SMART Machine Learning Lamp

TK3: Ubiquitous / Mobile Computing

Fuchs, Max
2908207

Gärtner, Christoph (Group AD)
2766889

Hofmann, Tobias
2350059

Kalabić, Edin
2401115

Linke, Alexander
2274630

July 17, 2017

1 Introduction

The gain in polularity of artifical intelligence in the recent years motivated our decision to build a smart lamp – with machine learning as driving actor. The lamp ought to be smart enough to toggle itself **on** and **off**, solely based on data. Data which is collected at the beginning, in the so-called *training-phase*, by manual user interaction with a physical switch.
Our concise two minute video (`https://www.youtube.com/watch?v=AOq6Cx7RbSw`) will give a short visual overview of the project.

1.1 Hardware

To keep monetary costs as low as possible, we tried using only the available hardware of prior excercises, i. e. a Raspberry Pi, several ESP32 developement boards and a broad range of available sensors+actors. But to avoid having to deal with mains electricity, because we as computer scientists should not act on exposed 230V wires with superficial knowledge, we were given three *Phlips Hue*[1] lamps with a network bridge for control.

1.2 Theoretical Layout

The Raspberry Pi ended up as the projects control center. All sensory data, as well as onboard metrics like *time of day* and *daylight* are flowing to the Pi, where it is logged during the training-phase. The training-phase is exactly one week after first start-up or after initating a reset. After training-phase, the Raspberry Pi would query the trained model continuously with current sensor values, to decide wether to turn the lamp on or off.
Because the Pi is very limited in regards to RAM and processing power, we couldn't use a machine learning technique like deep neural networks to train the model. Whereas only classification would be possible. But because the model also had to be learned on the Pi itself, we moved away from the idea of a neural network and towards a decision tree.
Unlike a neural network, a decision tree requires discrete values, which forced us to lessen resolution of continuous values like *time of day*.
Following is the complete list of used metrics and their resolutions:

- Time of Day
  - Early Morning
  - Morning
  - Early Afternoon
  - Late Afternoon
  - Night
- Daylight
  - True / False
- Weekday
  - Mo – So
- One entry per networked device
  - Device Status (Connected, Disconnected)
- Lamp Status
  - On / Off

[1]`http://www2.meethue.com/de-DE`

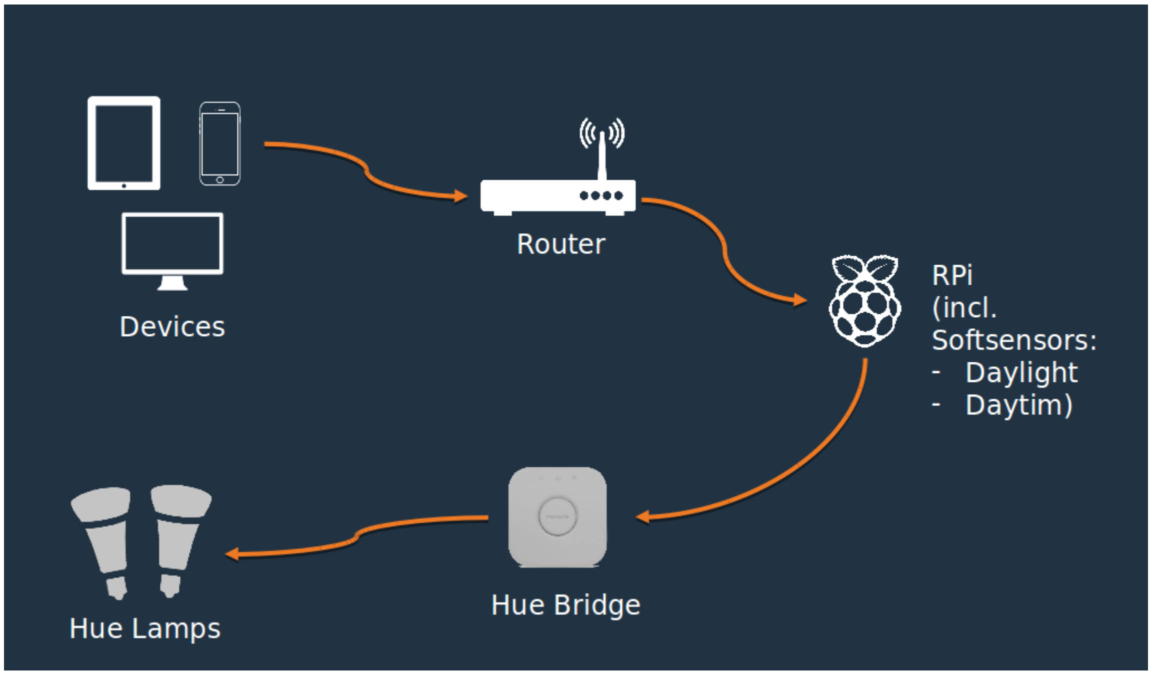

Figure 1: Architecture

2 Instructions for Code Execution

Get a MQTT Broker running in your network (e.g. on the Raspberry Pi). Adjust the IP Adress (192.168.1.226) in RaspberryPi/DataCollector.py and Linksys_Router/publish_wifi_clients.py to match the ip of the client that is running your MQTT broker.
Get yourself a Linksys WRT1900AC Router with openWRT operating system. The router needs to be in a network with the MQTT Broker and should be configured as an Access Point on your WLAN Devices. Edit the devices Dictionary in Linksys_Router/publish_wifi_clients.py to match your personal Wifi devices and respective MAC-Addresses. Install Python, pip and the paho library on the router. Run the Linksys_Router/publish_wifi_clients.py script on the Router.
Set up a phillips HUE in your network. Adjust RaspberryPI/LampControl.py to match your Hue's ip address.
Copy the RaspberryPI folder to your Raspberry Pi and run the DataCollector Script. This will store information about your lamps current status, daytime, connected devices, etc. in 'decision_tree_data' and 'raw' files. Let the data collection run for a few weeks.
After collecting data, run the RaspberryPi/ModelTrainer.py on your Raspberry. It will build a decision tree model based on the collected data stored in decision_tree_data.
Once the model is trained you can run RaspberryPI/Evaluator.py on your Raspberry. This script will switch the Hue on and off according to the decision tree model.

3 Implementation

Our code is completely written in Python. Python allows us to quickly test and generate functioning examples. Also many machine learning libraries provide excellent Python APIs, such as the libraries we used in the implementation.
Our Code is structured in two main logic parts, consisting of several smaller, partly shared, modules and classes.

3.1 Training Phase

The class *DataColletor* can be queried for the most recent sensor data. *DataColletor* internally listens for MQTT events and caches the latest value. At query time, all MQTT values are augmented with computed properties like time of day, status of the lamp, or daylight.
DataWriter.py is responsible to store the training data at a pre-defined interval, by querying *DataColletor*. The query-result is simply appended in a text file.
After the collection phase, the decision tree is computed. It gives us insight in when to turn the lamp on or off. This tree is computed in *ModelTrainer.py*.

3.2 Evaluation Phase

When the model is sucessfully created, it is evaluated. This is also done, just like the data collection, in a pre-defined interval. *Evaluator.py* gets the current sensor values, by quering *DataColletor* and with them as input, in turn queries the prior computed decision tree model. The decision tree query now results in wether the lamp should be turned on or off. The class *LampControl* provides an interface for doing just that.
LampControl sends API requests via HTTP to the *Philips Hue Bridge*.

4 Conclusion and Outlook

Due to the very short time we had to complete this project, we couldn't collect as much training data as we wanted. Partly also because of problems with the Raspberry Pi's timezone setting over Ethernet. Our final tree reflects just that constrained enviroment. Despite only turning the lights on at 2 distinct junctions, the tree still achieves a 91% accuracy on the trained data.
Interesting would be a result with complete full week long training.

4.1 Future Work

Future work could improve the SMART Machine Learning Lamp in the following ways.
The lamps underlying decision tree model currently cannot be easily adjusted after the initial training, without a complete new training-phase. Future work could improve upon that, and continuously improve the model with the help of ongoing user interaction with the lamp. This also includes when new devices that should impact the decision are brought to the household.
*Resposibilities

- Logo / Video: Alexander Linke, Edin Kalabić
- Presentation: Tobias Hofmann, Alexander Linke, Max Fuchs

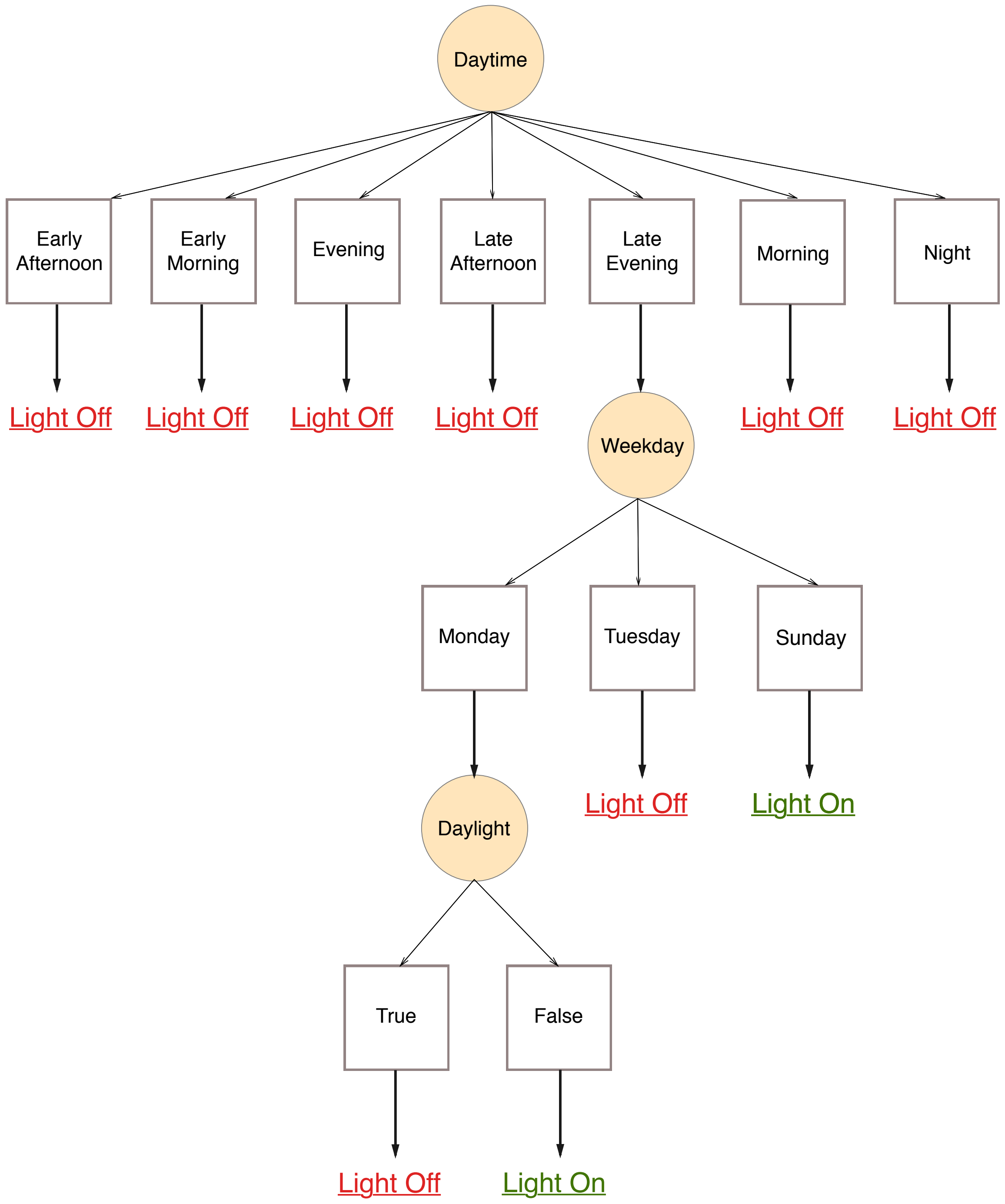

Figure 2: Resulting decision tree

- Documentation: All
- Lamp Control: Alexander Linke, Edin Kalabić
- Softsensor Daylight: Alex, Max Fuchs, Edin Kalabić
- Softsensor Wifi-Devices: Max Fuchs
- Evaluator: Christoph Gärtner, Max Fuchs
- Decision Tree Model Trainer: Max Fuchs
- Data Collection: Christoph Gärtner, Max Fuchs
- Class structure: Christoph Gärtner