list*) – Data_size for where the prediction were made.
- **p_error** (*list*) – Error for where the prediction were made.

globalscaling(*globalscaledata*, *train_features*)
All sub-groups of traindata are scaled same.

Parameters **globalscaledata** (*string*) – The data will be scaled globally if requested.

prediction_error(*test_features*, *test_targets*, *coef*, *s_tar*, *m_tar*)
Calculate the error of the prediction with the model.

Parameters

- **test_features** (*array*) – Independet data for testing the model.
- **test_targets** (*array*) – Dependent data to test the model.
- **coef** (*array*) – The coeffieiceints which makes up the model.
- **s_tar** (*string*) – Standard devation or (max-min), for the dependent train_targets.
- **m_tar** (*array*) – Mean for the dependent train_targets.

scaling_data(*train_features*, *train_targets*, *test_features*, *s_tar*, *m_tar*, *s_feat*, *m_feat*)
Scaling the data if requested.

Parameters

- **train_feature** (*array*) – Independent data used to train model.
- **train_targets** (*array*) – Dependent data used to train model.
- **test_features** (*array*) – Independent data used to test the model.
- **s_tar** (*array*) – Standard devation or (max-min), for the dependent train_targets.
- **m_tar** (*array*) – Mean for the dependent train_targets.
- **s_feat** (*array*) – Standard devation or (max-min), for the independent train_features.
- **m_feat** (*array*) – Mean for the independent train_features.

22.2 catlearn.learning_curve.feature_selection

Feature selection with lasso.

class catlearn.learning_curve.feature_selection.**feature_selection**(*train_features*, *train_targets*)

Bases: object

Class made to make it possible to select features.

Used with hierarchy cross-validation.

alpha_finder(*feat_vec*, *alpha_vec*, *feat*)
Find the alpha corresponding to the number of features.

Parameters

- **feat_vec** (*list*) – Features within the interval.
- **alpha_vec** (*list*) – Alphas within the interval.
- **feat** (*int*) – The group of feature searched.

alpha_refinment(*alpha*, *feat*, *splits=10*, *refsteps=1*, *upper=1.5*)
Find a more stringent alpha for the number of feature searched for.

Parameters

- **alpha** (*int*) – Initial alpha found for the nuumber of feature searched for. Will be used as a lower limit.

- **feat** (*int*) – The number of feature searched for.
- **splits** (*int*) – Increase of Number of alphas under inspection within interval.
- **refsteps** (*int*) – Number of refinements.
- **upper** – How many times alpha the upper limit should be.

feature_inspection(*lower=0*, *upper=1*, *interval=100*, *alpha_list=None*)
Generate interval used to search for the alpha.

Parameters

- **lower** (*int*) – Lower bound for the interval search.
- **upper** (*int*) – Upper bound for the interval search.
- **interval** (*int*) – Number of alphas in interval inspected.

interval_modifier(*feat_vec*, *alpha_vec*, *feat*, *splits*, *int_expand*)
Modifiy the interval under inspection by reduction or expantion.

Parameters

- **feat_vec** (*list*) – Features within the interval.
- **alpha_vec** (*list*) – Alphas within the interval.
- **feat** (*int*) – The group of feature searched.
- **splits** (*int*) – Increase of Number of alphas under inspection within interval.
- **int_expand** (*int*) – Number of times the number of alphas in interval is increased.

selection(*select_limit*)
Select the the feture/s that works best wtig L1.

22.3 catlearn.learning_curve.learning_curve

Generate the learning curve.

class catlearn.learning_curve.learning_curve.**LearningCurve**(*nprocs=1*)
Bases: object

Learning curve class. Test a model while varying the density of the training data.

run(*model*, *train*, *target*, *test*, *test_target*, *step=1*, *min_data=2*)
Evaluate a model versus training data size.

Parameters

- **model** (*object*) – A function that will train or load a regression model or classifier and make predictions for testing. model should accept the parameters:

  train_features : array test_features : array train_targets : list test_targets : list

  model should return either a float or a list of floats. The float or the first value of the list will be used as the fitness score.
- **train** (*array*) – An n, d array of training examples.
- **targets** (*test*) – A list of the target values.
- **test** (*array*) – An n, d array of test data.
- **targets** – A list of the test target values.

- **step** (*int*) – Incrementent the data set size by this many examples.
- **min_data** (*int*) – Smallest number of training examples to test.

Returns **output** – Each row is the output from the model object.

Return type array

catlearn.learning_curve.learning_curve.**feature_frequency**(*cv*, *features*, *min_split*, *max_split*, *smallest=False*, *new_data=True*, *ridge=True*, *scale=True*, *globalscale=True*, *normalization=True*, *featselect_featvar=False*, *featselect_featconst=True*, *select_limit=None*, *feat_sub=15*)

Function to extract raw data from the database.

Parameters

- **features** (*int*) – Number of features used for regression.
- **min_split** (*int*) – Number of datasplit in the smallest sub-set.
- **max_split** (*int*) – Number of datasplit in the largest sub-set.
- **new_data** (*string*) – Use new data or the previous data.
- **ridge** (*string*) – Ridge regulazer is deafult. If False, lasso is used.
- **scale** (*string*) – If the data are supposed to be scaled or not.
- **globalscale** (*string*) – Using global scaleing or not.
- **normalization** (*string*) – If scaled, normalized or standardized. Normalized is default.
- **feature_selection** (*string*) – Using feature selection with ridge, or plain vanilla ridge.
- **select_limit** (*int*) – Up to have many number of features used for feature selection.

catlearn.learning_curve.learning_curve.**hierarchy**(*cv*, *features*, *min_split*, *max_split*, *new_data=True*, *ridge=True*, *scale=True*, *globalscale=True*, *normalization=True*, *featselect_featvar=False*, *featselect_featconst=True*, *select_limit=None*, *feat_sub=15*)

Start the hierarchy.

Parameters

- **features** (*int*) – Number of features used for regression.
- **min_split** (*int*) – Number of datasplit in the smallest sub-set.
- **max_split** (*int*) – Number of datasplit in the largest sub-set.
- **new_data** (*string*) – Use new data or the previous data.
- **ridge** (*string*) – Ridge regulazer is deafult. If False, lasso is used.

- **scale** (*string*) – If the data are supposed to be scaled or not.
- **globalscale** (*string*) – Using global scaleing or not.
- **normalization** (*string*) – If scaled, normalized or standardized. Normalized is default.
- **feature_selection** (*string*) – Using feature selection with ridge, or plain vanilla ridge.
- **select_limit** (*int*) – Up to have many number of features used for feature selection.

22.4 catlearn.learning_curve.placeholder

Placeholder for now.

class catlearn.learning_curve.placeholder.**placeholder**(*PC*, *index_split*, *hv*, *indicies*, *hier_level*, *featselect_featvar*, *featselect_featconst*, *s_feat*, *m_feat*, *feat_sub=15*, *s_tar=None*, *m_tar=None*, *select_limit=None*, *selected_features=None*, *glob_feat1=None*, *glob_tar1=None*, *new_training=True*)

Bases: object

Used to make the hierarchey more easy to follow.

Placeholder for now.

get_data_scale(*split*, *set_size=None*, *p_error=None*, *result=None*)
Get the data for each sub-set of data and scales it accordingly.

Parameters

- **split** (*int*) – Which sub-set od data within hierarchy level.
- **result** (*list*) – Contain all the coefficien and omega2 for all training data.
- **set_size** (*list*) – Size of sub-set of data/features which the model is based on.
- **p_error** (*list*) – The prediction error for plain vanilla ridge.

getstats()
Used to get features for the frequencyplots.

predict_subsets(*result=None*, *set_size=None*, *p_error=None*)
Run the prediction on each sub-set of data on the hierarchy level.

Parameters

- **result** (*list*) – Contain all the coefficien and omega2 for all training data.
- **set_size** (*list*) – Size of sub-set of data/features which the model is based on.
- **p_error** (*list*) – The prediction error for plain vanilla ridge.

reg_data_var(*train_features*, *train_targets*, *test_features*, *test_targets*, *ridge*, *set_size*, *p_error*, *result*)
Ridge regression and calculation of prediction error.

Parameters

- **train_features** (*array*) – Independent data used to train the model.
- **train_targets** (*array*) – Dependent data used to train model.
- **test_features** (*array*) – Independent data used to test model.
- **test_target** (*array*) – Dependent data used to test model.
- **ridge** (*object*) – Generates the model based on the training data.
- **set_size** (*list*) – Size of sub-set of data/features which the model is based on.
- **p_error** (*list*) – The prediction error for plain vanilla ridge.
- **result** (*list*) – Contain all the coefficien and omega2 for all training data.

reg_feat_var(*train_features*, *train_targets*, *test_features*, *test_targets*, *ridge*, *set_size*, *p_error*, *result*)

Regression within a dataset with varying feature.

Parameters

- **train_features** (*array*) – Independent data used to train the model.
- **train_targets** (*array*) – Dependent data used to train model.
- **test_features** (*array*) – Independent data used to test model.
- **test_target** (*array*) – Dependent data used to test model.
- **ridge** (*object*) – Generates the model based on the training data.
- **p_error** (*list*) – The prediction error for feature selection corresponding to different feature set.
- **set_size** (*list*) – Different data/feature set used for feature selection.
- **result** (*list*) – Contain all the coefficien and omega2 for all training data.

22.5 catlearn.learning_curve.pltfile

CHAPTER 23

catlearn.preprocess

23.1 catlearn.preprocess.clean_data

Functions to clean data.

catlearn.preprocess.clean_data.**clean_infinite**(*train*, *test=None*, *targets=None*, *labels=None*, *mask=None*, *max_impute_fraction=0*, *strategy='mean'*)

Remove features that have non finite values in the training data.

Optionally removes features in test data with non fininte values. Returns a dictionary with the clean 'train', 'test' and 'index' that were removed from the original data.

Parameters

- **train** (*array*) – Feature matrix for the traing data.
- **test** (*array*) – Optional feature matrix for the test data. Default is None passed.
- **targets** (*array*) – An array of training targets.
- **labels** (*array*) – Optional list of feature labels. Default is None passed.
- **mask** (*list*) – Indices of features that are not subject to cleaning.
- **max_impute_fraction** (*float*) – Maximum fraction of values in a column that can be imputed. Columns with higher fractions of nans values will be discarded.
- **strategy** (*str*) – Imputation strategy.

Returns

data –

key value pairs

- **'train'** [array] Clean training data matrix.
- **'test'** [array] Clean test data matrix

- **'targets'** [list] Boolean list on whether targets are finite.
- **'labels'** [list] Feature labels of clean data set.

Return type dict

catlearn.preprocess.clean_data.**clean_skewness**(*train*, *test=None*, *labels=None*, *mask=None*, *skewness=3.0*)

Discards features that are excessively skewed.

Parameters

- **train** (*array*) – Feature matrix for the traing data.
- **test** (*array*) – Optional feature matrix for the test data. Default is None passed.
- **labels** (*array*) – Optional list of feature labels. Default is None passed.
- **mask** (*list*) – Indices of features that are not subject to cleaning.
- **skewness** (*float*) – Maximum allowed skewness thresshold.

catlearn.preprocess.clean_data.**clean_variance**(*train*, *test=None*, *labels=None*, *mask=None*)

Remove features that contribute nothing to the model.

Removes a feature if there is zero variance in the training data. If this is the case, then the model won't learn anything new from adding this feature as it will just act as a scalar.

Parameters

- **train** (*array*) – Feature matrix for the traing data.
- **test** (*array*) – Optional feature matrix for the test data. Default is None passed.
- **labels** (*array*) – Optional list of feature labels. Default is None passed.
- **mask** (*list*) – Indices of features that are not subject to cleaning.

catlearn.preprocess.clean_data.**remove_outliers**(*features*, *targets*, *con=1.4826*, *dev=3.0*, *constraint=None*)

Preprocessing routine to remove outliers by median absolute deviation.

This will take the training feature and target arrays, calculate any outliers, then return the reduced arrays. It is possible to set a constraint key ('high', 'low', None) in order to allow for outliers that are e.g. very low in energy, as this may be the desired outcome of the study.

Parameters

- **features** (*array*) – Feature matrix for training data.
- **targets** (*list*) – List of target values for the training data.
- **con** (*float*) – Constant scale factor dependent on the distribution. Default is 1.4826 expecting the data is normally distributed.
- **dev** (*float*) – The number of deviations from the median to account for.
- **constraint** (*str*) – Can be set to 'low' to remove candidates with targets that are too small/negative or 'high' for outliers that are too large/positive. Default is to remove all.

23.2 catlearn.preprocess.feature_elimination

Functions to select features for the fingerprint vectors.

class catlearn.preprocess.feature_elimination.**FeatureScreening**(*correlation='pearson'*, *iterative=True*, *regression='ridge'*, *random_check=False*)

Bases: object

Class for feature elimination based on correlation screening.

eliminate_features(*target*, *train_features*, *test_features*, *size=None*, *step=None*, *order=None*)

Function to eliminate features from training/test data.

Parameters

- **target** (*list*) – The target values for the training data.
- **train_features** (*array*) – Array of training data to eliminate features from.
- **test_features** (*array*) – Array of test data to eliminate features from.
- **size** (*int*) – Number of features after elimination.
- **step** (*int*) – Number of features to eliminate at each step.
- **order** (*list*) – Precomputed ordered indices for features.

Returns

- **reduced_train** (*array*) – Reduced training feature matrix, now n x size shape.
- **reduced_test** (*array*) – Reduced test feature matrix, now m x size shape.

iterative_screen(*target*, *feature_matrix*, *size=None*, *step=None*)

Function iteratively screen featues.

Parameters

- **target** (*list*) – The target values for the training data.
- **feature_matrix** (*array*) – The feature matrix for the training data.
- **size** (*int*) – Number of features to be returned. Default is number of data.
- **step** (*int*) – Step size by which to reduce the number of features. Default is n / log(n).

Returns

- **index** (*list*) – The ordered list of feature indices, top index[:size] will be indices for best features.
- **size** (*int*) – Number of accepted features.

screen(*target*, *feature_matrix*)

Feature selection based on SIS.

Further discussion on this topic can be found in Fan, J., Lv, J., J. R. Stat. Soc.: Series B, 2008, 70, 849.

Parameters

- **target** (*list*) – The target values for the training data.
- **feature_matrix** (*array*) – The feature matrix for the training data.

Returns

- **index** (*list*) – The ordered list of feature indices.
- **correlation** (*list*) – The ordered list of correlations between features and targets.

- **size** (*int*) – Number of accepted features following screening.

23.3 catlearn.preprocess.feature_engineering

Functions for feature engineering.

catlearn.preprocess.feature_engineering.**generate_features**(*p*, *max_num=2*, *max_den=1*, *log=False*, *sqrt=False*, *exclude=False*, *s=False*)

Generate composite features from a combination of input features.

developer note: This is currently scales *quite slowly* with max_den. There's surely a better way to do this, but it's apparently currently functional.

Parameters

- **p** (*list*) – User-provided list of physical features to be combined.
- **max_num** (*integer*) – The maximum order of the polynomial in the numerator of the composite features. Must be non-negative.
- **max_den** (*integer*) – The maximum order of the polynomial in the denominator of the composite features. Must be non-negative.
- **log** (*boolean (not currently supported)*) – Set to True to include terms involving the logarithm of the input features. Default is False.
- **sqrt** (*boolean (not currently supported)*) – Set to True to include terms involving the square root of the input features. Default is False.
- **exclude** (*bool*) – Set exclude=True to avoid returning 1 to represent the zeroth power. Default is False.
- **s** (*bool*) – Set True to return a list of strings and False to evaluate each element in the list. Default is False.

Returns **features** – A list of combinations of the input features to meet the required specifications.

Return type list

catlearn.preprocess.feature_engineering.**generate_positive_features**(*p*, *N*, *exclude=False*, *s=False*)

Generate list of polynomial combinations in list p up to order N.

Example: p = (a,b,c) ; N = 3

returns (order not preserved) [a*a*a, a*a*b, a*a*c, a*b*b, a*b*c, a*c*c, b*b*b, b*b*c, b*c*c, c*c*c, a*a, a*b, a*c, b*b, b*c, c*c, a, b, c]

Parameters

- **p** (*list*) – Features to be combined.
- **N** (*integer*) – The maximum polynomial coefficient for combinations. Must be non-negative.
- **exclude** (*bool*) – Set True to avoid returning 1 to represent the zeroth power. Default is False.
- **s** (*bool*) – Set True to return a list of strings and False to evaluate each element in the list. Default is False.

Returns **all_powers** – A list of combinations of the input features to meet the required specifications.

Return type list

catlearn.preprocess.feature_engineering.**get_ablog**(*A*, *a*, *b*)
Get all combinations x_ij = a*log(x_i) + b*log(x_j).

The sorting order in dimension 0 is preserved.

Parameters

- **A** (*array*) – An n x m matrix, where n is the number of training examples and m is the number of features.
- **a** (*float*) –
- **b** (*float*) –

Returns **new_features** – The n x triangular(m) matrix of new features.

Return type array

catlearn.preprocess.feature_engineering.**get_div_order_2**(*A*)
Get all combinations x_ij = x_i / x_j, where x_i,j are features.

The sorting order in dimension 0 is preserved. If a denominator is 0, Inf is returned.

Parameters **A** (*array*) – n x m matrix, where n is the number of training examples and m is the number of features.

Returns **new_features** – The n x m**2 matrix of new features.

Return type array

catlearn.preprocess.feature_engineering.**get_labels_ablog**(*l*, *a*, *b*)
Get all combinations ij, where i,j are feature labels.

Parameters

- **a** (*float*) –
- **b** (*float*) –

Returns **new_features** – List of new feature names.

Return type list

catlearn.preprocess.feature_engineering.**get_labels_order_2**(*l*, *div=False*)
Get all combinations ij, where i,j are feature labels.

Parameters **x** (*list*) – Length m vector, where m is the number of features.

Returns **new_features** – List of new feature names.

Return type list

catlearn.preprocess.feature_engineering.**get_labels_order_2ab**(*l*, *a*, *b*)
Get all combinations ij, where i,j are feature labels.

Parameters **x** (*list*) – Length m vector, where m is the number of features.

Returns **new_features** – List of new feature names.

Return type list

catlearn.preprocess.feature_engineering.**get_order_2**(*A*)
Get all combinations x_ij = x_i * x_j, where x_i,j are features.

The sorting order in dimension 0 is preserved.

Parameters **A** (*array*) – n x m matrix, where n is the number of training examples and m is the number of features.

Returns **new_features** – The n x triangular(m) matrix of new features.

Return type array

catlearn.preprocess.feature_engineering.**get_order_2ab**(*A*, *a*, *b*)
Get all combinations x_ij = x_i**a * x_j**b, where x_i,j are features.

The sorting order in dimension 0 is preserved.

Parameters

- **A** (*array*) – n x m matrix, where n is the number of training examples and m is the number of features.
- **a** (*float*) –
- **b** (*float*) –

Returns **new_features** – The n x triangular(m) matrix of new features.

Return type array

catlearn.preprocess.feature_engineering.**single_transform**(*A*)
Perform single variable transform x^2, x^0.5 and log(x).

Parameters **A** (*array*) – n x m matrix, where n is the number of training examples and m is the number of features.

Returns **new_features** – The n x m*3 matrix of new features.

Return type array

23.4 catlearn.preprocess.feature_extraction

Some feature extraction routines.

catlearn.preprocess.feature_extraction.**catlearn_pca**(*components*, *train_features*, *test_features=None*, *cleanup=False*, *scale=False*)
Principal component analysis varient that doesn't require scikit-learn.

Parameters

- **components** (*int*) – Number of principal components to transform the feature set by.
- **test_fpv** (*array*) – The feature matrix for the testing data.

catlearn.preprocess.feature_extraction.**pca**(*components*, *train_matrix*, *test_matrix*)
Principal component analysis routine.

Parameters

- **components** (*int*) – The number of components to be returned.
- **train_matrix** (*array*) – The training features.
- **test_matrix** (*array*) – The test features.

Returns

- **new_train** (*array*) – Extracted training features.
- **new_test** (*array*) – Extracted test features.

catlearn.preprocess.feature_extraction.**pls**(*components*, *train_matrix*, *target*, *test_matrix*)
Projection of latent structure routine.

Parameters

- **components** (*int*) – The number of components to be returned.
- **train_matrix** (*array*) – The training features.
- **test_matrix** (*array*) – The test features.

Returns

- **new_train** (*array*) – Extracted training features.
- **new_test** (*array*) – Extracted test features.

catlearn.preprocess.feature_extraction.**spca**(*components*, *train_matrix*, *test_matrix*)
Sparse principal component analysis routine.

Parameters

- **components** (*int*) – The number of components to be returned.
- **train_matrix** (*array*) – The training features.
- **test_matrix** (*array*) – The test features.

Returns

- **new_train** (*array*) – Extracted training features.
- **new_test** (*array*) – Extracted test features.

23.5 catlearn.preprocess.greedy_elimination

Greedy feature selection routines.

class catlearn.preprocess.greedy_elimination.**GreedyElimination**(*nprocs=1*, *verbose=True*, *save_file=None*)
Bases: object

The greedy feature elimination class.

greedy_elimination(*predict*, *features*, *targets*, *nsplit=2*, *step=1*)
Greedy feature elimination.

Function to iterate through feature set, eliminating worst feature in each pass. This is the backwards greedy algorithm.

Parameters

- **predict** (*object*) – A function that will make the predictions. predict should accept the parameters:

train_features : array test_features : array train_targets : list test_targets : list

predict should return either a float or a list of floats. The float or the first value of the list will be used as the fitness score.

- **features** (*array*) – An n, d array of features.
- **targets** (*list*) – A list of the target values.
- **nsplit** (*int*) – Number of folds in k-fold cross-validation.

Returns

output – First column is the index of features in the order they were eliminated.

Second column are corresponding cost function values, averaged over the k fold split.

Following columns are any additional values returned by predict, averaged over the k fold split.

Return type array

23.6 catlearn.preprocess.importance_testing

Functions to check feature significance.

class catlearn.preprocess.importance_testing.**ImportanceElimination**(*transform*, *nprocs=1*, *verbose=True*)

Bases: object

The feature importance elimination class.

importance_elimination(*train_predict*, *test_predict*, *features*, *targets*, *nsplit=2*, *step=1*)

Importance feature elimination.

Function to iterate through feature set, eliminating least important feature in each pass. This is the backwards elimination algorithm.

Parameters

- **train_predict** (*object*) – A function that will train a model. The function should accept the parameters:

  train_features : array train_targets : list

  predict should return a function that can be passed to test_predict.
- **test_predict** (*object*) – A function that will accept a trained model object and return a float or a list of test metrics. The first returned metric will be used to eliminate features.
- **features** (*array*) – An n, d array of features.
- **targets** (*list*) – A list of the target values.
- **nsplit** (*int*) – Number of folds in k-fold cross-validation.
- **step** (*int*) – Optional number of features to eliminate in each round.

Returns

output – First column is the index of features in the order they were eliminated.

Second column are corresponding cost function values, averaged over the k fold split.

Following columns are any additional values returned by test_predict, averaged over the k fold split.

Return type array

catlearn.preprocess.importance_testing.**feature_invariance**(*args*)
Make a feature invariant.

Parameters **args** (*list*) – A list of arguments:

index [int] The index of the feature to be shuffled.

train_features [array] The original training data matrix.

test_features [array] The original test data matrix.

Returns

- **train** (*array*) – Feature matrix with a shuffled feature column in matrix.
- **test** (*array*) – Feature matrix with a shuffled feature column in matrix.

catlearn.preprocess.importance_testing.**feature_randomize**(*args*)
Make a feature random noise.

Parameters **args** (*list*) – A list of arguments:

index [int] The index of the feature to be shuffled.

train_features [array] The original training data matrix.

test_features [array] The original test data matrix.

Returns

- **train** (*array*) – Feature matrix with a shuffled feature column in matrix.
- **test** (*array*) – Feature matrix with a shuffled feature column in matrix.

catlearn.preprocess.importance_testing.**feature_shuffle**(*args*)
Shuffle a feature.

The method has a number of advantages for measuring feature importance. Notably the original values and scale of the feature are maintained.

Parameters **args** (*list*) – A list of arguments:

index [int] The index of the feature to be shuffled.

train_features [array] The original training data matrix.

test_features [array] The original test data matrix.

Returns

- **train** (*array*) – Feature matrix with a shuffled feature column in matrix.
- **test** (*array*) – Feature matrix with a shuffled feature column in matrix.

23.7 catlearn.preprocess.scaling

Functions to process the raw feature matrix.

catlearn.preprocess.scaling.**min_max**(*train_matrix*, *test_matrix=None*, *local=True*)
Normalize each feature relative to the min and max.

Parameters

- **train_matrix** (*list*) – Feature matrix for the training dataset.
- **test_matrix** (*list*) – Feature matrix for the test dataset.
- **local** (*boolean*) – Define whether to scale locally or globally.

catlearn.preprocess.scaling.**normalize**(*train_matrix*, *test_matrix=None*, *mean=None*, *dif=None*, *local=True*)

Normalize each feature relative to mean and min/max variance.

Parameters

- **train_matrix** (*list*) – Feature matrix for the training dataset.
- **test_matrix** (*list*) – Feature matrix for the test dataset.
- **local** (*boolean*) – Define whether to scale locally or globally.
- **mean** (*list*) – List of mean values for each feature.
- **dif** (*list*) – List of max-min values for each feature.

catlearn.preprocess.scaling.**standardize**(*train_matrix*, *test_matrix=None*, *mean=None*, *std=None*, *local=True*)

Standardize each feature relative to the mean and standard deviation.

Parameters

- **train_matrix** (*array*) – Feature matrix for the training dataset.
- **test_matrix** (*array*) – Feature matrix for the test dataset.
- **mean** (*list*) – List of mean values for each feature.
- **std** (*list*) – List of standard deviation values for each feature.
- **local** (*boolean*) – Define whether to scale locally or globally.

catlearn.preprocess.scaling.**target_center**(*target*)

Return a list of normalized target values.

Parameters **target** (*list*) – A list of the target values.

catlearn.preprocess.scaling.**target_normalize**(*target*)

Return a list of normalized target values.

Parameters **target** (*list*) – A list of the target values.

catlearn.preprocess.scaling.**target_standardize**(*target*)

Return a list of standardized target values.

Parameters **target** (*list*) – A list of the target values.

catlearn.preprocess.scaling.**unit_length**(*train_matrix*, *test_matrix=None*, *local=True*)

Normalize each feature vector relative to the Euclidean length.

Parameters

- **train_matrix** (*list*) – Feature matrix for the training dataset.
- **test_matrix** (*list*) – Feature matrix for the test dataset.
- **local** (*boolean*) – Define whether to scale locally or globally.

CHAPTER 24

catlearn.regression

24.1 catlearn.regression.gpfunctions

24.1.1 catlearn.regression.gpfunctions.covariance

Generation of covariance matrix.

catlearn.regression.gpfunctions.covariance.**get_covariance**(*kernel_list*, *log_scale*, *matrix1*, *matrix2=None*, *regularization=None*, *eval_gradients=False*)

Return the covariance matrix of training dataset.

Parameters

- **kernel_list** (*dict of dicts*) – A dict containing all dictionaries for the kernels.
- **log_scale** – Flag to define if the hyperparameters are log scale.
- **train_matrix** (*list*) – A list of the training fingerprint vectors.
- **test_matrix** (*list*) – A list of the test fingerprint vectors.
- **regularization** (*None or float*) – Smoothing parameter for the Gramm matrix.

24.1.2 catlearn.regression.gpfunctions.default_scale

Scale everything within regression functions.

class catlearn.regression.gpfunctions.default_scale.**ScaleData**(*train_features*, *train_targets*)

Bases: object

Class to perform default scaling in the regression functions.

Will standardize both the features and the targets. These can then be rescaled before being returned. The parameters can be accessed from the class with:

ScaleData.feature_data['mean']

This can be accessed from the gp with:

gp = GaussianProcess(...) gp.scaling.feature_data['mean']

rescale_targets(*predictions*)
Rescale predictions.

Parameters **predictions** (*list*) – The predicted values from the GP.

Returns **p** – The rescaled predictions.

Return type array

test(*test_features*)
Scale the test features.

Parameters **test_features** (*array*) – Feature matrix for the test data.

Returns **scaled_features** – The scaled features for the test data.

Return type array

train()
Scale the training features and targets.

Returns

- **feature_data** (*array*) – The scaled features for the training data.
- **target_data** (*array*) – The scaled targets for the training data.

24.1.3 catlearn.regression.gpfunctions.hyperparameter_scaling

Utility to scale hyperparameters.

catlearn.regression.gpfunctions.hyperparameter_scaling.**hyperparameters**(*scaling*, *kernel_list*)
Scale the hyperparameters.

catlearn.regression.gpfunctions.hyperparameter_scaling.**rescale_hyperparameters**(*scaling*, *kernel_list*)
Rescale hyperparameters.

24.1.4 catlearn.regression.gpfunctions.io

Functions to read and write models to file.

catlearn.regression.gpfunctions.io.**read**(*filename*, *ext='pkl'*)
Function to read a pickle of model object.

Parameters

- **filename** (*str*) – The name of the save file.
- **ext** (*str*) – Format to save GP, can be pkl or hdf5. Default is pkl.

Returns **model** – Python GaussianProcess object.

Return type obj

catlearn.regression.gpfunctions.io.**read_train_data**(*filename*)
Function to read raw training data.

Parameters **filename** (*str*) – The name of the save file.

Returns

- **train_features** (*arr*) – Arry of the training features.
- **train_targets** (*list*) – A list of the training targets.
- **regularization** (*float*) – The regularization parameter.
- **kernel_list** (*list*) – The dictionary containing parameters for the kernels.

catlearn.regression.gpfunctions.io.**write**(*filename*, *model*, *ext='pkl'*)
Function to write a pickle of model object.

Parameters

- **filename** (*str*) – The name of the save file.
- **model** (*obj*) – Python GaussianProcess object.
- **ext** (*str*) – Format to save GP, can be pkl or hdf5. Default is pkl.

catlearn.regression.gpfunctions.io.**write_train_data**(*filename*, *train_features*, *train_targets*, *regularization*, *kernel_list*)
Function to write raw training data.

Parameters

- **filename** (*str*) – The name of the save file.
- **train_features** (*arr*) – Arry of the training features.
- **train_targets** (*list*) – A list of the training targets.
- **regularization** (*float*) – The regularization parameter.
- **kernel_list** (*dict*) – The list containing dictionaries for the kernels.

24.1.5 catlearn.regression.gpfunctions.kernel_scaling

Function to scale kernel hyperparameters.

catlearn.regression.gpfunctions.kernel_scaling.**kernel_scaling**(*scale_data*, *kernel_list*, *rescale*)
Base hyperparameter scaling function.

Parameters

- **scale_data** (*object*) – Output from the default scaling function.
- **kernel_list** (*list*) – Dictionary containing all dictionaries for the kernels.
- **rescale** (*boolean*) – Flag for whether to scale or rescale the data.

24.1.6 catlearn.regression.gpfunctions.kernel_setup

Functions to prepare and return kernel data.

catlearn.regression.gpfunctions.kernel_setup.**kdict2list**(*kdict*, *N_D=None*)
Return ordered list of hyperparameters.

Assumes function is given a dictionary containing properties of a single kernel. The dictionary must contain either the key 'hyperparameters' or 'theta' containing a list of hyperparameters or the keys 'type' containing the type name in a string and 'width' in the case of a 'gaussian' or 'laplacian' type or the keys 'degree' and 'slope' in the case of a 'quadratic' type.

Parameters

- **kdict** (*dict*) – A kernel dictionary containing the keys 'type' and optional keys containing the hyperparameters of the kernel.
- **N_D** (*none or int*) – The number of descriptors if not specified in the kernel dict, by the lenght of the lists of hyperparameters.

catlearn.regression.gpfunctions.kernel_setup.**kdicts2list**(*kernel_list*, *N_D=None*)
Return ordered list of hyperparameters given the kernel dictionary.

The kernel dictionary must contain one or more dictionaries, each specifying the type and hyperparameters.

Parameters

- **kernel_list** (*dict*) – A dictionary containing kernel dictionaries.
- **N_D** (*int*) – The number of descriptors if not specified in the kernel dict, by the length of the lists of hyperparameters.

catlearn.regression.gpfunctions.kernel_setup.**list2kdict**(*hyperparameters*, *kernel_list*)
Return updated kernel dictionary with updated hyperparameters from list.

Assumed an ordered list of hyperparametersthe and the previous kernel dictionary. The kernel dictionary must contain a dictionary for each kernel type in the same order as their respective hyperparameters in the list hyperparameters.

Parameters

- **hyperparameters** (*list*) – All hyperparameters listed in the order they are specified in the kernel dictionary.
- **kernel_list** (*dict*) – A dictionary containing kernel dictionaries.

catlearn.regression.gpfunctions.kernel_setup.**prepare_kernels**(*kernel_list*, *regularization_bounds*, *eval_gradients*, *N_D*)
Format kernel_listionary and stores bounds for optimization.

Parameters

- **kernel_list** (*dict*) – List containing all dictionaries for the kernels.
- **regularization_bounds** (*tuple*) – Optional to change the bounds for the regularization.
- **eval_gradients** (*boolean*) – Flag to change kernel setup based on gradients being defined.
- **N_D** (*int*) – Number of dimensions of the original data.

24.1.7 catlearn.regression.gpfunctions.kernels

Contains kernel functions and gradients of kernels.

catlearn.regression.gpfunctions.kernels.**AA_kernel**(*theta*, *log_scale*, *m1*, *m2=None*, *eval_gradients=False*)

Generate the covariance between data with a Aichinson & Aitken kernel.

Parameters

- **theta** (*list*) – [l, n, c]
- **log_scale** (*boolean*) – Scaling hyperparameters in the kernel can be useful for optimization.
- **m1** (*list*) – A list of the training fingerprint vectors.
- **m2** (*list*) – A list of the training fingerprint vectors.

Returns **k** – The covariance matrix.

Return type array

catlearn.regression.gpfunctions.kernels.**constant_kernel**(*theta*, *log_scale*, *m1*, *m2=None*, *eval_gradients=False*)

Return constant to add to the kernel.

Parameters

- **theta** (*list*) – A list of widths for each feature.
- **log_scale** (*boolean*) – Scaling hyperparameters in the kernel can be useful for optimization.
- **eval_gradients** (*boolean*) – Analytical gradients of the training features can be included.
- **m1** (*list*) – A list of the training fingerprint vectors.
- **m2** (*list*) – A list of the training fingerprint vectors.

Returns **k** – The covariance matrix.

Return type array

catlearn.regression.gpfunctions.kernels.**constant_multi_kernel**(*theta*, *log_scale*, *m1*, *m2=None*, *eval_gradients=True*)

Return constant to add to the kernel.

Parameters

- **theta** (*list*) – A list containing the constants.
- **log_scale** (*boolean*) – Scaling hyperparameters in the kernel can be useful for optimization.
- **eval_gradients** (*boolean*) – Analytical gradients of the training features can be included.
- **m1** (*list*) – A list of the training fingerprint vectors.
- **m2** (*list*) – A list of the training fingerprint vectors.

Returns **k** – The covariance matrix.

Return type array

catlearn.regression.gpfunctions.kernels.**gaussian_dk_dwidth**(*k*, *m1*, *kwidth*, *log_scale=False*)

Return gradient of the gaussian kernel with respect to the j'th width.

Parameters

- **k** (*array*) – n by n array. The (not scaled) gaussian kernel.
- **m1** (*list*) – A list of the training fingerprint vectors.
- **kwidth** (*float*) – The full list of widths
- **log_scale** (*boolean*) – Scaling hyperparameters in kernel can be useful for optimization.

catlearn.regression.gpfunctions.kernels.**gaussian_kernel**(*theta*, *log_scale*, *m1*, *m2=None*, *eval_gradients=False*)

Generate the covariance between data with a Gaussian kernel.

Parameters

- **theta** (*list*) – A list of widths for each feature.
- **log_scale** (*boolean*) – Scaling hyperparameters in the kernel can be useful for optimization.
- **eval_gradients** (*boolean*) – Analytical gradients of the training features can be included.
- **m1** (*list*) – A list of the training fingerprint vectors.
- **m2** (*list*) – A list of the training fingerprint vectors.

Returns **k** – The covariance matrix.

Return type array

catlearn.regression.gpfunctions.kernels.**gaussian_xx_gradients**(*m1*, *kwidth*, *k*)

Gradient for k(x, x).

Parameters

- **m1** (*array*) – Feature matrix.
- **kwidth** (*list*) – List of lengthscales for the gaussian kernel.
- **k** (*array*) – Upper left portion of the overall covariance matrix.

catlearn.regression.gpfunctions.kernels.**gaussian_xxp_gradients**(*m1*, *m2*, *kwidth*, *k*)

Gradient for k(x, x').

Parameters

- **m1** (*array*) – Feature matrix.
- **m2** (*array*) – Feature matrix typically associated with the test data.
- **kwidth** (*list*) – List of lengthscales for the gaussian kernel.
- **k** (*array*) – Upper left portion of the overall covariance matrix.

catlearn.regression.gpfunctions.kernels.**laplacian_dk_dwidth**(*k*, *m1*, *kwidth*, *log_scale=False*)

catlearn.regression.gpfunctions.kernels.**laplacian_kernel**(*theta*, *log_scale*, *m1*, *m2=None*, *eval_gradients=False*)

Generate the covariance between data with a laplacian kernel.

Parameters

- **theta** (*list*) – A list of widths for each feature.
- **log_scale** (*boolean*) – Scaling hyperparameters in the kernel can be useful for optimization.
- **m1** (*list*) – A list of the training fingerprint vectors.
- **m2** (*list or None*) – A list of the training fingerprint vectors.

Returns **k** – The covariance matrix.

Return type array

catlearn.regression.gpfunctions.kernels.**linear_kernel**(*theta*, *log_scale*, *m1*, *m2=None*, *eval_gradients=False*)

Generate the covariance between data with a linear kernel.

Parameters

- **theta** (*list*) – A list containing constant offset.
- **log_scale** (*boolean*) – Scaling hyperparameters in the kernel can be useful for optimization.
- **eval_gradients** (*boolean*) – Analytical gradients of the training features can be included.
- **m1** (*list*) – A list of the training fingerprint vectors.
- **m2** (*list or None*) – A list of the training fingerprint vectors.

Returns **k** – The covariance matrix.

Return type array

catlearn.regression.gpfunctions.kernels.**noise_multi_kernel**(*theta*, *log_scale*, *m1*, *m2=None*, *eval_gradients=False*)

Return constant to add to the kernel.

Parameters

- **theta** (*list*) – A list containing the constants to be added in the diagonal of the covariance matrix .
- **eval_gradients** (*boolean*) – Analytical gradients of the training features can be included.
- **m1** (*list*) – A list of the training fingerprint vectors.
- **m2** (*list*) – A list of the training fingerprint vectors.

Returns **k** – The covariance matrix.

Return type array

catlearn.regression.gpfunctions.kernels.**quadratic_dk_ddegree**(*k*, *m1*, *degree*, *log_scale=False*)

catlearn.regression.gpfunctions.kernels.**quadratic_dk_dslope**(*k*, *m1*, *slope*, *log_scale=False*)

catlearn.regression.gpfunctions.kernels.**quadratic_kernel**(*theta*, *log_scale*, *m1*, *m2=None*, *eval_gradients=False*)

Generate the covariance between data with a quadratic kernel.

Parameters

- **theta** (*list*) – A list containing slope and degree for quadratic.
- **log_scale** (*boolean*) – Scaling hyperparameters in the kernel can be useful for optimization.
- **m1** (*list*) – A list of the training fingerprint vectors.
- **m2** (*list or None*) – A list of the training fingerprint vectors.

Returns **k** – The covariance matrix.

Return type array

catlearn.regression.gpfunctions.kernels.**scaled_sqe_kernel**(*theta*, *log_scale*, *m1*, *m2=None*, *eval_gradients=False*)

Generate the covariance between data with a Gaussian kernel.

Parameters

- **theta** (*list*) – A list of hyperparameters.
- **log_scale** (*boolean*) – Scaling hyperparameters in the kernel can be useful for optimization.
- **m1** (*list*) – A list of the training fingerprint vectors.
- **m2** (*list*) – A list of the training fingerprint vectors.

Returns **k** – The covariance matrix.

Return type array

catlearn.regression.gpfunctions.kernels.**sqe_kernel**(*theta*, *log_scale*, *m1*, *m2=None*, *eval_gradients=False*)

Generate the covariance between data with a Gaussian kernel.

Parameters

- **theta** (*list*) – A list of widths for each feature.
- **log_scale** (*boolean*) – Scaling hyperparameters in the kernel can be useful for optimization.
- **m1** (*list*) – A list of the training fingerprint vectors.
- **m2** (*list*) – A list of the training fingerprint vectors.

Returns **k** – The covariance matrix.

Return type array

24.1.8 catlearn.regression.gpfunctions.log_marginal_likelihood

Log marginal likelihood calculator function.

catlearn.regression.gpfunctions.log_marginal_likelihood.**dK_dtheta_j**(*theta*, *train_matrix*, *kernel_list*, *Q*)

Return the jacobian of the log marginal likelyhood.

This is calculated with respect to the hyperparameters, as in: Equation 5.9 in C. E. Rasmussen and C. K. I. Williams, 2006

Parameters

- **theta** (*list*) – A list containing the hyperparameters.
- **train_matrix** (*list*) – A list of the test fingerprint vectors.
- **kernel_list** (*list*) – A list of kernel dictionaries.
- **Q** (*array.*) –

catlearn.regression.gpfunctions.log_marginal_likelihood.**log_marginal_likelihood**(*theta*, *train_matrix*, *targets*, *kernel_list*, *scale_optimizer*, *eval_gradients*, *cinv=None*, *eval_jac=False*)

Return the negative of the log marginal likelyhood.

Equation 5.8 in C. E. Rasmussen and C. K. I. Williams, 2006

Parameters

- **theta** (*list*) – A list containing the hyperparameters.
- **train_matrix** (*list*) – A list of the test fingerprint vectors.
- **targets** (*list*) – A list of target values.
- **kernel_list** (*dict*) – A list of kernel dictionaries.
- **scale_optimizer** (*boolean*) – Flag to define if the hyperparameters are log scale for optimization.
- **eval_gradients** (*boolean*) – Flag to specify whether to compute gradients in covariance.
- **cinv** (*array*) – Pre-computed inverted covariance matrix.
- **eval_jac** (*boolean*) – Flag to specify whether to calculate gradients for hyperparameter optimization.

24.1.9 catlearn.regression.gpfunctions.sensitivity

Function performing GP sensitivity analysis.

class catlearn.regression.gpfunctions.sensitivity.**SensitivityAnalysis**(*train_matrix*, *train_targets*, *test_matrix*, *kernel_list*, *init_reg=0.001*, *init_width=10.0*)

Bases: object

Perform sensitivity analysis to estimate important features.

backward_selection(*predict=False*, *test_targets=None*, *selection=None*)

Feature selection with backward elimination.

Parameters

- **predict** (*boolean*) – Specify whether to make predictions on test data.
- **test_targets** (*list*) – A list of test targets to calculate errors, if known.
- **selection** (*int, list*) – Specify the number or range of features to consider.

24.1.10 catlearn.regression.gpfunctions.uncertainty

Function performing uncertainty analysis.

catlearn.regression.gpfunctions.uncertainty.**get_uncertainty**(*kernel_list*, *test_fp*, *ktb*, *cinv*, *log_scale*)

Function to calculate uncertainty.

Parameters

- **kernel_list** (*list*) – List containing all dictionaries for the kernels.
- **test_fp** (*array*) – Test feature set.
- **ktb** (*array*) – Covariance matrix for test and training data.
- **cinv** (*array*) – Covariance matrix for training dataset.
- **log_scale** (*boolean*) – Flag to define if the hyperparameters are log scale.

Returns **uncertainty** – The uncertainty on each prediction in the test data. By default, this includes a measure of the noise on the data.

Return type list

24.2 catlearn.regression.cost_function

Functions to calculate the cost statistics.

catlearn.regression.cost_function.**get_error**(*prediction*, *target*, *metrics=None*, *epsilon=None*, *return_percentiles=True*)

Return error for predicted data.

Discussed in: Rosasco et al, Neural Computation, (2004), 16, 1063-1076.

Parameters

- **prediction** (*list*) – A list of predicted values.
- **target** (*list*) – A list of target values.

- **metrics** (*list*) – Define a list of additional cost functions to be returned. Can currently be 'log' and 'insensitive'.
- **epsilon** (*float*) – insensitivity value.
- **return_percentiles** (*boolean*) – Return some percentile statistics with the predictions.

24.3 catlearn.regression.gaussian_process

Functions to make predictions with Gaussian Processes machine learning.

class catlearn.regression.gaussian_process.**GaussianProcess**(*train_fp*, *train_target*, *kernel_list*, *gradients=None*, *regularization=None*, *regularization_bounds=None*, *optimize_hyperparameters=False*, *scale_optimizer=False*, *scale_data=False*)

Bases: object

Gaussian processes functions for the machine learning.

optimize_hyperparameters(*global_opt=False*, *algomin='L-BFGS-B'*, *eval_jac=False*, *loss_function='lml'*)

Optimize hyperparameters of the Gaussian Process.

This function assumes that the descriptors in the feature set remain the same. Optimization is performed with respect to the log marginal likelihood. Optimized hyperparameters are saved in the kernel dictionary. Finally, the covariance matrix is updated.

Parameters

- **global_opt** (*boolean*) – Flag whether to do basin hopping optimization of hyperparameters. Default is False.
- **algomin** (*str*) – Define scipy minimizer method to call. Default is L-BFGS-B.

predict(*test_fp*, *test_target=None*, *uncertainty=False*, *basis=None*, *get_validation_error=False*, *get_training_error=False*, *epsilon=None*)

Function to perform the prediction on some training and test data.

Parameters

- **test_fp** (*list*) – A list of testing fingerprint vectors.
- **test_target** (*list*) – A list of the the test targets used to generate the prediction errors.
- **uncertainty** (*boolean*) – Return data on the predicted uncertainty if True. Default is False.
- **basis** (*function*) – Basis functions to assess the reliability of the uncertainty predictions. Must be a callable function that takes a list of descriptors and returns another list.
- **get_validation_error** (*boolean*) – Return the error associated with the prediction on the test set of data if True. Default is False.

- **get_training_error** (*boolean*) – Return the error associated with the prediction on the training set of data if True. Default is False.
- **epsilon** (*float*) – Threshold for insensitive error calculation.

Returns

data – Gaussian process predictions and meta data:

prediction [vector] Predicted mean.

uncertainty [vector] Predicted standard deviation of the Gaussian posterior.

training_error [dictionary] Error metrics on training targets.

validation_error [dictionary] Error metrics on test targets.

Return type dictionary

predict_uncertainty(*test_fp*)
Return uncertainty only.

Parameters **test_fp** (*list*) – A list of testing fingerprint vectors.

update_data(*train_fp*, *train_target=None*, *gradients=None*, *scale_optimizer=False*)
Update the training matrix, targets and covariance matrix.

This function assumes that the descriptors in the feature set remain the same. That it is just the number of data ponts that is changing. For this reason the hyperparameters are not updated, so this update process should be fast.

Parameters

- **train_fp** (*list*) – A list of training fingerprint vectors.
- **train_target** (*list*) – A list of training targets used to generate the predictions.
- **scale_optimizer** (*boolean*) – Flag to define if the hyperparameters are log scale for optimization.

update_gp(*train_fp=None*, *train_target=None*, *kernel_list=None*, *scale_optimizer=False*, *gradients=None*, *regularization_bounds=(1e-06*, *None)*, *optimize_hyperparameters=False*)
Potentially optimize the full Gaussian Process again.

This alows for the definition of a new kernel as a result of changing descriptors in the feature space. Other parts of the model can also be changed. The hyperparameters will always be reoptimized.

Parameters

- **train_fp** (*list*) – A list of training fingerprint vectors.
- **train_target** (*list*) – A list of training targets used to generate the predictions.
- **kernel_list** (*dict*) – This dict can contain many other dictionarys, each one containing parameters for separate kernels. Each kernel dict contains information on a kernel such as: - The 'type' key containing the name of kernel function. - The hyperparameters, e.g. 'scaling', 'lengthscale', etc.
- **scale_optimizer** (*boolean*) – Flag to define if the hyperparameters are log scale for optimization.
- **regularization_bounds** (*tuple*) – Optional to change the bounds for the regularization.

24.4 catlearn.regression.ridge_regression

Modified ridge regression function from Keld Lundgaard.

class catlearn.regression.ridge_regression.**RidgeRegression**(*W2=None*, *Vh=None*, *cv='loocv'*, *Ns=100*, *wsteps=15*, *rsteps=3*)

Bases: object

Ridge regression class to find an optimal model.

Regualization fitting can be performed with wither the loocv or bootstrap.632 method. The loocv method is faseter, but it is better to use bootstrap when there is highly correlated training data.

RR(*X*, *Y*, *omega2*, *p=0.0*, *featselect_featvar=False*)
Ridge Regression (RR) solver.

Cost is (Xa-y)**2 + omega2*(a-p)**2, SVD of X.T X, where T is the transpose V, W2, Vh = X.T*X

Parameters

- **X** (*array*) – Feature matrix for the training data.
- **Y** (*list*) – Target data for the training sample.
- **p** (*float*) – Define the prior function.
- **omega2** (*float*) – Regularization strength.

Returns

- **coefs** (*list*) – Optimal coefficients.
- **neff** (*float*) – Number of effective parameters.

bootstrap_calc(*X*, *Y*, *p*, *omega2*, *samples*, *W2_samples*, *Vh_samples*)
Calculate optimal omega2 from bootstrap.

Parameters

- **X** (*array*) – Feature matrix for the training data.
- **Y** (*list*) – Target data for the training sample.
- **p** (*float*) – Define the prior function.
- **omega2** (*float*) – Regularization strength.
- **samples** (*list*) – Sample index for bootstrap.
- **W2_samples** (*array*) – Sigular values for samples.
- **Vh_samples** (*array*) – Right hand side of sigular matrix for samples.

find_optimal_regularization(*X*, *Y*, *p=0.0*)
Find regualization value to minimize Expected Prediction Error.

Parameters

- **X** (*array*) – Feature matrix for the training data.
- **Y** (*list*) – Target data for the training sample.
- **p** (*float*) – Define the prior function. Default is zero.

Returns **omega2_min** – Regularization corresponding to the minimum EPE.

Return type float

get_coefficients(*train_targets*, *train_features*, *reg=None*, *p=0.0*)
Generate the omgea2 and coef value's.

Parameters

- **train_targets** (*array*) – Dependent data used for training.
- **train_features** (*array*) – Independent data used for training.
- **reg** (*float*) – Precomputed optimal regaluzation.
- **p** (*float*) – Define the prior function. Default is zero.

predict(*train_matrix*, *train_targets*, *test_matrix*, *test_targets=None*, *coefficients=None*, *reg=None*, *p=0.0*)
Function to do ridge regression predictions.

regularization(*train_targets*, *train_features*, *coef=None*, *featselect_featvar=False*)
Generate the omgea2 and coef value's.

Parameters **train_targets** (*array*) – Dependent data used for training.

train_features [array] Independent data used for training.

coef [int] List of indices in the feature database.

24.5 catlearn.regression.scikit_wrapper

Regression models to assess features using scikit-learn framework.

class catlearn.regression.scikit_wrapper.**RegressionFit**(*train_matrix*, *train_target*, *test_matrix=None*, *test_target=None*, *method='ridge'*, *predict=False*)
Bases: object

Class to perform a fit to specified regression model.

feature_select(*size=None*, *iterations=100000.0*, *steps=None*, *line_search=False*, *min_alpha=1e-08*, *max_alpha=0.1*, *eps=0.001*)
Find index of important featurs.

Parameters

- **size** (*int*) – Number best features to return.
- **iterations** (*float*) – Maximum number of iterations taken minimizing the regression function. Implemented in elastic net and lasso.
- **steps** (*int*) – Number of steps to be taken in the penalty function of LASSO.
- **min_alpha** (*float*) – Starting penalty when searching over range. Default is 1.e-8.
- **max_alpha** (*float*) – Final penalty when searching over range. Default is 1.e-1.

CHAPTER 25

catlearn.active_learning package

25.1 Submodules

25.2 catlearn.active_learning.acquisition_functions module

GP acquisition functions.

catlearn.active_learning.acquisition_functions.**EI**(*y_best*, *predictions*, *uncertainty*, *objective='max'*)

Return expected improvement acq. function.

Parameters

- **y_best** (*float*) – Condition
- **predictions** (*list*) – Predicted means.
- **uncertainty** (*list*) – Uncertainties associated with the predictions.

catlearn.active_learning.acquisition_functions.**PI**(*y_best*, *predictions*, *uncertainty*, *objective*)

Probability of improvement acq. function.

Parameters

- **y_best** (*float*) – Condition
- **predictions** (*list*) – Predicted means.
- **uncertainty** (*list*) – Uncertainties associated with the predictions.

catlearn.active_learning.acquisition_functions.**UCB**(*predictions*, *uncertainty*, *objective='max'*, *kappa=1.5*)

Upper-confidence bound acq. function.

Parameters

- **predictions** (*list*) – Predicted means.

- **uncertainty** (*list*) – Uncertainties associated with the predictions.
- **kappa** (*float*) – Constant that controls the explotation/exploration ratio in UCB.

catlearn.active_learning.acquisition_functions.**classify**(*classifier, train_atoms, test_atoms, targets, predictions, uncertainty, train_features=None, test_features=None, objective='max', k_means=3, kappa=1.5, metrics=['optimistic', 'UCB', 'EI', 'PI']*)

Classify ranked predictions based on acquisition function.

Parameters

- **classifier** (*func*) – User defined function to classify an atoms object.
- **train_atoms** (*list*) – List of atoms objects from training data upon which to base classification.
- **test_atoms** (*list*) – List of atoms objects from test data upon which to base classification.
- **targets** (*list*) – List of known target values.
- **predictions** (*list*) – List of predictions from the GP.
- **uncertainty** (*list*) – List of variance on the GP predictions.
- **train_features** (*array*) – Feature matrix for the training data.
- **test_features** (*array*) – Feature matrix for the test data.
- **k_means** (*int*) – Number of cluster to generate with clustering.
- **kappa** (*float*) – Constant that controls the explotation/exploration ratio in UCB.
- **metrics** (*list*) – list of strings. Accepted values are 'cdf', 'UCB', 'EI', 'PI', 'optimistic' and 'pdf'.

Returns **res** – A dictionary of lists containg the fitness of each test point for the different acquisition functions.

Return type dict

catlearn.active_learning.acquisition_functions.**cluster**(*train_features, targets, test_features, predictions, k_means=3*)

Penalize test points that are too clustered.

Parameters

- **train_features** (*array*) – Feature matrix for the training data.
- **targets** (*list*) – Training targets.
- **test_features** (*array*) – Feature matrix for the test data.
- **predictions** (*list*) – Predicted means.
- **k_means** (*int*) – Number of clusters.

catlearn.active_learning.acquisition_functions.**optimistic**(*y_best, predictions, uncertainty*)

Find predictions that will optimistically lead to progress.

Parameters

- **y_best** (*float*) – Condition
- **predictions** (*list*) – Predicted means.
- **uncertainty** (*list*) – Uncertainties associated with the predictions.

catlearn.active_learning.acquisition_functions.**optimistic_proximity**(*y_best*, *predictions*, *uncertainty*)

Return uncertainties minus distances to y_best.

Parameters

- **y_best** (*float*) – Condition
- **predictions** (*list*) – Predicted means.
- **uncertainty** (*list*) – Uncertainties associated with the predictions.

catlearn.active_learning.acquisition_functions.**probability_density**(*y_best*, *predictions*, *uncertainty*)

Return probability densities at y_best.

Parameters

- **y_best** (*float*) – Condition
- **predictions** (*list*) – Predicted means.
- **uncertainty** (*list*) – Uncertainties associated with the predictions.

catlearn.active_learning.acquisition_functions.**proximity**(*y_best*, *predictions*, *uncertainty=None*)

Return negative distances to y_best.

Parameters

- **y_best** (*float*) – Condition
- **predictions** (*list*) – Predicted means.
- **uncertainty** (*list*) – Uncertainties associated with the predictions.

catlearn.active_learning.acquisition_functions.**random_acquisition**(*y_best*, *predictions*, *uncertainty=None*)

Return random numbers for control experiments.

Parameters

- **y_best** (*float*) – Condition
- **predictions** (*list*) – Predicted means.
- **uncertainty** (*list*) – Uncertainties associated with the predictions.

catlearn.active_learning.acquisition_functions.**rank**(*targets, predictions, uncertainty, train_features=None, test_features=None, objective='max', k_means=3, kappa=1.5, metrics=['optimistic', 'UCB', 'EI', 'PI']*)

Rank predictions based on acquisition function.

Parameters

- **targets** (*list*) – List of known target values.
- **predictions** (*list*) – List of predictions from the GP.
- **uncertainty** (*list*) – List of variance on the GP predictions.
- **train_features** (*array*) – Feature matrix for the training data.
- **test_features** (*array*) – Feature matrix for the test data.
- **k_means** (*int*) – Number of cluster to generate with clustering.
- **kappa** (*float*) – Constant that controls the explotation/exploration ratio in UCB.
- **metrics** (*list*) – list of strings. Accepted values are 'cdf', 'UCB', 'EI', 'PI', 'optimistic' and 'pdf'.

Returns **res** – A dictionary of lists containg the fitness of each test point for the different acquisition functions.

Return type dict

25.3 catlearn.active_learning.algorithm module

Class to automate building a surrogate model.

class catlearn.active_learning.algorithm.**ActiveLearning**(*surrogate_model, train_data, target*)

Bases: object

Active learning class, intended for screening or optimizing in a predefined and finite search space.

acquire(*unlabeled_data, batch_size=1*)

Return indices of datapoints to acquire, from a predefined, finite search space.

Parameters

- **unlabeled_data** (*array*) – Data matrix representing an unlabeled search space.
- **initial_subset** (*list*) – Row indices of data to train on in the first iteration.
- **batch_size** (*int*) – Number of training points to acquire (move from test to training) in every iteration.

Returns

- **to_acquire** (*list*) – Row indices of unlabeled data to acquire.
- *score* – User defined output from predict.

ensemble_test(*size, initial_subset=None, batch_size=1, n_max=None, seed_list=None, nprocs=None*)

Return a 3d array of test results for a surrogate model. The third dimension expands the ensemble of tests.

Parameters

- **size** (*int*) – How many tests to run.
- **initial_subset** (*list*) – Row indices of data to train on in the first iteration.
- **batch_size** (*int*) – Number of training points to acquire (move from test to training) in every iteration.
- **n_max** (*int*) – Max number of training points to test.
- **seed_list** (*list*) – List of integer seeds for shuffling training data.
- **nprocs** (*int*) – Number of processors for parallelization

Returns **ensemble** – size by iterations by number of metrics array of test results.

Return type array

test_acquisition(*initial_subset=None*, *batch_size=1*, *n_max=None*, *seed=None*)

Return an array of test results for a surrogate model.

Parameters

- **initial_subset** (*list*) – Row indices of data to train on in the first iteration.
- **batch_size** (*int*) – Number of training points to acquire (move from test to training) in every iteration.
- **n_max** (*int*) – Max number of training points to test.

25.4 Module contents

CHAPTER 26

catlearn.estimator package

26.1 Submodules

26.2 catlearn.estimator.general_gp module

Function to setup a general GP.

class catlearn.estimator.general_gp.**GeneralGaussianProcess**(*clean_type='eliminate'*, *dimension='single'*, *kernel='general'*)

Bases: object

Define a general setup for the Gaussin process.

This should not be used to try and obtain highly accurate solutions. Though it should give a reasonable model.

gaussian_process_predict(*test_features*)

Function to make GP predictions on tests data.

Parameters **test_features** (*array*) – The array of test features.

Returns **prediction** – The prediction data generated by the Gaussian process.

Return type dict

train_gaussian_process(*train_features*, *train_targets*)

Generate a general gaussian process model.

Parameters

- **train_features** (*array*) – The array of training features.
- **train_targets** (*array*) – A list of training target values.

Returns **gp** – The trained Gaussian process.

Return type object

26.3 catlearn.estimator.general_kernel module

Setup a generic kernel.

catlearn.estimator.general_kernel.**default_lengthscale**(*features*, *dimension='single'*)
Generate defaults for the kernel lengthscale.

Parameters

- **features** (*array*) – The feature matrix for the training data.
- **dimension** (*str*) – The number of parameters to return. Can be 'single', or 'features'.

Returns **std** – The standard deviation of the features.

Return type array

catlearn.estimator.general_kernel.**general_kernel**(*features*, *dimension='single'*)
Generate a default kernel.

catlearn.estimator.general_kernel.**smooth_kernel**(*features*, *dimension='single'*)
Generate a default kernel.

26.4 catlearn.estimator.general_preprocess module

A default setup for data preprocessing.

class catlearn.estimator.general_preprocess.**GeneralPrepreprocess**(*clean_type='eliminate'*)
Bases: object

A general purpose data preprocessing class.

process(*train_features*, *train_targets*, *test_features=None*)
Processing function.

Parameters

- **train_features** (*array*) – The array of training features.
- **train_targets** (*array*) – A list of training target values.
- **test_features** (*array*) – The array of test features.

transform(*features*)
Function to transform a new set of features.

Parameters **features** (*array*) – A new array of features to clean. This will most likely be the new test features.

Returns **processed** – A cleaned and scaled feature set.

Return type array

26.5 Module contents

CHAPTER 27

catlearn.optimize package

27.1 Submodules

27.2 catlearn.optimize.constraints module

catlearn.optimize.constraints.**apply_mask**(*list_to_mask=None*, *mask_index=None*)

catlearn.optimize.constraints.**create_mask**(*ini*, *constraints*)

catlearn.optimize.constraints.**unmask_geometry**(*org_list*, *masked_geom*, *mask_index*)

27.3 catlearn.optimize.convergence module

27.4 catlearn.optimize.functions_calc module

class catlearn.optimize.functions_calc.**GoldsteinPrice**(***kwargs*)

Bases: ase.calculators.calculator.Calculator

GoldsteinPrice potential.

calculate(*atoms=None, properties=['energy', 'forces'], system_changes=['positions', 'numbers', 'cell', 'pbc']*)

Do the calculation.

properties: list of str List of what needs to be calculated. Can be any combination of 'energy', 'forces', 'stress', 'dipole', 'charges', 'magmom' and 'magmoms'.

system_changes: list of str List of what has changed since last calculation. Can be any combination of these six: 'positions', 'numbers', 'cell', 'pbc', 'initial_charges' and 'initial_magmoms'.

Subclasses need to implement this, but can ignore properties and system_changes if they want. Calculated properties should be inserted into results dictionary like shown in this dummy example:

```
self.results = {'energy': 0.0,
                'forces': np.zeros((len(atoms), 3)),
                'stress': np.zeros(6),
                'dipole': np.zeros(3),
                'charges': np.zeros(len(atoms)),
                'magmom': 0.0,
                'magmoms': np.zeros(len(atoms))}
```

The subclass implementation should first call this implementation to set the atoms attribute.

implemented_properties = ['energy', 'forces']

nolabel = True

class catlearn.optimize.functions_calc.**Himmelblau**(***kwargs*)
Bases: ase.calculators.calculator.Calculator

Himmelblau potential.

calculate(*atoms=None, properties=['energy', 'forces'], system_changes=['positions', 'numbers', 'cell', 'pbc']*)
Do the calculation.

properties: list of str List of what needs to be calculated. Can be any combination of 'energy', 'forces', 'stress', 'dipole', 'charges', 'magmom' and 'magmoms'.

system_changes: list of str List of what has changed since last calculation. Can be any combination of these six: 'positions', 'numbers', 'cell', 'pbc', 'initial_charges' and 'initial_magmoms'.

Subclasses need to implement this, but can ignore properties and system_changes if they want. Calculated properties should be inserted into results dictionary like shown in this dummy example:

```
self.results = {'energy': 0.0,
                'forces': np.zeros((len(atoms), 3)),
                'stress': np.zeros(6),
                'dipole': np.zeros(3),
                'charges': np.zeros(len(atoms)),
                'magmom': 0.0,
                'magmoms': np.zeros(len(atoms))}
```

The subclass implementation should first call this implementation to set the atoms attribute.

implemented_properties = ['energy', 'forces']

nolabel = True

class catlearn.optimize.functions_calc.**ModifiedHimmelblau**(***kwargs*)
Bases: ase.calculators.calculator.Calculator

Himmelblau potential.

calculate(*atoms=None, properties=['energy', 'forces'], system_changes=['positions', 'numbers', 'cell', 'pbc']*)
Do the calculation.

properties: list of str List of what needs to be calculated. Can be any combination of 'energy', 'forces', 'stress', 'dipole', 'charges', 'magmom' and 'magmoms'.

system_changes: list of str List of what has changed since last calculation. Can be any combination of these six: 'positions', 'numbers', 'cell', 'pbc', 'initial_charges' and 'initial_magmoms'.

Subclasses need to implement this, but can ignore properties and system_changes if they want. Calculated properties should be inserted into results dictionary like shown in this dummy example:

```
self.results = {'energy': 0.0,
                'forces': np.zeros((len(atoms), 3)),
                'stress': np.zeros(6),
                'dipole': np.zeros(3),
                'charges': np.zeros(len(atoms)),
                'magmom': 0.0,
                'magmoms': np.zeros(len(atoms))}
```

The subclass implementation should first call this implementation to set the atoms attribute.

implemented_properties = ['energy', 'forces']

nolabel = True

class catlearn.optimize.functions_calc.**MullerBrown**(***kwargs*)
Bases: ase.calculators.calculator.Calculator

Muller-brown potential.

calculate(*atoms=None, properties=['energy', 'forces'], system_changes=['positions', 'numbers', 'cell', 'pbc']*)
Do the calculation.

properties: list of str List of what needs to be calculated. Can be any combination of 'energy', 'forces', 'stress', 'dipole', 'charges', 'magmom' and 'magmoms'.

system_changes: list of str List of what has changed since last calculation. Can be any combination of these six: 'positions', 'numbers', 'cell', 'pbc', 'initial_charges' and 'initial_magmoms'.

Subclasses need to implement this, but can ignore properties and system_changes if they want. Calculated properties should be inserted into results dictionary like shown in this dummy example:

```
self.results = {'energy': 0.0,
                'forces': np.zeros((len(atoms), 3)),
                'stress': np.zeros(6),
                'dipole': np.zeros(3),
                'charges': np.zeros(len(atoms)),
                'magmom': 0.0,
                'magmoms': np.zeros(len(atoms))}
```

The subclass implementation should first call this implementation to set the atoms attribute.

default_parameters = {'p1': [-200.0, -1.0, 0.0, -10.0, 1.0, 0.0], 'p2': [-100.0, -1.

implemented_properties = ['energy', 'forces']

nolabel = True

class catlearn.optimize.functions_calc.**MultiModal**(***kwargs*)
Bases: ase.calculators.calculator.Calculator

MultiModal potential.

calculate(*atoms=None, properties=['energy', 'forces'], system_changes=['positions', 'numbers', 'cell', 'pbc']*)
Do the calculation.

properties: list of str List of what needs to be calculated. Can be any combination of 'energy', 'forces', 'stress', 'dipole', 'charges', 'magmom' and 'magmoms'.

system_changes: list of str List of what has changed since last calculation. Can be any combination of these six: 'positions', 'numbers', 'cell', 'pbc', 'initial_charges' and 'initial_magmoms'.

Subclasses need to implement this, but can ignore properties and system_changes if they want. Calculated properties should be inserted into results dictionary like shown in this dummy example:

```
self.results = {'energy': 0.0,
                'forces': np.zeros((len(atoms), 3)),
                'stress': np.zeros(6),
                'dipole': np.zeros(3),
                'charges': np.zeros(len(atoms)),
                'magmom': 0.0,
                'magmoms': np.zeros(len(atoms))}
```

The subclass implementation should first call this implementation to set the atoms attribute.

implemented_properties = ['energy', 'forces']

nolabel = True

class catlearn.optimize.functions_calc.**NoiseHimmelblau**(***kwargs*)
Bases: ase.calculators.calculator.Calculator

NoiseHimmelblau potential.

calculate(*atoms=None, properties=['energy', 'forces'], system_changes=['positions', 'numbers', 'cell', 'pbc']*)
Do the calculation.

properties: list of str List of what needs to be calculated. Can be any combination of 'energy', 'forces', 'stress', 'dipole', 'charges', 'magmom' and 'magmoms'.

system_changes: list of str List of what has changed since last calculation. Can be any combination of these six: 'positions', 'numbers', 'cell', 'pbc', 'initial_charges' and 'initial_magmoms'.

Subclasses need to implement this, but can ignore properties and system_changes if they want. Calculated properties should be inserted into results dictionary like shown in this dummy example:

```
self.results = {'energy': 0.0,
                'forces': np.zeros((len(atoms), 3)),
                'stress': np.zeros(6),
                'dipole': np.zeros(3),
                'charges': np.zeros(len(atoms)),
                'magmom': 0.0,
                'magmoms': np.zeros(len(atoms))}
```

The subclass implementation should first call this implementation to set the atoms attribute.

implemented_properties = ['energy', 'forces']

nolabel = True

class catlearn.optimize.functions_calc.**Rosenbrock**(***kwargs*)
Bases: ase.calculators.calculator.Calculator

Himmelblau potential.

calculate(*atoms=None, properties=['energy', 'forces'], system_changes=['positions', 'numbers', 'cell', 'pbc']*)
Do the calculation.

properties: list of str List of what needs to be calculated. Can be any combination of 'energy', 'forces', 'stress', 'dipole', 'charges', 'magmom' and 'magmoms'.

system_changes: list of str List of what has changed since last calculation. Can be any combination of these six: 'positions', 'numbers', 'cell', 'pbc', 'initial_charges' and 'initial_magmoms'.

Subclasses need to implement this, but can ignore properties and system_changes if they want. Calculated properties should be inserted into results dictionary like shown in this dummy example:

```
self.results = {'energy': 0.0,
                'forces': np.zeros((len(atoms), 3)),
                'stress': np.zeros(6),
                'dipole': np.zeros(3),
                'charges': np.zeros(len(atoms)),
                'magmom': 0.0,
                'magmoms': np.zeros(len(atoms))}
```

The subclass implementation should first call this implementation to set the atoms attribute.

implemented_properties = ['energy', 'forces']

nolabel = True

27.5 catlearn.optimize.get_real_values module

27.6 catlearn.optimize.io module

catlearn.optimize.io.**array_to_ase**(*input_array*, *num_atoms*)

Converts a flat array into an ase structure (list).

Parameters

- **input_array** (*ndarray*) – Structure.
- **num_atoms** (*int*) – Number of atoms.

Returns **pos_ase** – Position of the atoms in ASE format.

Return type list

catlearn.optimize.io.**array_to_atoms**(*input_array*)

Converts an input flat array into atoms shape for ASE.

Parameters **input_array** (*ndarray*) – Structure.

Returns

- **pos_ase** (*list*)
- *Position of the atoms in ASE format.*

catlearn.optimize.io.**ase_to_catlearn**(*list_atoms*)

Converts a trajectory file from ASE to a list of train, targets and gradients. The first and last images of the trajectory file are also included in this dictionary.

list_atoms [string] List Atoms objects in ASE format. The file must be in the current working directory.

results: dict Dictionary that contains list of train (including constraints), targets and gradients, number of atoms for the atomistic structure, images included in the trajectory file and Atoms structures of the initial and final endpoints of the NEB.

catlearn.optimize.io.**print_cite_mlmin**()

catlearn.optimize.io.**print_cite_mlneb**()

catlearn.optimize.io.**print_info**(*self*)
Output of the ML-Min surrogate machine learning algorithm.

catlearn.optimize.io.**print_info_neb**(*self*)
Prints the information of the surrogate model convergence at each step.

catlearn.optimize.io.**print_time**()

catlearn.optimize.io.**print_version**(*version*)

catlearn.optimize.io.**store_results_neb**(*self*)
Function that dumps the predicted discrete and interpolated M.E.P. curves in csv files for plotting

catlearn.optimize.io.**store_trajectory_neb**(*self*)

27.7 catlearn.optimize.mlneb module

class catlearn.optimize.mlneb.**ASECalc**(*gp*, *index_constraints*, *scaling_targets*, *finite_step=0.0001*, ***kwargs*)
Bases: ase.calculators.calculator.Calculator

CatLearn/ASE calculator.

calculate(*atoms=None, properties=['energy', 'forces'], system_changes=['positions', 'numbers', 'cell', 'pbc', 'initial_charges', 'initial_magmoms']*)
Do the calculation.

properties: list of str List of what needs to be calculated. Can be any combination of 'energy', 'forces', 'stress', 'dipole', 'charges', 'magmom' and 'magmoms'.

system_changes: list of str List of what has changed since last calculation. Can be any combination of these six: 'positions', 'numbers', 'cell', 'pbc', 'initial_charges' and 'initial_magmoms'.

Subclasses need to implement this, but can ignore properties and system_changes if they want. Calculated properties should be inserted into results dictionary like shown in this dummy example:

```
self.results = {'energy': 0.0,
                'forces': np.zeros((len(atoms), 3)),
                'stress': np.zeros(6),
                'dipole': np.zeros(3),
                'charges': np.zeros(len(atoms)),
                'magmom': 0.0,
                'magmoms': np.zeros(len(atoms))}
```

The subclass implementation should first call this implementation to set the atoms attribute.

implemented_properties = ['energy', 'forces']

nolabel = True

class catlearn.optimize.mlneb.**MLNEB**(*start*, *end*, *prev_calculations=None*, *n_images=0.25*, *k=None*, *interpolation='linear'*, *mic=False*, *neb_method='improvedtangent'*, *ase_calc=None*, *restart=True*, *force_consistent=None*)
Bases: object

run(*fmax=0.05*, *unc_convergence=0.05*, *steps=500*, *trajectory='ML_NEB_catlearn.traj'*, *acquisition='acq_5'*, *dt=0.025*, *ml_steps=750*, *max_step=0.25*, *sequential=False*, *full_output=False*)
Executing run will start the NEB optimization process.

Parameters

- **fmax** (*float*) – Convergence criteria (in eV/Angs).
- **unc_convergence** (*float*) – Maximum uncertainty for convergence (in eV).
- **steps** (*int*) – Maximum number of iterations in the surrogate model.
- **trajectory** (*string*) – Filename to store the output.
- **acquisition** (*string*) – Acquisition function.
- **dt** (*float*) – dt parameter for MDMin.
- **ml_steps** (*int*) – Maximum number of steps for the NEB optimization on the predicted landscape.
- **max_step** (*float*) – Early stopping criteria. Maximum uncertainty before stopping the optimization in the predicted landscape.
- **sequential** (*boolean*) – When sequential is set to True, the ML-NEB algorithm starts with only one moving image. After finding a saddle point the algorithm adds all the images selected in the MLNEB class (the total number of NEB images is defined in the 'n_images' flag).
- **full_output** (*boolean*) – Whether to print on screen the full output (True) or not (False).

Returns

Return type Minimum Energy Path from the initial to the final states.

catlearn.optimize.mlneb.**create_ml_neb**(*is_endpoint*, *fs_endpoint*, *images_interpolation*, *n_images*, *constraints*, *index_constraints*, *scaling_targets*, *iteration*, *gp=None*)

Generates input NEB for the GPR.

catlearn.optimize.mlneb.**eval_and_append**(*self*, *interesting_point*)

Evaluates the energy and forces (ASE) of the point of interest for a given atomistic structure.

Parameters

- **self** (*arrays*) – Previous information from the CatLearn optimizer.
- **interesting_point** (*ndarray*) – Atoms positions or point in space.

Returns

Return type Append function evaluation and forces values to the training set.

catlearn.optimize.mlneb.**get_energy_catlearn**(*self*, *x=None*)

Evaluates the objective function at a given point in space.

Parameters

- **self** (*arrays*) – Previous information from the CatLearn optimizer.
- **x** (*array*) – Array containing the atomic positions (flatten).

Returns **energy** – The function evaluation value.

Return type float

catlearn.optimize.mlneb.**get_fmax**(*gradients_flatten*)

Function that print a list of max. individual atom forces.

catlearn.optimize.mlneb.**get_forces_catlearn**(*self*, *x=None*)

Evaluates the forces (ASE) or the Jacobian of the objective function at a given point in space.

Parameters

- **self** (*arrays*) – Previous information from the CatLearn optimizer.
- **x** (*array*) – Atoms positions or point in space.

Returns **forces** – Forces of the atomic structure (flatten).

Return type array

catlearn.optimize.mlneb.**get_results_predicted_path**(*self*)
Obtain results from the predicted NEB.

catlearn.optimize.mlneb.**train_gp_model**(*list_train*, *list_targets*, *list_gradients*, *index_mask*, *path_distance*, *fullout=False*)
Train Gaussian process

27.8 catlearn.optimize.tools module

catlearn.optimize.tools.**plotneb**(*trajectory='ML_NEB_catlearn.traj'*, *view_path=True*)
Plot NEB path from a trajectory file containing the optimized images. This is meant to be used with ML-NEB. The error bars show the uncertainty for each image along the path.

27.9 catlearn.optimize.warnings module

27.10 Module contents

CHAPTER 28

catlearn.utilities

28.1 catlearn.utilities.clustering

Simple k-means clustering.

catlearn.utilities.clustering.**cluster_features**(*train_matrix*, *train_target*, *k=2*, *test_matrix=None*, *test_target=None*)

Function to perform k-means clustering in the feature space.

Parameters

- **train_matrix** (*list*) – Feature matrix for the training dataset.
- **train_target** (*list*) – List of target values for training data.
- **k** (*int*) – Number of clusters to divide data into.
- **test_matrix** (*list*) – Feature matrix for the test dataset.
- **test_target** (*list*) – List of target values for test data.

28.2 catlearn.utilities.database_functions

Functions to create databases storing feature matrix.

class catlearn.utilities.database_functions.**DescriptorDatabase**(*db_name='descriptor_store.sqlite'*, *table='Descriptors'*)

Bases: object

Store sets of descriptors for a given atoms object assigned a unique ID.

The descriptors for a given system can be stored in the ase.atoms object, though we typically find this method to be slower.

create_column(*new_column*)
Function to create a new column in the table.

The new column will be initialized with None values.

Parameters **new_column** (*str*) – Name of new feature or target.

create_db(*names*)
Function to setup a database storing descriptors.

Parameters **names** (*list*) – List of heading names for features and targets.

fill_db(*descriptor_names*, *data*)
Function to fill the descriptor database.

Parameters

- **descriptor_names** (*list*) – List of descriptor names for features and targets.
- **data** (*array*) – First row should contain string of UUIDs, thereafter array should contain floats corresponding to the descriptor names provided.

get_column_names()
Function to get the of a supplied table column names.

query_db(*unique_id=None*, *names=None*)
Return single row based on uuid or all rows.

Parameters

- **unique_id** (*str*) – If specified, the data corresponding to the given UUID will be returned. If None, all rows will be returned.
- **names** (*list*) – If specified, only the data corresponding to provided column names will be returned. If None, all columns will be returned.

update_descriptor(*descriptor*, *new_data*, *unique_id*)
Function to update a descriptor based on a given uuid.

Parameters

- **descriptor** (*str*) – Name of descriptor to be updated.
- **new_data** (*float*) – New value to be entered into table.
- **unique_id** (*str*) – The UUID of the entry to be updated.

class catlearn.utilities.database_functions.**FingerprintDB**(*db_name='fingerprints.db'*, *verbose=False*)
A class for accessing a temporary SQLite database.

This function works as a context manager and should be used as follows:

with FingerprintDB() as fpdb: (Perform operation here)

This syntax will automatically construct the temporary database, or access an existing one. Upon exiting the indentation, the changes to the database will be automatically commited.

create_table()
Create the database table framework used in SQLite.

This includes 3 tables: images, parameters, and fingerprints.

The images table currently stores ase_id information and a unqiue string. This can be adapted in the future to support atoms objects.

The parameters table stores a symbol (10 character maximum) for convenient reference and a description of the parameter.

The fingerprints table holds a unique image and parmeter ID along with a float value for each. The ID pair must be unique.

fingerprint_entry(*ase_id*, *param_id*, *value*)
Enter fingerprint value to database for given ase and parameter ID.

Parameters

- **ase_id** (*int*) – The ase unique ID associated with an atoms object in the database.
- **param_id** (*int or str*) – The parameter ID or symbol associated with and entry in the paramters table.
- **value** (*float*) – The value of the parameter for the atoms object.

get_fingerprints(*ase_ids*, *params=[]*)
Return values of provided parameters for each ase_id provided.

Parameters

- **ase_id** (*list*) – The ase ID(s) associated with an atoms object in the database.
- **params** (*list*) – List of symbols or int in parameters table to be selected.

Returns **fingerprint** – An array of values associated with the given parameters (a fingerprint) for each ase_id.

Return type array

get_parameters(*selection=None*, *display=False*)
Return integer values corresponding to parameter IDs.

The array returned will be for a set of provided symbols. If no selection is provided, return all symbols.

Parameters

- **selection** (*list*) – List of symbols in parameters table to be selected.
- **display** (*bool*) – If True, print parameter descriptions.

Returns **res** – Return the integer values of selected parameters.

Return type array

image_entry(*asedb_entry=None*, *identity=None*)
Enter a single ase-db image into the fingerprint database.

This table can be expanded to contain atoms objects in the future.

Parameters

- **d** (*object*) – An ase-db object which can be parsed.
- **identity** (*str*) – An identifier of the users choice.

Returns **d.id** – The ase ID colleted for the ase-db object.

Return type int

parameter_entry(*symbol=None*, *description=None*)
Function for entering unique parameters into the database.

Parameters

- **symbol** (*str*) – A unique symbol the entry can be referenced by. If None, the symbol will be the ID of the parameter as a string.
- **description** (*str*) – A description of the parameter.

28.3 catlearn.utilities.distribution

Pair distribution function.

catlearn.utilities.distribution.**pair_deviation**(*images*, *cutoffs*, *bins=33*, *bounds=None*, *mic=True*, *element=None*)

Return distribution of deviations from atom-pair nominal bond length.

Parameters

- **images** (*list*) – List of atoms objects.
- **cutoffs** (*dictionary*) – Subtract elemental cutoff radii from distances. This is a useful for testing cutoff radii.
- **bins** (*int*) – Number of bins
- **bounds** (*tuple*) – Optional upper and lower bound of distances.
- **mic** (*boolean*) – Use minimum image convention. Set to False for non-periodic structures.
- **subset** (*list*) – Optionally select a subset of atomic indices to include.

catlearn.utilities.distribution.**pair_distribution**(*images*, *bins=101*, *bounds=None*, *mic=True*, *element=None*)

Return the pair distribution function from a list of atoms objects.

Parameters

- **images** (*list*) – List of atoms objects.
- **bins** (*int*) – Number of bins
- **bounds** (*tuple*) – Optional upper and lower bound of distances.
- **mic** (*boolean*) – Use minimum image convention. Set to False for non-periodic structures.
- **subset** (*list*) – Optionally select a subset of atomic indices to include.

28.4 catlearn.utilities.neighborlist

Functions to generate the neighborlist.

catlearn.utilities.neighborlist.**ase_connectivity**(*atoms*, *cutoffs=None*, *count_bonds=True*)

Return a connectivity matrix calculated of an atoms object.

If no neighborlist or connectivity matrix is attached to the atoms object, a new one will be generated. Multiple connections are counted.

Parameters

- **atoms** (*object*) – An ase atoms object.
- **cutoffs** (*list*) – A list of cutoff radii for the atoms, ordered by atom index.

Returns **conn** – An n by n, where n is len(atoms).

Return type array

`catlearn.utilities.neighborlist.`**`ase_neighborlist`**(*atoms*, *cutoffs=None*)

Make dict of neighboring atoms using ase function.

This provides a wrapper for the ASE neighborlist generator. Currently default values are used.

Parameters

- **`atoms`** (*`object`*) – Target ase atoms object on which to get neighbor list.
- **`cutoffs`** (*`list`*) – A list of radii for each atom in atoms.
- **`rtol`** (*`float`*) – The tolerance factor to allow for small variation in the cutoff radii.

Returns **neighborlist** – A dictionary containing the atom index and each neighbor index.

Return type dict

`catlearn.utilities.neighborlist.`**`catlearn_neighborlist`**(*atoms*, *dx=None*, *max_neighbor=1*, *mic=True*)

Make dict of neighboring atoms for discrete system.

Possible to return neighbors from defined neighbor shell e.g. 1st, 2nd, 3rd by changing the neighbor number.

Parameters

- **`atoms`** (*`object`*) – Target ase atoms object on which to get neighbor list.
- **`dx`** (*`dict`*) – Buffer to calculate nearest neighbor pairs in dict format: dx = {atomic_number: buffer}.
- **`max_neighbor`** (*`int or str`*) – Maximum neighbor shell. If int is passed this will define how many shells to consider. If 'full' is passed then all neighbor combinations will be included. This might get expensive for particularly large systems.

Returns **connection_matrix** – An array of the neighbor shell each atom index is located in.

Return type array

28.5 catlearn.utilities.penalty_functions

Class with penalty functions.

`class` `catlearn.utilities.penalty_functions.`**`PenaltyFunctions`**(*targets=None*, *predictions=None*, *uncertainty=None*, *train_features=None*, *test_features=None*)

Bases: `object`

Base class for penalty functions.

`penalty_close`(*c_min_crit=100000.0*, *d_min_crit=1e-05*)

Penalize data that is too close.

Pass an array of test features and train features and returns an array of penalties due to 'too short distance' ensuring no duplicates are added.

Parameters

- **`d_min_crit`** (*`float`*) – Critical distance.

- **c_min_crit** (*float*) – Constant for penalty minimum distance.
- **penalty_min** (*array*) – Array containing the penalty to add.

penalty_far(*c_max_crit=100.0*, *d_max_crit=10.0*)
Penalize data that is too far.

Pass an array of test features and train features and returns an array of penalties due to 'too far distance'. This prevents to explore configurations that are unrealistic.

Parameters

- **d_max_crit** (*float*) – Critical distance.
- **c_max_crit** (*float*) – Constant for penalty minimum distance.
- **penalty_max** (*array*) – Array containing the penalty to add.

28.6 catlearn.utilities.sammon

Function to compute Sammon's error between original and reduced features.

catlearn.utilities.sammon.**sammons_error**(*original*, *reduced*)
Sammon error.

Parameters

- **original** (*array*) – The original feature set.
- **reduced** (*array*) – The reduced feature set.

Returns **error** – Sammon's error value.

Return type float

28.7 catlearn.utilities.utilities

Some useful utilities.

catlearn.utilities.utilities.**formal_charges**(*atoms*, *ion_number=8*, *ion_charge=-2*)
Return a list of formal charges on atoms.

Parameters

- **atoms** (*object*) – ase.Atoms object representing a chalcogenide. The default parameters are relevant for an oxide.
- **anion_number** (*int*) – atomic number of anion.
- **anion_charge** (*int*) – formal charge of anion.

Returns **all_charges** – Formal charges ordered by atomic index.

Return type list

catlearn.utilities.utilities.**geometry_hash**(*atoms*)
A hash based strictly on the geometry features of an atoms object.

Uses positions, cell, and symbols.

This is intended for planewave basis set calculations, so pbc is not considered.

Each element is sorted in the algorithem to help prevent new hashs for identical geometries.

catlearn.utilities.utilities.**holdout_set**(*data*, *fraction*, *target=None*, *seed=None*)

Return a dataset split in a hold out set and a training set.

Parameters

- **matrix** (*array*) – n by d array
- **fraction** (*float*) – fraction of data to hold out for testing.
- **target** (*list*) – optional list of targets or separate feature.
- **seed** (*float*) – optional float for reproducible splits.

catlearn.utilities.utilities.**target_correlation**(*train, target, correlation=['pearson', 'spearman', 'kendall']*)

Return the correlation of all columns of train with a target feature.

Parameters

- **train** (*array*) – n by d training data matrix.
- **target** (*list*) – target for correlation.

Returns **metric** – len(metric) by d matrix of correlation coefficients.

Return type array

CHAPTER 29

Indices and tables

- genindex
- modindex

Python Module Index

C

Index

C

D

E

F

G

H

I

K

L

M

Q

R

S

T

U

V

W

X

Z