Screening Workflows And Nanomaterials Documentation

Release 0.7.0

Felipe Zapata

Apr 19, 2023

Contents:

CHAPTER 1

Screening Workflows And Nanomaterials

Swan is a Python pacakge to create statistical models using machine learning to predict molecular properties. See Documentation.

1.1 Installation

- Download miniconda for python3: miniconda (also you can install the complete anaconda version).
- Install according to: installConda.
- Create a new virtual environment using the following commands:
  - `conda create -n swan`
- Activate the new virtual environment
  - `conda activate swan`

To exit the virtual environment type `conda deactivate`.

1.1.1 Dependencies installation

- Type in your terminal:

  ```
  conda activate swan
  ```

Using the conda environment the following packages should be installed:

- install RDKit and H5PY:
  - *conda install -y -q -c conda-forge h5py rdkit*
- install Pytorch according to this recipe
- install Pytorch_Geometric dependencies.
- install DGL using conda

1.1.2 Package installation

Finally install the package:

- Install **swan** using pip: - `pip install git+https://github.com/nlesc-nano/swan.git`

Now you are ready to use *swan*.

Notes:

- Once the libraries and the virtual environment are installed, you only need to type `conda activate swan` each time that you want to use the software.

CHAPTER 2

Tutorial to Generate Statistical Models

In this tutorial we explore how to create and train statistical models to predict molecular properties using the Pytorch library. We will use smiles to represent the molecules and use the csv file format to manipulate the molecules and their properties.

As an example, we will predict the *activity coefficient_* for a subset of carboxylic acids taken from the *GDB-13 database_*. Firstly, We randomly takes a 1000 smiles from the database and compute the *activity coefficient_* using the *COSMO approach_*. We store the values in the *thousand.csv_* file.

A peek into the file will show you something like:

```
smiles,E_solv,gammas
OC(=O)C1OC(C#C)C2NC1C=C2,-11.05439751550119,8.816417146193844
OC(=O)C1C2NC3C(=O)C2CC13O,-8.98188869016993,52.806217658944995
OC(=O)C=C(C#C)C1NC1C1CN1,-11.386853547889574,6.413128231164093
OC(=O)C1=CCCCC2CC2C#C1,-10.578966144649726,1.426566948888662
```

Where the first column contains the index of the row, the second the solvation energy and finally the *activity coefficients_* denoted as *gammas*. Once we have the data we can start exploring different statistical methods.

swan offers a thin interface to Pytorch. It takes yaml file as input and either train an statistical model or generates a prediction using a previously trained model. Let's briefly explore the *swan* input.

2.1 Simulation input

A typical *swan* input file looks like:

```
dataset_file:
  tests/test_files/thousand.csv
properties:
  - gammas

use_cuda: True

```

(continues on next page)

(continued from previous page)

```
featurizer:
  fingerprint: atompair

model:
  name: FingerprintFullyConnected
  parameters:
     input_features: 2048  # Fingerprint size
     hidden_cells: 200
     output_features: 1  # We are predicting a single property

torch_config:
  epochs: 100
  batch_size: 100
  optimizer:
    name: sgd
    lr: 0.002
```

dataset_file: A csv file with the smiles and other molecular properties.

properties: the columns names of hte csv file representing the molecular properties to fit.

featurizer: The type of transformation to apply to the smiles to generates the features. Could be either **fingerprint** or **graph**.

Have a look at the *Available models*.

2.2 Training a model

In order to run the training, run the following command:

```
modeller --mode train -i input.yml
```

swan will generate a log file called *output.log* with a timestamp for the different steps during the training. Finally, you can see in your *cwd* a folder called *swan_models* containing the parameters of your statistical model.

It is possible to restart the training procedure by providing the `--restart` option like:

```
modeller --mode train -i input.yml --restart
```

2.3 Predicting new data

To predict new data you need to provide some smiles for which you want to compute the properties of interest, in this case the *activity coefficient_*. For doing so, you need to provide in the *dataset_file* entry of the *input.yml* file the path to a csv file containing the smiles, like the *smiles.csv_*:

```
,smiles
0,OC(=O)C1CNC2C3C4CC2C1N34
1,OC(=O)C1CNC2COC1(C2)C#C
2,OC(=O)CN1CC(=C)C(C=C)C1=N
```

Then run the command:

```
modeler --mode predict -i input.yml
```

swan will look for a *swan_model.pt* file with the previously trained model and will load it.

Finally, you will find a file called “predicted.csv” with the predicted values for the activity coefficients.

CHAPTER 3

Available models

Currently **Swan** Implements the following models:

3.1 Fully Connected Neural Network

A standard fully connected neural network that takes *fingerprints* as input features. To use the model you need to specify in the `model` section of the input YAML file the following:

```
model:
  name: FingerprintFullyConnected
  parameters:
    input_features: 2048
    hidden_cells: 100
    output_features: 1
```

The model takes 3 additional optional parameters: * `input_features`: fingerprint size. Default 2048. * `hidden_cells`: Hiden number of cell(or nodes). Default 100. * `num_labels`: the amount of labels to predict. Default 1.

Also, the model requires as a `featurizer` a fingerprint calculator that can be provided like:

```
featurizer:
  fingerprint: atompair
```

Available fingerprints algorithms are: `atompair` (default), `morgan` or `torsion`. These algorithms are provided by RDKIT descriptor package.

3.2 Message Passing Neural Network

Implementation of the message passing neural network (MPNN) reported at https://arxiv.org/abs/1704.01212. If you don't have an idea what a MPNN is have a look at this introduction to Graph Neural Networks.

To train your model using the MPNN you need to provide the following section in the YAML input file:

```
model:
  name: MPNN
  parameters:
    output_channels: 10
    num_labels: 1
    batch_size: 128
    num_iterations: 3
```

The optional parameters for the model are: :: * `output_channels` Channels in the Convolution. default 10. * `num_labels`: the amount of labels to predict. Default 1. * `batch_size`: the size of the batch used to train the model. Default 128. * `num_iterations`: number of steps to interchange messages for each epoch. Default 3.

Additionally the model requires the use of the following featurizer:

```
featurizer:
 graph: molecular
 file_geometries: geometries.json
```

Where `file_geometries` is a JSON file containing an array of molecules on PDB format. Check the example file If the `file_geometries` is not set in the input the model will try to use the RDKit geometries.

CHAPTER 4

Training and validation

The training and validation functionality is implemented by the *Modeller* class.

CHAPTER 5

API Data Representation

5.1 Data Base Class

5.2 Graph Data Base Class

5.3 Fingerprints Data

5.4 Torch Geometric Data

5.5 DGL Data

CHAPTER 6

API Statistical Models

Available models

6.1 Deep Feedforward Network

6.2 Message Passing Graph Neural Network

6.3 Equivariant Neural Networks

CHAPTER 7

Indices and tables

- genindex
- modindex
- search