QSARBO

QSARBO is a hands-off quantiative structural analysis relationship (QSAR) modeling tool that can be used by researchers and academics. Some features of QSARBO are:

- Random Forest
- Artificial Neural Network
- Combinatorial QSAR
- Automatic Hyperparameter Optimization

QSARBO derives its name from the main tool behind automatic hyperparameter tuning: bayesian optimization.

New Features!

QSARBO, during the development phase of 2019-2020, has added these new features.

- Automatic machine learning using [TPOT]
- Descriptor-free modeling using [LSTM]
- Enriched documentation
- Classification features are now available for autoML
- Able to test hyperparameters for combinatorial QSAR
- Visualizations involving Williams Plot for applicability domain

Purpose

The objective of QSARBO is to provide a tool for scientists and researchers that want a *first pass* look at the impact of a chemical structure on a certain property. QSARBO **does not replace** other popular QSAR tools such as [autoQSAR] and [QSAR Toolbox]. Rather, it should be used as a *first glance* type of tool and a part of the rapid response toolkit. Based on testing, QSARBO will produce conservative estimates, meaning that even if QSARBO suggests no relationship, other tools may predict otherwise.

QSARBO also has a second objective: the ability for researchers to tweak hyperparameters and models in a template-model form. QSARBO provides four models in template form: random forest, neural network, combinatorial QSAR, and LSTM. Because QSARBO was developed with a heavy focus on one dataset, the

models may (and should be) changed based on the chemical space that the user is exploring. QSARBO may be a faster, more efficient approach for users that may want to tweak small aspects of the template without having to write the models from scratch.

One disclaimer: the developers of QSARBO decided that most users would like to see all three models (RF, NN, and combiQSAR) before testing. **Therefore, testing parameters is only functional if combiQSAR is chosen.**

Acknowledgement

The developers of QSARBO would like to thank UES for their mentorship. This package is the property of UES Inc. and Air Force Research Laboratory.

The descriptor-free modeling is taken from this paper: https://arxiv.org/abs/1712.02034

QSARBO is based on the work by PyQSAR, which contained the first version of the random forest model as well as the data wrangling and pre-processing steps.

License

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

Installation

QSARBO requires the following packages/dependencies:

- RDkit
- Mordred Calculator
- [TPOT]
- TensorFlow
- OpenBabel
- GPy, GPyOPt

All are tested at Python 3.7 in MacOS Catalina Version 10.15.4. The versions of each dependency can be found in the list provided in the QSARBO repository.

Step 1

Create a virtual Conda environment after cloning the GitHub repository.

```
$ git clone https://github.com/spencerhongcornell/QSARBayesOpt.git
$ cd QSARBO
$ conda create -n qsar
$ conda activate qsar
```

Step 2

Install dependencies.

```
$ conda install -c rdkit -c mordred-descriptor mordred
$ conda install -c openbabel openbabel=2.4.1
$ pip install -r requirements.txt
$ pip install tpot
```

Step 3

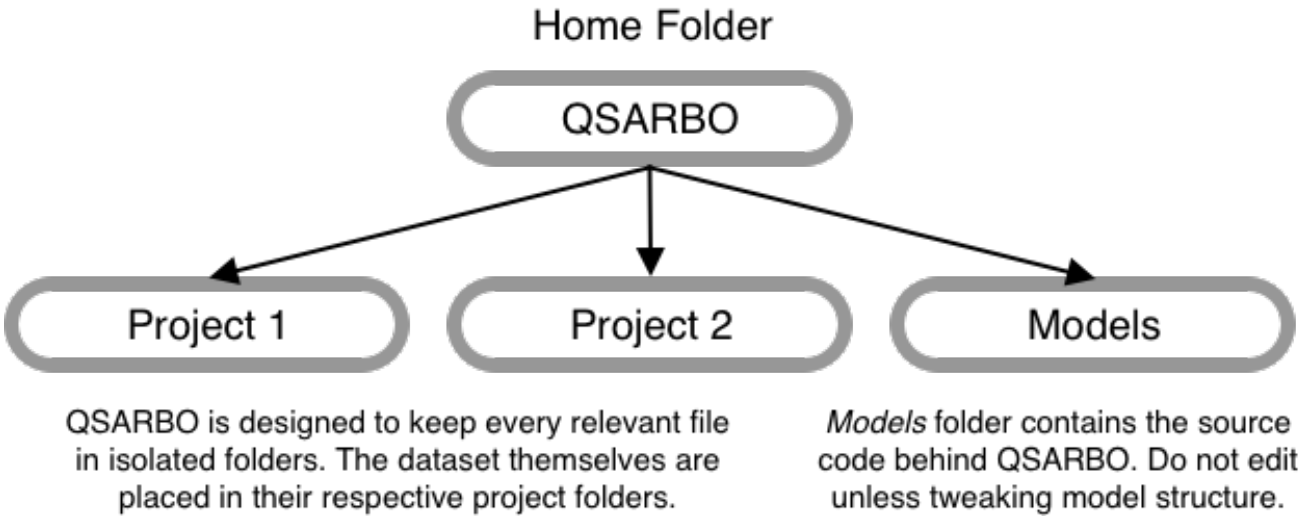
Let's pull test data files using git-lfs. Note: homebrew is required.

```
$ brew install git-lfs
$ git lfs install
$ git-lfs pull
```

You are now ready to use QSARBO.

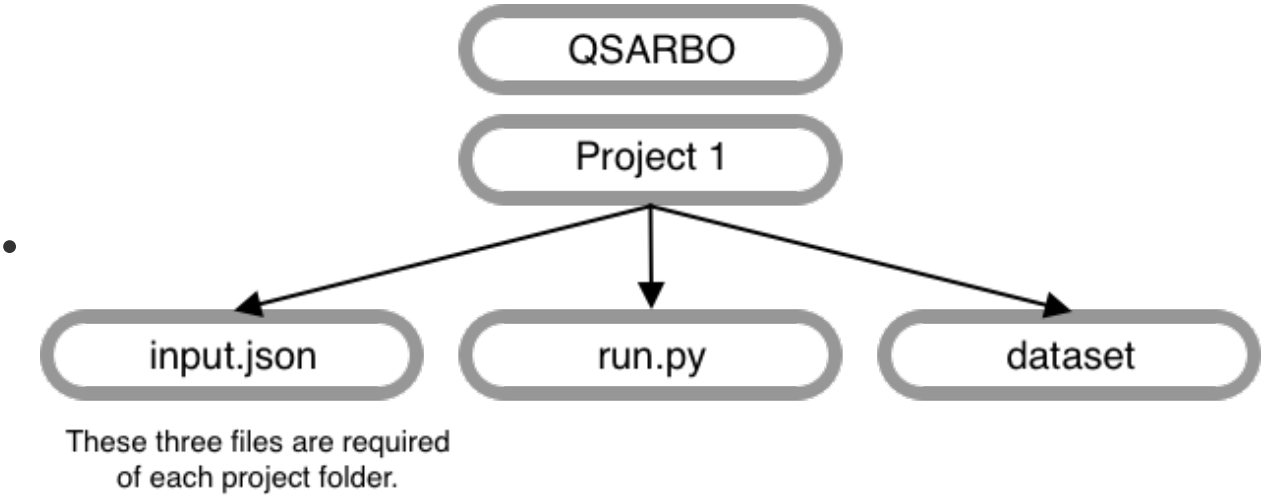
Model Hierachy

QSARBO is designed to reduce hard-coded parameters and file names as much as possible.



Three files are required for each project folder: `input.json`, `run.py`, and a dataset file (.csv preferred).

- `input.json` is the json file that contains all parameters that require editing by the user.
- `run.py` is a template run script that does not have to be changed, but must be included in every folder.
- the raw, full dataset should be included in csv format. QSARBO does not support API calls or SQL queries.



These three files can be copied from other folders into new project folders. Do not change the name of these three files.

Tutorial with Ruark Dataset (~50 minutes, excluding runtime)

This will be a tutorial and a proof-of-concept for QSARBO using the Ruark dataset. First, navigate to the QSARBO folder. As mentioned previously, we are copying the `input.json` and `run.py` from the tutorial folder, as well as the dataset.

```
$ mkdir ruark
$ cp tutorial/run.py ruark
$ cp tutorial/input.json ruark
$ cp tutorial/ruark_dataset.csv ruark
```

We now have all three required files in our `ruark` folder. Let's now explore the `input.json` file. The file contains two fields for each question: "_description" and "content". Description field describes how to choose the value for the content field (case-sensitive!). We summarize these parameters in the table below.

Parameters in `input.json`	**Specifics**
classification or regression?	Type "regression" or "classification".
test or train?	Type "1" for training, and "2" for testing. You cannot choose testing without having run training at least once.

which_models?	Type "1" to only train random forest (RF) models. Type "2" to train only neural network (NN) models. Type "3" for a combinatorial QSAR model, combining both RF and NN.
developer_mode?	This is an artifact of QSARBO development team. Multiple runs of each model are recorded with individual runtimes and accuracies. **Recommended to type "0" for most situations.**
autoML?	Type "1" to run the models with TPOT, an autoML package. Type "0" to ignore autoML. If autoML is chosen, all models, not just RF/NN, will be tested. This supersedes the "which_models?" question.
column_SMILES	Type (case-sensitive) the name of the column in the dataset that contains the SMILES.
column_activity	Type (case-sensitive) the name of the column in the dataset that contains the activity you wish to predict. For classification, only values of 0's and 1's are accepted.
folder_name	Type (case-sensitive) the name of the project folder this `input.json` and the dataset are located within.
dataset_name	Type (case-sensitive) the name of the dataset with the .csv ending.
chemID	Type (case-sensitive) the name of the column that you wish to keep in the prediction phase for future identification of the chemical (i.e. CAS #). **Recommendation: this feature is deprecated for all non-RF models. Please put "NA" for now.** Please identify all chemicals by SMILES.
num_cores	Type the number of cores you wish to dedicate to this project during modeling. Most scripts are parallelized.
elements_kept	List, in the form of element numbers, the elements that you would like to keep in the dataset for study. **Recommended: QSARBO does not support certain**

	elements due to the poor support by some of the dependencies. For most QSAR studies, keep the list to 1, 6, 7, 8, 9, 15, 16, 17, and 35.
correlation_threshold	A float that marks the threshold for removing highly correlated features during data cleaning. Adapted from https://chrisalbon.com/machine_learning/feature_selection/drop_highly_correlated_features/. **Recommended: 0.95**.
std_threshold	A float that marks the threshold value for removing low standard deviated features. Ranges from 0 to 1. **Recommended: 0.25**.
valid_split	A decimal value from 0 to 1 that marks the percentage of the **whole** dataset to be treated as the validation dataset.
cvfolds?	Type the number of cross-validation folds you would like to achieve.
test_split	A decimal value from 0 to 1 that marks the percentage of the **training** dataset to be treated as the testing dataset (internal validation).
saved_descriptors?	The model saves previous descriptors in between runs (within the same project). There is no need to recalculate the descriptors. **Recommended: type "True" if this is not the first time you have run the code in this project. Type "False" if this is the first time you have run the code.**
saved_hyperparameters?	Type "True" if you have run the code in this project before. Type "False" if you have not.

We will first check regression of Ruark dataset with both RF and NN models. The `input.json` copied from the tutorial folder will have the correct parameters typed-in already for your benefit, but make sure:

- the "classification or regression?" question is marked "regression"
- the "test or train?" question is marked "1"
- the "which models?" question is marked "3"

Training New Model

Notice that the folder name being `ruark` was intentional as it is a typed parameter in the `input.json` file. After verifying that these fields are in, you may type:

```
$ python run.py
```

inside the `ruark` folder.

If the following error persists:

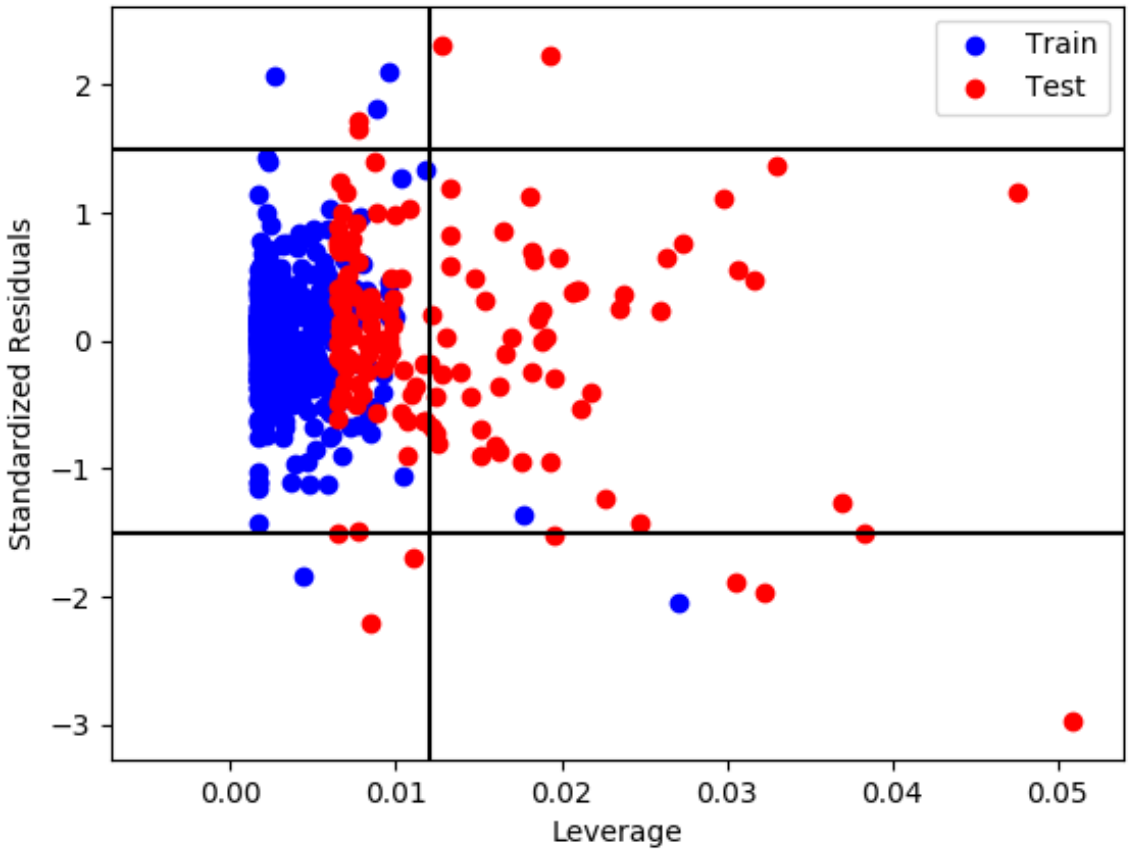
> *RuntimeError: Python is not installed as a framework. The Mac OS X backend will not be able to function >correctly if Python is not installed as a framework. See the Python documentation for more information on >installing Python as a framework on Mac OS X. Please either reinstall Python as a framework, or try one of >the other backends. If you are using (Ana)Conda please install python.app and replace the use of 'python' >with 'pythonw'. See 'Working with Matplotlib on OSX' in the Matplotlib FAQ for more information.*

Then, follow the solution in this link: https://stackoverflow.com/questions/21784641/installation-issue-with-matplotlib-python.

After the run finishes, the final output should be a list of four numbers. The numbers mean the following in this order: time taken for the run (in seconds), the training set score, the testing set score, and the validation set score. Furthermore, in the `/visualizations` folder, there is a Williams plot to calculate the applicable domain. In the `/predictions` folder, the predicted and the true values can be compared for both the testing and the validation datasets.

Visualizations and Results

QSARBO automatically saves key visualizations and datapoints that can be used to make other graphics of your choice. In the `/visualizations` folder, two files should exist: `rf0.png` and `nn0.png`. These are the Williams Plot visualizations.



Furthermore, in the `/predictions` folder, two .csv files exist: one that contains the comparisons between the predicted and actual values for the testing dataset and one that contains the comparisons between the predicted and actual values for the validation dataset. Use these .csv files to create graphs in third-party.

Testing New Chemicals

Now that we've trained the models, we would like to use the trained models to test new, unknown chemicals. To do so, follow these steps:

1. Place the dataset that contains the unknown chemical SMILES in the same project folder. **In the tutorial, the dataset that contains unknown, dummy chemicals is called `tobetested.csv`.**
2. Edit the `input.json` file so that the dataset name now reflects this new dataset that contains the unknown SMILES, **not the original dataset name that wass used for training the model.**
3. Edit the `input.json` file for the appropriate column names. Type "2" for the "test or train?" question.

Then, type the following in your terminal (`run.py` is not changed):

```
$ python run.py
```

After it finishes running, there should be a file in `/predictions` called `unknowntested.csv`. The results from testing the unknown chemicals are located in that file.

Tutorial for Classification

To run classification, very similar approach is used. In the `/tutorial` folder, Follow the approach for regression, but the following parameters are different in the `input.json` file.

- For the "classification or regression>" question, the answer is "classification"
- Again, please type "1" for the "test or train"
- Please choose combiQSAR, as it will test for both RF and NN models
- For the tutorial, the "column_activity" field is the "logki-class"
- Please change the folder name if it has been changed

Then, type

```
$ python run.py
```

Similar to regression, results are found in `/predictions` folder. One thing to note here: **Because we are doing consensus voting with only two models, the combiQSAR results in the `combinations.csv` results indicate whether or not the two models agreed (in either marking it True or False) on its results. The decidability becomes clearer on the results that have "1" in this column.**

Tutorial for Automatic Machine Learning

To run the autoML (will test most machine learning models in our pipeline), please do the following:

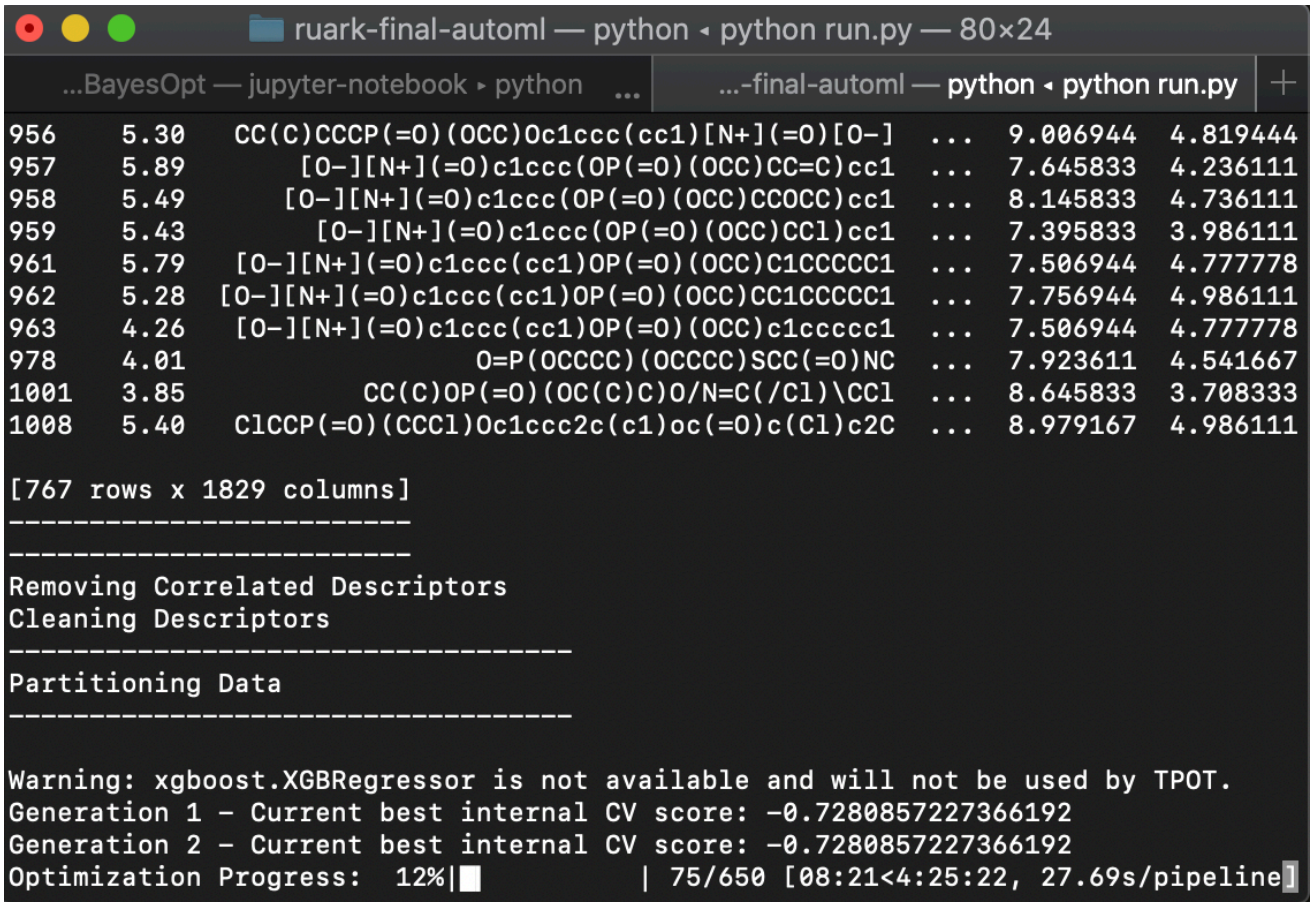
- Type "1" for the "autoML?" question
- Rest of the fields may stay the same as the previous tutorial. Please create a new folder for the autoML project and indiate the folder name in the `input.json`.
- autoML can do either classification or regression. Please mark accordingly in the json file.

This is taken from TPOT:

> *AutoML algorithms can take a long time to finish their search.*

> *AutoML algorithms aren't as simple as fitting one model on the dataset; they are considering multiple machine learning algorithms (random forests, linear models, SVMs, etc.) in a pipeline with multiple preprocessing steps (missing value imputation, scaling, PCA, feature selection, etc.), the hyperparameters for all of the models and preprocessing steps, as well as multiple ways to ensemble or stack the algorithms within the pipeline.*

Status of the autoML finding the best pipeline is shown as an output on the terminal screen. **The results are placed in the `/predictions` folder. Also, the best performing pipeline is saved as `tpot_regression.py` or `tpot_classification.py`.**



Disclaimer: the Ruark dataset takes about > 24 hours for the autoML regression.

LSTM Modeling Tutorial

LSTM is a class of deep learning technique that do not require traditional QSAR features that are derived from the data itself. LSTM can be utilized to extract features from the structure itself. The proof-of-concept was published in the paper provided: https://www.frontiersin.org/articles/10.3389/frai.2019.00017/full

LSTM is not yet incorporated within the QSARBO library, but the regression predictive model can still be used using the script `lstm.py`. Instead of using the `input.json` file, we will be using arguments directly in the terminal. Inside the project folder that contains the dataset for LSTM regression, type:

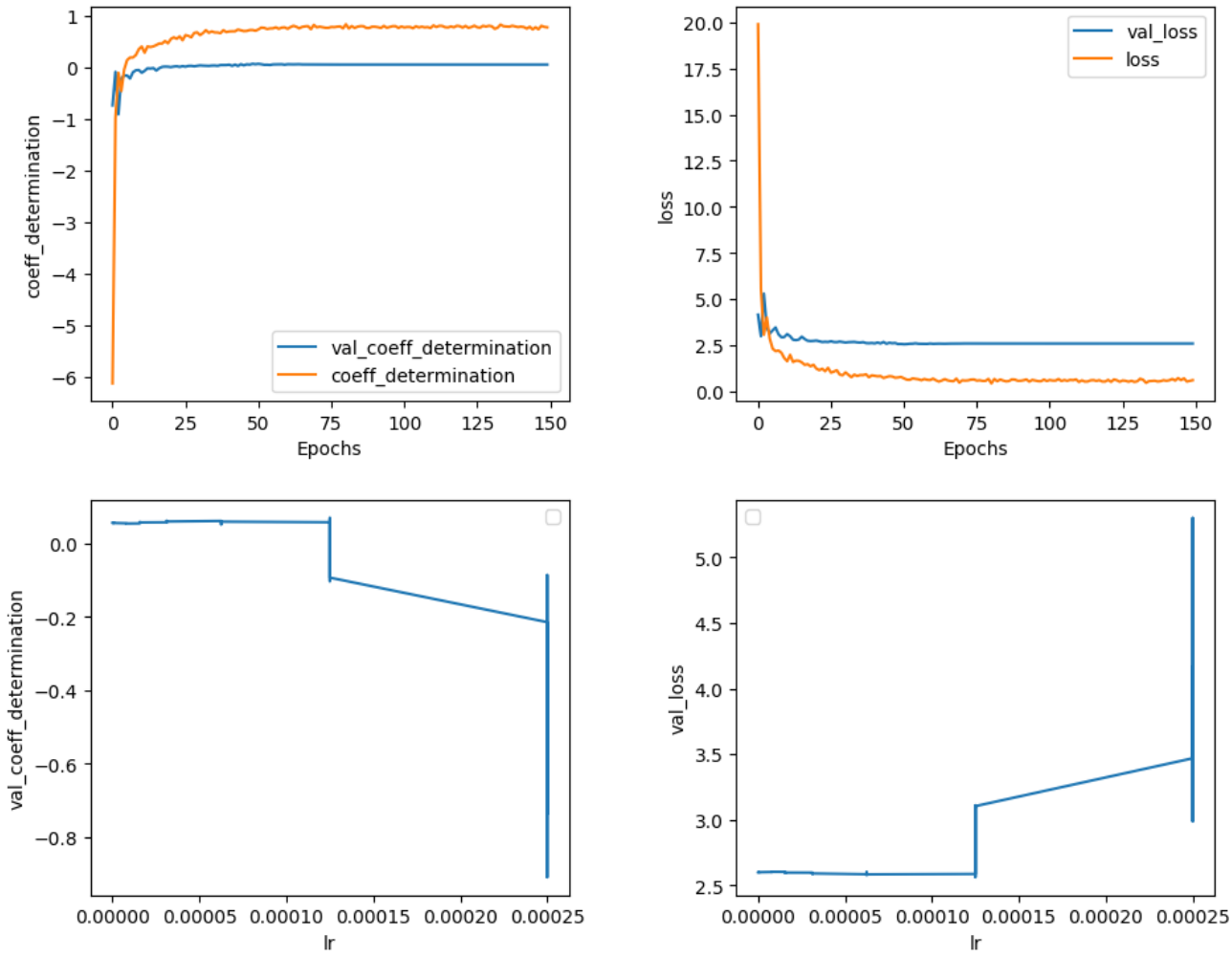
```
$ python [name of dataset in csv] [name of the column containing
activity] [name of the column containing SMILES]
```

For example, for the tutorial, we will use:

```
$ python lstm.py Ruark_OPData.csv logki SMILES
```

The dataset contains a field called "SPLIT" where 1 marks the testing set and 0 marks the training set. This is a required field in the dataset.

Results can be found in `lstm_results.csv` as well as a visualization plot in `results.png`.



Tweaking Models

The random forest model and the neural network models can be found in two places: `models/runner_rf.py`, `models/runner_nn.py` and `models/classes/nn.py` for regression. For classification, choose `models/classes/nnc.py`. The model structure in this code can be changed, either by changing the layer counts, the activation functions, or the optimizers.