Appliance Analysis and Modification Toolkit

Johannes Hölzl

May 20, 2015

Abstract

For trying out newly developed algorithms, it is important to have some kind of appliance set or house to test it. Hence, a tool for creating a simulated house is needed.

Contents

1 Introduction

The Appliance Analysis and Modification Toolkit is web based application, running on a tornado webserver.

2 Usage

2.1 Starting the Service

To start the server, run the *server.py* script. In case of errors, make shure that every required external module has been installed. Also use *Python 3* as this project was also written for this version.

Using the web browser, it is now possible to access program.

2.2 Select Datasets

On the left-most side it is possible to filter appliances by datasets. There can be more chosen datasets than just one (use of checkboxes and not radio buttons). The affected types of appliances are now shown in the second column.

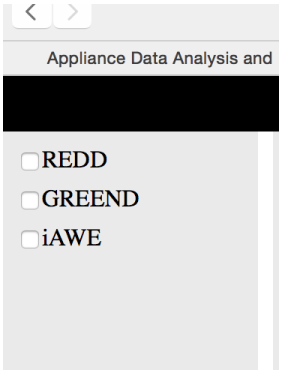

Figure 1: Use this Checkboxes to use specific datasets.

2.3 Select Appliances

There are three relative sizes of appliances: small, medium and large.

The green *Add* button appends an instance of this appliance to the list of chosen appliances. This is only possible after the size has been selected.

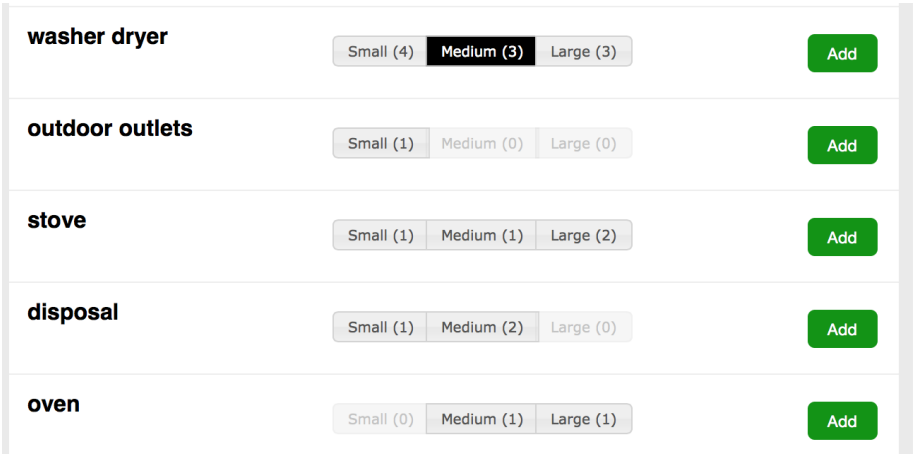

Figure 2: Available appliances.

To find appliances easier a search function was implemented. Also sorting by name could be easily done by clicking the *Sort by name* button.

Figure 3: Search available appliances.

2.4 Chosen Appliances

The third column shows the selected appliances. It is possible to have multiple appliance of the same kind in this list. The size can be changed even at this point. The red *Remove* button can be used to remove the particular appliance from the list of chosen appliances.

While hovering over the *Edit* field, a menu is shown. The Resampling has to be entered in a *pandas* interval format. Default is at *1S*. Leaving this textfield empty performs no resampling at all. The second input element is responsible for the kind of the filling method used. Lastly, the size of the median filter could be entered. Similar as on the first text box, no input results in an absence of the median filter.

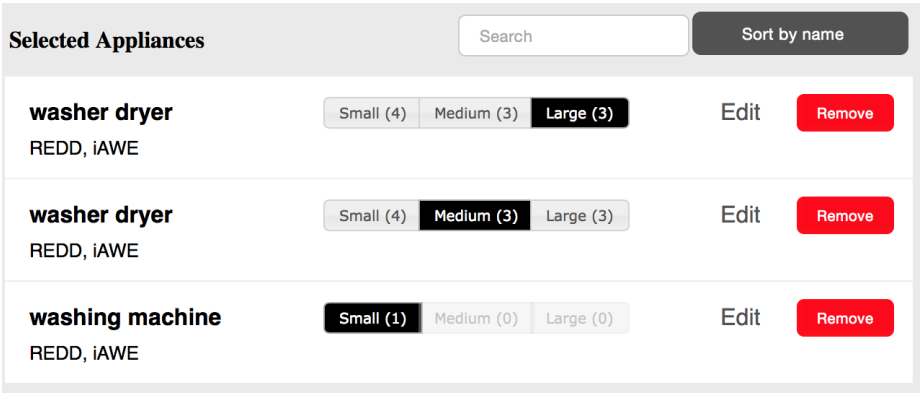

Figure 4: List of selected appliances.

Figure 5: Edit a selected appliance.

2.5 Final Preferences

The final column features some options which affect the whole dataset. The entered value for noise should represent the amplitude of the noise itself in Watts. The value for the missing data interpreted in such a way, that in average every moment, which position can be diveded by the entered number will be deleted. E.g. the entered number is ten, in average every tenth entry will be deleted from the final result.

After that, the time interval can be selected. There a presets like one hour, one day and one week. Another time interval should be given in the *pandas* conform interval notation. It is also possible to calculate the total complexity of the final house. This will take some time to calculate. The *Generate* button triggers the generation of the result. The current state of this is shown in the gray box below and the progress bar above it. After the result has been successfully generated, it can be downloaded as a *csv* file, in which the first column represents the timestamp, the last column shows the total power draw at this moment and the ones in between show the power values of the single appliances.

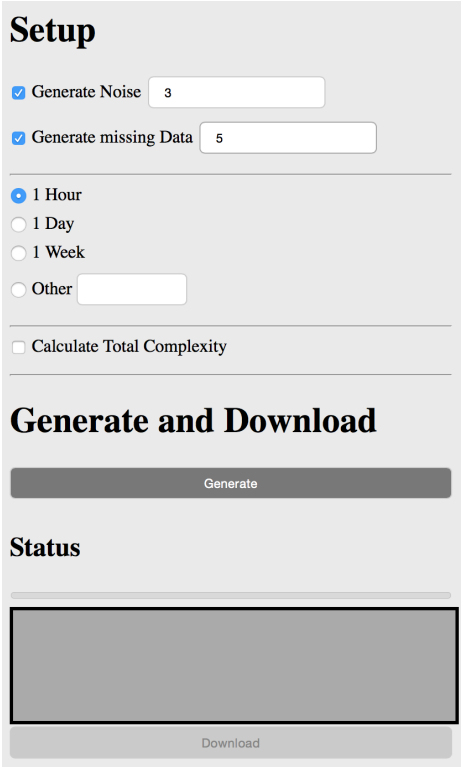

Figure 6: Final settings.

A small part of the resulting file is shown below.

```
1061604040.0, 0.000000, 0.000000, 237.061000, 238.383951
1061604041.0, 0.000000, 0.000000, 237.061000, 235.818284
1061604043.0, 0.000000, 0.000000, 237.061000, 240.043915
1061604044.0, 0.000000, 0.000000, 237.061000, 234.733919
1061604045.0, 0.000000, 0.000000, 237.061000, 235.399529
```

3 Appliance Features

3.1 State Reckoning

A state is recognized as one if the absolute power difference of two consecutive power values is greater than a given threshold.

3.2 Getting Appliance States by Edge Detection

A more advanced technique for getting the states of an appliance is by using edges in the course of the power consumed by the appliance. If there is an edge

with an amplitude larger than a predefined value this state will be appended to the list of already known states.

The edge detection is used for gathering the states which are then used to calculate the complexity of the dataset.

4 Resampling

The *Pandas* module is used for the resampling process.

5 Filling Missing

Also for filling in and calculating missing data the *Pandas* module is used. See the documentation of this project for more information.

6 Occurance per day

It is possible to gather information about how many and which states an appliance has per day. This data can be obtained using the methods *get_states_per_day* and *get_count_per_day*. These return a dict with the date as keys and the count or a list of states as values. They are both using the private method *__get_occurrence_per_day*.

7 Python Classes

The Python part of this project mainly consists of classes which are directly responsible for web communication.

7.1 GenerateOrderHandler

This class represents a handler which is mainly used for calling the *GenerateOrder*-class. Before it does so, it reads all necessary values and passes them to the *GenerateOrder*-class.

7.2 GenerateOrder

This class is responsible for generating the fake house. It fetches data from predefined paths (*REDD*, *GREEND*, *iAWE*) and also generates missing data using the method `add_missing()`. Noise is generated using the `add_noise()`-method. For calculating the complexity of this scenario, the class provides the

required method (`calc_complexity()`). Finally, when the all the processing is finished, the result is exported to a *.csv* file using the method `write_csv()`.

7.3 StatusBuffer

The *StatusBuffer*-class provides ressources for managing currently connected clients. When an update is available, an object of this class sets an result for the `Future` objects of the particular objects, so they know an update for the client is due.

7.4 UpdateStatusHandler

For handling and supplying information to the status field of the client, the *UpdateStatusHandler*-class is used. It does that using long-polling. The content of the update dict is not important at this point.

7.5 CheckboxesAllHandler

When clicking checkboxes at the client side, the server is used to provide the new list of available appliances to the client.

7.6 DownloadHandler

If the download button is pressed, the client sends a request to the server, which is handled by this class.

7.7 ApplianceDataset

This is one of the most important classes, as it is used to save and process appliance data. Also median filtering is done here by the `med()`-method. States of the appliance are calculated within the `get_appliance_state_by_edge_detection()` and `get_appliance_state()` methods. Basic modifying is done by the `resample()` and `fill_missing()` methods. For gathering important information about the occurrence per day of the particular appliance the methods `get_count_per_day()` and `get_states_per_day()` are accessing `__get_occurrence_per_day()`.