matminer Documentation

Release 0.1.0

Anubhav Jain

Aug 02, 2017

Contents

matminer is an open-source Python library for performing data mining and analysis in the field of Materials Science. It is meant to make accessible the application of state-of-the-art statistical and machine learning algorithms to materials science data with just a *few* lines of code. It is currently in development, however it is a **working code**.

CHAPTER 1

Citing matminer

We are currently in the process of writing a paper on matminer - we will update the citation information once it is submitted.

CHAPTER 2

Example notebooks

A few examples demonstrating some of the features available in matminer have been created in the form of Jupyter notebooks:

(Note: the Jupyter (Binder) links below are recommended as Github does not render interactive Javascript code or images.)

1. Get all experimentally measured band gaps of PbTe from Citrine's database: Jupyter Github
2. Compare and plot experimentally band gaps from Citrine with computed values from the Materials Project: Jupyter Github
3. Train and predict band gaps using matminer's tools to retrieve computed band gaps and descriptors from the Materials Project, and composition descriptors from pymatgen: Jupyter Github
4. Training and predict bulk moduli using matminer's tools to retrieve computed bulk moduli and descriptors from the Materials Project, and composition descriptors from pymatgen: Jupyter Github

You can also use the Binder service (in beta) to launch an interactive notebook upon a click. Click the button below to open the tree structure of this repository and navigate to the folder **example_notebooks** in the current working directory to use/edit the above notebooks right away! To open/run/edit other notebooks, go to "File->Open" within the page and navigate to the notebook of your choice.

CHAPTER 3

Installation

There are a couple of quick and easy ways to install matminer:-

- **Quick install**

(Note: this may not install the latest changes to matminer. To install the version with the latest commits, skip to the next steps)

For a quick install of matminer, and all of its dependencies, simply run the command in a bash terminal:

```
$ pip install matminer
```

or, to install matminer in your user $HOME folder, run the command:

```
$ pip install matminer --user
```

One way to obtain `pip` if not already installed is through `conda`, which is useful when you are working with many python packages and want to use separate configuration settings and environment for each package. You can then install matminer and packages required by it in its own environment. Some useful links are here and here.

- **Install in developmental mode**

To install the full and latest source of the matminer code in developmental mode, along with its important dependencies, clone the Git source in a folder of your choosing by entering the following command:

```
$ git clone https://github.com/hackingmaterials/matminer.git
```

and then entering the cloned repository/folder to install in developer mode:

```
$ cd matminer
$ python setup.py develop
```

Depending on how many of the required dependencies were already installed on your system, you will see a few or many warnings, but everything should be installed successfully.

CHAPTER 4

Overview

Below is a general workflow that shows the different tools and utilities available within matminer, and how they could be implemented with each other, as well as with external libraries, in your own materials data mining/analysis study.

Here's a brief description of the available tools (please find implementation examples in a dedicated section elsewhere in this document):

Data retrieval tools

- Retrieve data from the biggest materials databases, such as the Materials Project and Citrine's databases, in a Pandas dataframe format

The MPDataRetrieval and CitrineDataRetrieval classes can be used to retrieve data from the biggest open-source materials database collections of the Materials Project and Citrine Informatics, respectively, in a Pandas dataframe format. The data contained in these databases are a variety of material properties, obtained in-house or from other external databases, that are either calculated, measured from experiments, or learned from trained algorithms. The `get_dataframe` method of these classes executes the data retrieval by searching the respective database using user-specified filters, such as compound/material, property type, etc , extracting the selected data in a JSON/dictionary format through the API, parsing it and output the result to a Pandas dataframe with columns as properties/features measured or calculated and rows as data points.

For example, to compare experimental and computed band gaps of Si, one can employ the following lines of code:

```
from matminer.data_retrieval.retrieve_Citrine import CitrineDataRetrieval
from matminer.data_retrieval.retrieve_MP import MPDataRetrieval

df_citrine = CitrineDataRetrieval().get_dataframe(formula='Si', property='band gap',
                                                   data_type='EXPERIMENTAL')
df_mp = MPDataRetrieval().get_dataframe(criteria='Si', properties=['band_gap'])
```

MongoDataRetrieval is another data retrieval tool developed that allows for the parsing of any MongoDB collection (which follows a flexible JSON schema), into a Pandas dataframe that has a format similar to the output dataframe from the above data retrieval tools. The arguments of the `get_dataframe` method allow to utilize MongoDB's rich and powerful query/aggregation syntax structure. More information on customization of queries can be found in the MongoDB documentation.

Data descriptor tools

- Decorate the dataframe with composition, structural, and/or band structure descriptors/features

In this module of the matminer library, we have developed utilities to help describe the material by their composition or structure, and represent them in a numeric format such that they are readily usable as features in a data analysis study to predict a target value.

The `get_pymatgen_descriptor` function is used to encode a material's composition using tabulated elemental properties in the pymatgen library. There are about 50 attributes available in the pymatgen library for most elements in the periodic table, some of which include electronegativity, atomic numbers, atomic masses, sound velocity, boiling point, etc. The `get_pymatgen_descriptor` function takes as input a material composition and name of the desired property, and returns a list of floating point property values for each atom in that composition. This list can than be fed into a statistical function to obtain a single heuristic quantity representative of the entire composition. The following code block shows a few descriptors that can be obtained for $LiFePO_4$:

```
from matminer.descriptors.composition_features import get_pymatgen_descriptor
import numpy as np

avg_mass = np.mean(get_pymatgen_descriptor('LiFePO4', 'atomic_mass'))     # Average␣
↪atomic mass
std_num = np.std(get_pymatgen_descriptor('LiFePO4', 'Z'))     # Standard deviation of␣
↪atomic numbers
range_elect = max(get_pymatgen_descriptor('LiFePO4', 'X')) - \
           min(get_pymatgen_descriptor('LiFePO4', 'X'))      # Maximum difference in␣
↪electronegativity
```

The function `get_magpie_descriptor` operates in a similar way and obtains its data from the tables accumulated in the Magpie repository, some of which are sourced from elemental data compiled by Mathematica (more information can be found here). Some properties that don't overlap with the pymatgen library include heat capacity, enthalpy of fusion of elements at melting points, pseudopotential radii, etc.

Some other descriptors that can be obtained from matminer include:

1. Composition descriptors
   (a) Cohesive energy
   (b) Band center
2. Structural descriptors
   (a) Packing fraction
   (b) Volume per site

   (c) Radial and electronic radial distribution functions
   (d) Structural order parameters

3. Band-structure descriptors
   (a) Branch point energy
   (b) Absolute band positions
4. Mechanical properties
   (a) Thermal stress
   (b) Fracture toughness
   (c) Brittleness index
   (d) Critical stress
   (e) bulk/elastic, rigid, and shear moduli
   (f) bulk modulus from coordination number
   (g) Vicker's hardness
   (h) Lame's first parameter
   (i) p-wave modulus
   (j) Sound velocity from elastic constants
   (k) Steady-state and maximum allowed heatflow
   (l) Strain energy release rate
5. Thermal condutivity models
   (a) Cahill model
   (b) Clarke model
   (c) Callaway model
   (d) Slack model
   (e) Keyes model

Plotting tools

- Plot data from either arrays or dataframes using either Plotly or matplotlib

In the figrecipes module of the matminer library, we have developed utilities that wrap around two popular plotting libraries, Plotly and matplotlib to produce various types of plots that plot data from either arrays or dataframes. The Plotly part of this module contains classes/functions that wrap around its Python API library and follows its JSON schema. The figrecipes module is aimed at making it easy for the user to create plots from their data using just a few lines of code, utilizing the wide and flexible functionality of Plotly and matplotlib, while at the same time sheilding the complexities involved.

A few examples demonstrating usage can be found in the notebook hosted on Jupyter and Github

Changelog

v0.1.0

- Add MPDS data retrieval (E. Blokhin)
- Add partial RDF descriptor (L. Ward)
- Add local environment motif descriptors (N. Zimmermann)
- fix misc. bugs and installation issues (A. Dunn, S. Bajaj, L. Ward)

For changelog before v0.1.0, consult the git history of matminer.