oximachine_featurizer

Release v0.3.2+0.gc188ac6.dirty

Kevin Maik Jablonka, Daniele Ongari, Mohamad Moosavi, Berend

Apr 30, 2021

CONTENTS

oximachine_featurizer can be used to featurize MOFs (using matminer featurizers) and parse the CSD and Materials Project.

Technical details about the featurization and case studies are discussed in our preprint

Jablonka, Kevin Maik; Ongari, Daniele; Moosavi, Seyed Mohamad; Smit, Berend (2020): Using Collective Knowledge to Assign Oxidation States. ChemRxiv. Preprint. https://doi.org/10.26434/chemrxiv.11604129.v1

CHAPTER

ONE

CONTENTS

1.1 Getting started

1.1.1 Installation

We recommend installing oximachine_featurizer in a clean virtual environment environment (e.g., a conda environment) The latest stable release can be installed from the Python package index (PyPi):

```
pip install oximachine_featurizer
```

The development version can be installed directly from GitHub

```
pip install git+https://github.com/kjappelbaum/oximachine_featurizer.git
```

Some parts of the code are accelerated using just-in-time compilation (jit) using numba. This can benefit from threading layers. You can enable this using `pip install tbb`. If you do not do so, you might see warnings like `The TBB threading layer requires TBB version 2019.5 or later`.

1.1.2 Featurizing a structure

To featurize one structure with the default options you can use the following Python snippet

```
from oximachine_featurizer import featurize
X, metal_indices, metals = featurize(structure)
```

Where `structure` is a `pymatgen.Structure` object. Under the hood, this function calls two different classes, the *`GetFeatures`* class that computes all features that we considered during development and the *`FeatureCollector`* that selects the relevant ones.

Alternatively, if you want to featurize directly on the command line, you can use the following syntax

```
run_featurization <structurefile> <outname>
```

For example,

```
run_featurization examples/structures/ACODAA.cif test.npy
```

This command line tool will attempt to read the `structurefile` using pymatgen and then write the features as npy file file to `outname`. The numpy array in this file can be feed directly into the `StandardScaler` and `VotingClassifier` objects that can be created with the `learnmofox` Python package.

1.1.3 Additional tools

Scripts that are prefixed with an underscore are part of the private API and may contain hard coded paths. For example, `_run_featurization_slurm_serial.py` contains code that is specific to our cluster infrastructure.

Parsing the CSD

The *GetOxStatesCSD* can be used to retrieve the oxidation states from a list of CSD identifiers. This feature requires a CSD license and you need to export `CSD_HOME` for the CSD API to work.

You can for example use the following snippet of Python

```
from oximachine_featurizer.parse import GetOxStatesCSD
getoxstates_instance= GetOxStatesCSD(names_cleaned)

outputdict = getoxstates_instance.run_parsing(njobs=4)
```

`outputdict` will be a nested dictionary of the form `{'id': {'symbol': [oxidation states]}}`.

The `run_parsing` command line tool allows you to run the parsing for a folder of structures that are names with the CSD refcodes.

```
run_parsing <indir> <outname>
```

The output dictionary will be saved in to a pickle file with the name `outname`.

Parsing the Materials Project

Using this code requires that you export the `MP_API_KEY` environment variable containing your API key for the Materials Project. For example, the `oximachine_featurizer.run.run_mine_mp.py` script will retrieve all binary halides, sulfides, oxides, ... that are stable (zero energy above comblex hull) and calculate the oxidation states.

```
run_mine_mp
```

Will write a dataframe with the results `mp_parsing_results.csv` to the current working directory.

1.2 oximachine_featurizer API documentation

1.2.1 The featurization module

Featurization functions for the oxidation state mining project. Wrapper around matminer

class oximachine_featurizer.featurize.**FeatureCollector**(*inpath=None*, *labelpath=None*, *outdir_labels='data/labels'*, *outdir_features='data/features'*, *outdir_helper='data/helper'*, *percentage_holdout=0*, *outdir_holdout=None*, *forbidden_picklepath=None*, *exclude_dir=None*, *selected_features=['local_property_stats', 'column', 'row', 'valenceelectrons', 'diffto18electrons', 'sunfilled', 'punfilled', 'dunfilled', 'crystal_nn_fingerprint']*, *old_format=False*, *training_set_size=None*, *racsfile=None*, *selectedracs=['D_mc-I-0-all', 'D_mc-I-1-all', 'D_mc-I-2-all', 'D_mc-I-3-all', 'D_mc-S-0-all', 'D_mc-S-1-all', 'D_mc-S-2-all', 'D_mc-S-3-all', 'D_mc-T-0-all', 'D_mc-T-1-all', 'D_mc-T-2-all', 'D_mc-T-3-all', 'D_mc-Z-0-all', 'D_mc-Z-1-all', 'D_mc-Z-2-all', 'D_mc-Z-3-all', 'D_mc-chi-0-all', 'D_mc-chi-1-all', 'D_mc-chi-2-all', 'D_mc-chi-3-all', 'mc-I-0-all', 'mc-I-1-all', 'mc-I-2-all', 'mc-I-3-all', 'mc-S-0-all', 'mc-S-1-all', 'mc-S-2-all', 'mc-S-3-all', 'mc-T-0-all', 'mc-T-1-all', 'mc-T-2-all', 'mc-T-3-all', 'mc-Z-0-all', 'mc-Z-1-all', 'mc-Z-2-all', 'mc-Z-3-all', 'mc-chi-0-all', 'mc-chi-1-all', 'mc-chi-2-all', 'mc-chi-3-all']*, *drop_duplicates=True*)

Bases: object

convert features from a folder of pickle files to three pickle files for feature matrix, label vector and names list.

__init__ (*inpath=None*, *labelpath=None*, *outdir_labels='data/labels'*, *outdir_features='data/features'*, *outdir_helper='data/helper'*, *percentage_holdout=0*, *outdir_holdout=None*, *forbidden_picklepath=None*, *exclude_dir=None*, *selected_features=['local_property_stats', 'column', 'row', 'valenceelectrons', 'diffto18electrons', 'sunfilled', 'punfilled', 'dunfilled', 'crystal_nn_fingerprint']*, *old_format=False*, *training_set_size=None*, *racsfile=None*, *selectedracs=['D_mc-I-0-all', 'D_mc-I-1-all', 'D_mc-I-2-all', 'D_mc-I-3-all', 'D_mc-S-0-all', 'D_mc-S-1-all', 'D_mc-S-2-all', 'D_mc-S-3-all', 'D_mc-T-0-all', 'D_mc-T-1-all', 'D_mc-T-2-all', 'D_mc-T-3-all', 'D_mc-Z-0-all', 'D_mc-Z-1-all', 'D_mc-Z-2-all', 'D_mc-Z-3-all', 'D_mc-chi-0-all', 'D_mc-chi-1-all', 'D_mc-chi-2-all', 'D_mc-chi-3-all', 'mc-I-0-all', 'mc-I-1-all', 'mc-I-2-all', 'mc-I-3-all', 'mc-S-0-all', 'mc-S-1-all', 'mc-S-2-all', 'mc-S-3-all', 'mc-T-0-all', 'mc-T-1-all', 'mc-T-2-all', 'mc-T-3-all', 'mc-Z-0-all', 'mc-Z-1-all', 'mc-Z-2-all', 'mc-Z-3-all', 'mc-chi-0-all', 'mc-chi-1-all', 'mc-chi-2-all', 'mc-chi-3-all']*, *drop_duplicates=True*)

Initializes a feature collector.

WARNING! The fingerprint selection function assumes that the full feature vector in the pickle files has the columns as specified in FEATURE_LABELS_ALL

Keyword Arguments

- **inpath** (*Union[str, Path]*) – None)
- **labelpath** (*Union[str, Path]*) – None)
- **outdir_labels** (*Union[str, Path]*) – "data/labels")
- **outdir_features** (*Union[str, Path]*) – "data/features")
- **outdir_helper** (*Union[str, Path]) -- path to output directory for helper files (feature names, structure names*) – "data/helper")
- **percentage_holdout** (*float*) –
- **outdir_holdout** (*Union[str, Path]) -- directory into which the files for the holdout set are written (names, X and y*) –
- **forbidden_picklepath** (*Union[str, Path]*) – None)
- **exclude_dir** (*Union[str, Path]*) – None)
- **selected_features** (*List[str]*) – (default: ["crystal_nn_fingerprint","ward_prd","bond_orientational","behler_parinello"])
- **old_format** (*bool*) – {True})
- **training_set_size** (*int*) –
- **racsfile** (*str) -- path to file with RACs (pd.DataFrame saved as csv*) –
- **selectedracs** (*List[str]*) –

__weakref__
list of weak references to the object (if defined)

static create_dict_for_feature_table (*picklefile*)
Reads in a pickle with features and returns a list of dictionaries with one dictionary per metal site.

Parameters **picklefile** (*Union[str, Path]*) –

Return type List[dict]

Returns List[dict] – list of dicionary

static create_dict_for_feature_table_from_dict (*d*)
Reads in a pickle with features and returns a list of dictionaries with one dictionary per metal site.

Parameters **d** (*dict*) –

Return type List[dict]

Returns List[dict] – list of dicionary

static create_feature_list(*picklefiles*, *forbidden_list*, *old_format=True*)
Reads a list of pickle files into dictionary

Parameters

- **picklefiles** (*List[Union[str, Path]]*) –
- **forbidden_list** (*list) -- list of "forbidden" names (CSD naming convention*) – that will not be used
- **old_format** (*bool*) – "legacy" format. Default: True

Return type list

Returns list – parsed pickle contents

dump_featurecollection()
Collect features and write features, labels and names to seperate files

Return type None

static make_labels_table(*raw_labels*)
Read raw labeling output into a dictionary format that can be used to construct pd.DataFrames

Warning: assumes that each metal in the structure has the same oxidation states as it takes the first list element. Cases in which this is not fulfilled need to be filtered out earlier.

Parameters **raw_labels** (*Dict[str, dict]*) – {metal: [oxidationstates]}}

Returns , 'metal':, 'oxidationstate':}]

Return type List[dict] – list of dictionaries of the form [{'name'

class oximachine_featurizer.featurize.**GetFeatures**(*structure*, *outpath*)
Bases: object

Featurizer

__init__(*structure*, *outpath*)
Generates features for a structures

Parameters

- **structure** (*Structure*) – Pymatgen Structure object
- **outpath** (*Union[str, Path]*) – path to which the features will be dumped

Returns:

__weakref__
list of weak references to the object (if defined)

property cutoff
Chose a cutoff for a given structure

property featurizer
Return the featurizer (with the suitable cutoff)

classmethod from_file(*structurepath*, *outpath*)

Construct a featurizer class from path to structure and an output path

Parameters

- **structurepath** (*Union[str, Path]*) – Path to structure file
- **outpath** (*Union[str, Path]*) – Path to which the outputs should be written.

Returns Instance of the GetFeatures class

Return type object

classmethod from_string(*structurestring*, *outpath*)

Constructor for the webapp, using a string of a structure file, e.g., a CIF

Parameters

- **structurestring** (*str*) – Fileconent of a CIF as string
- **outpath** (*Union[str, Path]*) – Path to which the output should be written.

Raises **ValueError** – In case the CIF could not be parsed

Returns Instance of GetFeatures

Return type object

return_features()

Runs featurization and returns a list of dictionaries

Returns

List of dictionaries of the form {"metal": , "feature", [, "coords"},] i.e features for one metal site

Return type List[dict]

oximachine_featurizer.featurize.**featurize**(*structure*, *featureset=['local_property_stats', 'column', 'row', 'valenceelectrons', 'diffto18electrons', 'sunfilled', 'punfilled', 'dunfilled', 'crystal_nn_no_steinhardt']*)

Finds metals in the structure, featurizes the metal sites and collects the features

Parameters

- **structure** (*pymatgen.Structure*) – Structure to featurize
- **featureset** (*List[str]*) – Features to be used in the final output

Returns [description]

Return type Union[np.array, list, list]

oximachine_featurizer.featurize.**get_feature_names**(*selected_features*, *offset=0*)

Given a set of selected feature categories, return all feature names

Parameters

- **selected_features** (*List[str]*) – feature categories
- **offset** (*int, optional*) – To offset the feature ranges, to be used with RACs. Defaults to 0.

Returns list of feature names

Return type List[str]

1.2.2 The parsing module

Parsing functions for the oxidation state mining project

class oximachine_featurizer.parse.**GetOxStatesCSD**(*cds_ids*)

Bases: object

Main parsing class

__init__(*cds_ids*)

Parses CSD structures for oxidation states

Parameters **cds_ids** (*List[str]*) – list of CSD database identifiers

Returns None

__weakref__

list of weak references to the object (if defined)

parse_csd_entry(*database_id*)

Looks up a CSD id and runs the parsing

Parameters **database_id** (*str*) – CSD database identifier

Returns symbol - oxidation state dictionary

Return type dict

Exception:

returns empy dict upon exception (if it cannot find the structure in the database)

parse_name(*chemical_name_string*)

Takes the chemical name string from the CSD database and returns, if it finds it, a dictionary with the oxidation states for the metals

Parameters **chemical_name_string** (*str*) – full chemical name

Returns dictionary of symbol: oxidation states (list)

Return type dict

run_parsing(*njobs=4*)

Runs (concurrent) parsing over the list of database identifiers.

Parameters **njobs** (*int*) – maximum number of parallel workers

Returns

nested dictionary with {'id': {'symbol': [oxidation states]}}

Return type Dict[str, dict]

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

O

INDEX

Symbols

C

D

F

G

M

O

P

R