Metamodeling Framework Documentation

Release 0.1.0

Nicholas Long

Sep 19, 2019

CONTENTS:

The metamodeling framework was created to build models to use for estimating commercial building energy loads. The framework currently supports linear models, random forests, and support vector regressions. The framework handles the building, evalulation, and validation of the models. During each set of the process, the framework exports diagnostic data for the user to evaluate the performance of the reduced order models. In addition to building, evaluating, and validating the reduced order models, the framework is able to load previously persisted model to be used in third-party applications (e.g. Modelica).

The project was initially developed focusing on evaluating ambient loop district heating and cooling systems. As a result, there are several hard coded methods designed to evaluate and validate building energy modeling data. These are planned to be removed and made more generic in the coming months.

This documentation will discuss how to inspect, build, evaluate, and validate a simple dataset focused on commercial building energy consumption. The documentation will also demonstrate how to load and run an already built metamodel to be used to approximate building energy loads.

CHAPTER

ONE

INSTRUCTIONS

The framework requires Python 3. After installing Python and configuring Python 3, the framework can be installed from source code (recommended) or from PyPI.

CHAPTER

TWO

INSTALLATION FROM SOURCE

1) Install Python and pip
2) Clone this repository
3) Install the Python dependencies

```
pip install -r requirements.txt
```

4) (Optional) install graphviz to visualize decision trees
   - OSX: `brew install graphviz`

CHAPTER

THREE

BUILDING EXAMPLE MODELS

A small office example has been included with the source code under the tests directory. The small office includes 3,300 hourly samples of building energy consumption with several characteristics for each sample. The example shown here is only the basics, for further instructions view the complete documentation on readthedocs.

```
./meta-runner build -f metamodeling/tests/smoff_test/metamodels.json -a␣
↪smoff_test
./meta-runner evaluate -f metamodeling/tests/smoff_test/metamodels.json -a␣
↪smoff_test
./meta-runner validate -f metamodeling/tests/smoff_test/metamodels.json -a␣
↪smoff_test
```

CHAPTER

FOUR

INSTALLATION FROM PYPI

Not yet complete.

CHAPTER FIVE

EXAMPLE REPOSITORY

An example repository was developed using the Metamodeling Framework to evaluate the results of OpenStudio using PAT. There are several repositories to generate the datasets; however, the first link below contains a basic dataset in order to demonstrate the functionality of the Metamodeling Framework.

- Ambient Loop Metamodels

The two repositories below were used to generate the OpenStudio/EnergyPlus models used for the Metamodeling FRamework.

- OpenStudio's Parametric Analysis Tool Projects
- OpenStudio Measures

CHAPTER

SIX

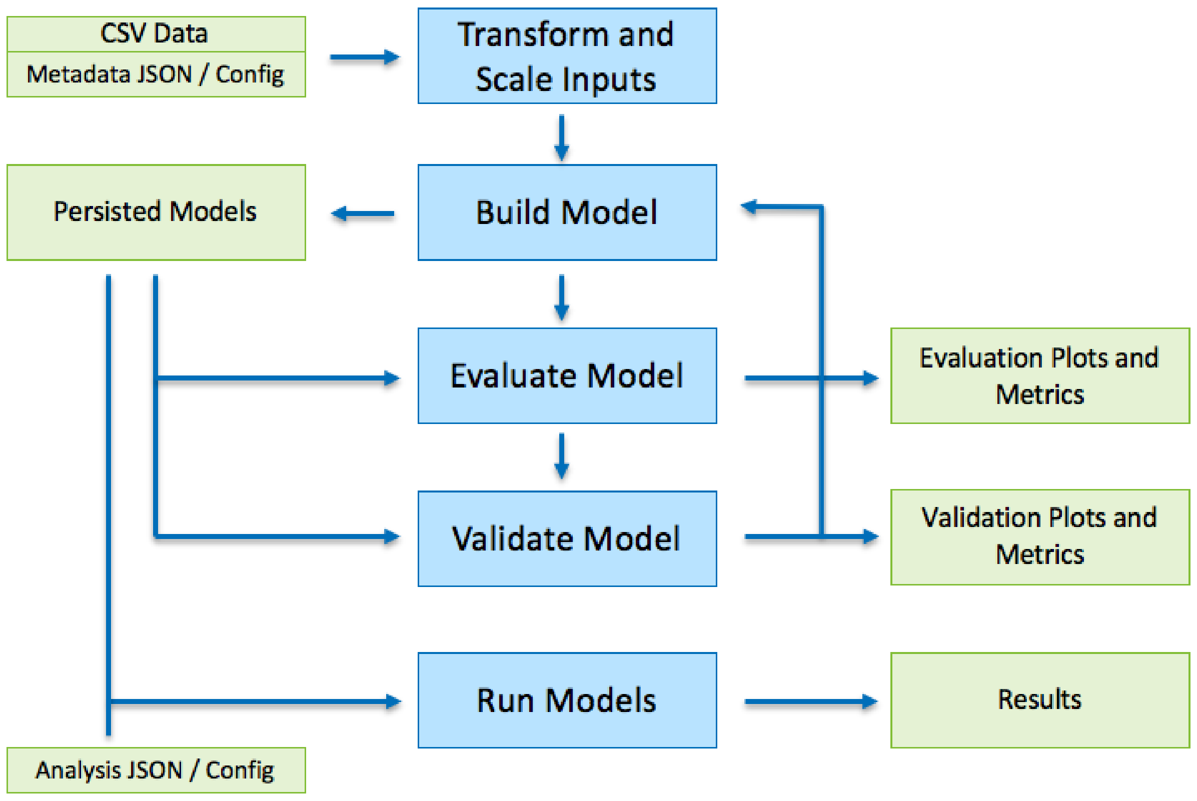
TO DOS

- Configure better CLI
- Allow for CLI to save results in specific location
- Remove downloaded simulation data from repository
- Write test for running the analysis_definition (currently untested!)

6.1 Getting Started

The Metamodeling Framework is designed to help users build, evaluate, validate, and run reduced order models. The image below shows the typical workflow and the required data. Each of the blue boxes represent a process and the green boxes represent either an input dataset or a output data.

In order to run the build method, the user must supply the data in CSV format with an accompanying JSON file which describes a) the build options, b) the response variables, and c) the covariates. An explanation and example of how the metadata JSON config file looks is shown in *example metadata json file*.



The four main functions of the meta-runner.py file includes:

1) Inspect

   Load the results dataframe and create a resulting dataframe (and CSV) describing the data. This is useful when determining what is in the dataframe and what the covariates and responses should be. This also calculates the means for all the variables which can be used to set as the default values for when running a parametric sweep with the resulting metamodel that is generated.

   Each model that is generated will create the statistics summary in the data directory.

```
-f FILE, --file FILE  Metadata file to use
-a ANALYSIS_MONIKER, --analysis-moniker ANALYSIS_MONIKER
                      Name of the Analysis Model
-m [{LinearModel,RandomForest,SVR}], --model-type [{LinearModel,
↪RandomForest,SVR}]
                      Type of model to build
-d DOWNSAMPLE, --downsample DOWNSAMPLE
                      Specific down sample value
```

```
./meta-runner inspect -a smoff_test
```

2) Build

   Use the build positional argument to build a new reduced order model as defined in the metamodels.json file. There are several arguments that can be passed with the build command including:

```
-f FILE, --file FILE  Metadata file to use
-a ANALYSIS_MONIKER, --analysis-moniker ANALYSIS_MONIKER
```

(continues on next page)

(continued from previous page)

```
                      Name of the Analysis Model
-m [{LinearModel,RandomForest,SVR}], --model-type [{LinearModel,
↪RandomForest,SVR}]
                      Type of model to build
-d DOWNSAMPLE, --downsample DOWNSAMPLE
                      Specific down sample value
```

```
./meta-runner build -a smoff_test
```

3) Evaluate

   Use the build positional argument to build a new reduced order model as defined in the metamodels.json file. There are several arguments that can be passed with the build command including:

```
-f FILE, --file FILE  Metadata file to use
-a ANALYSIS_MONIKER, --analysis-moniker ANALYSIS_MONIKER
                      Name of the Analysis Model
-m [{LinearModel,RandomForest,SVR}], --model-type [{LinearModel,
↪RandomForest,SVR}]
                      Type of model to build
-d DOWNSAMPLE, --downsample DOWNSAMPLE
                      Specific down sample value
```

```
./meta-runner evaluate -a smoff_test
```

4) Validate

   Use the build positional argument to build a new reduced order model as defined in the metamodels.json file. There are several arguments that can be passed with the build command including:

```
-f FILE, --file FILE  Metadata file to use
-a ANALYSIS_MONIKER, --analysis-moniker ANALYSIS_MONIKER
                      Name of the Analysis Model
-m [{LinearModel,RandomForest,SVR}], --model-type [{LinearModel,
↪RandomForest,SVR}]
                      Type of model to build
-d DOWNSAMPLE, --downsample DOWNSAMPLE
                      Specific down sample value
```

```
./meta-runner validate -a smoff_test
```

5) Run

   Use the build positional argument to build a new reduced order model as defined in the metamodels.json file. There are several arguments that can be passed with the build command including:

```
-ad ANALYSIS_DEFINITION, --analysis-definition ANALYSIS_DEFINITION
                        Definition of an analysis to run using the␣
↪Metamodels
-w WEATHER, --weather WEATHER
                        Weather file to run analysis-definition
-o OUTPUT, --output OUTPUT
                        File to save the results to
```

```
./meta-runner.py run -a smoff_parametric_sweep -m RandomForest -ad␣
↪examples/smoff-one-year.json -w examples/lib/USA_CO_Golden-NREL.724666_
↪TMY3.epw -d 0.15 -o output.csv
```

(continues on next page)

(continued from previous page)

6.2 Metadata Definition File

The JSON file shown below is meant to serve as documentation by example for the metamodel definition JSON file.

6.3 Analysis Definition

6.3.1 Static

This example configuration file shows setting all the covariates to a single value.

6.3.2 EPW Data Source

This example configuration file shows reading data from an EPW file for the metamodel covariates.

6.3.3 Single

This example configuration file shows sweeping over a single variable range.

6.3.4 Multiple

This example configuration file shows sweeping over multiple variable ranges. In this case the total number of samples will result in the full combinatorial of all the values in this file.

6.3.5 Code

Analysis Definition

Parser for analysis definition JSON files.

class `metamodeling.analysis_definition.analysis_definition.`**`AnalysisDefinition`**(*definition_file*)

Bases: `object`

Pass in a definition file and a weather file to generate distributions of models

`as_dataframe`()

Return the dataframe with all the data needed to run the analysis defined in the json file.

Note that the first field in the analysis definition json file must be a value or an EPW.

Returns pandas dataframe

`load_files`(*definition_file*)

`load_weather_file`(*weather_file*)

Load in the weather file and convert the field names to what is expected in the JSON file :return:

EPW File

Process an EPW file

class `metamodeling.analysis_definition.epw_file.`**`EpwFile`**(*filepath*)

Bases: `object`

`as_dataframe`()

Return the EPW file as a dataframe. This drops the data_source column for brevity.

Returns pandas DataFrame

`post_process_data`()

Add in derived columns

Returns

The analysis definition module is used for loading an already generated reduced order and running a subsequent analysis. The input is a JSON file that defines each of the covariates of interest. The analysis can take of

- Single value analysis, see *example*
- Sweep values over a year (as defined by an EPW file), see *example*
- Sweep values over specified ranges for single variable, see *example*
- Sweep values over specified ranges for multiple variable, see *example*

To run an analysis with a JSON file, first load a metamodel, then load the analysis defintion.

```
from metamodeling.analysis_definition.analysis_definition import AnalysisDefinition
from metamodeling.metamodels import Metamodels
```

6.4 Using the ROMs

6.4.1 CLI Example

This example CLI file shows how a simple application can be built that reads in all the values from the command line and reports the values of the responses passed.

```
# Single response - with only setting the inlet temperature
python analysis_cli_ex1.py -f smoff/metamodels.json -i 18

# Multiple responses - with only setting the inlet temperature
python analysis_cli_ex1.py -f smoff/metamodels.json -i 18 -r HeatingElectricity␣
↪DistrictHeatingHotWaterEnergy
```

6.4.2 Analysis Example

Example analysis script demonstrating how to programatically load and run already persisted reduced order models. This example loads two response variables (models) from the small office random forest reduced order models. The loaded models are then passed through the swee-temp-test.json analysis definition file. The analysis definition has few fixed covariates and a few covariates with multiple values to run.

```
python analysis_ex1.py
```

6.4.3 Sweep Example

Example analysis script demonstrating how to programatically load and run already persisted reduced order models using a weather file. This example is very similar to the analysis_ex1.py excpet for the analysis.load_weather_file method. This method and the smoff-one-year.json file together specify how to parse the weather file.

The second part of this script using seaborn to generate heatmaps of the two responses of interest. The plots are stored in a child directory. Run the example by calling the following:

```
python analysis_sweep_ex1.py
```

6.4.4 Modelica Example

This example file shows how to load the models using a method based approach for use in Modelica. The run_model takes only a list of numbers (int and floats). The categorical variables are converted as needed in order to correctly populate the list of covariates in the dataframe.

For use in Modelica ake sure that the python path is set, such as by running export PYTHONPATH='pwd'

Call the following bash command shown below to run this example. This example runs as an entrypoint; however, when connected to modelica the def run_model will be called directory. Also, note that the run_model method loads the models every time it is called. This is non-ideal when using this code in a timestep by timestep simulation. Work needs to be done to determine how to load the reduced order models only once and call the reduced order model yhat methods each timestep.

```
python analysis_modelica_ex1.py
```

6.5 Developer Notes

6.5.1 Building Documentation

```
$ sphinx-apidoc -o docs/source/modules . metamodeling
$ cd docs
$ make html
```

6.6 Code Documentation

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

INDEX