matsML: a machine-learning toolkit for materials science

Huan Tran
Georgia Institute of Technology
huantd@gmail.com

September 23, 2021

Contents

1 Overview

matsML is a python-based machine-learning (ML) toolkit for some generic problems in materials science. Being initiated to support some non-polymer ML works by Dr. Huan Tran, matsML was designed to be portable and self-contained, providing necessary materials to follow the workflows and reproduce the results reported. Given this objective, scripts used for the reported works can be found in some examples of matsML while others are for tutorial purposes and beyond. matsML is free at https://github.com/huantd/matsml.git.

A typical workflow of (the very rapidly evolving field of) materials informatics includes preparing/generating/collecting suitable data, featurizing/fingerprinting the data, learning the featurized data to make models, using the developed models to make predictions, inverting the models to solve inverse problems, and more. This toolkit *does not* aim at providing complete solutions to any of these steps. However, demonstrations for most of the typical workflow can be found in the examples of matsML. For each of them, a Jupyter Notebook (as well as a pdf version) is available and should be ready to run when matsML is installed correctly.

Most of the computed data referred to in this toolkit come from works by Huan Tran, others are from open literature. In cases of experimental data that are subjected to copyright and ownership, suitable freely available alternatives are provided for demonstration purpose. When these datasets are used for publication, respective references, given in Table 1 should be cited.

The toolkit and the documentation will gradually be improved. Questions, requests, and comments are welcome at huantd@gmail.com.

2 Technical details

2.1 Code and installation

matsML is available at https://github.com/huantd/matsml.git. Dependencies needed are

1. numpy (https://numpy.org/)
2. pandas (https://pandas.pydata.org/)
3. scikit-learn (https://scikit-learn.org/stable/)
4. keras (https://keras.io/)
5. tensorflow (https://www.tensorflow.org/)
6. tensorflow_probability (https://www.tensorflow.org/probability)
7. ase (https://wiki.fysik.dtu.dk/ase/)
8. dscribe (https://singroup.github.io/dscribe/latest/index.html)
9. matplotlib (https://matplotlib.org/)

Users are suggested to (1) create a devoted environment for this toolkit, e.g., using conda with python3, (2) install the dependencies, (3) obtain matsML by

```
git clone https://github.com/huantd/matsml.git
```

and then (4) install the source code of matsML by entering the matsML folder and issuing

```
python setup.py install
```

2.2 Asummary of functionalities

1. Some datasets (see Table 1 for a summary) provided at www.matsml.org and used for the toolkit
2. Fingerprinting (1) molecules with projected Coulomb matrix and Smooth Overlap of Atomic Positions (SOAP) and (2) crystals with projected Ewald sum matrix and SOAP.
3. Building and training ML models using Kernel Ridge Regression (with scikit-learn), Gaussian Process Regression (with scikit-learn), fully connected Neural Network (with TensorFlow), and probability Neural Network (with TensorFlow-Probability).

Some information on how to use matsML will be provided in the following Sections. However, *it will not be completed*, and will gradually be augmented. Readers are recommended to look at the examples, all of them will be ready to run, given that matsML is installed correctly. All the keys and structures in the examples are easy to recognize and understand.

3 Data

Two classes of data used in matsML (and obtained from www.matsml.org) are raw data and featured data. Raw data are given in terms of atomic configurations and associated properties, thus they must be featured before learning. For some reasons, raw data are not available for some examples, and in

these cases, featured data are provided for learning step. A list of these datasets can also be obtained by the following snippet

```
from matsml.data import Datasets

data=Datasets()
data.summary()
```

A summary of the available datasets is given in Table 1.

Table 1: Datasets available at `www.matsml.org` and can be retrieved using `matsML`. SOAP, PCM, and PESM stand for Smooth Overlap of Atomic Positions, projected Coulomb Matrix, and projected Ewald Sum Matrix, respectively.

Name	Description	Format	Prop.	Size	Ref.
`molecs_CH4`	Non-equilibrium configurations of CH_4 molecules & energy computed using BigDFT	xyz	E_{DFT}	10,000	
`fp_molecs_CH4_soap`	Fingerprinted version of `molecs_CH4` using SOAP	csv.gz	E_{DFT}	10,000	
`fp_molecs_CH4_pcm`	Fingerprinted version of `molecs_CH4` using PCM	csv.gz	E_{DFT}	10,000	
`crysts_MgSi`	Equilibrium configurations of 13 Mg/Si compounds & energy computed using VASP	poscar	E_{DFT}	329	[1]
`fp_crysts_MgSi_soap`	Fingerprinted version of `crysts_MgSi` using SOAP	csv.gz	E_{DFT}	329	[1]
`fp_crysts_MgSi_pesm`	Fingerprinted version of `crysts_MgSi` using PESM	csv.gz	E_{DFT}	329	[1]
`fp_hoips_S1_1dest`	Fingerprinted data of the computed band gap of 1,346 hybrid organic-inorganic perovskites	csv.gz	$E_{\mathrm{gap}}^{\mathrm{mean}}$	192	[2]
`fp_hoips_S2a_2dest`	Fingerprinted data of the computed band gap of 1,346 hybrid organic-inorganic perovskites	csv.gz	$E_{\mathrm{gap}}^{\mathrm{mean}}$ & $E_{\mathrm{gap}}^{\mathrm{std}}$	192	[2]
`fp_hoips_S2b_1dest`	Fingerprinted data of the computed band gap of 1,346 hybrid organic-inorganic perovskites	csv.gz	$E_{\mathrm{gap}}^{\mathrm{mean}}$ & $E_{\mathrm{gap}}^{\mathrm{std}}$	384	[2]
`fp_hoips_S3_4tfp`	Fingerprinted data of the computed band gap of 1,346 hybrid organic-inorganic perovskites	csv.gz	E_{gap}	1,346	[2]
`molecs_CH3NHOH`	Non-equilibrium configurations of CH_3-NH-OH molecules & energy computed using BigDFT	xyz	E_{DFT}	10,000	
`fp_molecs_CH3NHOH_soap`	Fingerprinted version of `molecs_CH3NHOH` using SOAP	csv.gz	E_{DFT}	10,000	
`fp_molecs_CH3NHOH_pcm`	Fingerprinted version of `molecs_CH3NHOH` using PCM	csv.gz	E_{DFT}	10,000	

Given its name, a dataset can be obtained by

```
from matsml.data import Datasets

# Load a dataset
dataset_name="molecs_CH3NHOH"
data=Datasets(dataset_name=dataset_name)
data.load_dataset()
```

4 Fingerprint

In this version of `matsML`, molecules (given in xyz format) can represented in projected Coulomb Matrix (key `pcm_molecs`) and Smooth Overlap of Atomic Positions (key `soap_molecs`) while atomic crystal (given in poscar format) can represented in projected Ewald Sum Matrix (key `pesm_crystals`) and Smooth Overlap of Atomic Positions (key `pesm_crystals`). When a dataset raw dataset is obtained as described in Sec. 3, it will contain a summary file and a folder of atomic structures. Both of them are needed to featurize/fingerprint the data as given in the example below

```
from matsml.fingerprint import Fingerprint

summary=os.path.join(os.getcwd(),'molecs_CH3NHOH/summary.csv')
data_loc=os.path.join(os.getcwd(),'molecs_CH3NHOH/')
n_atoms_max=8                # max number of atoms
fp_dim=50                    # intended fingerprint dimensionality
verbosity=0                  # verbosity, 0 or 1
species=["C","H", "N", "O"]  # All the species in the datasets
# projected Coulomb Matrix
data_params_pcm={"fp_type":"pcm_molecs","summary":summary,"data_loc":data_loc,
    "n_atoms_max":n_atoms_max,"fp_file":"fp_pcm.csv","fp_dim":fp_dim,
    "species":species,"verbosity":verbosity}
fp_pcm=Fingerprint(data_params_pcm)
fp_pcm.get_fingerprint()
# Smooth Overlap of Atomic Positions
data_params_soap={"fp_type":"soap_molecs","summary":summary,
    "data_loc":data_loc,"species":species,"n_atoms_max":n_atoms_max,
    "fp_file":"fp_soap.csv","fp_dim":fp_dim,"verbosity":verbosity}
fp_soap=Fingerprint(data_params_soap)
fp_soap.get_fingerprint()
```

5 ML model development

5.1 Models supported by `matsML`

Four learning algorithms are supported by `matsML`, and they are summarized in Table 2. Given a fingerprinted data file is prepared, these models can be constructed, trained, and tested by `matsML` as

Table 2: Learning algorithms supported by `matsML`.

Model	Description	Packages used
SVecR	Support Vector Regression	scikit-learn
RFR	Random Forest Regression	scikit-learn
KRR	Kernel Ridge Regression	scikit-learn
GPR	Gaussian Process Regression	scikit-learn
FCNN	Fully-connected neural net	TensorFlow, Keras
PrFCNN	Probabilistic fully-connected neural net	TensorFlow-Probability, Keras

```
from matsml.models import FCNN

model=FCNN(data_params=data_params_pcm,model_params=model_params)
model.train()
model.plot(pdf_output=True)
```

Here, `data_params` and `model_params` are two python dictionaries, specifying all the parameters needed for the dataset and the learning model, respectively. `data_params` is the same for all the models, and a sample of this parameter set is given below

```
data_file="fp_pcm.csv"    # Fingerprinted data file
id_col=["id"]             # column for data ID
y_cols=["target"]         # columns for (one or more) target properties
comment_cols=[]           # comment columns, anything not counted into
                          # ID, fingerprints, and target
n_trains=0.8              # 80% for training, 20% for validating
sampling="random"         # method for train/test spliting
x_scaling="minmax"        # method for x scaling
y_scaling="minmax"        # method for y scaling

# Dict of data parameters
data_params_pcm={"data_file":data_file,"id_col":id_col,"y_cols":y_cols,
    "comment_cols":comment_cols,"y_scaling":y_scaling,"x_scaling":x_scaling,
    "sampling":sampling,"n_trains":n_trains}
```

On the other hand, `model_params` is model dependent. Here are samples of the model parameters for Support Vector Regression (SVecR)

```
nfold_cv=5
model_file="model.pkl"
verbosity=0
rmse_cv=False
regular_param=2
kernel="rbf"
max_iter=-1

model_params={"kernel":kernel,"nfold_cv":nfold_cv,"regular_param":regular_param,
    "max_iter":max_iter,"model_file":model_file,"verbosity":verbosity,
    "rmse_cv":rmse_cv}
```

for Random Forest Regression (RFR)

```
nfold_cv=5
model_file="model.pkl"
verbosity=0
rmse_cv=False
n_estimators=200
random_state=11
criterion="mse"
max_depth=8

model_params={"nfold_cv":nfold_cv,"n_estimators":n_estimators,
    "random_state":random_state,"criterion":criterion,"max_depth":max_depth,
    "model_file":model_file,"verbosity":verbosity,"rmse_cv":rmse_cv}
```

for Kernel Ridge Regression (KRR)

```
nfold_cv=5
model_file="model.pkl"
alpha=[-2,5]
gamma=[-2,5]
n_grids=10
kernel="rbf"

model_params={"kernel":kernel,"nfold_cv":nfold_cv,"model_file":model_file,
    "alpha":alpha,"gamma":gamma,"n_grids":n_grids}
```

for Gaussian Process Regression (GPR)

```
nfold_cv=5
model_file="model.pkl"
verbosity=0
rmse_cv=True
n_restarts_optimizer=100

model_params={"nfold_cv":nfold_cv,"n_restarts_optimizer":n_restarts_optimizer,
    "model_file":model_file,"verbosity":verbosity,"rmse_cv":rmse_cv}
```

and for Fully-Connected Neural Net (FCNN) and Probabilistic Fully-Connected Neural Net (PrFCNN)

```
layers=[8,8]                # list of nodes in hidden layers
epochs=200                  # Epochs
nfold_cv=5                  # Number of folds for cross validation
use_bias=True               # Use bias term or not
model_file="model_nn.pkl"   # Name of the model file to be created
verbosity=0                 # Verbosity, 0 or 1
batch_size=32               # Default = 32
loss="mse"
activ_funct="selu"          # Options: "tanh", "relu", and more
optimizer="nadam"           # options: "Nadam", "Adam", and more

# Dict of model parameters
model_params={"layers":layers,"activ_funct":activ_funct,"epochs":epochs,
    "nfold_cv":nfold_cv,"optimizer":optimizer,"use_bias":use_bias,
    "model_file":model_file,"loss":loss,"batch_size":batch_size,
    "verbosity":verbosity,"rmse_cv":False}
```

While the deterministic learning algorithm FCNN is widely used, the probabilistic version of FCNN is not that popular. Discussion on this method and its applicability can be found in Ref. [2] and in the `ex3_hoips` example of `matML`. For more information on how to use this toolkit, readers are recommended to explore all the examples provided. For each of them, a Jupyter Notebook and a pdf version of the Notebook with all the results are available ane ready to run.

5.2 Training models with cross validation

`matsML` always uses multi-fold cross-validation to develop models. Within this procedure, a part of the dataset, called test set, is kept *totally unseen* from the training process. In particular, the training set will be splitted into k subsets. Then, k models, each of them is trained on $k-1$ subsets and tested on the k^{th} one, identifying the best model. All the parameters/weights/coefficients of this model will be saved as the final model, and it will be tested on the *unseen* test dataset prepared at the beginning of the process. The whole cross-validation process will be shown in the standard output of `matsML`.

We note that this is one of perhaps several possible variants of cross-validation procedures, and they may be considered in the future.

References

[1] T. D. Huan, Phys. Rev. Materials **2**, 023803 (2018).

[2] V. N. Tuoc, N. T. T. Nguyen, V. Sharma, and T. D. Huan, submitted (2021).