Incremental Learning from IoT for Smart Home Automation

Term Research Project Report

Santa Clara University
COEN 281

Nguyen Do
Quan Bach

Acknowledgements

The researchers of this project would like to extend their gratitude to Dr. Ming-Hwa Wang for providing us the opportunity to conduct this research and write this research paper. We are also thankful to his lectures in data mining and patterns recognition.

We also would like to express our gratitude to the Center of Advanced Studies in Adaptive Systems (CASAS) for providing us the data sets in this research.

Table of Contents

Page

List of Tables and Figures

Tables

Figures

Abstract

In the era of the industrial revolution 4.0, turning a regular boring home into a Smart Home has never been easier with the price of smart sensors and gadgets becoming much affordable. Home owners can equip smart sensors, install monitoring systems, integrate voice and remote control. Smart Homes, however, are very reactive and not really smart yet. Hence, there is a need for a central system to mine data and learn from the resident's occupancy and usage behavior. In this project, we demonstrated the possibility of such a system via the development of 3 different pruning techniques that can be applied to Internet of Things data prior to performing association rules: Naive Duration Based Pruning, Time Threshold Based Pruning and Duration Based Pruning. By mix-matching the techniques, we can apply them to different scenarios and we hope they will pave the way for future research.

I. Introduction

Objective

The objective of this research is to mine sensors and appliances data to detect usage correlation patterns. We aim to improve the Smart Home Management system (SHMS) by letting it continuously learn from these patterns to make automatic suggestions or decisions for the users without the manual SHMS configurations.

What is the problem

With the technological advancement in the Internet of Things (IoT) and SHMS, turning a regular home into a smart home has been easier. The lights turn on when a sensor is triggered. The heating/cooling system can be controlled remotely on smartphone devices. The SHMS can display usage data in real time. Nevertheless, smart homes still remain rather reactive and involve manual rule setup. Smart homes are lacking a central system to mine the residents data and to apply what they learn from those patterns.

Smart sensors have become very affordable and reliable, which enable the collection of a large amount of daily from a single home. The challenge of mining patterns from these large data sets has been taken on by many researchers, and there are quite a number of efficient algorithms have been proposed. However, these studies only focused on mining data from single ‘events’(i.e. occupancy, traffic in and out of a building) or from a single appliance.

There was a study and proposed algorithm to mine the correlation patterns among appliances and showed splendid results. The researchers of the paper introduced an algorithm that can efficiently mine the big database and detect meaningful patterns. This research has made clear the objective and necessity of incrementally mining patterns among sensors and appliances, with intention to expand to the learning of occupancy and electricity consumption data.

Why this project is related to this class

This project is related to this class because it aligns with the course objectives and learning outcomes. The project focuses on the problems of ming in large data sets and patterns recognition. We are not only implementing different data mining techniques, but also exploring different methods to learn from the mined patterns. Principally, we will research and implement frequent-sequence detection algorithms.

Through this research, we would deeply understand the significance of processing the data in such a way that not only preserving the information accuracy but also minimizing the requirements for time and memory extension.

Why other approach is No Good

The problem with other approaches does not come from the technical aspect, but rather the lack of diversity and generalization in data mining. Most of the studies for the past decade had only focused on mining the patterns of either individual appliance, occupancy, or energy consumption. There were many good algorithms that had been proposed and proven to work efficiently to mine patterns from the large data sets generated from the sensors. However, the applications of these

findings have not lived up to their expectations. Our smart homes have not yet become smart. The paper 'Significant Correlation Pattern Mining in Smart Homes' by Yi-Cheng Chen - Tamkang University, Wen-Chih Peng and Jiun-Long Huang - National Chiao Tung University, and Wang-Chien Lee - Pennsylvania State University has also pointed out the same problem and showed promising results in finding correlations patterns among appliances.

Why we think our approach is better

Our approach is better as we have several enhancements to previous studies including datasets, preprocessing, methods of mining and application. We use data from multiple sources, which is closer to the real life situations. Our data pool includes fifty motion sensors, door sensors, light sensors, fan sensors, temperature sensors, and energy usage. In this research, we will process and mine data from motion sensors with doors, and lights sensors to discover correlation patterns in usage behavior of the residents.

We intend to develop an evaluation method to learn from these patterns and then to make suggestions/decisions for the residents. The association rules coming out from the mining would get updated incrementally using the data collected in the future to self correct and self adapt to real changes of the users.

Statement of the problem

In the era of the Internet of Things, the availability and affordability of smart home gadgets, sensors, and system; has arisen the necessity of a central system to mine and learn from the correlation patterns between all the appliances and sensors within the house. As most of the previous studies have only focused on a single data source, we intend to explore different mining techniques to discover the correlation patterns between appliances and motion sensors. From the patterns we want to develop an evaluation process to learn and make suggestions/decisions for the resident.

Scrope of investigation

(1) Explore different mining techniques to tackle the large data sets problem

(2) Explore different algorithms to discover frequent sequence and correlations patterns

(3) Explore different probability models to develop rules for a central system to learn from the correlation patterns, then make suggestions/decisions.

II. Theoretical bases and literature review

Theoretical background of the problem

Internet of Things definitions:

Internet of Things: are internet-connected objects (things) that are able to collect and transmit information over wire or wireless network without human intervention (Luigi et al.). They can be described as an extension of the Internet and other network connections (ZWave/ZigBee) to different sensors and devices, such as light bulbs, doors, locks, motion sensors, temperature sensors, camera…

Sensors: these devices collect data about the environment status. The data measurement can be as simple as in binary form, such as Motion ON/OFF to complex multivariable, such as Temperature Level, Luminance Level, Fitness Performance Measure… In this paper, we consider sensors as triggers for our actuators.

Actuators: the device that can produce actual physical action from either programmable command or user manual control. A light switch, for example, can turn on via a signal command via network communication, or when a person presses on it. The actuator data could be collected in the similar manner as sensors. In fact, most devices can be both sensor and actuator.

Connection and identification: to communicate with other devices in the Internet of Things system, the device has to have a unique identifier, usually paired with its own IP address. In this paper, the device identifications in the data set are named as in the following table:

Name / Pattern	Device Type
Mxx	Motion sensor
Lxx	Light, Fan
Dxx	Door sensor
Txx	Temperature sensor
P001	Electricity power meter
Ixx	Item sensor for selected kitchen item

Table 1: Identification for Internet of Things devices used in this research.

Data streaming from Internet of Things:

In the Internet of Things data streaming model, the data arrives at a "very large scale and high dynamicity, as well as the great heterogeneity of the data and systems involved" (Benjamin), while the algorithms that process it must run under very strict constraints of space and time. Due to this characteristic, data streams present many challenges for Internet of Things data mining algorithms when running in real-time to archive the acceptable accuracy.

Our method does not focus on real time mining, rather, we focus on pre-computed information to generate decision trees for real time responses. However, we will still continuously collect the streaming data and retrain the model to learn new patterns or correct the wrong ones.

The importance of a device:

In text mining, the TF.IDF (Term Frequency multiply Inverse Document Frequency) is the method used to calculate the importance of a word to a document in a collection. The Term Frequency, TF is defined as:

$$TF(t) = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ the\ document}$$

Applying the same idea to rank the importance of IoT devices, we define DF (Device Frequency) as follow:

$$DF(t) = \frac{Number\ of\ times\ device\ t\ appears\ in\ a\ period\ of\ time}{Total\ number\ of\ devices\ in\ the\ same\ period\ of\ time}$$

Apriori, AprioriTid and AprioriHybrid algorithms

To find Association Rules, the Apriori algorithms are usually utilized. Association Rules find all itemsets with the support greater than the minimum support and then use the large itemsets to generate the desired rules that have confidence greater than the minimum confidence. The Apriori algorithm utilizes the monotonicity property of the subsets of the frequent itemsets as mean to prune the lower support itemsets. In general, here are the steps of a pass in the Apriori algorithm:

1. The candidate itemsets are generated with only the itemsets that are frequent (meet the minimum support) from the previous pass, ignoring the transactions in the database.
2. The frequent itemset of the previous pass is joined with itself to generate all itemsets with size larger than 1.
3. Each generated itemset that doesn't meet the minimum support will be discarded. The remaining itemsets are the candidate for the next pass.

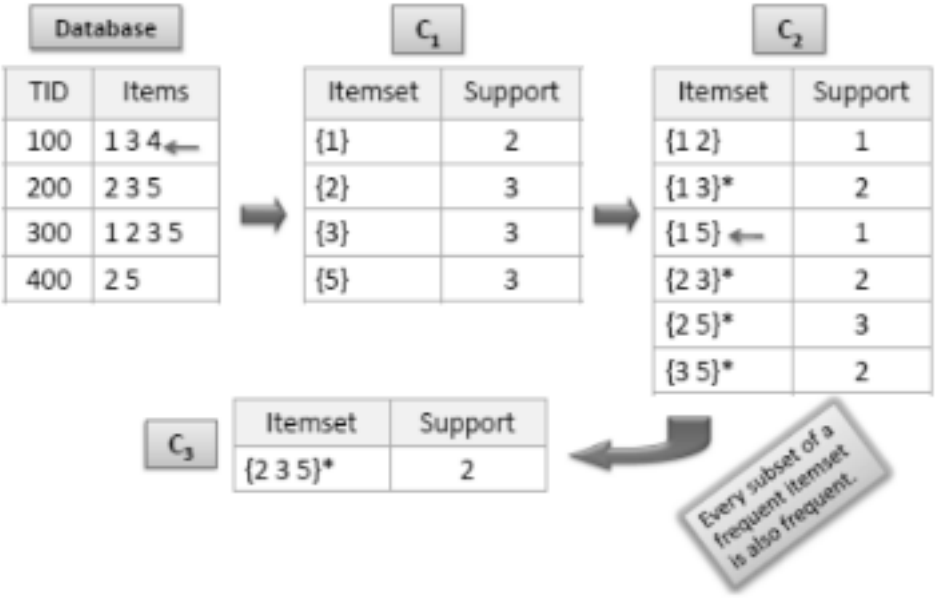


Figure 1. Example of the run of the Apriori algorithm with a minimum support 2

Another modified version of the Apriori algorithm is known as the AprioriTid algorithm. The processing of a pass is slightly different:

1. After the first pass, the AprioriTid algorithm will not use the database to count the support of candidate itemsets.

2. However, the candidate itemsets are generated similar to the Apriori algorithm.

3. In AprioriTid, a table C' is added to hold the TID (Transaction ID) and the candidate frequent itemsets of that transaction. This table is then utilized to count the support of each candidate itemset in the next pass.

Usually, the entries in C' would be much smaller than the original database, hence it would speed up the time used when counting support for later passes. The following figure shows an example of the AprioriTid algorithm.

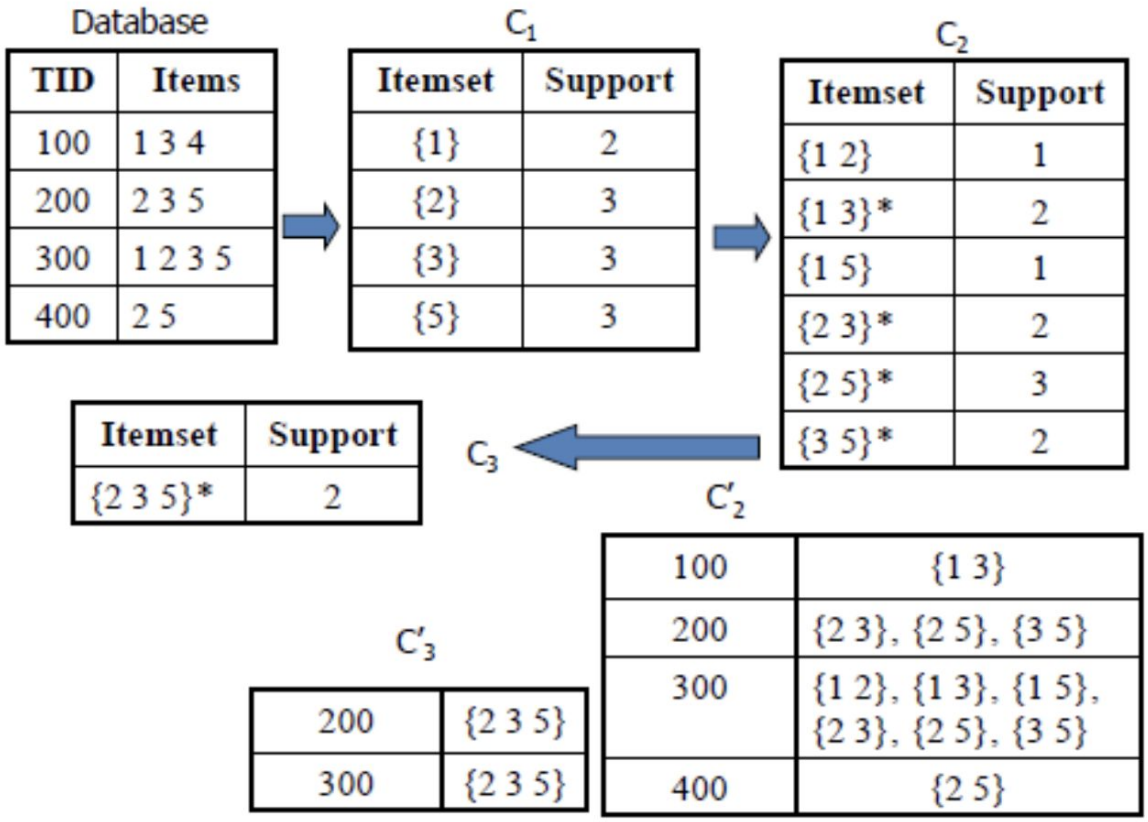


Figure 2. Example of the run of the AprioriTid algorithm with a minimum support 2

Finally, the AprioriHybrid algorithm invented by Agrawal and Srikant utilizes both Apriori and AprioriTid algorithms for a faster result. They have proven that the AprioriHybrid can outperform others by 3 to 10 times while scaling linearly with the database size.

Allen's thirteen temporal logic

In order to discover the correlation between different usage intervals, having a suitable representation is very important. Therefore, in this research, we organize the relationship between the intervals using Allen's thirteen temporal logics. Given two usage intervals A and B, the relationship between A and B can be fully represented by the figure below.

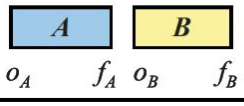
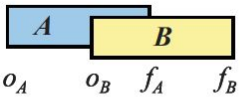
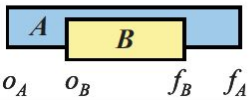
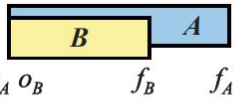
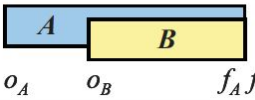
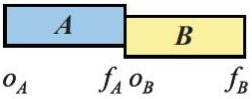
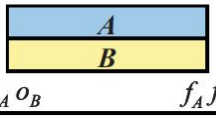
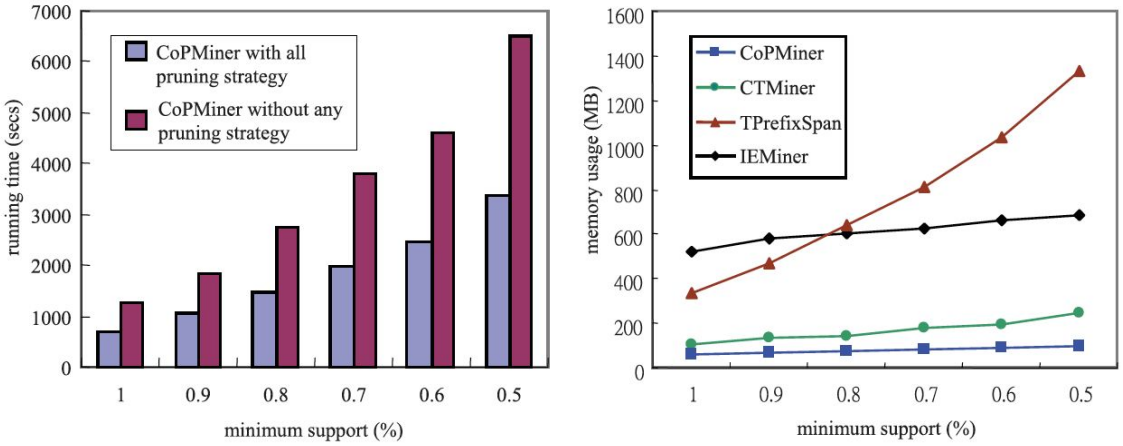
Allen's 13 temporal logics		pictorial example (o: on time, f: off time)	usage representation
temporal relation	inversed relation		
A before B	*B after A*		$\begin{pmatrix} A^+ & A^- & B^+ & B^- \\ o_A & f_A & o_B & f_B \end{pmatrix}$
A overlaps B	*B overlapped-by A*		$\begin{pmatrix} A^+ & B^+ & A^- & B^- \\ o_A & o_B & f_A & f_B \end{pmatrix}$
A contains B	*B during A*		$\begin{pmatrix} A^+ & B^+ & A^- & B^- \\ o_A & o_B & f_A & f_B \end{pmatrix}$
A starts B	*B started-by A*		$\begin{pmatrix} (A^+ & B^+) & B^- & A^- \\ o_A & o_B & f_B & f_A \end{pmatrix}$
A finished-by B	*B finishes A*		$\begin{pmatrix} A^+ & B^+ & (A^- & B^-) \\ o_A & o_B & f_A & f_B \end{pmatrix}$
A meets B	*B met-by A*		$\begin{pmatrix} A^+ & (A^- & B^+) & B^- \\ o_A & f_A & o_B & f_B \end{pmatrix}$
A equal B	*B equal A*		$\begin{pmatrix} (A^+ & B^+) & (A^- & B^-) \\ o_A & o_B & f_A & f_B \end{pmatrix}$

Figure 3. Allen's thirteen temporal logics

By representing the relationships with these temporal logics, the representation will be lossless, unambiguous and simple.

Related research to solve the problem

In the paper "Significant Correlation Pattern Mining in Smart Homes" by Yi-Cheng Chen - Tamkang University, Wen-Chih Peng and Jiun-Long Huang - National Chiao Tung University, and Wang-Chien Lee - Pennsylvania State University, the authors have developed an algorithm to efficiently mine the large data sets to detect usage frequent-sequences, then by implementing a statistical significance model to determine the probability of a frequent-sequence to happen, and then these patterns are used to detect abnormal activities in usage or make suggestions for saving power.



(e) The performance test of influence on all pruning strategies

(f) The memory usage test of different algorithms on different *min_sup*

Influence test of four pruning strategies and memory usage test on different *min_sup*.

Figure 4. Comparison of CoPMiner with and without pruning techniques from the paper 'Significant Correlation Pattern Mining in Smart Homes' by Yi-Cheng Chen, Wen-Chih Peng, Jiun-Long Huang, and Wang-Chien Lee

Their CoPMiner algorithm with four pruning techniques has shown to be much more efficient than other algorithms. Moreover, The authors also showed the significance of the four

pruning techniques by comparing the CoPMiner with pruning and without. The result in Figure 4. showed that the algorithm runs twice as fast with the four pruning techniques.

Another study called “Occupant Behavior Modeling for Smart Buildings: A CriticalReview of Data Acquisition Technologies and Modeling Methodologies” by Mengda Jia and Ravi S. Srinivasan - University of Florida, has also emphasized on the relation between occupancy and energy consumption.

In “Research and Application of Community Population Information Association Model Based on IoT Multi-device Mining” by Junwei Gang and Lin Qin, the researchers focused on mining large and complex Internet of Things data with applications around the population information. The following Figure 2. shows the framework of this research.

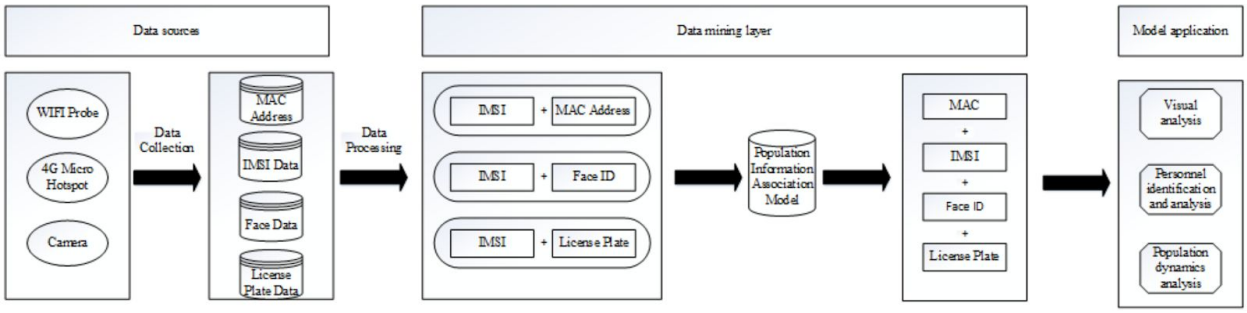


Figure 5. Population Information Association Model Framework from paper “Research and Application of Community Population Information Association Model Based on IoT Multi-device Mining” by Junwei Gang and Lin Qin

In their research, they assessed the two famous association algorithms in big data: the Apriori algorithm and the FP-Growth algorithm. While the FP-Growth algorithm can achieve a better performance, the Apriori algorithm has far simpler data structure. However, they found a drawback of the classical Apriori algorithm, which is “only suitable for single-dimensional association mining”; thus, it didn’t work with their data source as is. Their fundamental idea in the paper is to

combine calculation principles from the Apriori algorithm into the FP-Growth algorithm and call it "the improved FP-Growth algorithm." This algorithm successfully processed their population information data to one-to-one correspondence.

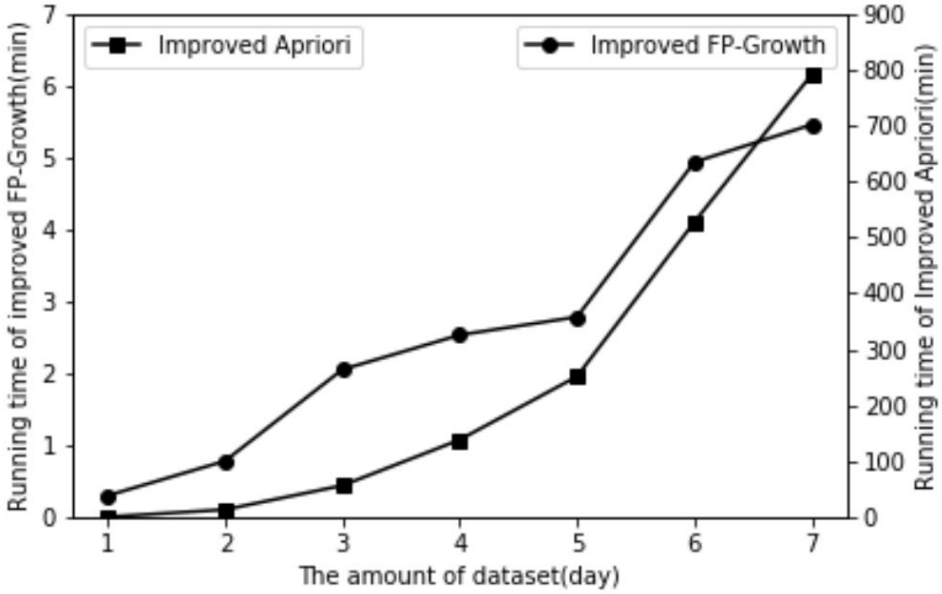


Figure 6. Comparison of Runtime performances for different models from paper "Research and Application of Community Population Information Association Model Based on IoT Multi-device Mining" by Junwei Gang and Lin Qin

The improved FP-Growth algorithm is done in 3 steps:

Step 1 consists of 2 scans to the database. The first scan calculates the support of each itemset. The second scan filters the itemsets that do not meet the support threshold, defined in advance to generate frequent itemsets.

Step 2 is the construction of the FP-Tree, using the itemsets from the previous step. Thus, this FP-Tree should only contain the frequent itemsets.

Step 3 proceeds with the actual mining on the FP-Tree. Then, with the mined frequent binomial sets, they calculate the confidence for each of the sets to determine the "strength of the association rule relationship." This confidence value is a key element for filtering undesirable association rules in their research.

Finally, they use one of the data sources as a key to join multiple association rules together from separate association models. The comparison that they conducted between the improved Apriori algorithm and their Improved FP-Growth algorithm in Figure 3. proves a significant enhancement in performance.

The 4th paper we looked at was "Pattern mining based compression of IoT data" by Dusan Ramljak, Amitangshu Pal and Krishna Kant discussed several pattern mining based compression algorithms for IoT data. Their algorithms are lossy with some trade-offs between the level of compressions and accuracy. These algorithms could be useful for us to deal with even larger data if we need to store historical data for incremental mining.

Advantage/Disadvantage of those research

Advantages: the above studies have a longer period of time to conduct their research. Also, they have access to many valuable equipment and help from the Taiwan government or private companies to collect real data. Therefore, their studies are usually thorough and widen in scopes. Furthermore, their research was published to The Association for Computing Machinery, a reliable and trustworthy source for computer scientists. With that, we can be confident in using their proposed methods and results as bases for our research. In fact, we will reuse or enhance some of their techniques to create our solution in this paper.

In the paper from Chen et al, the introduction to Allen's temporal logics could help us keep track of possible cases when the actuators react and potential overlapping that could cause false detections in our methods. The same paper also explores the idea of pruning to make the mining less noisy. They also surveyed multiple pruning techniques that could be applied to our scenario, such as spatial-pruning.

On the other hand, the paper from Gang et al introduces us to the idea of combining principles of different algorithms to archive a better performance. Since our data is complex, we think combining multiple techniques will give us the better results as well.

Disadvantage: the dataset used in their papers are not readily available for us to revalidate or use, so we cannot test or learn from their actual implementations. Some of the research papers were conducted in earlier years, when the technology of IoT and Smart Homes have not yet as advanced as of today. As a result, they are limited to the technologies available at the time their research was conducted. Moreover, all of the research papers above didn’t take into account time and incremental learning as important factors for their work.

Our solution to Solve This Problem

Our dataset is fairly complex due to the mix of multiple IoT devices, such as sensors and actuators. In addition, the frequency of recording the data from those devices are extremely high, from milliseconds to seconds. Therefore, the amount of data in our datasets is large and usually includes noises. Our proposed solution to the problem is to emphasize the pruning techniques to the existing data, and pre-categorize the data before mining.

Since human life is usually based on circles and set schedules, such as, 24 hour circle and weekly routine, we would like to divide our data into subsets by date of a week, time of a day, down to every single minute. Having each minute as a basket in the Apriori algorithm would help us identify the association between the sensors and the actuators much better with higher confidence. On top of that, comparing the mined Hourly Association Rules based on the same days of a week could further help us confirm the correctness of the rules.

After learning from the manual human trigger events, we will start to make predictions in the system and observe overwrites to the actuators from human reactions. We can call these human interventions as corrections to our prediction. Hence, we shall design an algorithm that can detect the corrections and discard the incorrect Associate Rules that were previously discovered. We believe that the correction mechanism will play a significant role in incremental learning capability of our method.

Where our solution differs

We intend to develop an evaluation method to learn from these patterns and then to make suggestions/decisions for the residents. The association rules coming out from the mining would get updated incrementally using the data collected in the future to self correct and self adapt to real changes of the users.

Also, we will divide our data into periods of time in order to analyze and discover patterns incrementally. Furthermore, we are looking forward to discovering correlation patterns between different sets of sensors and data points instead of focusing on a single type of sensor and an appliance.

Why is it better?

Our approach is better as we have several enhancements to previous studies including datasets, preprocessing, methods of mining and application. We use data from multiple sources, which is closer to the real life situations. Our data pool includes fifty motion sensors, door sensors, light sensors, fan sensors, temperature sensors, and energy usage. In this research, we will process and mine data from motion sensors with doors, and lights sensors to discover correlation patterns in usage behavior of the residents.

Furthermore, since our ideas follow the daily routine in human life and we organize the learning phase according to that, we should be able to produce more accurate results than taking all data into consideration.

III. Hypothesis

Positive hypothesis

Based on the papers we have been researching, the researchers have focused on mining the usage data of the resident as a whole to discover the usage patterns. They would receive the collected data as a whole and would develop algorithms and data structures to analyze and discover patterns from the whole period of time. However, we believe that human behaviors will change over time, hence their usage and interacting with their home will change too. This has motivated us to conduct this research on incremental learning from the change behaviors in order to enable our system to learn and adapt.

We hypothesize that the patterns in usage and behavior of residents of smart homes will change. Therefore, we believe that we will be able to find the significant change in resident usage behavior then we can design our central control system to be able to detect the changes and learn the new patterns incrementally.

IV. Methodology

How to generate/collect data input

We won't be generating our data. Our entire data sets for this project will be provided and can be downloaded from the Center of Advanced Studies for Adaptive System (CASAS). The data sets belong to a project called 'WSU Smart Apartment 2010 Two Resident Testbed'. The set includes the sensor events in which the start and end of various activities are annotated at the end of the corresponding sensor event lines. The sensors are categorized as mentioned in Figure 1. The dataset was recorded in space separated - CSV format. The following Figure shows a snippet of the dataset:

```
2009-08-24 00:00:00.000009 M046  OFF
2009-08-24 00:00:01.039408 M048  OFF
2009-08-24 00:00:19.034964 M050  ON     R1_Wandering_in_room begin
2009-08-24 00:00:19.078563 M044  ON
2009-08-24 00:00:21.029362 M046  ON
2009-08-24 00:00:21.095411 M044  OFF
2009-08-24 00:00:22.077674 M050  OFF
2009-08-24 00:00:25.061429 M046  OFF
2009-08-24 00:00:41.015317 M044  ON
….
2009-08-24 00:02:16.034348 M045  OFF
2009-08-24 00:02:16.078363 M045  ON
2009-08-24 00:02:19.020443 M050  ON
2009-08-24 00:02:20.025331 M045  OFF
2009-08-24 00:02:21.001098 M045  ON
2009-08-24 00:02:21.005446 M050  OFF
2009-08-24 00:02:47.069235 M045  OFF
2009-08-24 00:02:48.004345 P001  1002.9
2009-08-24 00:02:48.008569 P001  684
```

Figure 7. An example of the dataset from

"WSU Smart Apartment 2010 Two Resident Testbed"

The data sets include the collected data from these sensors, which are installed in two WSU apartments during the 2009-2010 academic year. The apartment housed two residents, R1 and R2, at this time and they performed their normal daily activities. The participants also notated their activities during their time in the apartment. However, we will not use those notations for our mining. We will use them for the verification step. The the sensors layouts in the two apartments are also included in the following figures:

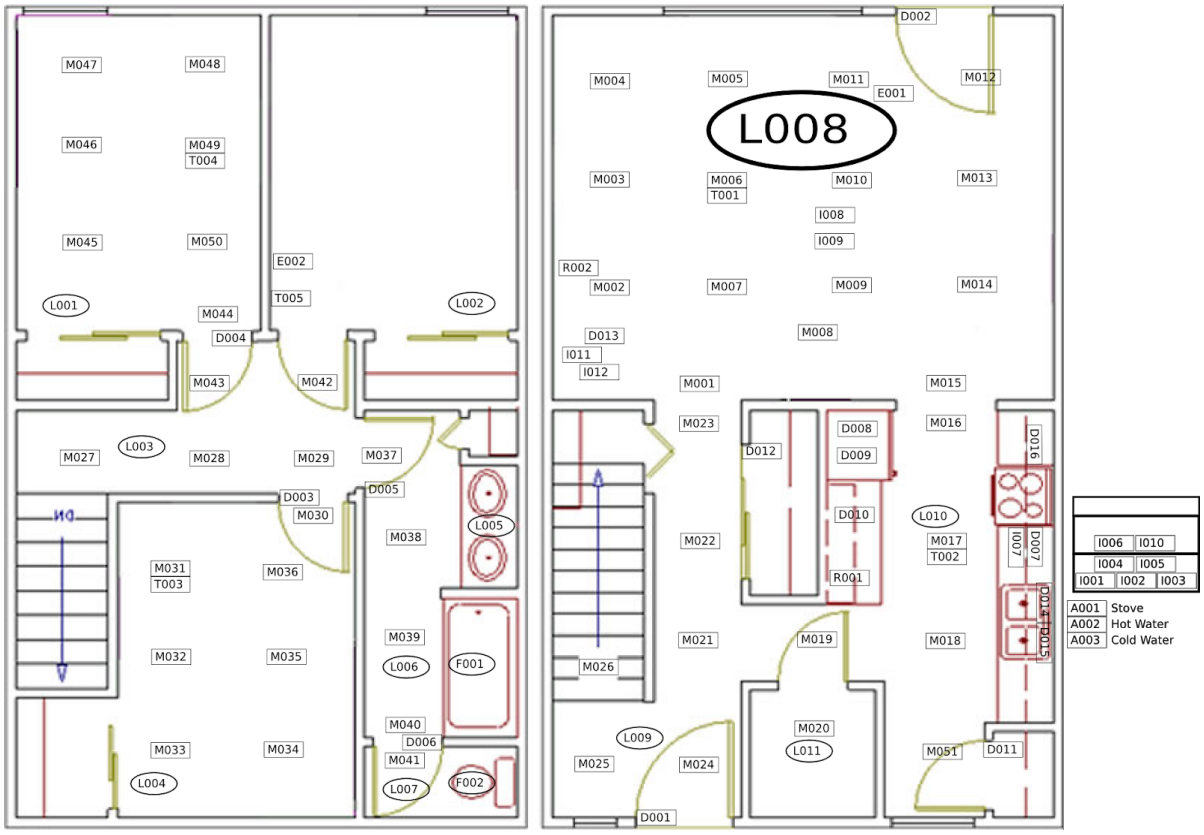


Figure 8. Sensors layout in apartment 1

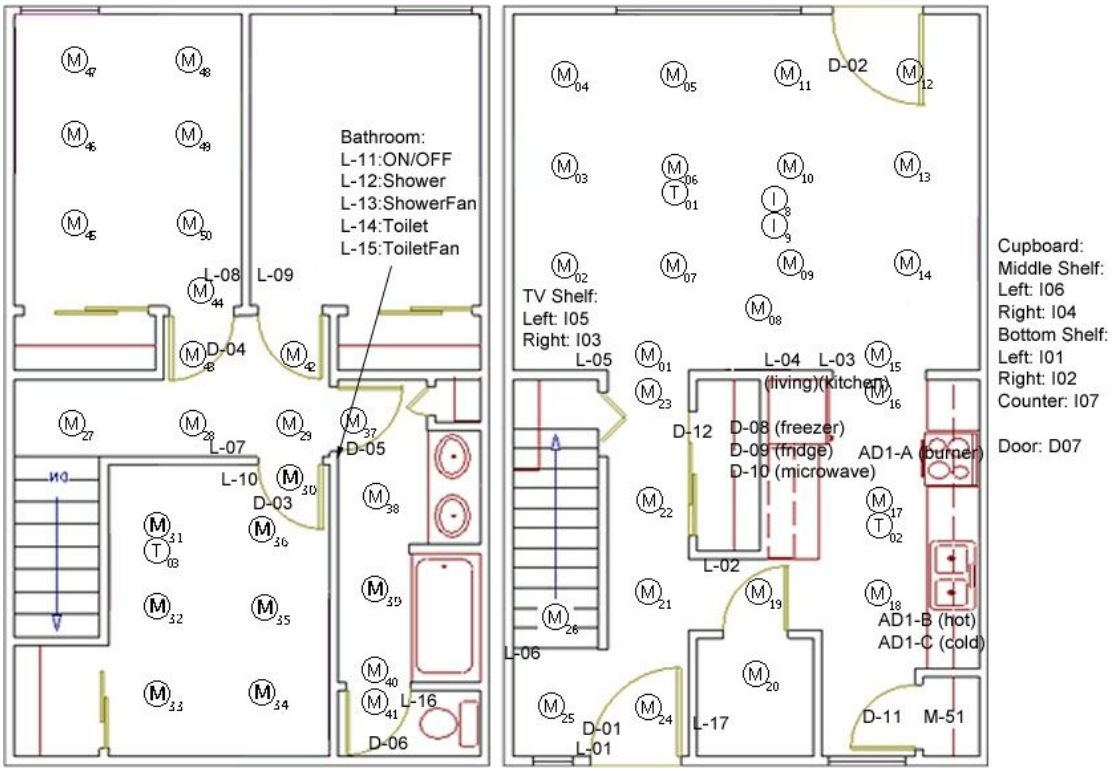


Figure 9. Sensors layout in apartment 2

How to solve the problem

Algorithm design

Spatial-pruning: in order to reduce the time complexity and memory usage, we implement spatial-pruning technique to exclude the possibility of including the sequence with motion sensors that are too far away from each other. This can be done by implementing a threshold max_distance based on the Euclidean distance between two sensors.

Noise-reduction-pruning: we will filter significant patterns based on the triggers of the actuators. Frequent-sequences that do not end with a triggered actuator will be ignored since

we aim to find the correlation patterns between the motion sensors and actuators. Therefore, sequences without the involvement of an actuator are categorized as noise and won't be stored to analyze.

Hybrid partitioning: For horizontal partitioning, we will filter the data further to the date of a week and time of a day, could be down to minutes for each basket as a trigger of the sensor and actuator reaction are effective in minute level. We will compare and combine the frequent itemsets based on the same days of a week.

For device partitioning, we will separate the sensors and actuators records. This will help us to apply the noise-reduction-pruning technique: only consider the basket that has both sensors and actuators. Otherwise, there will be nothing to learn and we should be able to safely discard such baskets.

Duration-pruning: From the hybrid partitioning, we also implement what we defined as duration pruning to find the routine activities of the time partitions. For each of the partitioned windows, we will scan the database for ON and OFF of each sensor ID and compute the total duration of occupancy for motion sensors and usage for actuators (appliances). From the duration accumulated we will compare with the duration threshold and determine the routine activities. These activities will then be converted into baskets ready for Apriori to find association rules.

AprioriHybrid: we will use this algorithm on our partitioned data to mine the association rules. Since we will combine multiple days, this algorithm should help us run Apriori in an efficient way.

Repeat with correction detection: after learning the initial rules, we will update our system by continuing to re-mine and re-learning the future data. If we see the actuators changes in a specific time unit, we would consider that as a correction user has made and consider that action at a higher weight.

Language used

Python 3.5

MatLab

Tools used

IDE: JupyterLab 2.2.4

Build machine configuration:

macOS Catalina; 2.5GHz Intel Core i7; 16GB 1600MHz DDR3.

Windows 10 Pro; 2.6GHz Intel Xeon E5-2660; 16GB 2333MHz DDR4.

External libraries:

pandas/numpy

scikit-learn

efficient-apriori

How to generate output

Our program will generate the discovered frequent correlation patterns between motion sensors and the triggers of the actuators. Also, we will show the association rules generated by the program based on the frequent correlation patterns.

How to test against hypothesis

We hypothesize the existence of the change in correlation patterns over time between the motion sensors and the triggered actuators. Our hypothesis will fail in the case when there is no change in the correlation patterns in all the analyzed and mined time intervals. Hence, if the set of association rules will remain constant over time, our hypothesis would be wrong.

V. Implementation

Code base

Our main code base is mainly written in Python 3. Due to the nature of the Internet of Things, our code should be able to run in multiple environments. To replicate this scenario, we deployed and executed our codes in 2 different Operating Systems: Windows and Mac using Anaconda 3 as an installation platform. Thanks to the Environment config file, we were able to successfully replicate the same environment among the platforms and proven the portability of the code base on different machines. The entire code base is shown in the appendices section.

Besides the main Python code, we also use MatLab for sketching and analyzing the events visually using heatmaps.

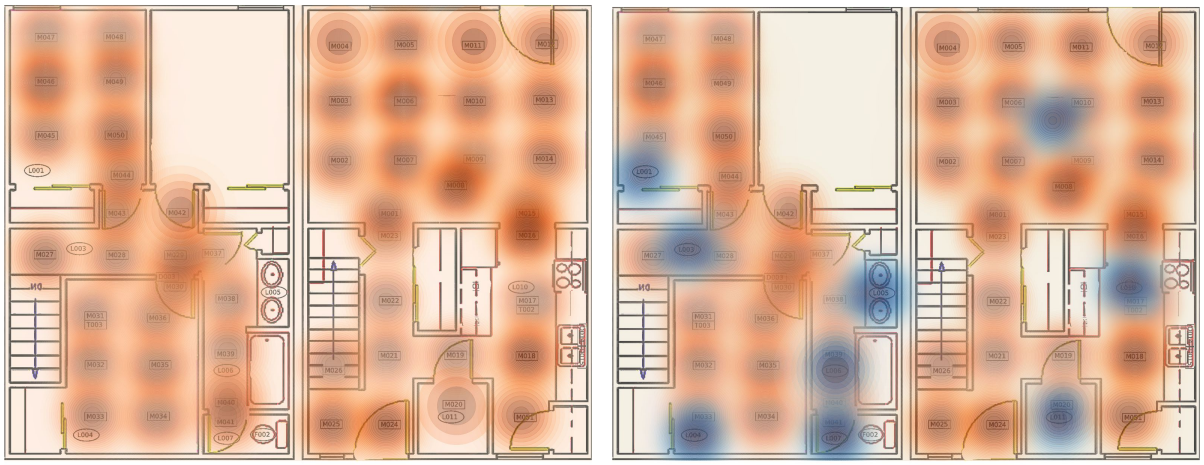


Figure 10. Heatmaps showing the differences between 2 weeks. On the left, there were no actuators, on the right, we have an interesting scenario worth mining.

We use Github and Gitflow for managing our collaboration, code review and code sharing between the team members. Our repository is located and can be accessed publicly at Github: https://github.com/nguyenshane/IoTMining

For performance measurement, we use the time library provided by Python and print the delta between the start and end of the pruning algorithms.

Design Document and Flow Chart

Our research started with a naive design solution, which we called the Naive Duration Based Pruning. In this algorithm, we looked for ON and OFF events and extracted the duration between them. The maximum duration we looked for was 15 minutes. With the naive algorithm design, we were able to generate the dataset and preliminary association rules that could be used to prove our hypothesis. However, we did see the slowness and complexity of the computation, while the pruned dataset is still quite large.

This led us to find different ways to mine the data. Our hope is to innovate ways to lower the computational complexity as well as the input dataset. We came up with 2 different strategies as shown in the figure below:

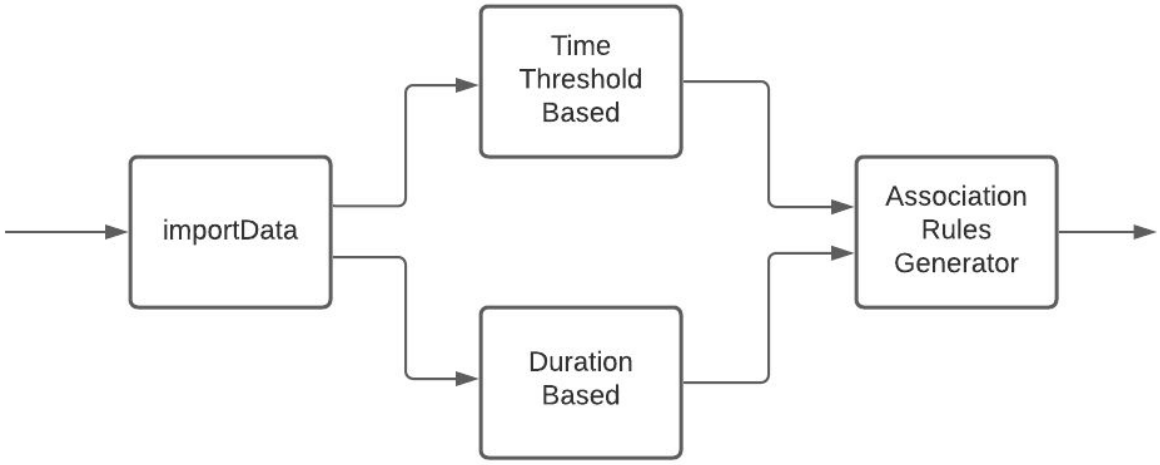


Figure 11. Flowchart of the pruning processes and association rules generation

We will examine each box in the flowchart:

- Import data: By running this file, it will pick up datasets specified in the datasets array, import to numpy array with various preliminary filters and then partition into weeks and write to the npy/prunedDataByWeek directory.
- Time Threshold Pruning: This process will pick up datasets in npy/prunedDataByWeek, prune the data based on 1 minute time threshold, combine with deduplication and filter segments that don't contain interesting device types to mine.
- Duration Pruning: This process will pick up datasets in npy/prunedDataByWeek, prune the data based on longer minute time threshold, combine with filtering for ON and OFF events. This contains the naive method we discussed above, and also the upgradedDurationPruning() method, which improves the performance by using predefined label sets and numpy filtering.
- Association Rules Generator: This function will search in the imported and pruned data, apply the apriori algorithm for each sliding window of 4 weeks and generate the rules.

V. Data Analysis and Discussion

Output and Data Analysis

After processing the data and performing a sliding window of size four to find the association rules in the data of four weeks each, we have generated thirty three sets of association rules out of thirty six weeks of data. An example an association rule we generate is showed in the following:

```
{BA_sink_Light} -> {BA_fan}
{BA_fan} -> {BA_sink_Light}
{LivRoom} -> {Liv_Light, Kitchen_Light}
```

The association rules above mean when the bathroom sink light is on, the bathroom fan must be on and vice versa. The third association rule means when the living room is occupied, the living room light and the kitchen light must be on. The first two association rules belong to the segment of a Thursday morning and the third one belongs to a Tuesday afternoon. However, these association rules are not fixed. As mentioned above, we generate our association rules by the time partitions of the day and by a sliding window of four weeks to mimic the incremental learning over time of collecting resident usage behavior data. We also find that the association rules change drastically over time and vary as time progresses.

Abnormal case explanation

We set out to build our program to detect the relationship between motion sensors and actuators. When we ran our program the first couple of test runs, we did not find any association rules generated from the first few weeks. At first, we thought there were logic errors in our program design, but testing with a test dataset yielded nominal results. Then, we decided to examine the data source itself and found out that for the six to seven weeks of living in the apartment, the two residents made no interaction with the lighting system (actuators). We do not know this was the residents actual behaviors or some error in collecting the data.

Discussion

We have essentially proved our hypotheses from the changing sizes and the variations of the association rules by the time partition of the day. From our results, we have shown that there are meaningful relationships and correlation patterns in usage across devices. Also, we have shown that a central system canlearn from the resident occupancy and usage behavior and partially take control of the smart home without any hard/fixed instructions setup by the residents. The following figures show the change overtime of the size of the association rules.

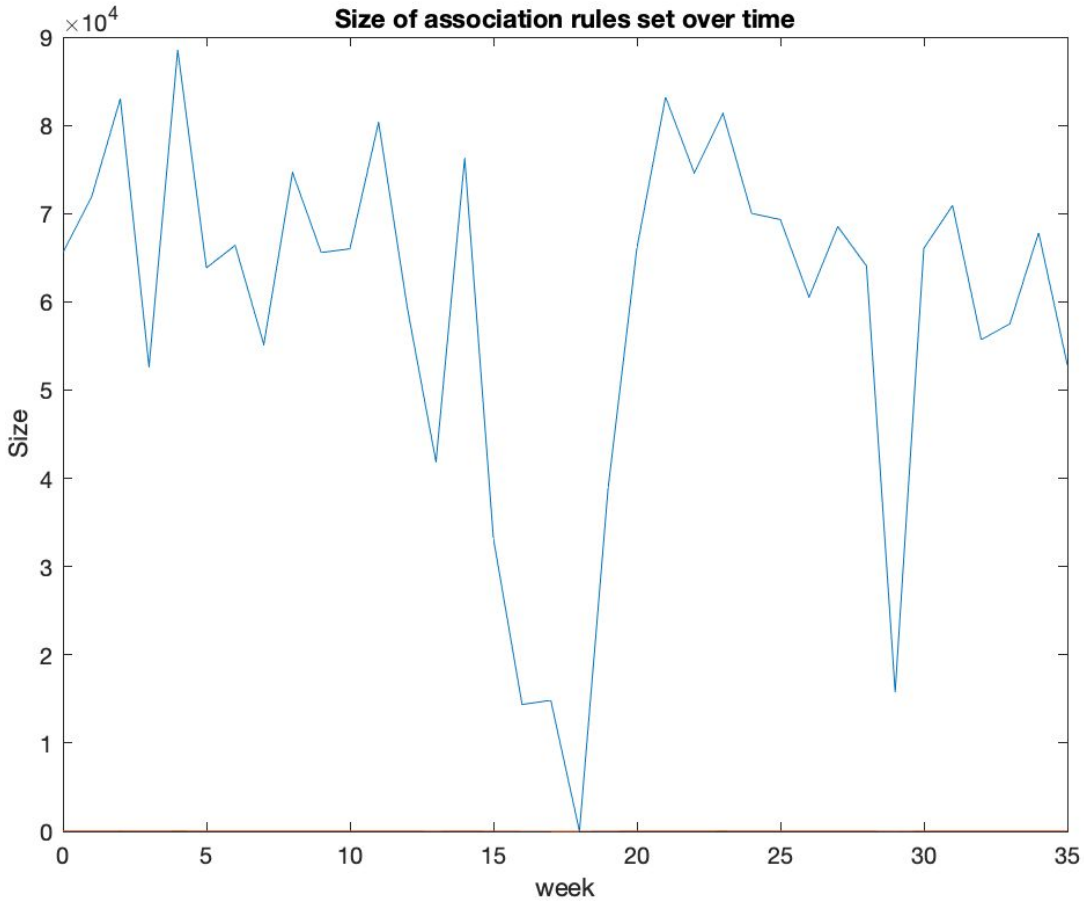


Figure 12. Size of association rules over time

Through the iterations of the 3 pruning methods, we also learned that the size of the dataset can be trimmed down significantly while keeping the accuracy and characteristic of the events. The Duration Pruning method is a great candidate for edge computing, where the computational time is usually limited. However, the Time Threshold Pruning method can be a good resource for discovering events and rules from multiple agnostic devices as it doesn't require the label sets. The following chart shows the differences between 3 methods we researched and developed:

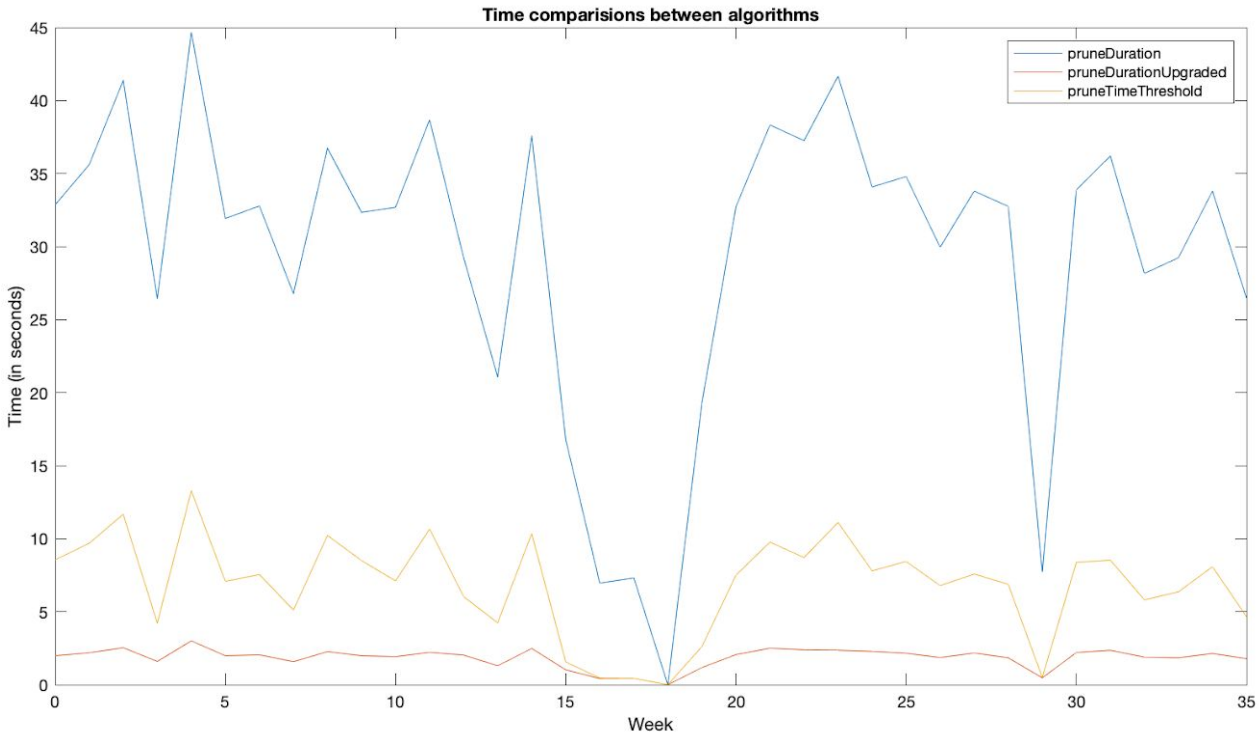


Figure 13. Comparisons between 3 pruning techniques in this research

VI. Conclusions and Recommendations

Conclusion

In this project, we aim to prove the possibility and necessity of a central system to mine occupancy and usage data from a smart home. We set out to find the relationship and patterns among devices rather than just focusing on one device. We hypothesize that the behavior of the residents will change over time and only an incrementally learning system can adapt to these changes. We have got our data set from a real living situation of two voluntary residents in Kyoto and set out to mine the patterns between motion sensors and lighting sensors. We have found meaningful association rules between these motion sensors and the lighting system. From these association rules, a central system can partially take control of the smart home without any preset/fixed instructions input by the residents. All they have to do is to live in and interact with the house.

Recommendations

In this project, we have found meaningful information in the interaction between motion sensors that trigger the lighting system (i.e. actuators as defined). We believe that there will be a lot of meaningful information that can be found from analyzing the correlation patterns between all the sensors in a smart home system. When combined with the incremental learning method we implemented in this project, we believe a central system can partially take control of the smart home based on the residents usage behavior. Therefore, we recommend extending future projects to more kinds of sensors and devices.

Furthermore, when analyzing the association rules generated by our program, we have noticed that there might be meaningful relationships between all these appliances, and these relations are formed by the residents. We believe a graph can be built from the frequent pairs with the edges are the implication of the association rules. From this graph, we hypothesize the existence of communities or clusters in the way the appliances are used.

On another note, we have also noticed that the amount of appliances used in a time partition are much less than the amount that aren't not used. Therefore, we also recommend mining the un-usage data and we believe that there is meaningful information in those data.

VIII. Bibliography

Atzori, Luigi & Iera, Antonio & Morabito, Giacomo. (2016). Understanding the Internet of Things: definition, potentials, and societal role of a fast evolving paradigm. Ad Hoc Networks. 56. 10.1016/j.adhoc.2016.12.004.

Billet, Benjamin. (2015). Data stream management system for the future Internet of Things.

Agrawal, Rakesh & Srikant, Ramakrishnan. (2000). Fast Algorithms for Mining Association Rules. Proc. 20th Int. Conf. Very Large Data Bases VLDB. 1215.

Chen, Yi-Cheng & Peng, Wen-Chih & Huang, Jiun-Long & Lee, Wang-Chien. (2015). Significant Correlation Pattern Mining in Smart Homes. ACM Transactions on Intelligent Systems and Technology. 6. 1-23. 10.1145/2700484.

Jia, Mengda & Srinivasan, Ravi. (2015). Occupant behavior modeling for smart buildings: A critical review of data acquisition technologies and modeling methodologies. 10.1109/WSC.2015.7408496.

Geng, Junwei & Qin, Lin. (2019). Research and Application of Community Population Information Association Model Based on IoT Multi-device Mining. BDIOT 2019: Proceedings of the 3rd International Conference on Big Data and Internet of Things. 93-98. 10.1145/3361758.3361767.

Ramijak, Dusan & Pal, Amitangshu & Kant, Krishna. (2018). Pattern mining based compression of IoT data. 1-6. 10.1145/3170521.3170533.

IX. Appendices

importData.py

```
#!/usr/bin/env python3

# Research: Incremental Learning from IoT for Smart Home Automation
# Authors: Nguyen Do, Quan Bach
# Usage:
# Import Data
# By running this file, it will pick up datasets in specified in datasets
array,
# import to numpy array with some filters and then partition into weeks

import os
import utils
from datetime import datetime, time, timedelta
from utils import sensorFilter
import numpy as np

datasets = ["./dataset/data"]
datasetsNames = [i.split('/')[-1] for i in datasets]

def loadDataset(filename):
    dataTable = []
    with open(filename, 'rb') as features:
        database = features.readlines()
        for i, line in enumerate(database):  # each line
            lineList = line.decode().split()  # split a line into a list
separated by spaces.
            # A line looks like this
            # 2009-08-24 00:00:19.034964    M050    ON  R1_Wandering_in_room
begin
            # Turn them into the following fields
            timestamp = None
            originalSensor = None
```

```
            sensor = None
            value = None
            activity = None
            dayOfWeek = None
            partitionTimeOfDay = None
            try:
                if 'M' == lineList[2][0] or 'L' == lineList[2][0]:
                    # choose only M, L sensors

                    originalSensor = str(np.array(lineList[2]))
                    sensor = sensorFilter(originalSensor)

                    if (sensor is None):
                        continue

                    if not ('.' in str(np.array(lineList[0])) +
str(np.array(lineList[1]))):
                        lineList[1] = lineList[1] + '.000000'
                    timestamp = (datetime.strptime(str(np.array(lineList[0]))
+ str(np.array(lineList[1])),

"%Y-%m-%d%H:%M:%S.%f"))
                    dayOfWeek = timestamp.weekday()
                    partitionTimeOfDay =
utils.timeInPartition(timestamp.hour)

                    value = (str(np.array(lineList[3])))

                    if len(lineList) == 4:  # if activity does not exist
                        activity = ''
                    else:  # if activity exists
                        activity = str(' '.join(np.array(lineList[4:])))
                        if 'begin' in activity:
                            #activity = re.sub('begin', '', des)
                            if activity[-1] == ' ':  # if white space at the
end
```

```
                                activity = activity[:-1]  # delete white
space
                            if 'end' in activity:
                                if activity[-1] == ' ':  # if white space at the
end
                                    activity = activity[:-1]  # delete white
space
                        dataTable.append([timestamp,
                                          dayOfWeek,
                                          partitionTimeOfDay,
                                          originalSensor,
                                          sensor,
                                          value,
                                          activity])
                except IndexError:
                    print(i, line)


    features.close()

    return dataTable

def partitionDataByWeek(path):
    dataTable = np.load(path, allow_pickle=True)
    startDate = datetime.combine(dataTable[0][:1][0].date(), time(0))
    finalDate =
datetime.combine(dataTable[dataTable.shape[0]-1][:1][0].date()
                                 + timedelta(days=1), time(0))

    currentWeek = 0

    while True:
        nextEndDate = startDate + timedelta(days=7)
        idx = ((dataTable[:, 0] >= startDate)
            & (dataTable[:, 0] < nextEndDate)
            & (dataTable[:, 0] < finalDate))
```

```
        if not os.path.exists('./npy/dataByWeek/'):
            os.makedirs('./npy/dataByWeek/')

        np.save('./npy/dataByWeek/week' + str(currentWeek), dataTable[idx])
        print('Saved ./npy/dataByWeek/week' + str(currentWeek))

        if (nextEndDate >= finalDate):
            break
        else:
            startDate = nextEndDate
            currentWeek += 1

if __name__ == '__main__':

    if not os.path.exists('npy/datanpy.npy'):
        # Generate the full dataset
        for filename in datasets:
            datasetName = filename.split("/")[-1]
            print('Loading ' + datasetName + ' dataset ...')
            dataTable = loadDataset(filename)

            dataTable = np.array(dataTable, dtype=object)
            if not os.path.exists('npy'):
                os.makedirs('npy')

            np.save('./npy/' + datasetName + 'npy', dataTable)
            print('Saved ' + datasetName)
    # Partition full dataset to week
    partitionDataByWeek('npy/datanpy.npy')
```

timeThresholdPruning.py

```
#!/usr/bin/env python3

# Research: Incremental Learning from IoT for Smart Home Automation
# Authors: Nguyen Do, Quan Bach
# Usage:
# Time Threshold Pruning
# By running this file, it will pick up datasets in npy/prunedDataByWeek,
# prune the data based on minute time threshold, combine with deduplication
# and filter segments that doesn't have interesting device type to learn

import utils
import os
from datetime import timedelta
import numpy as np
import time

def timeThresholdPruning(week, dataTable):
    startProcessTime = time.process_time()
    if (dataTable is None):
        filename = "./npy/dataByWeek/week" + str(week) + ".npy"
        dataTable = np.load(filename, allow_pickle = True)

    dataSize = len(dataTable)

    prunedDataTable = []
    startTime = dataTable[0][:1][0]

    dataTableIndex = 0
    segmentCount = 0

    while True:

        nextTimeSegment = startTime + timedelta(minutes=
utils.timePruningThreshold)
        #print('next time', nextTimeSegment)
```

```
        idx = (dataTable[:, 0] >= startTime) & (dataTable[:, 0] <
nextTimeSegment)

        currentSegment = dataTable[idx]

        dataTableIndex += len(currentSegment) - 1
        #print('segment', currentSegment, segmentCount)

        prunedSegment = pruneByDevice(currentSegment, segmentCount)

        if (len(prunedSegment) > 0):
            prunedDataTable.extend(prunedSegment)

        if (len(dataTable) > 1):
            segmentCount += 1

            # trim off dataTable as we go
            dataTable = dataTable[len(currentSegment): len(dataTable), :]

            if (len(dataTable) == 0):
                break
            startTime = dataTable[0][:1][0]
        else:
            break

    prunedDataTable = np.array(prunedDataTable, dtype=object)
    if not os.path.exists('npy/prunedDataByWeek'):
        os.makedirs('npy/prunedDataByWeek')
    np.save('./npy/prunedDataByWeek/week' + str(week), prunedDataTable)

    endProcessTime = time.process_time() - startProcessTime
    print('week:', week, 'excecution time:', endProcessTime)
    if not os.path.exists('./ProgOutput/'):
        os.makedirs('./ProgOutput/')
    outFile = open('./ProgOutput/pruneFineTimeThreshold-Measure.txt','a+')
    outFile.writelines("{}, {}, {}\n".format(week, dataSize, endProcessTime))
    outFile.close()
```

```
    print('prunedDataTable', prunedDataTable)

    return

def pruneByDevice(segment, segmentCount):
    deviceList = segment[:, 4]
    for index, device in enumerate(deviceList):
        if (device != None):
            if ('Light' in device) or ('fan' in device):
                segment = pruneDuplication(segment)
                newCol = np.full((1, len(segment)), segmentCount)
                segment = np.insert(segment, 3, newCol,  axis=1)
                return segment.tolist()

    return []

def pruneDuplication(segment):
    onIdx = (segment[:, 5] == "ON")
    onSegment = segment[onIdx]
    uniqueKeys, indices = np.unique(onSegment[:, 4], return_index=True)

    newSegment = onSegment[indices]
    return newSegment

if __name__ == '__main__':
    path = './npy/dataByWeek/'
    weekCount = len([name for name in os.listdir(path) if
os.path.isfile(os.path.join(path, name))])
    for i in range (0, weekCount):
        filename = "./npy/dataByWeek/week" + str(i) + '.npy'

        timeThresholdPruning(i, None)


#timeThresholdPruning(9, None)
```

durationPruning.py

```
#!/usr/bin/env python3

# Research: Incremental Learning from IoT for Smart Home Automation
# Authors: Nguyen Do, Quan Bach
# Usage:
# Duration Pruning
# By running this file, it will pick up datasets in npy/prunedDataByWeek,
# prune the data based on longer minute time threshold, combine with
filtering
# for ON and OFF events. upgradedDurationPruning() improves the performance
by using
# predefined labelset and numpy filtering.

from utils import labelSet, timeStampDiff, durationThreshold, basketsKeySet
from collections import defaultdict
import numpy as np
import os
import time

def durationPruning(week, filename):
    startProcessTime = time.process_time()
    """Return a map of routine items i.e. a baskets for A-priori"""
    """keys of the map are the combinations of dayOfWeek + partitionTimeOfDay
i.e. 00, 01, 02...62 """
    # an item is routine if its duration is longer then the time pruning
threshold
    # a routine item can be occupancy: motion sensors ;and usage: lightning
sensor

    routineItemsMap = {}
    dataTable = np.load(filename, allow_pickle = True)
    dataSize = len(dataTable)

    for key in basketsKeySet:
        timeStampList = []
        routineItems = []
```

```
        for elem in labelSet: # scan the data for each sensor ID
            totalDuration = 0
            endPoint = dataTable.shape
            lookingForOn = True # flag to alternate looking for ON and not ON
(i.e. OFF)
            for row in range(0,endPoint[0]):
                keyInTable = str(dataTable[row][1])+str(dataTable[row][2])
                if key == keyInTable:
                    #in case reach the end but OFF not found (i.e. device has
only ON)
                    if row == (endPoint[0]-1) and not lookingForOn:
                        timeStamp = str(dataTable[row][0]).split()
                        timeStampList.append(timeStamp[1])

                    #looking for ON
                    if elem == dataTable[row][3] and lookingForOn and "ON" ==
str(dataTable[row][4]):
                        timeStamp = str(dataTable[row][0]).split()
                        timeStampList.append(timeStamp[1])
                        lookingForOn = False

                    #looking for OFF
                    if elem == dataTable[row][3] and not lookingForOn and
"OFF" == str(dataTable[row][4]):
                        timeStamp = str(dataTable[row][0]).split()
                        timeStampList.append(timeStamp[1])
                        lookingForOn = True

                    #compute the duration of the intervals between ON and OFF
                    if len(timeStampList) == 2:
                        totalDuration +=
timeStampDiff(timeStampList[0],timeStampList[1])
                        timeStampList.clear()

            if totalDuration > durationThreshold : #duration pruning
                routineItems.append((elem,int(totalDuration)))
```

```
        routineItemsMap[key] = routineItems
    endProcessTime = time.process_time() - startProcessTime
    print('week:', week, 'excecution time:', endProcessTime)
    if not os.path.exists('./ProgOutput/'):
        os.makedirs('./ProgOutput/')
    outFile = open('./ProgOutput/pruneDuration-Measure.txt','a+')
    outFile.writelines("{}, {}, {}\n".format(week, dataSize, endProcessTime))
    outFile.close()
    return routineItemsMap



def upgradedDurationPruning(week, filename):
    startProcessTime = time.process_time()
    """Return a map of routine items i.e. a baskets for A-priori"""
    """keys of the map are the combinations of dayOfWeek + partitionTimeOfDay
i.e. 00, 01, 02...62 """
    # an item is routine if its duration is longer then the time pruning
threshold
    # a routine item can be occupancy: motion sensors ;and usage: lightning
sensor
    #upgraded version to run faster by taking advantages of numpy table
features

    routineItemsMap = defaultdict(list)
    routineItemMapIDOnly = defaultdict(list)
    dataTable = np.load(filename, allow_pickle = True)
    dataSize = len(dataTable)

    for elem in labelSet:
        for i in range(0,7):
            for j in range(0,3):
                idx = ((dataTable[:,4] == elem) & (dataTable[:,1] == i) &
(dataTable[:,2] == j))
                opTable = dataTable[idx]
                lookingForOn = True
                endPoint = opTable.shape
```

```
                timeStampList = []
                totalDuration = 0
                firstOnFound = False
                firstOn = ''
                lastOff = ''
                for row in range(0,endPoint[0]):

                    #looking for last off
                    if "OFF" == str(opTable[row][5]):
                        lastOff = str(opTable[row][0]).split()[1]

                    #in case reach the end but OFF not found (i.e. device has
only ON)
                    if row == (endPoint[0]-1) and not lookingForOn:
                        timeStamp = str(opTable[row][0]).split()
                        timeStampList.append(timeStamp[1])

                    #looking for ON
                    if lookingForOn and "ON" == str(opTable[row][5]):
                        if not firstOnFound:
                            firstOn = str(opTable[row][0]).split()[1]
                            firstOnFound = True
                        timeStamp = str(opTable[row][0]).split()
                        timeStampList.append(timeStamp[1])
                        lookingForOn = False

                    #looking for OFF
                    if not lookingForOn and "OFF" == str(opTable[row][5]):
                        timeStamp = str(opTable[row][0]).split()
                        timeStampList.append(timeStamp[1])
                        lookingForOn = True

                    #compute the duration of the intervals between ON and OFF
                    if len(timeStampList) == 2:
                        totalDuration +=
timeStampDiff(timeStampList[0],timeStampList[1])
                        timeStampList.clear()
```

```
                if totalDuration > durationThreshold : #duration pruning
                    key = str(i) + str(j)

routineItemsMap[key].append((elem,int(totalDuration),str(firstOn),str(lastOff
)))
                    routineItemMapIDOnly[key].append(elem)
    endProcessTime = time.process_time() - startProcessTime
    print('week:', week, 'excecution time:', endProcessTime)
    if not os.path.exists('./ProgOutput/'):
        os.makedirs('./ProgOutput/')
    outFile = open('./ProgOutput/pruneDurationUpgraded-Measure.txt','a+')
    outFile.writelines("{}, {}, {}\n".format(week, dataSize, endProcessTime))
    outFile.close()
    return routineItemsMap, routineItemMapIDOnly


if __name__ == '__main__':
    path = './npy/dataByWeek/'
    weekCount = len([name for name in os.listdir(path) if
os.path.isfile(os.path.join(path, name))])
    for i in range (0,weekCount):
        filename = "./npy/dataByWeek/week" + str(i) + '.npy'
        #uncomment to demonstrate the different between two algorithms
        #durationPruning(i, filename) #old version, much slower
        table1, table2 = upgradedDurationPruning(i, filename)
        currentWeek = '===== Week ' + str(i+1) + ' ====='
        print(currentWeek)
        for key in basketsKeySet:
            print("{} : {}".format(key,table2[key]))
        print('\n')

        if not os.path.exists('./ProgOutput/'):
                os.makedirs('./ProgOutput/')
        outFile = open('./ProgOutput/weeklyRoutineActivities.txt','a+')
        currentWeek = '===== Week ' + str(i+1) + ' ====='
        outFile.write(currentWeek)
```

```
        outFile.write('\n')
        for key in basketsKeySet:
            outFile.writelines("{} : {}".format(key,table2[key]))
            outFile.write('\n')
            outFile.write('\n')
        outFile.close()



#test output

#table = upgradedDurationPruning('./npy/dataByWeek/week11.npy')
# table = durationPruning('./npy/dataByWeek/week11.npy')

#for key in basketsKeySet:
    #print("{} : {}".format(key,table[key]))
```

timeThresholdBasedAssociationRulesGenerator.py

```
#!/usr/bin/env python3

# Research: Incremental Learning from IoT for Smart Home Automation
# Authors: Nguyen Do, Quan Bach
# Usage:
# Time Threshold Based Association Rules Generator
# By running rulesGenerator(), this function will get into the imported and
pruned data,
# runs the apriori algorithm per sliding window of 4 weeks and generate the
rules

import utils
from utils import sizeOfSlidingWindow
import os
from datetime import datetime, time, timedelta
import numpy as np
from numpy.core.defchararray import find
from efficient_apriori import apriori

def findFrequentSets(weeks):
    dataTable = None
    baskets = {} # dictionary
    for week in weeks:
        filename = "./npy/prunedDataByWeek/week" + str(week) + ".npy"
        weekDataTable = []
        try:
            weekDataTable = np.load(filename, allow_pickle = True)
        except:
            print(filename, "doesn't exist.")


        if (len(weekDataTable) > 0):
            for dayInWeek in range(0,6):
                for partitionTimeIndex in utils.timePartitionMap:
                    idx = (weekDataTable[:, 1] == dayInWeek) &
(weekDataTable[:, 2] == int(partitionTimeIndex))
```

```
                    currentDataTable = weekDataTable[idx]
                    id = str(dayInWeek) + partitionTimeIndex
                    # print('currentDataTable', tuple(set(currentDataTable[:,
3])))

                    if (len(currentDataTable) == 0):
                        continue

                    uniqueSegments = np.unique(currentDataTable[:,4])

                    for uniqueSegment in uniqueSegments:
                        if (not id in baskets):
                            baskets[id] = [tuple(set(currentDataTable[:,
5]))]
                        else:
                            baskets[id].extend([tuple(set(currentDataTable[:,
5]))])

    rulesList = []
    itemsetsList = []
    sizeOfRules = []
    for id in baskets:
        itemsets, rules = apriori(baskets[id], min_support=0.5,
min_confidence=1, max_length=2)
        itemsetsList.append(itemsets)

        rulesFilter = filter(lambda rule: findActuator(rule), rules)
        filteredRules = []
        for rule in rulesFilter:
            filteredRules.append(rule)

        rulesList.append([id, filteredRules])

    return itemsetsList, rulesList, len(baskets)

def findActuator(tup):
```

```
    return all((any(substr in e for substr in ['Light', 'fan'])) for e in
tup.rhs)

def rulesGenerator():

    path = './npy/prunedDataByWeek/'
    weekCount = len([name for name in os.listdir(path) if
os.path.isfile(os.path.join(path, name))])

    #compute the total number of sliding window
    windowCount = weekCount - sizeOfSlidingWindow + 1  #algor:(n-k+1) with k
is the size of sliding window

    if not os.path.exists('./ttRules/'):
            os.makedirs('./ttRules/')

    ruleSizeFile = open('./ttRules/ruleSize.txt', 'a+')
    for i in range(0, windowCount+1):
        itemsetsList, rulesList, basketsSize =
findFrequentSets(tuple(range(i, i+3)))
        outFile = open('./ttRules/f' + str(i) + 't' +str(i+3) + '.txt', 'w')
        outFile.write("size of baskets: " + str(basketsSize))
        outFile.write("\n")
        for i in range(0, basketsSize):
            ruleSizeFile.write("\n")
            outFile.write("date time segment: " + str(rulesList[i][0]) +
"\n")
            outFile.write("size of rule list: " + str(len(rulesList[i][1])))
            outFile.write("\n")
            outFile.write(str(rulesList[i]))
            outFile.write("\n")
            outFile.write("\n")
        outFile.close()

rulesGenerator()
```

Utils.py

```
# Research: Incremental Learning from IoT for Smart Home Automation
```

```
# Authors: Nguyen Do, Quan Bach
# Usage:
# Utils
# This file contains configurable variables, mapping and common functions

sizeOfSlidingWindow = 4 #the number of weeks in the sliding window for
apriori
timePruningThreshold = 1 # time threshold in minute
durationThreshold = 30 # duration threshold in minute

startDate = "2009-08-24" # start and end date got from the study
endDate = "2010-05-01"

timePartitionMap = {"0": {"start": 6, "end": 11}, # Morning
                    "1": {"start": 12, "end": 19}, # Afternoon
                    "2": {"start": 20, "end": 5}} # Everning

lightningMap = {"L001" : "R1room_Light",
                "L002" : "R3room_Light",
                "L003" : "uHall_Light",
                "L004" : "R2room_Light",
                "L005" : "BA_sink_Light",
                "L006" : "BA_tub_Light",
                "L007" : "BA_fan",
                "L008" : "Liv_Light",
                "L009" : "dHall_Light",
                "L010" : "Kitchen_Light"}

"""
Organizing sensors IDs into sets.

"""

LivingRoomSensorSet =    ( "M001",
                           "M002",
                           "M003",
                           "M004",
```

```
                          "M005",
                          "M006",
                          "M007",
                          "M008",
                          "M009",
                          "M010",
                          "M011",
                          "M012",
                          "M013",
                          "M014",
                          "M015")
R1roomSensorSet =      ("M044",
                        "M045",
                        "M046",
                        "M047",
                        "M048",
                        "M049",
                        "M050")

R2roomSensorSet =     ("M030",
                       "M031",
                       "M032",
                       "M033",
                       "M034",
                       "M035",
                       "M036")


upstairsHallSensorSet =     ("M027",
                             "M028",
                             "M029")

downstairsHallSensorSet =     ("M021",
                               "M022",
                               "M023",
                               "M024",
                               "M025",
```

```
                              "M026")

kitchenSensorSet =    ("M016",
                       "M017",
                       "M018",
                       "M051" )

bathroomSensorSet =    ("M037",
                        "M038",
                        "M039",
                        "M040",
                        "M041")


labelSet = ("LivRoom",
            "R1room",
            "R2room",
            "UpstairsHall",
            "DownstairsHall",
            "Kitchen",
            "Bathroom",
            "R1room_Light",
            "R3room_Light",
            "uHall_Light",
            "R2room_Light",
            "BA_sink_Light",
            "BA_tub_Light",
            "BA_fan",
            "Liv_Light",
            "dHall_Light",
            "Kitchen_Light")

labelTestSet = ("Kitchen_Light",
                "R1room",
                "R2room",
                "Liv_Light",
                "dHall_Light")
```

```
basketsKeySet = ("00", "01", "02",

                 "10", "11", "12",

                 "20", "21", "22",

                 "30", "31", "32",

                 "40", "41", "42",

                 "50", "51", "52",

                 "60", "61", "62")


sensorGroupList = [  LivingRoomSensorSet,
                     R1roomSensorSet,
                     R2roomSensorSet,
                     upstairsHallSensorSet,
                     downstairsHallSensorSet,
                     kitchenSensorSet,
                     bathroomSensorSet]




"""
Function sensorFilter
this funcntion will read in the sensor ID and return the primary ID of each
group
primary ID for each group is the frist element of that group
"""
def sensorFilter(sensorID):
    for element in sensorGroupList:
            if (element.count(sensorID) > 0):
                return labelSet[sensorGroupList.index(element)]
```

```
    if sensorID in lightningMap.keys():
        return lightningMap.get(sensorID)
    return None #return None if not found in the group list




def timeInRange(start, end, x):
    """Return true if x is in the range [start, end]"""
    if start <= end:
        return start <= x <= end
    else:
        return start <= x or x <= end

def timeInPartition(x):
    for (key) in timePartitionMap:
        if (timeInRange(timePartitionMap[key]["start"],
            timePartitionMap[key]["end"], x)):
            return int(key)

def timeStampDiff (timeStamp1, timeStamp2):
    """ Return the absolute different between two timeStamps"""
    #a timeStamp has this format HH:MM:SS:mmmmmmm
    time1 = timeStamp1.split(':')
    time2 = timeStamp2.split(':')
    return abs((float(time1[0])*60 + float(time1[1]) +
float(time1[2])*(1/60)) - (float(time2[0])*60 + float(time2[1]) +
float(time2[2])*(1/60)))
```

heatmapGenerate.py

```
#!/usr/bin/env python3

# Research: Incremental Learning from IoT for Smart Home Automation
# Authors: Nguyen Do, Quan Bach
# Usage:
# Heatmap Generator
# By running this file, it will pick up datasets in npy/prunedDataByWeek,
# and generate heat maps for analysis purpose

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from PIL import Image

#2D Gaussian function
def twoD_Gaussian(x, y, xo, yo, sigma_x, sigma_y):
    a = 1./(2*sigma_x**2) + 1./(2*sigma_y**2)
    c = 1./(2*sigma_x**2) + 1./(2*sigma_y**2)
    g = np.exp( - (a*((x-xo)**2) + c*((y-yo)**2)))
    return g.ravel()


def transparent_cmap(cmap, N=255):
    "Copy colormap and set alpha values"

    mycmap = cmap
    mycmap._init()
    mycmap._lut[:,-1] = np.linspace(0, 0.6, N+4)
    return mycmap


if __name__ == "__main__":

    dataTable = np.load('./npy/dataByWeekNoFilter/week0.npy',
allow_pickle=True)
    sensorArr = dataTable[:,3]
```

```
    unique, counts = np.unique(sensorArr, return_counts=True)
    countDict = dict(zip(unique, counts))

    if bool(countDict):
        maxCount = max(countDict.values())

    locationDict = {}
    filename = "./sensorLocation.txt"
    with open(filename, 'rb') as locations:
        lines = locations.readlines()
        for line in lines:
            lineList = line.decode().split()
            locationDict.update({str(lineList[0]):
(float(lineList[1]),float(lineList[2]))})


    #Use base cmap to create transparent
    redcmap = transparent_cmap(plt.cm.Oranges)
    bluecmap = transparent_cmap(plt.cm.Blues)


    # Import image and get x and y extents
    I = Image.open('./heatmap.png')
    p = np.asarray(I).astype('float')
    w, h = I.size
    y, x = np.mgrid[0:h, 0:w]

    #Plot image and overlay colormap
    fig, ax = plt.subplots(1, 1)

    for key in locationDict:
        if key in countDict.keys():
            Gauss = twoD_Gaussian(x, y, locationDict[key][0] *x.max(),
locationDict[key][1] *y.max(), .08*x.max(), .08*y.max())
            if str(key)[0] == "M":
                value = int(((countDict[key])/maxCount)*200)
```

```
                cb = ax.contourf(x, y, Gauss.reshape(x.shape[0], y.shape[1]),
value , cmap=redcmap)
            else:
                value = int(((countDict[key]*101)/maxCount)*100)
                cb = ax.contourf(x, y, Gauss.reshape(x.shape[0], y.shape[1]),
value , cmap=bluecmap)


    plt.axis('off')
    ax.imshow(I)
    plt.show()
```