2017

AutoPlug: An Automated Metadata Service for Smart Outlets

Lurdh Pradeep Reddy Ambati
University of Massachusetts Amherst

AUTOPLUG : AN AUTOMATED METADATA SERVICE FOR SMART OUTLETS

A Masters Thesis Presented

by

LURDH PRADEEP REDDY AMBATI

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

September 2017

Electrical and Computer Enginering

AUTOPLUG : AN AUTOMATED METADATA SERVICE FOR SMART OUTLETS

A Masters Thesis Presented

by

LURDH PRADEEP REDDY AMBATI

Approved as to style and content by:

David E Irwin, Chair

Marco F. Duarte, Member

C. Mani Krishna, Member

Christopher V. Hollot, Department Head
Electrical and Computer Enginering

ACKNOWLEDGMENTS

First and foremost, I would like to thank my supervisor, David Irwin, for his excellent advice and constant support, and for being an endless source of motivation throughout the course of my graduate studies.

Thanks as well to my fellow students Dong Chen, Srini Iyengar, Xue Ouyang, Jonathan, Supreeth Shastri, Noman Bashir and Akansa Singh. You took the daunting process of spending over two years working seemingly endless hours and made it truly fun. I hope I have helped you with your work as much as you helped me with mine.

Finally, I would like to thank my parents Anthony and Prabhavathi, my sister Praveena, and my brother-in-law Rajasekhar and my friends, for their unwavering support and understanding. Thank you for your constant encouragement and for listening to the long technical explanations you may or may not have asked for. I could never have finished Masters without your support and encouragement.

ABSTRACT

AUTOPLUG : AN AUTOMATED METADATA SERVICE FOR SMART OUTLETS

SEPTEMBER 2017

LURDH PRADEEP REDDY AMBATI

B.E., CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor David Irwin

Low-cost network-connected smart outlets are now available for monitoring, controlling, and scheduling the energy usage of electrical devices. As a result, such smart outlets are being integrated into automated home management systems, which remotely control them by analyzing and interpreting their data. However, to effectively interpret data and control devices, the system must know the type of device that is plugged into each smart outlet. Existing systems require users to manually input and maintain the outlet metadata that associates a device type with a smart outlet. Such manual operation is time-consuming and error-prone: users must initially inventory all outlet-to-device mappings, enter them into the management system, and then update this metadata every time a new device is plugged in or moves to a new outlet. Inaccurate metadata may cause systems to misinterpret data or issue incorrect control actions.

To address the problem, we propose AutoPlug, a system that automatically identifies and tracks the devices plugged into smart outlets in real time without user intervention. AutoPlug combines machine learning techniques with time-series analysis of device energy data in real time to accurately identify and track devices on startup, and as they move

from outlet-to-outlet. We show that AutoPlug achieves ∼90% identification accuracy on real data collected from 13 distinct device types, while also detecting when a device changes outlets with an accuracy >90%. We implement an AutoPlug prototype on a Raspberry Pi and deploy it live in a real home for a period of 20 days. We show that its performance enables it to monitor up to 25 outlets, while detecting new devices or changes in devices with <50s latency.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

The U.S. Energy Information Administration estimates that commercial and residential buildings account for 41% of U.S. energy usage, and over 75% of its electricity usage [24]. As a result, gathering detailed energy usage from buildings to optimize their energy consumption is critically important. Due to the high price of networked sensors, prior researchers have focused on analyzing power data from a single building-wide energy sensor to disaggregate it and estimate the energy usage of individual devices [19]. Unfortunately, such energy disaggregation, which is also known as Non-Intrusive Load Monitoring (NILM), is often highly inaccurate even in buildings with only a small number of devices [4]. However, recently, low-cost network-enabled energy sensors and switches have become widely available to consumers. The presence of these sensors can both aid in disaggregation or remove the need for it entirely. For example, many commercially-available smart power outlets cost <$50, including the Belkin WeMo [34], Insteon iMeter [20], and Z-Wave Smart Energy Switch [1]. In addition, research prototypes now exist that cost less than $20 [13]. These "smart" sensors and switches have the potential to enable deep visibility and control of the energy usage for each individual electrical device in a building. Ultimately, smart sensors and switches are the foundation of "smart" buildings that collect energy usage data from devices, combine it with external data on the environment, forecasts, energy prices, user occupanncy and comfort, etc., and analyze it to coordinate control of devices to optimize for energy usage, cost, user comfort, etc.

Smart energy sensors and switches may either be embedded into a device itself, or be attached externally to the device, e.g., as part of a power outlet. Embedding sensing and

switching functions into devices enables users to perform a one-time association between a device's unique identifier, e.g., its MAC or other layer-2 address, and its building management system (BMS). While this association is often done manually, given well-defined standards, resource discovery protocols could also be developed to automate the device's initial configuration with the BMS. However, embedding such functions into devices is likely only feasible for devices that are large enough to warrant the additional complexity. The numerous miscellaneous electrical loads (MELs), which comprise a rapidly growing portion of building energy usage [15], are likely too small and inexpensive to warrant their own embedded sensing and control functions. In addition, existing appliances that do not have smart functions will continue to operate for many years. Further, this approach requires BMSs to interact over the network with untrusted devices that visitors may bring into the building, e.g., to register them with the BMS, which is a security concern both for the BMS and for visitors.

Thus, a more general approach is to separate the energy sensor/switch from the devices, often by embedding these functions into each building outlet. This approach requires instrumenting only a building's outlets, rather than its devices. As a result, the BMS need only be configured once based on the unique identifier associated with each outlet, and also its location (which is generally not available from device-level sensors). In addition, since the outlets are part of the building's administrative domain, they can be trusted by the BMS, alleviating it from interacting over the network with untrusted devices from visitors. However, such external sensing poses a significant metadata challenge: since the sensors are built into outlets, rather than devices, users must manually associate the outlet with the respective device that is plugged into it. Further, users must alter this device-to-outlet mapping every time devices are unplugged or move to a new outlet. While some devices, such as a refrigerator, rarely if ever move, other devices, such as laptops, frequently change outlets. Companies typically provide smartphone or desktop apps to configure and monitor smart outlets, as well as schedule remote control

of devices, e.g., to turn them on or off at specific times or based on custom triggers. These applications also provide basic energy data analytics, such as a device's energy consumption. The market for smart outlets and other home automation devices is expected to grow by 60% from 2012 to 2018 [12].

Energy data recorded by smart outlets is much less useful to a BMS if the data is not correctly associated with a device, as it prevents a BMS from providing an accurate per-device breakdown of energy usage and also may result in incorrect remote control actions, e.g., by switching the wrong devices on or off. The configuration of current applications for controlling smart outlets and collecting their energy data is manual, and typically based on the outlet and not the device. Thus, users can only view the energy usage of outlets or automate the control of specific outlets, and not devices. Providing such energy data and control for devices, regardless of the outlet they are plugged into, is more natural for users, as energy-efficiency optimizations are based on devices not outlets.

To address the problem, we design AutoPlug, an automated metadata service for smart outlets, which can automatically identify and track the devices plugged into smart outlets based on their energy data in real time. We present our system as a service, deployable in a wireless gateway that communicates with smart outlets, and has the ability to identify the appliance plugged into the outlets. This gateway maintains a record of both previously identified devices, as well as a real-time record of the smart outlet→device mappings. This gateway could be incorporated into hubs like the Amazon Echo [3] and Google home [18]. For example, the Amazon Echo can already communicate with Belkin Wemo, ZWave, and Zigbee sensors and switches.

AutoPlug assumes a smart building that is equipped with smart outlets capable of recording and wirelessly transmitting their power consumption in real-time, e.g., at a 1Hz resolution, to a centralized gateway. The outlets may also be remotely controlled by the gateway, e.g., switched on or off. Our hypothesis is that combining machine learning techniques with analytical time-series models of device usage will result in accurate iden-

tification and tracking of devices on startup, and as they move from outlet-to-outlet in real time.

1.1 Contributions

In this thesis, we make the following contributions:

- **Real-time tracking**: We design AutoPlug to be a real-time system that identifies when a device moves from one outlet to another. Prior work [9] has not emphasized real-time identification and tracking of changing outlet metadata, and has instead focused narrowly on identification via classification over long time windows. Our basic approach is to combine time-series pattern matching techniques to recognize when the pattern of energy usage of an outlet changes, which indicates a new device has been plugged in.

- **ML Feedback**: AutoPlug uses the device tracking information to improve the offline machine learning techniques by enabling them to accurately configure the time period over which they analyze the data. If a device change has been detected in an outlet, then AutoPlug dynamically re-configures the analyzing time period such that it considers the time since the device change. Prior work [9, 2] generally performs the classification over a static time period, e.g., every 24 hours, which may result in inaccuracy if the device plugged into the outlet changes one or more times within the 24 hour period. We show that our approach is more accurate than the prior work [9] for device identification.

- **Implementation and Evaluation**: We implement AutoPlug prototype on a Raspberry Pi and deploy it live in a real home for a period of 20 days. We evaluate AutoPlug's accuracy on multiple data sets and in addition, we also evaluate its performance in terms of latency on multiple platforms. Prior works [9, 2] focus only on offline machine learning analysis, and ignore performance considerations. Our

results show that AutoPlug achieves ∼90% identification accuracy on real data collected from 13 distinct device types, and is also able to accurately detect when a device changes outlets with accuracy >90%. In addition, we show that AutoPlug is able to monitor up to 50 outlets on a Raspberry Pi 2 while detecting new devices or changes in devices with only a 100s latency.

1.2 Problem Statement

We define AutoPlug's outlet metadata problem as a combination of two distinct, but interlinked sub-problems. The first sub-problem is to identify the device D that is plugged into a smart outlet O_i over a period $[t_{start}, t_{end}]$, given time-series power data $P(t)$ from $[t_{start}, t_{end}]$. This problem is similar to the machine learning classification problem explored in prior work [9, 27], where the task is to map a given feature vector, which is based on processed time-series data, to a device label. As in prior work, AutoPlug processes the time-series power data to form a feature vector based on the data's statistical metrics. We then use well-known feature vectors from representative devices with known labels as training data to the classifier. After building the model, the classifier outputs a device's label based on an input feature vector. One notable difference between prior work and AutoPlug is the selection of the interval $[t_{start}, t_{end}]$ over which the classification occurs. Prior work generally performs this classification over a static time period, e.g., every 24 hours, which may result in inaccuracy if the device plugged into the outlet changes one or more times within the 24 hour period. In this case, the feature vector represents a variety of different features from multiple different devices. Instead, AutoPlug dynamically sets the interval based on the sub-problem below.

Our second sub-problem is to identify when a device is newly plugged into an outlet or changes from one outlet to another. MELs are often plugged into and out of outlets, especially in shared spaces such as living rooms or kitchens. We call this sub-problem "swap" detection using the same terminology from prior work, which first identified this

problem [27]. However, prior work only applied the same classification techniques as above to detect such swaps. Unfortunately, the machine learning classification problem above is not well-suited to dynamically detecting such changes in outlets in real time, as these classifications are trained based on device features, rather than the "features" of a change. That is, they attempt to simply map features over a given time period to a single device label. Thus, prior approaches cannot accurately detect the presence of multiple devices over a time period. Given a smart outlet O_i and time series power data $P(t)$, swap detection is the problem of determining the time t_{change} when a new device is plugged into an outlet and is turned on. Swap detection has two key metrics: the accuracy of t_{change} and the latency to detect a change has occurred.

1.3 Application areas

Potential applications of our system include device resource discovery, device activity recognition, energy attribution, etc. First, Homes with many smart outlets deployed may use the system to discover the status of the outlets and identity of loads[1] plugged in. The system would maintain the mapping of the smart outlet and its host appliance and these mappings can be accessible to users through a dashboard or a smartphone application. Without the accurate outlet-appliance mappings, it is fairly difficult for a user to spatially locate the devices (assuming the smart outlet's position is fixed).

Second, our system aids in device activity recognition, for example, it can provide information like when was the last time a coffee maker was active, when was the microwave oven last active etc. If we configure the AutoPlug to notify a user of certain events like garage door opening/closing, toggling coffee maker etc., then AutoPlug can notify the user in the case of a respective event happening. From the dashboard application perspective, this feature/aspect can be important as this can track the device as well as its activities.

[1]In this document, we use terms "load", "device" and "appliance" interchangeably

Third, it can be used to detect fault appliances. Since the AutoPlug analyzes the time series power data of each appliance, a faulty appliance can be identified as either an appliance which draws significantly more power than average that appliance used to draw or an appliance whose power consumption increases over time. In such a situation, AutoPlug notifies the user regarding the deteriorating device. The system could even compute and notify how long the device will be operational before complete breakdown, if the user takes no action to replace the device. An example of such a devices is generally those which are less energy efficient over their lifetime or which operate continuously over long periods of time like a refrigerator, air conditioner etc.

CHAPTER 2

BACKGROUND

In this chapter, we will discuss various existing techniques which aim to disaggregate a smart meter data into individual appliance energy consumption, techniques which deal with identifying the appliances based on the data or tags given by building managers and device modeling. We first describe non-intrusive methods for load monitoring. We then move onto the device modeling and techniques addressing the problem of identifying the devices plugged into the smart outlets/plugs.

2.1 Non Intrusive Load Monitoring (NILM)

The goal of Non-Intrusive Appliance Load Monitoring is to break down the aggregate energy consumption of household into individual appliance energy consumption. Hart (1992) introduced this field in his seminal work [19], which outlined a set of principles NILM algorithms should follow, a taxonomy of the potential approaches, a set of features that such approaches could use to distinguish between appliances and the use of finite state machines to model appliances. NILM techniques require the prior knowledge of the accurate appliance model. These models are required to track the appliance's load signature in the given aggregated energy consumption data of a household. There exist various techniques of NILM in the literature.

First one is Hart algorithm [19] for NILM, it is a model driven approach. In this approach, each appliance is modeled as a finite state machine. This approach first detects the edges in smart meter data, where an edge refers to a power surge or drop in power by a large margin in the data, for example, a +180W power change and -130W power

change due to the refrigerator will be detected as edges. After detecting all the edges in the data, clustering is used to cluster together similar changes in power between two steady states. After this On-off pairs of each appliance are grouped together i.e. similar power changes are grouped together. Next, simultaneous changes will be separated for example if a step change of +700 W is observed, this can be due to +500 W and +200 W appliances turning on. Finally, based on the finite state machine of each appliance model, On-off pairs can be identified as which appliance they belong to. Limitation of this approach is its performance degrades as we consider low power consuming appliances and it is best suited for on-off appliances only.

Second is Combinational Optimization (CO) [10] for NILM. CO is a topic that consists of finding an optimal object from a finite set of objects. At any given time, an appliance can only be in a single state. CO assigns each load a state and calculates the total power drawn by all the appliances in a household. Error in this assignment is the difference between the actual power drawn by all the appliances and power calculated above. CO seeks to minimize this by finding an optimal combination of the appliance in different states which will minimize the error term.

Apart from these approaches, there have been prior works which took a different approach to track/monitor the loads in a smart meter data. Powerplay [7] is a model driven approach for monitoring an individual electrical load's energy usage by analyzing a building's smart meter data. It takes an *online* tracking approach and employs a feature driven approach for tracking the loads. Powerplay tracks the individual device events in the real time by continuously tracking the device as a smart meter generates new data.

In the recent work [23], energy disaggregation of smart meter data is done by applying neural networks. Neural nets described in this approach once trained, they do not need ground truth appliance data from each house. Also, this approach requires substantial training data as the deep neural nets use large number of parameters.

2.2 Device Modeling

Accurate modeling of electrical loads characterizes the device usage and behavior, which is key in interpreting the energy data. Modern electrical appliances demonstrate complex energy usage pattern, that is hard to characterize using simple on/off model. It is particularly important when the only data available is aggregated, as is typically the case with a single energy meter providing energy data from the entire house. NILM techniques depend on accurate models to disaggregate the smart meter data, inferring occupancy patterns [25], and reducing peak demand by opportunistic load scheduling [8].

In the past, Sean Barker et al. [6] proposed an empirical or analytic modeling of electrical loads based on fundamental electrical characteristics (e.g., resistive, inductive, or non-linear loads). In this work, authors developed a framework to characterize/describe the energy use of modern devices that is more accurate than simple on-off models. Recently, Srini Iyengar et al. [21] developed an automated modeling framework(NIMD) for residential electrical loads, this work is similar to aforementioned work. NIMD enables simple construction of highly detailed power traces for any devices from given sample data. It was shown that generated traces closely approximate the ground truth data. Such a framework can be used for wide range of applications like generating training data for NILM algorithms, or for classification based techniques like NILI [9] etc.

Most prior works in modeling devices have been in the context of NILM (improving its accuracy). A recent work [2] aims to track the devices plugged in smart plugs in real time uses device modeling techniques to detect the devices, where they aim to infer the appliance energy consumption model from the given input time-series and identify the appliance as belonging to one of the defined appliance models.

2.3 Identifying appliances plugged into smart outlets

This section discusses the existing approaches to identify the devices plugged into the outlets and to track the devices which move from one outlet to another outlet frequently.

All the approaches described below assume a home where it is instrumented with smart outlets and smart outlet data is measured at a constant interval.

There have been several works proposed in the recent past that address the problem of identifying the appliance plugged into a smart outlet based on outlet data. Few of the works were based on the voltage/current data, other were based on the power data. These works are in the context of a home, the spatial location of the outlets is ignored in these approaches.

The fundamental approach of these techniques involves transforming the outlet data into a compact set of features, which characterize the energy consumption of the outlet. And then the off-the-shelf classifiers are trained on those feature vectors and are used to label the outlets. Usually, substantial amount of training data is required for these classifiers to accurately identity/predict the outlet's label.

Sean Barker et al. [9] address the problem of automatically identifying the devices plugged into the outlets. The approach presented in this work is based on the extracting the features from the input time-series power data, these features include statistical, and histogram based values to represent the device energy consumption. The proposed system uses a C4.5 classifier for classification and the data window is a day long and the data resolution is 1 second. The evaluation results show an accuracy of 93% in the case of "observed" devices. However, this approach doesn't take into the account the outlet changes.

Leonardi et al. [27] proposed a similar approach to the above work, apart from that this work presents a novel approach for detecting the new devices plugged in (new devices introduced in the respective house/environment) and detecting the swap of devices in a smart outlet. In this work, authors consider only the statistical features as part of the feature vector. The approach to detect new devices or swap of a device is to check if the energy usage pattern fits one of the existing models (classification model). However, this

approach doesn't take into account that appliance can operate in multiple modes, as such appliances like microwave oven, washing machine etc. have multiple operational modes.

The above approaches discussed share a common drawback; all of the works are offline analysis and ignore the performance aspect of the system as the machine learning and statistical techniques used in those approaches have high computational overhead.

CHAPTER 3

DESIGN

We distill AutoPlug's two sub-problems of outlet metadata maintenance—device classification and swap detection—into the two design pipelines in Figure 3.1. The device classification sub-problem includes feature extraction from time-series power data, as a pre-processing step, followed by model building based on training data from existing device energy usage traces, and then load classification based on the learned model, which provides the output AutoPlug uses to update its device-to-outlet mapping, i.e., by modifying the database that stores the mapping. In contrast, the swap detection pipeline has only two stages: the active period extraction as a pre-processing step followed by time-series similarity matching. In active period extraction, AutoPlug divides the input time-series power data into distinct device active periods, which represent contiguous time periods where a device is active and consuming electricity. Note that if there is no energy consumption by an outlet, AutoPlug cannot determine whether a device is unplugged or whether it is simply not turned on.

3.1 Device Classification and Labeling

For device classification and labeling, similar to prior work [9], we first perform feature extraction by transforming a given window of time-series power data into a reduced set of statistical features, called a feature vector, that serves as input to a classifier. AutoPlug extracts features from both the raw data, as well as processed data consisting of a new time-series of energy deltas that represent the difference between consecutive power readings in the raw data. We use the latter time-series because changes in power are

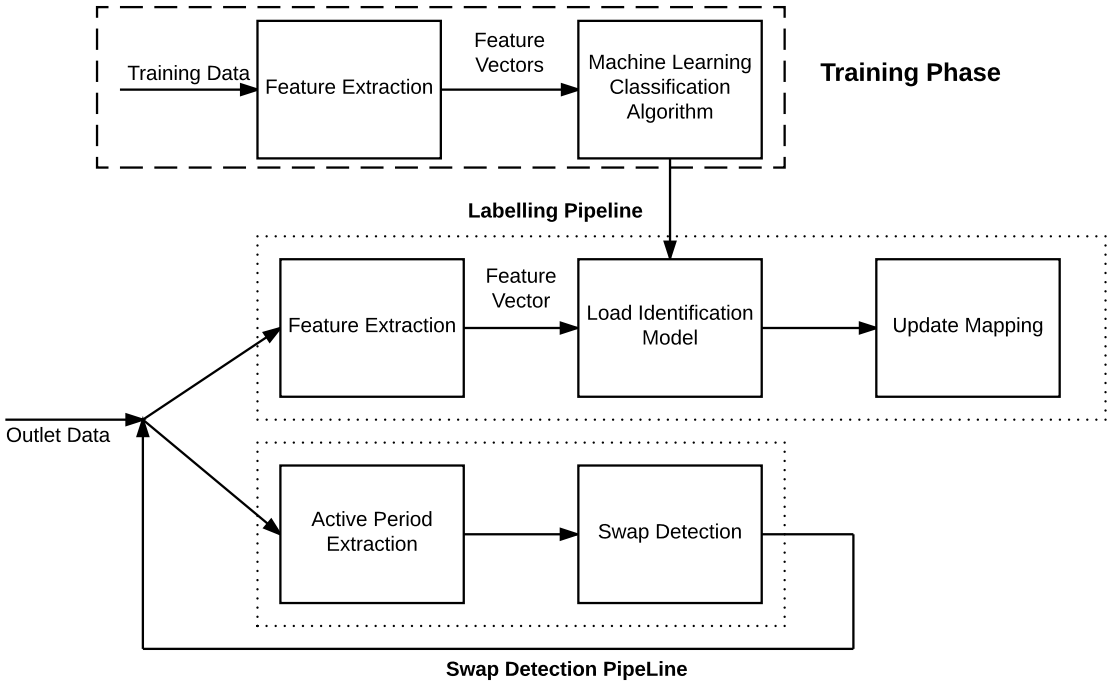


Figure 3.1: AutoPlug Design Block Diagram

often more identifiable than the raw power level of a device. Below, we briefly review the specific features AutoPlug's classifier employs for model training and device identification. Note that these features are similar to features used in prior work [9, 27].

3.1.1 Statistical Features

We compute a simple set of statistical features for the two time-series above. Common features include the average, maximum, minimum, and standard deviation over each input time-series. These statistical features provide the classifier model characteristic and discriminative information for a specific device. In addition, we also compute an additional metric for our feature set: the number of energy deltas greater than a threshold value Δ_{OSC}. This metric gives insight into the dynamic behavior of the device's energy consumption, i.e., the frequency and magnitude of its variations in power, as shown in the equation below (where p_i is the average power of i^{th} outlet, and $\delta_{>}(x, y) = 1$ if $|y - x| > threshold$ and 0 otherwise).

$$\Delta_{OSC} = \sum_{i=2}^{N} \delta_{>}(p_i(t_i), p_i(t_i + 1)) \tag{3.1}$$

p_i is average power of ith outlet

where $\delta_{>}(x, y) = 1$ if $|y - x| > threshold$, O otherwise

The *threshold* value depends on the input time-series data and varies across the appliances and appliance models.

3.1.2 Duty cycle

The duty cycle is the fraction of time a device has been active during a given window of time. This feature is useful in distinguishing continuously running devices from devices that run for shorter periods. The duty cycle feature indicates if an outlet's device is idle or active in the recent time-series window. We compute the duty cycle as the number of power readings greater than a threshold value divided by the total number of readings. This threshold value varies depending on the input time-series data.

3.1.3 Histogram Features

Devices also exhibit patterns of energy usage that are not captured by aggregate statistical metrics. Similar to prior work [9], to capture this, we separate the energy delta values of a device's time-series power data into separate bins of a histogram, which indirectly captures a device's energy usage pattern as a set of features amenable to classification. The selection of bin sizes is configurable, and affects the model's accuracy. We use 8 different overlapping bins spanning from 10W to 2500W. Each overlapping bin width is X to 5X, where X represents the starting power value for a bin. For example, our first bin is 10W-50W. Bin starting values are 5, 10, 25, 50, 100, 200, 300, 400, and 500.

For each bin, we calculate two features: a) a bin size, which represents the number of values that have populated the respective bin and b) an average time interval between

the energy deltas in each bin. Thus, for 8 bins, there will be a total of 16 features that characterize the waveform of the time series data.

3.2 Database Schema

We maintain a table of seen devices and their key characteristics [1]. AutoPlug updates the table whenever it updates an outlet's label. Each device entry in the table has a name field, outlet name, peak power, average power, energy consumed, and last active time. The table is initialized when the user deploys AutoPlug and users can set the expiration time for each device record (or can manually erase the database/table entry).

3.3 Detecting Outlet Changes

As discussed earlier, classification is not sufficient to accurately identify devices that change outlets in real time. In this case, the feature vector from an outlet's time-series energy data may represent a combination of two or more devices. The classifier, however, will provide only a single label, which may not match any of the devices plugged in, as the aggregate features above may significantly diverge from the individual features of any single device. Thus, detecting outlet changes is critical to the consistent maintenance of outlet metadata. Since standard classification is not well-suited to detect such real-time swaps, we design the detection technique below.

3.3.1 Active Period Extraction

First, to detect a device change in a smart outlet, we extract the active periods from the outlet's time-series power data. Each device alternates between active periods, where it consumes significant energy, and inactive periods, where it does not. Since some devices consume a small amount of standby, or vampire, power when inactive, we assume a device

[1] In this thesis, we use terms "sequence", "trace" and "time-series" interchangeably

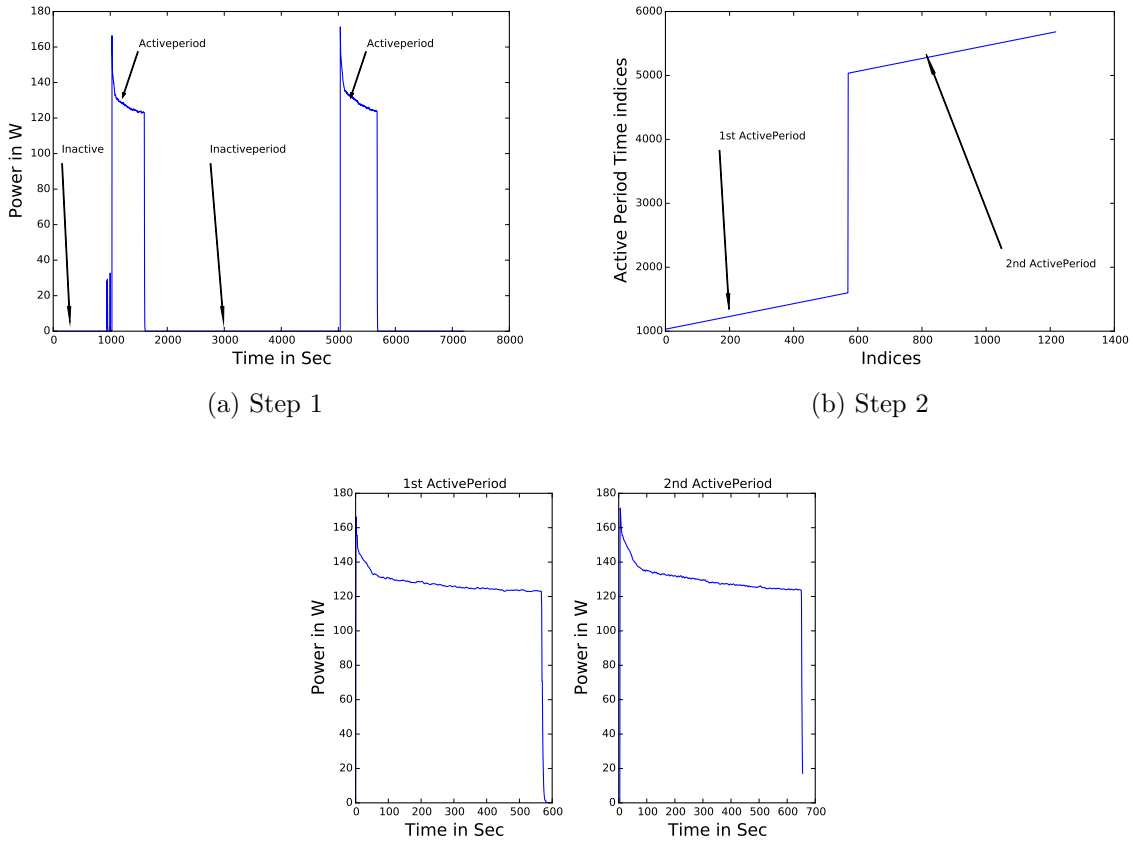
is inactive if its power usage falls below a small threshold. Based on empirical data across a wide set of devices we set this threshold to 5W. The active period is then a continuous time period where the device operates over a power greater than this threshold. We delineate separate active periods if the inactive period is greater than a separation time threshold, e.g., one minute. That is, if there is an inactive period of greater than one minute we consider there to be two active periods before and after the inactive period. If the inactive period is less than one minute, we discard the inactive period and assume it was part of a brief lull in operation of a device's active period.

Note that we have tried more advanced techniques for extracting the active period, one was change point detection to find the change points in the input power data, where a change can correspond to the device being ON or device going to idle state. Unlike the thresholding method, where we need to calculate the threshold for each input, change point detection doesn't require setting the threshold for each input. We have observed that both change point detection and thresholding yield similar output, and thresholding is more computationally efficient than change point detection. Considering the fact that thresholding is faster and computationally cheaper, we use thresholding in our work.

3.3.2 Time-series Matching

After we extract each active period from the input time-series power data in real time, we then compare it with the previous active period to determine if the device has changed outlets. In each case, AutoPlug signals a change in the device if the new active period is significantly different than the previous active period. We combine two different approaches to perform this comparison.

- **Time-series Distance**. There are multiple functions available to compute the distance between two time-series, such as Euclidean distance or Dynamic Time Warping (DTW) [30]. DTW finds an optimal match between two time-series which allows for stretched and compressed sections of the sequences. DTW improves on



(a) Step 1

(b) Step 2

(c) Step 3

Figure 3.2: Demonstration of active period extraction for a refrigerator trace

Euclidean distance, as it is less sensitive to slight differences in the alignment and shape of the time-series pattern, i.e., it is able to slightly "warp" each time-series to better align them and reduce the distance. Thus, DTW is robust to data sequences of different lengths unlike with Euclidean distance [30], as traces are "warped" non-linearly in the time dimension to compute a measure of their similarity. However, the DTW algorithm is expensive, as it has $O(n^2)$ time complexity, where n is length of longest data sequence. Thus, the longer the sequence in length, the more time it takes for AutoPlug to compute the DTW distance, which may not scale well on embedded devices like a Raspberry Pi or Arduino, commonly used as gateway devices. As we show in our experiments, we coarsen our data (from 1Hz resolution to 0.2 Hz resolution) before applying DTW to improve performance. Thus, in this

approach, we compute the DTW distance between two consecutive active periods and signal a change when it exceeds a specified threshold.

- **Curve Fitting**. Another approach is to fit a function to the data, e.g., such as a logarithmic growth function, and then compare the parameters of the best fit function for both active periods. In this case, we signal a change if the percentage difference between the parameters exceeds a threshold, which we determine empirically. Curve fitting is a method to construct the best fit of a mathematical function for the input data sequence, given the curve type or reference mathematical distribution. In this approach, we compute the parameters of the best fit logarithmic growth function to the active period, as prior work shows that this function approximates the energy usage pattern on startup for a wide range of devices [6].

$$p(t) = \begin{cases} p_{base} + \lambda * ln(t), 0 < t < t_{active} \\ \\ p_{off}, t > t_{active} \end{cases}$$

Using the logarithmic growth function, curve fitting on a given data set computes two parameters p_{base} and λ. p_{base} is the starting power level of the best curve fit and λ is growth parameter. In our approach, we compute parameters for both the active periods and then we compare the respective p_{base} parameters, and finally compute similarity S as the percentage difference in p_{base} of the both, where p_{base1} and p_{base2} are parameters for the active periods, respectively.

$$S = \frac{|p_{base1} - p_{base2}|}{max(p_{base1}, p_{base2})} * 100 \tag{3.2}$$

- **Approach Selection**. We use the DTW approach and Curve fitting approach above in different circumstances. Specifically, if the length of an active period is short, e.g., less than three minutes in our experiments, then we compare two sequences using the second approach, since the logarithmic growth characteristic of

many devices is generally short-lived. In contrast, if one of the active periodś length is long, e.g., greater than three minutes, we use DTW, as longer active periods tend to exhibit more variations in power usage that do not permit a single curve fitting.

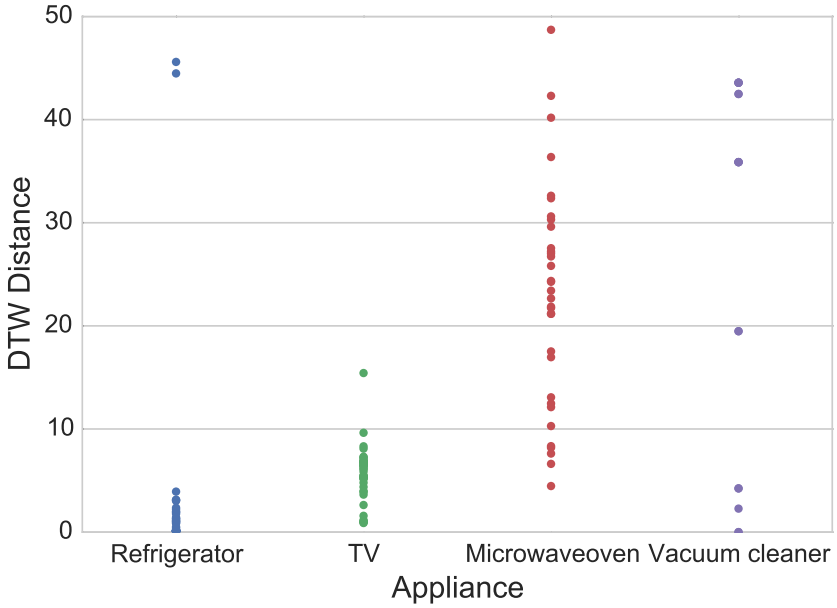


Figure 3.3: Strip plot of DTW distances between sequences of the same appliance, broken down by appliance

AutoPlug signals a change if the similarity score or DTW distance exceeds a threshold, which we determine empirically. As an example, we measure the DTW distance between two active periods for four device types and illustrate the results in a strip-plot in Figure 3.3. The figure shows that the DTW distance for the refrigerator and TV are well below 10 (with few exceptions), but that the microwave and vacuum have DTW distances scattered in the range of 0 to 50. Thus, selecting the DTW threshold for the microwave and vacuum is more difficult than for the refrigerator and TV. However, this is due largely to the shorter operating cycle of the microwave and vacuum, which in this case is below our threshold of three minutes. The Figure 3.4 shows that the curve fitting approach's similarity for the microwave oven and vacuum cleaner are well below 10 (with few exceptions). Thus, AutoPlug uses

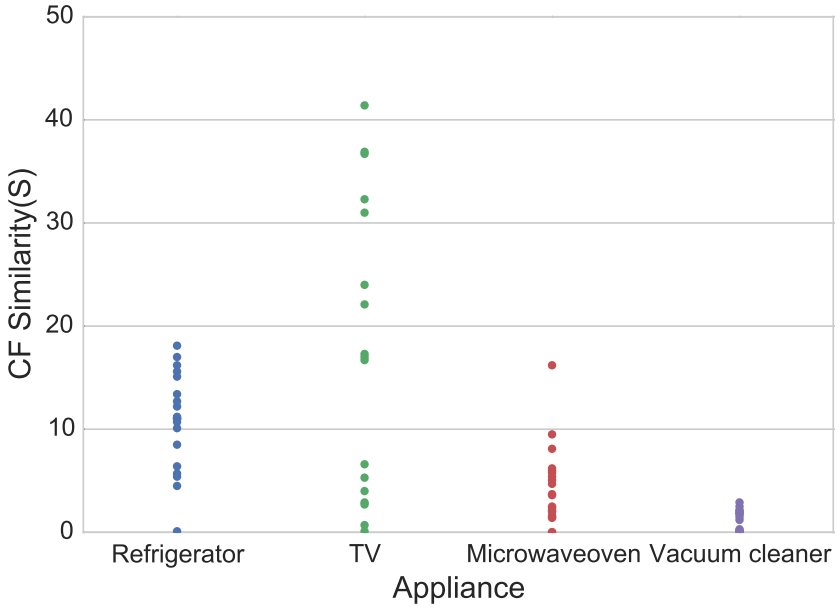


Figure 3.4: Strip plot of Curve fitting distances between sequences of the same appliance, broken down by appliance

curve fitting for these shorter active periods, as the DTW distance threshold is more variable for these periods.

3.3.3 Window Size and Update Frequency

AutoPlug adapts the data window size and frequency at which it runs the classification problem above. Prior work [9] uses a static window size of 24 hours and updated the classification offline once per day. Instead, AutoPlug sets the window size and update interval dynamically when it detects a change in the outlet. That is, the window size for the classification of an outlet starts from the last change detected to the current time. In addition, after a change AutoPlug periodically re-runs the classification, as the classification accuracy increases as more data is collected after a device swap. The period at which AutoPlug re-runs the classification based on new data is frequent, e.g., every 15 minutes, as new data significantly improves classification immediately after an outlet change. AutoPlug stops re-running the classification when the "confidence" in the labeling both reaches a specified threshold and does not significantly improve with new data. Here,

"confidence" refers to the probability assigned by classifier to the output label. Note that this approach results in AutoPlug potentially mis-labeling a device immediately after a change, as there is not much data, and then correcting itself as it collects more data.

CHAPTER 4

IMPLEMENTATION

We have implemented Autoplug in python using the Scikit-learn [29] and Scipy [28] stack. Scikit-learn is an open-source machine learning library for python, which has a collection of classification, regression and clustering algorithms. SciPy has a collection of powerful scientific computing libraries for data processing and visualization, as well as modules for performing curve fitting. We use the implementation of Dynamic Time Warping from a standard machine learning library for python. AutoPlug maintains a simple database table where each row stores a device label, an outlet label, a start time for the association, and the duration of the association.

4.1 Data Sets

For the classification technique, for training and initial evaluation we use device-level data from a public data set - Tracebase [31] and the data collected from real a home through eGauge equipment [14].

4.1.1 Tracebase Repository

The Tracebase repository was set up by a group at Darmstadt University, and contains individual appliance data from an unspecified number of households in Germany. The repository contains a total of 1883 days of power readings, recorded at 1 second intervals, across 158 appliance in- stances (e.g. a Bosch Logixx KSV36AW41G refrigerator), of 43 different appliance types (e.g. refrigerator) in 2012. Since the core aim was to create an appliance database, no household aggregate measurements were also collected.

Since the Tracebase repository contains many examples of different appliance instances of the same type, it provides an ideal data set from which to investigate the diversity of appliances within an appliance type.

4.1.2 eGauge Data

We have collected the data from a real home using the eGauge equipment. eGauge is an affordable, flexible, secure, web-based electric energy and power meter. eGauge provides XML API, using which we collect the data at 1 second resolution. The data collected from this home contains 30 days of recorded data from 6 devices belonging to these categories: washing machine, refrigerator, lamps, dish washer, freezer, and TV.

4.1.3 Reference Energy Disaggregation Data Set

The Reference Energy Disaggregation Data set(REDD) [26] was collected by a group at MIT from 6 households in the Greater Boston area, MA, USA. The data set contains both household and circuit-level data over various durations. Current and voltage data are recorded at high frequency (15 KHz) for mains circuits, while device-level or circuit-level data was recorded at low frequency(3-4 sec interval), of 30 different appliance types.

4.1.4 Miscellaneous

Apart from the above data sets, we have collected the data from the devices that users upgrade from time to time like TV, laptop etc. Table 4.1 shows the list of device's data we incorporate into the complete data set.

4.1.5 Virtual Data Set

We generate virtual data sets using Tracebase repository data and real home deployment data, such that the virtual data set resembles the appliance usage as in a real home environment. In a real home environment, some devices, such as a refrigerator rarely if ever move, other devices such as laptops frequently change outlets. So, we manipulate

Device List
Coffee maker
Dish Washer
Freezer
Lamp
Laptop
Laundry Dryer
Microwave Oven
Printer
Refrigerator
Toaster
TV
Washing Machine
Vacuum Cleaner

Table 4.1: List of Devices that AutoPlug can identify

the Tracebase data to reflect such behavior, such as laptop/lamp device data will have outlet changes, refrigerator/microwave oven data will not outlet changes etc.

These virtual data set contains data of Refrigerator, TV, Microwave oven as the static outlet data and as far as dynamic outlets are concerned, they host lamp, laptop, and vacuum cleaner data.

4.2 Classifiers

In this work, we investigate three classifiers and pick the best classifier that yields high labeling accuracy. We evaluate the classifiers based on the accuracy calculated by using cross validation algorithm. Cross validation technique is a assessing/bench marking technique for classification/regression, where it estimates the performance of the algorithm.

4.2.1 Random forest classifier

Random forest classifer is an ensemble classifier based on bagging technique, that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes. Bagging description:

Given a Training Set Tr (with corresponding labels Lr), bagging randomly sub-samples from it K times (sampling with replacement) to form K train sets - $(Tr_1, ..., Tr_K)$. Each of these sets are used to train a independent instance of same regressor/classifier to obtain K different regression functions - $f_1(x), ..., f_K(x)$. After training, predictions for the unseen samples can be made by averaging the predictions from all the model trained previously.

Random forest classifiers are fast and accurate, are robust to outliners in data. Random forest classifiers have limitations that all decision tree have, is the danger of overfitting when you have many high-cardinality categorical variables.

4.2.2 Support Vector Machine

We also consider a classifier using the support vector machine (SVM), a discriminative classification algorithm that maps the input feature space into second, linearly separable feature space using a kernel function.

Support vector machines are effective in high dimensional spaces. But, they are slow due to mapping the input feature space to higher dimensional space and re-mapping to original feature space.

4.2.3 Naive Bayes

Finally, we consider Naive bayes for classification due to its simplicity and its efficiency. Naive bayes algorithm assumes that all the features are conditionally independent of each other. Naive bayes classification algorithm assigns probability to each label for the given input and outputs the label which has highest probability.

Naive bayes yields high accuracy in case of the data/features where the features are orthogonal to each other, if not the accuracy can be poor.

4.3 Live Deployment

We have implemented an AutoPlug prototype system on a Raspberry Pi 2, and deployed it in a real home. We instrument the home with four Belkin Wemo Insight

Switches [34]. The Belkin Wemo Insight Switches are WiFi-enabled smart outlets, which are programmatically accessible via the ouimeaux python API [22]. The Raspberry Pi acts as a gateway to gather outlet power data, using the `ouimeaux` API to poll each outlet for its energy usage at a 1Hz resolution, and implement AutoPlug's techniques from the previous section.

Raspberry Pi 2 acts as a gateway to switches and using ouimeaux API gateway polls each outlet at a regular interval of 1 second respectively. Gateway updates the outlet mappings at a dynamic interval, taking latest two hours of data as input and updates the database with the corresponding classification output.

CHAPTER 5

EVALUATION

In this chapter, we evaluate AutoPlug's accuracy on our multiple public data sets, on data collected during the live deployment, and on the virtual datasets we construct from real data. In addition, we also evaluate its performance, in terms of the average computation time per update, i.e. latency, on multiple platforms, including a Raspberry Pi and Macmini.

5.1 Accuracy

AutoPlug operates in an online fashion by continuously polling each smart outlet every second, such that it updates the smart outlet's label at a dynamic interval based on the detection of a swap. We evaluate both classification accuracy and swap detection accuracy. As we discuss, there is a trade-off between the latency of detecting a swap and its accuracy.

5.1.1 Classification Accuracy

We divide our dataset into blocks of two hours each, and then we transform each data block into a feature vector as described previously, which we use for initial classification. Similar to prior work [9], we consider two scenarios of evaluation for the classification: identifying previously observed (or "seen") devices and previously unobserved (or "unseen") devices. Identifying seen devices is the case where respective device data is already in our training data, where with unseen devices the classifier predicts the label of a device whose data is not in the training data.

Classifier	Accuracy(%)	Training-Time(sec)
Naïve Bayes	69.23	0.165
Support Vector Machine	76.60	260.23
Random Forest Classifier	89.94	1.670

Table 5.1: Accuracy of different classifiers on our dataset.

- **Identifying Seen Devices:** We evaluate AutoPlug's device identification accuracy when we train and test the classifier on the complete dataset. We use this experiment to evaluate the accuracy of multiple classifiers, including Random Forest, Support Vector Machine, and Naïve Bayes. We perform 5-fold cross validation on the complete dataset to measure the model accuracy of above classifiers.

  Table 5.1 shows that the accuracy is highest for the Random Forest Classifier. Its accuracy of 90% is similar to the 93% accuracy presented in prior work on NILI, which addresses a similar problem [9]. However, NILI only works on a dataset size that includes 24 hours of energy data, which results in more information in each instance and apart from that, the respective data set was missing common appliance data like microwave oven, toaster, vacuum cleaner and laptop charger. In addition, NILI only updates each outlet's label once per day, while our approach performs an update at least once every 2 hours, and is dynamically determined. Table 5.1 is for static devices that do not change outlets. If we insert a change in outlet for each device, AutoPlug with the Random Forest Classifier maintains an accuracy of 89%, while NILI accuracy drops to 79%. (For this experiment part, we train the classifier on the both the Tracebase and live deployment data, and we test it on the virtual data sets we generate.)

- **Identifying Unseen Devices:** In general, training data for devices is not known *a priori*, and thus identifying previously unobserved devices is also important. For this experiment, based on our results above, we train the Random Forest Classifier

on the complete dataset and then compute testing accuracy for the data collected during the live deployment, which includes devices not in the training data. Note that the live deployment also included devices changing outlets. We observe that overall AutoPlug accuracy in this live deployment is close to 76% where the AutoPlug correctly labels 236 instances out of 310 instances. In comparison, the NILI approach that assumes devices never change outlets yielded an accuracy of only 64%. Figure 5.1 shows the per device break down of the accuracy.

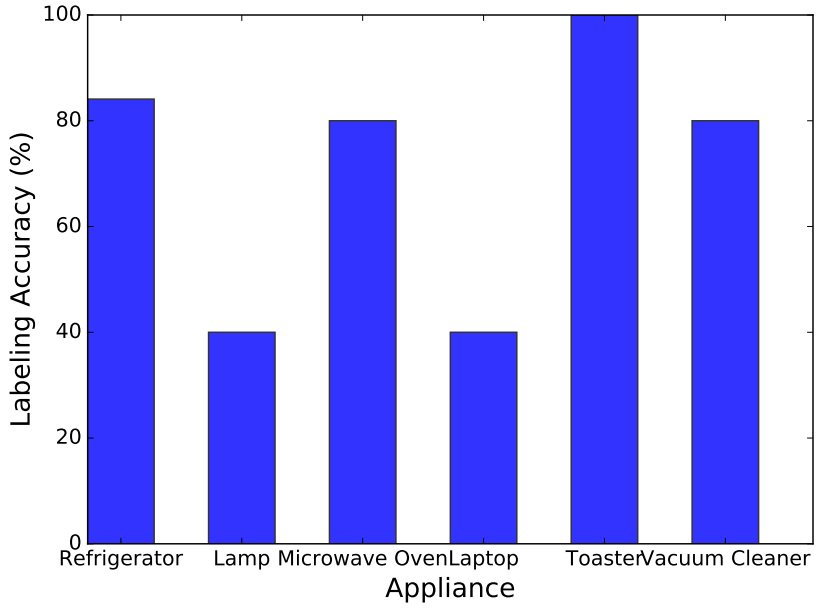


Figure 5.1: Detailed identification evaluation per device for unseen devices with random forest classifier

Finally, Table 5.2 shows the confusion matrix for the classification of previously unobserved devices. We see some devices exhibit worse accuracy than others(i.e. bars representing lamp and laptop in Figure 5.1). For example, every instance of toaster was correctly identified, which is understandable given the energy profile of toasters are similar in data set and unobserved toaster. The same holds true for Refrigerator, Vacuum cleaner, and Microwave oven, wherein accuracy is high in comparison to lamp and laptop. The lamps, on the other hand are frequently

mislabeled as TV-LCD, due to the fact that in the live deployment, the lamp's power output was 70W, while in the training data set the lamp data in some cases was 120W and higher. As a result, the Random Forest Classifier fails to classify the 70W lamp correctly. Similarly the laptops were mislabeled as lamp. Regardless, we see that Random Forest Classifier was able to accurately predict the device types with unique energy characteristics(refrigerator), and high power consumption with respectable accuracy.

	Coffee Maker	Dishwasher	Lamp	Laptop	MW Oven	Refrigerator	TV-LCD	Toaster	V Cleaner
Lamp	0	0	**10**	0	0	0	15	0	0
Laptop	0	0	15	**10**	0	0	0	0	0
MW Oven	3	2	0	0	**20**	0	0	0	0
Refrigerator	0	0	2	0	0	**165**	18	0	0
Toaster	0	0	0	0	0	0	0	**25**	0
V Cleaner	3	0	0	0	0	1	0	0	**12**

Table 5.2: Confusion Matrix for the classification of "unseen" devices.

5.1.2 Detecting Outlet Changes

Since the output of swap detection is a binary classification, e.g., either a swap is detected or it is not, we evaluate its accuracy based on the false positive rate, false negative rate and the Matthews Correlation Coefficient (MCC) [5]. A false positive represents the number of instances in which AutoPlug detects a swap that is incorrect. Similarly, false negatives represent instances where the AutoPlug fails to detect a swap. Similarly, the MCC is a quality measure of binary classification that takes both the false positives and false negatives into account, and represents a balanced measure of a binary classifier's performance. The MCC's value is between -1 and $+1$, where $+1$ indicates a perfect prediction, 0 indicates a random prediction and -1 indicates a total mismatch of observations and predictions.

False Positive rate is calculated as:

$$FPR = \frac{FP}{(TP + FP)} \tag{5.1}$$

False Negative rate is calculated as:

$$FNR = \frac{FN}{(TN + FN)} \tag{5.2}$$

MCC can be calculated as:

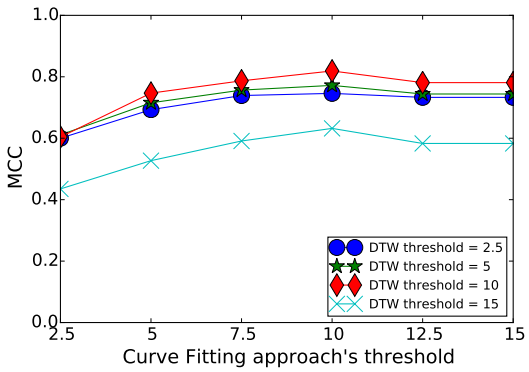
$$MCC = \frac{TPxTN - FPxFN}{\sqrt[2]{(TP + FP)(TN + FN)(TP + FN)(TN + FP)}} \tag{5.3}$$

where FP denotes false positives, FN denotes false negatives , TP denotes true positives, and TN denotes true negatives.

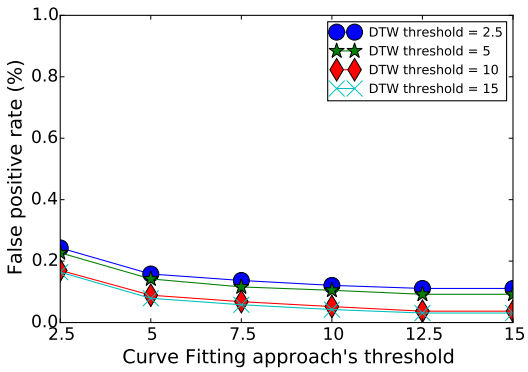
To evaluate swap detection, we compute the above metrics for our live deployment data and virtual data sets assembled from the Tracebase [31] and eGauge [14]. During the live deployment, we plug in six different devices into smart outlets, such that three of them operate in the living room and the other three operate in the kitchen. We then swap one device with other that operates in the same room.

Our false positive rate and false negative rate will depend on the threshold values for swap detection. To find the optimal threshold value for both of our swap detection approaches, we consider our DTW approach's threshold from 2.5 to 15 and our curve fitting approach's threshold from 2.5 to 15. We then compute the MCC, false positive, and false negative rate for each set of threshold values. Figure 5.2(b), and (c) shows that the false positive rate increases as the threshold values decrease, while the false negative rate decreases as the threshold values decrease. Thus, a threshold combination of 10 and 10 for both the approaches respectively is a suitable choice as this keeps the false positive rate and false negative rate relatively low and maximizes the MCC value, shown in (a). As the figure shows, the false positive and false negative rates of AutoPlug with optimal threshold values are 5.2% and 7.5%, which translates to 92% swap detection rate.

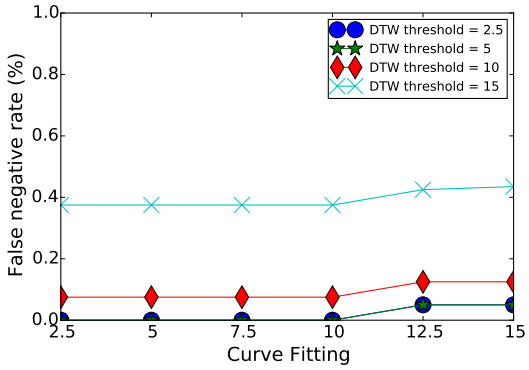
Figure 5.2(a) shows that AutoPlug's MCC for swap detection is near 0.8. We also compared this with an approach that detects changes solely based on classification, as in



(a) Step 1



(b) Step 2



(c) Step 3

Figure 5.2: Swap detection evaluation over different threshold values

[27], and not detecting changes in the pattern of time-series data. For this classification-based approach, we compare the output classification label of an outlet from the present data window to previous data window. If the labels do not match then we signal that a device has changed outlets; if they do match, then we signal a device has not changed outlets. This classification-based approach yielded an MCC of 0.31 and a false positive rate of 16%. This approach differs from AutoPlug in that the classification requires much more data to accurately re-classify a device, while our time-series-based approach is able to re-classify as soon as it recognizes that the pattern does not match the data pattern. Since such classification is done over data features that are aggregate statistical metrics of the data, more data results in a higher classification accuracy.

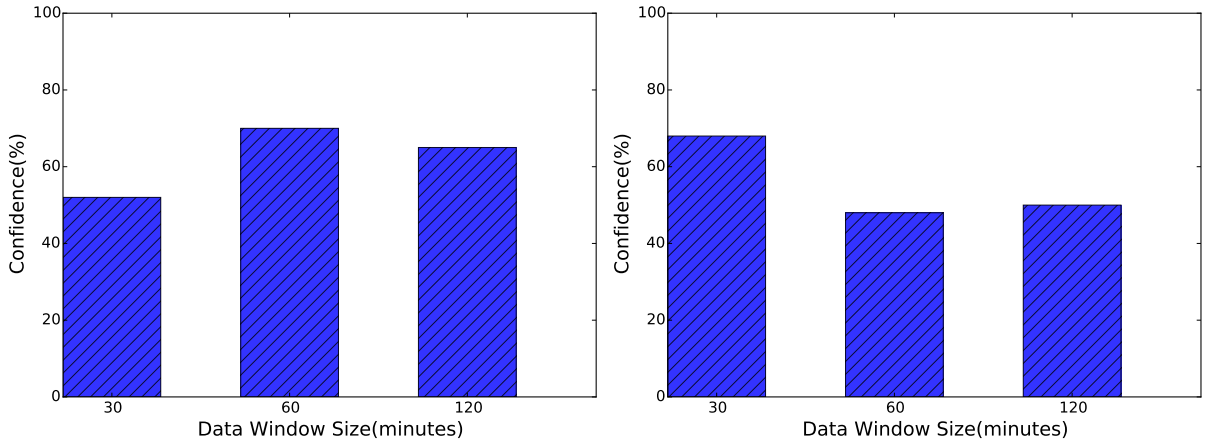
5.1.3 Dynamically Setting the Window Size

Recall that, unlike prior work, AutoPlug dynamically sets its window size and update frequency for classification based on its swap detection. AutoPlug automatically starts a new window after detecting a swap and then continuously updates the classification as new data becomes available. AutoPlug stops the classification when the confidence level for the device reaches a threshold and stops improving. The confidence level is the probability assigned to the output label from the probability distribution over the set of labels considered by the classifier (Random forest classifier implementation provides these probability values for each prediction). If the confidence level is over 50%, then it indicates that the classifier output is likely correct, and less likely if under 50%.

We compute the confidence levels for four devices: a lamp, laptop, microwave oven and refrigerator over three data window sizes (0.5 hours, 1 hour, and 2 hours) after a change. We then compute confidence levels over multiple active periods of the device and average those values, respectively. Figure 5.3 shows the results, which indicate that the microwave oven can be clearly labeled in 0.5 hours, but refrigerator and laptop need more time for accurate detection. Note in the laptop and refrigerator cases, when data window size was 0.5 hours, the classifier failed to assign their correct label, but AutoPlug corrects itself once more data becomes available.

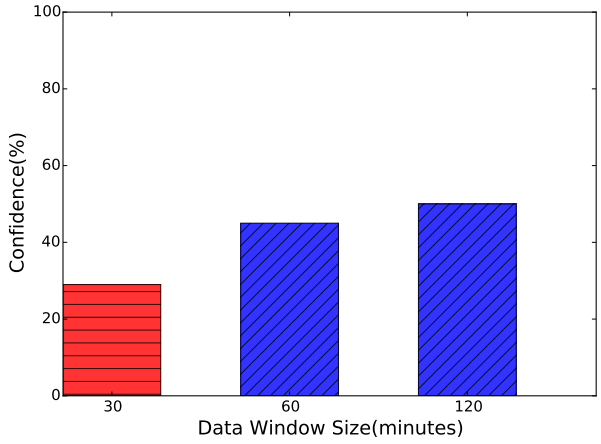
5.2 Performance

We next measure the average computation time per update i.e. latency, to evaluate AutoPlug's performance. Since AutoPlug stores only recent two hour power data per outlet, I/O is negligible, and computation time will depend on a) the number of outlets being monitored and b) the computational overhead of the swap detection process, especially the DTW approach given its high overhead. Here, we configure AutoPlug in two configurations—A and B—such that A is the default configuration and B adds a processing step that reduces the resolution of the active periods by a factor of five be-

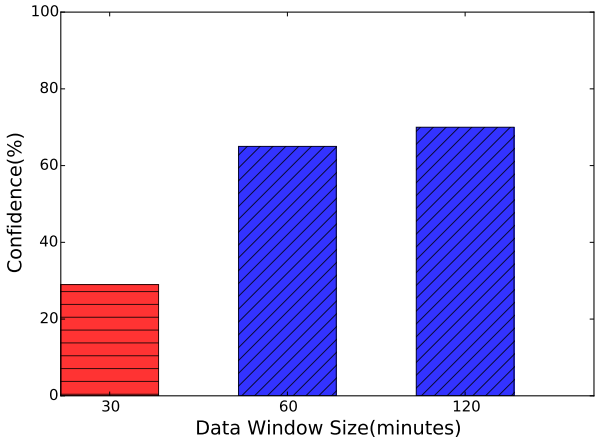


(a) Lamp

(b) Microwave oven



(c) Laptop



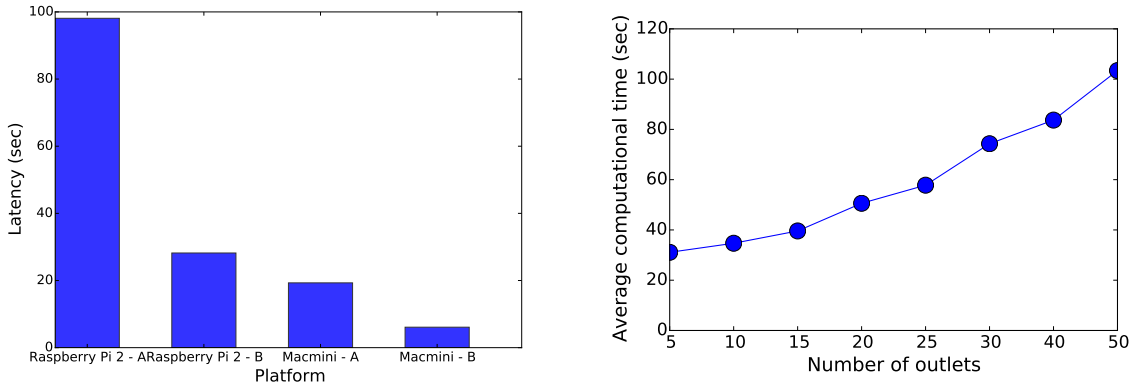
(d) Fridge

Figure 5.3: Confidence Level versus data window size for different devices.

fore applying DTW, e.g., from 1Hz resolution to 0.2Hz resolution. Note that, we have evaluated accuracy of system in configuration B.

We first investigate how the latency varies for AutoPlug running on different platforms. For this experiment we implement AutoPlug on a Raspberry Pi 2 running Ubuntu Linux, as well as a Macmini running OSX. From Figure 5.4(a) we see that the latency is modest in the case of Macmini, but increases for Raspberry Pi 2, particularly when AutoPlug is in configuration A, due to the computational overhead of DTW. Next, we measure the latency of the AutoPlug for varying number of smart outlets. We conduct this experiment

with AutoPlug in simulation (using configuration B) implemented on a Raspberry Pi 2. Also, for this experiment we assume that all the outlets are active. From Figure 5.4(b), we see that the latency increases as the number of outlets increases. When the number of active outlets increases beyond 25, there is a sharp increase in latency relatively, due to the CPU nearing saturation.



(a) Average computation time over different platforms

(b) Average computation time of the Autoplot over varying number of outlets

Figure 5.4: Performance evaluation of AutoPlug, in a) A and B indicates configurations of AutoPlug

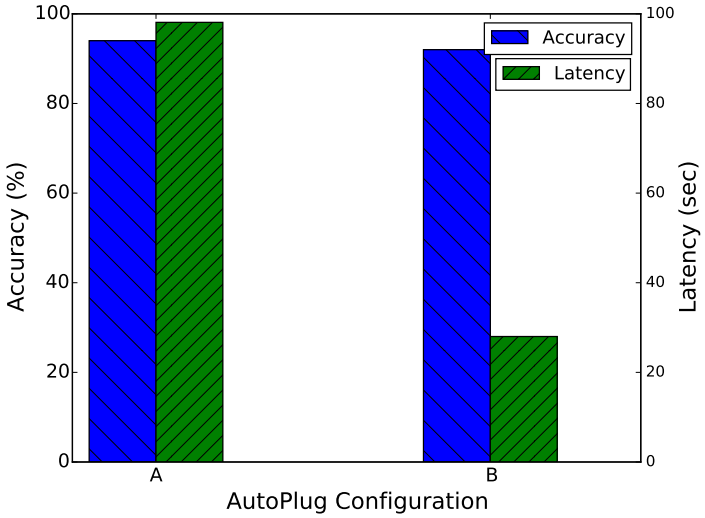


Figure 5.5: Accuracy and latency for two AutoPlug configurations.

Finally, Figure 5.5 shows the trade-off between AutoPlug's latency and swap detection accuracy. Here, we configure AutoPlug on a Raspberry Pi 2 monitoring four smart outlets. We see that latency varies by a large margin from configuration A to B, but accuracy in both the configurations is above 92% with configuration A being marginally higher, as expected due to the higher data resolution. However, the increase in accuracy over configuration B, which reduces the resolution of the data to reduce DTW overhead, is not significant, indicating the benefit of this optimization.

CHAPTER 6

CASE STUDY - ENERGY ATTRIBUTION

The electricity utility companies provide monthly report to users, a report that provides aggregate household energy usage and comparative details like how efficient one's energy usage compared to their neighbors. The aggregate energy usage details doesn't provide per-device energy usage or breakdown of energy usage per device. Often, it is desirable to have per device energy usage or the percentage of energy use attributable to certain devices. The access to such information can be key to:

- Identifying the inefficient devices or the devices that are running for a longer time than actually needed. For example, even when there are no occupants in a house and if lamps are active in that respective house, then switching off lamps will save energy resources.

- Detecting faulty or malfunctioning device, especially continually running devices like refrigerator. For example, a faulty refrigerator can be identified as either an appliance which draws significantly more power than average that refrigerator used to draw or whose power consumption increases over time.

The Non intrusive load monitoring(NILM) analysis on smart meter data disaggregates the house-level smart meter trace into energy usage data for individual devices. The disaggregated data can provide above benefits, but NILM algorithms have accuracy issues and require prior knowledge of the accurate appliance model (ground truth device level data). The poor disaggregation accuracy might result in inferring wrong conclusions about energy consumption, and hence the benefits provided by NILM analysis is limited.

Autoplug can address this issue, as it can monitor devices at circuit level(i.e. per plug) and combined with its high accuracy in detecting the appliances as well as tracking them.

6.1 Methodology

For every time window, Autoplug updates smart outlet's label and for each time window, AutoPlug computes energy consumed in kWH(kilo watt hour), last active time etc for the corresponding label and updates the sqlite database (that AutoPlug maintains and regularly updates). So, the database provides the details of each appliance like energy consumed, last active time, total active time duration etc over a period.

6.2 Experiment

In this case study evaluation, we compute the energy attribution error (per device) as the difference between the estimated energy consumption (database provides this value) and the ground truth data per device in the test data, and we compute the average of those errors as the average error.

For this experiment, based on the evaluation results in previous chapter, we train the Random forest classifier on complete dataset and set the optimal threshold values for both of our swap detection approaches and then perform the experiment on the two data sets, data collected from live deployment and REDD data set. We use the REDD [26] data set(device-level data) because it is widely used in NILM papers [17] and we want to compare (the state of the art) NILM algorithm's disaggregation accuracy with AutoPlug's energy attribution accuracy. In REDD data set we use House-1 data and devices we consider are refrigerator, dish washer, washing machine, lamps, and micro wave oven.

6.3 Results

Table 6.1 shows the energy estimation of live deployment data using AutoPlug. Some devices exhibit worse accuracy than others and results are comparable in terms of accuracy

with classification accuracy we reported in the last chapter. Autoplug estimates energy of high power devices like toaster, micro wave oven with high accuracy. But for the laptop, the accuracy is poor as laptop is frequently mislabeled as lamp. Average error for the experiment was 18%.

Devices	Estimated(kWH)	Ground Truth(kWH)	Error(%)
Lamp	2.01	1.635	18
Laptop	0.70	1.319	-46
Microwave Oven	1.273	1.213	5
Refrigerator	4.53	5.168	-12.34
Toaster	0.8	0.8	0.0

Table 6.1: Energy Estimation of Live Deployment Data.

Devices	Estimated(kWH)	Ground Truth(kWH)	Error(%)
Dishwasher	9.5	15.18	-37
Lamp	11.719	18.6	-36
Microwave Oven	12.5	11.57	8
Refrigerator	23.1	21.05	9.26
Washing Machine	23.59	25.139	6.16

Table 6.2: Energy Estimation of REDD Dataset

Second, table 6.2 presents the experiment results on REDD dataset. Here, the energy attribution error was high for lamp and dish washer. REDD dataset is 3 second interval, where as AutoPlug was designed for analyzing 1 second resolution data, this affects the waveform features in the feature vector. Suman Giri et al . [17] proposed NILM framework requires training on the test data and also inputting the device events like switch on and off time to generate the models . On the other hand, AutoPlug is easy to configure and doesn't require training on the test data. There model/framework achieves 77% energy attribution accuracy in the case of REDD data set and in comparison, Autoplug's accuracy was 81% accurate. Note that the NILM algorithms performance degrades as we consider low power consuming appliances, on the other hand AutoPlug can identify and track low power consuming appliances as well, given that we include similar device's data in training data set.

CHAPTER 7

RELATED WORK

This chapter lists and briefly discusses the related works addressing the problem of the identifying the devices plugged into the smart outlets published in the literature.

In the recent past, several works have been published which deal with the classification of devices on the basis of their power consumption. In NILI [9], authors address the problem of device identification by transforming the energy time-series data into a set of features and then use a classifier to identify the appliances. A.Leonardi et al. [27] take a similar approach for labeling the smart outlets, apart from this they describe a methodology to detect new devices introduced in a home and to detect swap of devices. These works [32], [35] try to address the device identification problem and follow the approach of training the off-the-shelf classifier and applying it to label the outlets, but not evaluated extensively especially in the case of unseen devices. What distinguishes our work from the approaches described above is that we employ time series and device modeling techniques to detect device swaps, moreover our approach frequently updates the smart outlets at an interval less than 2 hours, dynamically determined. Apart from that, above works focus only on offline analysis, and ignore performance considerations, where as in this work we investigate the performance of AutoPlug in terms of latency. One of the recent works, Muhammad Aftab et al. [2] proposed a different solution to the problem that we addressed. There proposed solution uses device modeling techniques to identify the loads as well as for tracking the loads in the real time, where they aim to infer the appliance energy consumption model from the given input time-series and identify the appliance as belonging to one of the defined appliance models. Their work

differs from our approach of combining machine learning and statistical techniques, and was published after our work [33].

Further, the problem of automated metadata in the context of the commercial buildings was addressed in these recent works [16, 11]. In Jungkun et al. [16], authors develop a framework to automatically infer a sensor label based on its time-series data using off-the-shelf classifier. Bhattacharya et al. [11] present a syntactic and data-driven approach to parsing sensor names to common name spaces. These works target building automation/management systems and the commercial building space where metadata associations seldom change, on the other hand, AutoPlug targets home automation systems and the residential home space where metadata association in the context of smart outlets change frequently.

CHAPTER 8

CONCLUSION

In this work, we present AutoPlug, a system for identification and tracking loads plugged in a smart outlet in real time. We now summarize the work and give the directions for the future work.

8.1 Conclusion

We first defined the problem statement of AutoPlug in chapter 1. We identified two key requirements that must be fulfilled to address the problem of identification and tracking of loads plugged into the smart outlets. The requirements were device identification and device swap detection. Further, we have stated the potential application of our system.

We then presented the background of existing techniques in the field of the Non intrusive load monitoring for smart meter data, device modeling and of existing works in automatically identification of devices plugged into smart outlets based on the time-series data. We have showed that the existing works in the "automatically identification of devices plugged into smart outlets", are related to offline analysis, ignore performance metrics. Apart from that, these works doesn't address the issue of device swap in a smart outlet.

We then described the design of the AutoPlug. AutoPlug design consists of two pipelines. One is for the device identification and another is for the device swap detection. In device identification pipeline, first we transform the input time-series power data into a feature vector (compact set of features) and use these feature vectors as input to the machine learning classifier, which then outputs the device label based on the input feature

vector. Device swap detection involved extraction of the active periods as preprocessing step. We then use pattern matching to compare these active periods with each other to detect a swap. Finally, we defined the size of the data window and update interval, which is dynamic based on the swap detection output.

Chapter 4 presented the implementation details of the AutoPlug. We have briefed about the data sets we have used in our system for training and testing of device identification and swap detection process. We then described live deployment details like AutoPlug prototype and the real home where it is deployed.

In the chapter 5 we evaluated the system on our multiple data sets and live deployment data. We first evaluated the accuracy of the AutoPlug. We have shown that AutoPlug achieves 90% identification accuracy on real data collected from 15 distinct device types, and is also able to detect device changing outlets with accuracy >90%. And then we evaluated the performance of AutoPlug in terms of latency. For which we show that its performance enables it to monitor up to 50 outlets, while detecting new devices or changes in devices with 30s latency.

Chapter 6 presented the case study application on AutoPlug, which is energy attribution. The goal of this application is to provide the energy breakdown by appliance i.e. per appliance energy consumption over a period in a household. We performed the experiment on live deployment data and REDD dataset . We have shown that AutoPlug was able to yield an accuracy of 82% and 81% for the both data sets respectively. In the latter experiment, we compare our results with a NILM algorithm results, and we show that AutoPlug is more accurate and also easy to configure.

8.2 Future Work

Future work will focus on the exploring the case studies and implementing applications based on the AutoPlug work. We aim to realize the useful end-user applications to take

advantage of the AutoPlug's capbailities in identifying and tracking the loads. The work so far presented is a prerequisite to the end-user applications.

1. Smart device scheduling or load deferral : Energy grid companies provide real time electricity pricing i.e. per hour or per 5 minutes electricity price per KWh. This can be used to suggestion for deferring the heavy loads like washing machine, dish washer etc to decrease the overall cost of electricity for house holds. Such suggestions do not prevent the household occupant from performing there desired task, but instead provides useful information for reducing the electricity bill.

Apart from these we aim to implement the functionality of swap identification, where in the AutoPlug identifies the swapped devices based on the database/table that it maintains. Furthermore, we aim to improve the accuracy and reduce the latency of the system by optimizing the techniques that we use or by using new techniques.

BIBLIOGRAPHY

[1] `http://aeotec.com/z-wave-plug-in-switch`.

[2] Aftab, M., Chau, C., and Khonji, M. Real-time Appliance Identification using Smart Plugs. In *e-Energy* (June 2017).

[3] Amazon. `http://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-WiFi-Alexa/dp/B00X4WHP5E`.

[4] Armel, K., Gupta, A., Shrimali, G., and Albert, A. Is disaggregation the holy grail of energy efficiency? the case of electricity. *Energy Policy 52*, 1 (January 2013).

[5] Baldi, Pierre, Brunak, SÃÿren, Chauvin, Yves, Andersen, Claus A. F., and Nielsen, Henrik. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics 16*, 5 (2000), 412–424.

[6] Barker, S., Kalra, S., Irwin, D., and Shenoy, P. Empirical Characterization and Modeling of Electrical Loads in Smart Homes. In *IGCC* (June 2013).

[7] Barker, S., Kalra, S., Irwin, D., and Shenoy, P. PowerPlay: Creating Virtual Power Meters through Online Load Tracking in Smart Homes. In *BuildSys* (November 2014).

[8] Barker, S., Mishra, A., Irwin, D., Shenoy, P., and Albrecht, J. SmartCap: Flattening Peak Electricity Demand in Smart Homes. In *PerCom* (March 2012).

[9] Barker, S., Mushtag, M., Irwin, D., and Shenoy, P. Non-Intrusive Load Identification for Smart Outlets. In *SmartGridComm* (November 2014).

[10] Batra, N., Kelly, J., Parson, O., Dutta, H., Knottenbelt, W., Rogers, A., Singh, A., and Srivastava, M. Nilmtk: An open source toolkit for non-intrusive load monitoring. In *e-Energy* (June 2014).

[11] Bhattacharya, A., Hong, D., Culler, D., Ortiz, J., Whitehouse, K., and Wu, E. Automated Metadata Construction To Support Portable Building Applications. In *BuildSys* (November 2015).

[12] Brennan, M. House of the future:How automation tech is transforming the homes. *in Forbes* (October 2013).

[13] DeBruin, S., Ghena, B., Kuo, Y., and Dutta, P. PowerBlade: A Low-Profile, True-Power, Plug-Through Energy Meter. In *SenSys* (November 2015).

[14] eGauge Energy Monitoring Solutions. http://www.egauge.net/, 2013.

[15] Analysis and Representation of Miscellaneous Electric Loads in NEMS. Tech. rep., U.S. Energy Information Administration, December 2013.

[16] Gao, J., Ploennigs, J., and Berges, M. A Data-driven Meta-data Inference Framework for Building Automation Systems. In *BuildSys* (November 2015).

[17] Giri, S., and Berges, M. An energy estimation framework for event-based methods in Non-intrusive Load Monitoring. *Energy Conversion and Management 90* (2015), 488–498.

[18] Google. https://madeby.google.com/home/.

[19] Hart, G. Nonintrusive Appliance Load Monitoring. *Proceedings of the IEEE 80*, 12 (December 19 92).

[20] iMeter Solo. http://www.insteon.net/2423A1-iMeter-Solo.html.

[21] Iyengar, S., Irwin, S., and Shenoy, P. Non-Intrusive Model Detection: Automated Modeling of Residential Electrical Loads. In *e-Energy* (June 2016).

[22] Jones, Eric, Oliphant, Travis, Peterson, Pearu, et al. SciPy: Open source scientific tools for Python, 2001–.

[23] Kelly, J., and W.Knottenbelt. Neural Nilm: Deep Neural Networks Applied to Energy Disaggregation. In *BuildSys* (November 2015).

[24] Kelso, J., Ed. *2011 Buildings Energy Data Book*. Department of Energy, March 2012.

[25] Kleiminger, W., Beckel, C., Staake, T., and Santini, S. Occupancy Detection from Electricity Consumption Data. In *BuildSys* (November 2013).

[26] Kolter, J. Zico, and Johnson, Matthew J. REDD: A public data set for energy disaggregation research. In *SustKDD* (August 2011).

[27] Leonardi, A., Ziekow, H., and D. Konchalenkov, A. Rogozina. Detecting Smart Plug Configuration Changes in Smart Homes. In *Smart SysTech* (July 1-2, 2014).

[28] McCracken, Ian, et al. ouimeaux: Open source control for belkin wemo devices, 2013–.

[29] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[30] Ratanamahatana, C.A., and Keogh, E. Three Myths about Dynamic Time Warping. In *SIAM International Conference on Data Mining* (April 21-23, 2005).

[31] Reinhardt, A., Baumann, P., Burgstahler, D., Hollick, M., Chonov, H., Werner, M., and Steinmetz, R. On the Accuracy of Appliance Identification Based on Distributed Load Metering Data. In *SustainIT* (October 2012).

[32] Ridi, A., Gisler, C., and J.Hennebert. Automatic identification of electrical appliances using smart plugs. In *WoSSPA* (May 2013).

[33] Venkatesh, J., Aksanli, B., Rosing, T., Junqua, J., and Morin, P. HomeSim: Comprehensive, Smart, Residential Energy Simulation and Scheduling. In *IGCC* (June 2013).

[34] Belkin wemo insight switch. `http://www.belkin.com/us/p/P-F7C029/`, February 2016.

[35] Zuffrey, D., Gisler, C., Khaled, A., and Hennebert, J. Machine learning approaches for electric appliance identification. In *ISSPA* (July 2012).