PREDICTION OF INHABITANT ACTIVITIES

IN SMART ENVIRONMENTS

By

BRYAN D. MINOR

A dissertation submitted in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

AUGUST 2015

To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of

BRYAN D. MINOR find it satisfactory and recommend that it be accepted.

Diane J. Cook, Ph.D., Co-Chair

Thomas R. Fischer, Ph.D., Co-Chair

Janardhan Rao Doppa, Ph.D.

ACKNOWLEDGMENTS

This work has been the culmination of many years of support and encouragement from many mentors, family, and friends.

I would like to thank my co-advisor Dr. Diane Cook, who has been an invaluable help in my research. She provided me an opportunity to join the CASAS group as an undergraduate freshman which led tremendously to my interest in smart environments. Her suggestions along the way and her willingness to help each of her students reach their potential have been greatly encouraging.

I am also grateful to my co-advisor Dr. Thomas Fischer for his support in both my undergraduate and graduate work. It was in his class that I found an interest in signal processing and he has continued to help me pursue that interest in my research. I am thankful for his continued insights.

I would also like to thank Dr. Janardhan Rao Doppa for his input and support. His suggestions have helped spur new solutions to many of the problems we have faced.

I would also like to thank others from the CASAS and IGERT groups who have supported this work, including Aaron, Brian, Randall, and Jim, among many others.

I would also like to thank Matt, Tyrel, Jason C. and Adam R. for their friendship and encouragement. All of you have supported me throughout this whole process and I am thankful for your friendship over the years. My parents, Pam and Dave, have been key to my success, along with my brother Jason. Your love and support have always been steadfast. My wonderful wife Ciara has been my biggest supporter and encouragement throughout this journey. I am

so thankful for you! Finally I would like to thank the Lord, Who has enabled me to be who I am today through His goodness.

PREDICTION OF INHABITANT ACTIVITIES

IN SMART ENVIRONMENTS

Abstract

by Bryan D. Minor, Ph.D.
Washington State University
August 2015

Co-Chairs: Diane J. Cook and Thomas R. Fischer

Human activity prediction is a challenging problem which poses a number of machine learning challenges. Predicting occurrences of activities is valuable in understanding human behavior and creating services that not only react to human activities, but proactively anticipate and prepare for future activities in advance. We hypothesize that human activities can be predicted from smart environment sensor data. In order to validate this hypothesis we introduce machine learning-based methods to create an Activity Predictor (AP). AP learns a model of inhabitant behavior from sensor observations of their daily lives. It then automatically generates numeric predictions of the time until future occurrences of activities of interest. These predictions can be used to facilitate a variety of smart environment services including activity prompting and energy management systems.

We explore the utility of drawing contextual features from sensor event data in order to inform the predictor. These features draw on the information provided in the temporal distribution and the values of the sensor events. We further develop a method of using information about activity relationships to provide joint activity prediction. These connections are established through the use of prediction lag features which supplement other contextual information in a recurrent model. We also propose a method of using a combination of independent and

recurrent predictors in a two-pass fashion to further enhance activity learning.

We utilize data collected from several CASAS smart environment datasets to validate the methods in support of our hypothesis. We also explore the use of various evaluation metrics and underscore the importance of using a combination of metrics to fully understand activity prediction performance. Our findings suggest that the proposed automated solutions are able to predict human activities with reasonable accuracy. To further evaluate our methods we have deployed them as part of two pilot tests of our CAFE prompting app. Predictions were found to be less than 30 minutes in most cases and the prompts were found to have a noticeable effect on participants' behavior.

Contents

List of Tables

List of Figures

Dedication

This work is dedicated to my wife, Ciara, and to my parents Pam and Dave. Without their support I would not be where I am today.

CHAPTER 1. INTRODUCTION

Predicting human activities is an important component of services across a variety of areas. In order to provide useful and meaningful interactions with human users, it is important to be able to predict how they will act. Traffic management systems need to anticipate drivers' movements to ensure the most efficient flow of vehicles in a city. Electrical distributors rely on predictive systems to help them manage generation resources and meet peaks in demand. Retailers make major business decisions based on predictions of consumer purchases throughout the year. These and many other important systems rely on accurate predictions of human behavior. They require knowing what will happen and when, with incorrect predictions leading to results ranging from inconvenience to large economic losses.

However, despite the importance placed on activity prediction, it is often a difficult task to do accurately. Data on human behavior can be scarce or difficult to keep updated. Humans' behavior may also shift over time as their lives change, such that simple rule-based predictions do not suffice. Although many human behaviors are cyclic in nature, they can vary in ways that are difficult for algorithms to grasp. These and many other concerns make activity prediction an important yet difficult challenge.

The goal of this project is to demonstrate the ability to predict human activities through contextual information found in observations of their actions. In many situations where activity prediction is warranted, sensor systems can be used to collect observations of users' behavior over time. These observations are rich in information that can be used to predict users' future activities. However, the challenge lies in extracting this information from the data and using it

to form accurate predictions of future activities. In this dissertation we explore ways to address this challenge.

Although prediction of human activities is important for a variety of domains, we examine the problem in the context of predicting activities in smart environments. Many of the techniques developed here are applicable to other prediction problems with similar characteristics, however. A smart environment is defined by Cook and Das [1] as "one that is able to acquire and apply knowledge about an environment and also adapt to its inhabitants in order to improve their experience in that environment." Smart environments utilize a variety of sensors and actuators to monitor inhabitants' behavior and improve their comfort, safety, and overall wellbeing.

Much recent research has focused on utilizing smart environments to help adults "age in place" by providing technologies that allow older adults to live independently in their own homes longer. This is an important application: it is estimated that the population of United States adults over age 64 will reach 88.5 million by 2050, twice the 2010 number of 40.2 million [2]. Many of these older adults will also be suffering from cognitive illnesses. The number of those suffering from dementia is expected to increase from 18 million in 2004 to 34 million by 2025 [3]. Many of those suffering from cognitive illnesses will require increased care, leading to increasing demand for healthcare services. The use of smart environments can help these adults live safely in their homes and reduce some of the need for caregiver support.

One aspect of daily living that can be improved with a smart environment is the regular, independent performance of activities of daily living (ADLs). ADLs consist of daily functional tasks, such as cooking, eating, or taking medicine [4]. Many of those suffering from cognitive impairments have difficulty initiating and completing ADLs in order to live independently. A

smart environment can be utilized to provide monitoring and feedback to both inhabitants and caregivers regarding performance of ADLs. However, a smart environment must be able to effectively predict when activities will occur in order to provide these services.

To provide effective activity prediction in a smart environment (and many similar settings) an activity prediction method should meet the following goals:

1. **Predict the numeric time of activity occurrences.** Many smart environment services are dependent upon accurate knowledge of when future activities will occur. Activity reminder prompts must be issued at the correct time to maximize effectiveness. Energy-saving actions often require preparing in advance to adjust for expected activities in the most efficient manner. Thus, it is important for the activity prediction method to provide accurate numeric predictions indicating when activities are predicted to occur.

2. **Utilize sensor data with minimal input from users.** Inhabitant and caregiver time is limited and therefore valuable. The activity prediction method should require as little input as possible for initial setup. During operation, required input should be kept to a minimum to avoid placing additional burden on inhabitants. To this end, the activity prediction algorithm should be able to run autonomously without specific knowledge of inhabitants' schedules or preferences. Instead, it should utilize contextual information from the smart environment sensor data to form activity predictions.

3. **Model non-linear activity relationships.** The relationships between sensor events and activities is complex and often non-linear. The activity prediction method must be able to model these non-linear relationships effectively.

4. **Form predictions quickly.** The predictions are often used in near-real-time situations

as new sensor events are collected. In order to facilitate this, the prediction method should be able to compute new predictions quickly.

We hypothesize that activity prediction can be automated and that by introducing machine learning and time series techniques to automate activity prediction these goals can be met. In this dissertation we present activity prediction methods which validate this hypothesis. These methods combine the extraction of useful features from smart environment data with various regression learning techniques to form activity predictions.

The major contributions of this dissertation include:

1. Present a general framework for activity prediction using feature extraction and regression learners to predict multiple activities in a smart environment.

2. Analyze various evaluation metrics and their effectiveness in describing activity prediction performance.

3. Describe useful features for activity prediction and compare their effect on activity prediction performance.

4. Explore ways in which predictions for various activities can be used jointly to improve performance.

5. Evaluate the effectiveness of the predictions for use in prompting applications.

In the remainder of this dissertation we describe the problem of activity prediction and our contributions to the field. We begin with a review of related work (Chapter 2). Then, we formally describe the activity prediction problem (Chapter 3) and present our Activity Predictor algorithms (Chapter 4). We review various metrics for evaluating performance (Chapter 5),

then describe the CASAS smart home system, which is used to validate our proposed methods (Chapter 6). We then describe experiments to validate our proposed methods (Chapter 7). Finally, we present results from two prompting applications (Chapter 8) and conclusions (Chapter 9).

CHAPTER 2. RELATED WORK

Previous research related to activity prediction has been performed across a number of disciplines. Researchers have used activity prediction in fields such as power systems, economics, and healthcare, sometimes for many decades. Solutions to the problem in the literature range from sequential prediction to numeric time series forecasting methods.

In this chapter we first describe work in activity sequence prediction in smart homes and works based on time series models. We then discuss previous works related to activity prompting. Lastly, we examine works related to joint predictor models.

2.1 Sequence Prediction

Work on activity prediction has followed two general approaches. The first approach is sequence prediction. The goal of this approach is to predict the next symbol in a sequence given the symbols that occur before it. A predictor is trained on a sequence of symbols $S = s_1, s, s_2, \ldots, s_t$ and is then used to predict the next symbol, s_{t+1} at time $t + 1$. In activity prediction, the sequence of symbols represents a sequence of consecutive activities. An expert or activity recognition algorithm is used to determine a sequence of activities that occur from a series of sensor events. The goal of the sequence predictor is to then predict which activity will occur next.

Many sequence prediction techniques use derivatives of the LZ78 compression algorithm [5]. This algorithm was adapted by Feder et al. for use in sequence prediction [6]. These models analyze the sequence of symbols that occurs and forms a Markov model for predicting what

symbol will be next. The model was further refined using an information-theoretic framework for mobility tracking in cellular networks with LeZi-Update [7].

LZ78 was adapted for smart environment activity prediction with the Active LeZi algorithm [8–10]. Active LeZi provides better prediction convergence by using a window of sequence contexts, providing reasonable sequence prediction accuracy. A further refinement for smart environment applications is the SPEED algorithm, which uses appliance usage patterns in the home [11]. Information about activity relationships has also been used with Active LeZi [12]. Other sequence prediction algorithms use Markov models [13] or Bayesian networks [14]. Others have explored sequence prediction for predicting human action from video observations [15–17] and user locations [18]. The sequence prediction method was expanded to include predicting if event sequences would happen within a specified time interval by [19].

While these sequence prediction algorithms are able to predict the next activity in a sequence, they have two major drawbacks which limit their use. Firstly, they do not provide any information about the timing of the next activity. The algorithms simply predict the next activity in a sequence - but they do not indicate whether it might happen in a few minutes or a few hours. This reduces their usability for prompting applications where knowledge of activity times is important for ensuring prompt effectiveness. Lack of knowledge about the time until predicted activities also hinders energy management applications where heating or cooling systems must be adjusted in preparation for anticipated activities. A second problem with many sequence prediction algorithms is that they do not predict activities beyond the short-term sequence horizon. Activities which may occur further in the future may not be easily predicted.

The activity prediction algorithms developed in this dissertation attempt to address these

issues. They predict the time duration until each activity occurs, allowing prompts and other time-sensitive items to be scheduled. Since all activities are predicted simultaneously, services relying on the predictions are aware of the time until all activities of interest, regardless of the time until they will occur. Furthermore, the features generated for the activity prediction and the subsequent model-learning process allow temporal relationships between activities to be learned.

2.2 Time-Based Prediction

Another approach to activity prediction is the use of time-based prediction models. Rather than providing a discrete classification for the next activity in a sequence, they predict the continuous-valued time until an activity will next occur. This allows these methods to address some of the issues found in sequence prediction techniques.

Time-based prediction techniques have traditionally centered around time series forecasting techniques. A wide variety of time series forecasting methods have been developed over several decades, especially in the business domain [20–23]. Some of the most popular models are the autoregressive moving average (ARMA) and autoregressive integrated moving average (ARIMA) models [24]. These models have been widely studied in the literature for a variety of applications [25–27]. However, they rely on the assumption that the process being modeled is both linear and stationary. These assumptions do not necessarily hold in the case of activity forecasting, as the activity relationships tend to be nonlinear and may change over time. Some nonlinear time series models are available, such as heteroscedastic models [28]. Neural networks are also popular amongst nonlinear methods [29, 30].

Additionally, many time series methods are univariate: they use previous values of the variable to be predicted as the only inputs to the model. This can prove challenging in activity prediction situations, as we usually do not know these previous values until after the activity occurs. Univariate models also fail to take into account the rich context that can be obtained from other information found in sensor event data. Inclusion of this data is useful for improving the accuracy of activity predictions. While multivariate models are available [31,32], they have not been studied as thoroughly and tend to overfit the data [20].

While time series methods have been studied extensively in the literature, little work has been conducted in predicting human activities, particularly within smart homes. Some have studied the use of time series techniques for predicting electrical demand [29, 30, 33–35]. Research into smart home activity prediction has generally been limited to the prediction of inhabitant wellness [36–40]. These works focus on predicting the duration of inhabitant activities over time to detect irregularities that may indicate health concerns. However, they do not predict when those activities will occur during the day. Mozer [41] describes a system to predict when an inhabitant will enter a room for lighting automation, but the predictions are made only a few seconds in advance. Moutacalli et al. [42] use ARIMA models to model the start times of activities and time offsets between activities to aid in activity recognition, though they only use univariate models. Others have studied forecasting with more complex models [43, 44], however these works include assumptions about the distribution of values or the context in which they can be used.

Most work in predicting inhabitant activities in smart environments is focused on predicting appliance usage. Some methods rely on discretization into time periods to predict appliance usage based on historic patterns [45]. One method models the states and state transitions of

human agents to optimize energy consumption [46]. This can be used to predict how long an inhabitant will perform the activity at the current state and the time until they start the next likely activity. (In some sense, it adapts sequence prediction by including information related to activity duration and spacing.) Barbato et al. [47] describe an approach in which appliance usage history is examined to determine if an appliance will be used on a given day. If so, the historical usage patterns from previous days are used to predict the times and durations of appliance usage on that day.

While these methods incorporate elements of inhabitant activity prediction, it is not the focus of these works. To the best of our knowledge, the methods described in this dissertation are among the first to numerically predict general activity times from environmental sensors in smart environments. These methods do not rely on models of specific appliance usage, but rather on the sensors available within the environment. They are also adaptable to predict activities with different sensors and contexts, even outside the smart home.

2.3 Prompting

Reminder prompts for activity initiation are an important aspect of a smart environment that can be used to help inhabitants live independently. Prompting systems have been developed using a variety of methods for many contexts. Many early systems were rule-driven or required knowledge of users' daily schedules. Lim et al. [48] use a rule-based system for delivering medicine reminder prompts. The Autominder system relies on a pre-programmed user schedule to find prompting rules based on a dynamic Bayesian network [49], reinforcement learning [50], or a Markov decision process [51]. Markov decision processes have also been used

to deliver prompts for the hand-washing process [52]. While these systems can adjust prompts based on user activities, they require input of a user's daily activities or predefined activity steps. In recent years a large number of smart phone reminder apps have become available, but most require self-programming of reminders and only provide time-based reminders rather than using context.

Recent work has focused on automated approaches for issuing prompts, without relying on predefined rules or schedules. The PUCK system [53] uses a variety of classification models to classify situations to determine if a prompt should be issued. It utilizes data sampling techniques to address the imbalance between prompt and non-prompt situations and improve performance. However, it only analyzes recent events to decide if a prompt is needed, instead of predicting future prompt times.

Holder and Cook [54] developed a system which learns the relationships between different activities in order to determine prompt times. They examine the distribution of activity times relative to each other and establish corresponding reference activities that are related to activities of interest. When an activity recognition algorithm detects that a reference activity has occurred, a prompt can be scheduled based on the average time until the activity of interest should occur. However, this model relies on the average relationships between activities, as well as the accuracy of the activity recognition algorithm in detecting the reference activity.

The activity prediction methods we have developed allow for automatic discovery of action relationships. They also allow for flexible prompt triggering mechanisms which can be based on a number of factors, including the numeric time predictions for multiple activities.

2.4 Joint Prediction

A key consideration in the development of prompting algorithms is determining the relationship between different activities performed by the inhabitant. Basic activity prediction algorithms can take into account general features about the sequence of sensor events to form their predictions, and in fact we describe such an algorithm in this dissertation. However, these simple features ignore the relationships between different activities, a potentially rich source of contextual information. Instead, each activity is predicted independently of the others, based solely on the sensor data features. In order to improve upon this, we would like to predict activities jointly to incorporate information about their joint dynamics.

However, it is challenging to properly model the various interrelationships between activities and how the collection of activities affects when all of the activities occur. One option would be to use a graphical model, such as a Markov model or conditional random field [55], to model the relationships between input and output variables. However, modeling all of these relationships can lead to a highly complex graphical model, making it difficult to train and draw inferences. Simplification attempts that attempt to make the model more tractable will often lead to reductions in accuracy and performance. Heuristic solutions such as loopy belief propagation could be used, but these solutions can be difficult to characterize and apply to new problems.

As an alternative approach, we can instead use imitation learning to view the problem as a supervised learning problem. Imitation learning has the goal of training a learner to imitate the actions of an expert in a sequential decision-making task, such as playing a video game or driving a vehicle. Usually, this is done by having the expert demonstrate the behavior to

generate a series of trajectories to use as training data. Supervised learning is then used to train a predictor from this training data, often with a common classification or regression learner.

We use a form of imitation learning called exact imitation learning. This method uses an expert to generate training data (in this case lagged prediction values for each activity), which the learner tries to imitate. In this way, the prediction model can learn about relationships between activities, yet still use a standard regression learner without much added complexity. This also allows us to utilize certain reductions and results from imitation learning to improve prediction performance [56–66].

One potential drawback of an exact imitation learning scheme lies in error propagation. Errors at the start of a prediction sequence may accumulate and amplify, causing large errors later in the trajectory [59, 67]. In order to address this issue, more advanced imitation learning techniques could be utilized. One example of this is the DAgger algorithm [60], which iterates through the training of multiple predictors. At each iteration, additional training data is generated to address any errors, and a new predictor is learned on the aggregated training set. Since DAgger can be seen as a no-regret online learning algorithm, it has nice properties that assure its performance [60].

We have discussed the current state of activity prediction in a variety of fields and using a variety of methods. Sequence prediction algorithms are popular for activity prediction in a variety of contexts and are among the most well-studied methods for predicting inhabitant activities in smart environments. However, they do not provide indications of the timing of those activities, making them difficult to use for time-sensitive applications. A variety of temporal techniques have been developed for generating prompts in a smart environment, but many require coordination with the user. Our prediction algorithms improve upon these by

providing information about activity timings and automatic learning of users' activity habits. We also incorporate imitation learning to simplify the challenge of jointly predicting multiple interdependent user activities. In the next chapter we formally present the activity prediction problem.

CHAPTER 3. ACTIVITY PREDICTION PROBLEM

In this chapter we present a formal definition of the activity prediction problem. Activity prediction is a useful concept in a number of disciplines and can be defined in a variety of ways. We consider the prediction of activity occurrence times from a series of sensor events. Although we predict inhabitant activities using smart environment sensor data, the methods described are applicable to other contexts with a similar structure.

Consider the situation where we wish to predict when a user will initiate certain activities. Let $A = (a_1, a_2, \ldots, a_T)$ be the set of activities that we wish to predict. We do not observe the activities the user performs directly. Instead, we observe a sequence of sensor events generated by sensors observing the user's actions and their environment. We represent the sequence of sensor events as $\Lambda = (\lambda_1, \lambda_2, \ldots, \lambda_N)$, where each event λ_i has a corresponding occurrence time t_i. Each sensor event has a known timestamp, along with information about what sensor generated the event and the value observed by the sensor. These sensor events form a sequence of observations related to the user's activity. In activity prediction, we want to predict the amount of time that will elapse before the next occurrence of an activity.

In order to form proper predictions, we must first determine what defines an activity occurrence. We consider the situation where each sensor event has an associated activity label, as shown in Table 3.1. These labels indicate what activity is believed to be occurring when the sensor event is generated. Labels may be added to sensor events through human annotators or through an automated activity recognition system. These methods are detailed in Section 6.3.

Based on these activity labels, the sensor events can be separated into activity "occur-

Table 3.1: Labeled sensor events. Each event contains the date and time the event was generated, the sensor that generated it, and the sensor message sent. The activity labels generated by activity recognition or human annotation are applied to each sensor event. The example events are drawn from our CASAS smart home system, described in Chapter 6.

Date	Time	Sensor	Message	Activity
2012-07-20	06:36:25.8	M002	ON	Bathe
2012-07-20	06:36:25.8	LS001	27	Bathe
2012-07-20	06:36:26.9	M001	ON	Bathe
2012-07-20	06:36:27.1	M002	OFF	Dress
2012-07-20	06:36:27.2	D003	OPEN	Dress
2012-07-20	06:36:29.7	LS001	36	Dress
2012-07-20	06:36:31.0	M004	ON	Personal Hygiene
2012-07-20	06:36:32.3	M001	ON	Toilet
2012-07-20	06:36:32.9	M005	ON	Personal Hygiene
2012-07-20	06:36:33.4	M004	OFF	Personal Hygiene

Table 3.2: Sensor events segmented into activities. Sensor events from Table 3.1 are grouped into "occurrences" based on consecutive events from the same activity. These occurrences are separated here by the horizontal rules.

Date	Time	Sensor	Message	Activity
2012-07-20	06:36:25.8	M002	ON	Personal Hygiene
2012-07-20	06:36:25.8	LS001	27	Personal Hygiene
2012-07-20	06:36:26.9	M001	ON	Personal Hygiene
2012-07-20	06:36:27.1	M002	OFF	Dress
2012-07-20	06:36:27.2	D003	OPEN	Dress
2012-07-20	06:36:29.7	LS001	36	Dress
2012-07-20	06:36:30.1	M001	OFF	Dress
2012-07-20	06:36:31.0	M004	ON	Personal Hygiene
2012-07-20	06:36:32.3	M001	ON	Toilet
2012-07-20	06:36:32.9	M005	ON	Personal Hygiene
2012-07-20	06:36:33.4	M004	OFF	Personal Hygiene

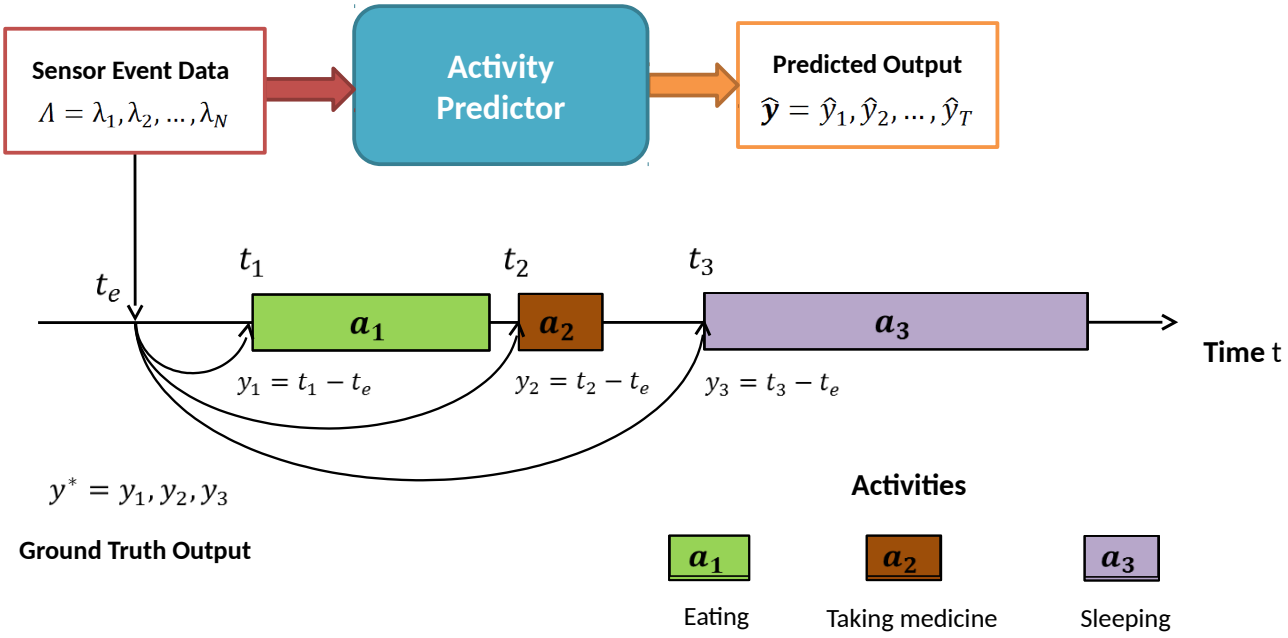


Figure 3.1: Overview of the predictor process. The sensor events Λ are used as input. In this example, we wish to predict the activities a_1 (eating), a_2 (taking medicine), and a_3 (sleeping), whose next occurrences start at times t_1, t_2, and t_3, respectively. Given the current event λ_e at time t_e, we wish to predict the times between t_e and the start of each activity, t_i. That is, we wish to predict the vector $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2, \hat{y}_3)$, where $\hat{y}_i$ is the predicted value of the ground-truth $y_i = t_i - t_e$. The ground truth vector is $\mathbf{y}^* = (y_1, y_2, y_3)$.

rences". An occurrence is simply a grouping of consecutive events with the same activity label. It represents an individual instance of the inhabitant performing an activity. For example, the sequence of labeled events from Table 3.1 can be segmented into occurrences as shown in Table 3.2. Note that activity occurrences can be as short as one event in length. This is particularly common occurrence when two activities are interwoven.

With this definition of an activity occurrence in mind, we wish to predict the time until the next occurrence of the activity of interest. That is, we wish to predict the length of time

between any event λ_e at time t_e and the time of the first sensor event in the next occurrence of the activity, t_i. This leads to a ground truth vector of relative activity times $\mathbf{y}^* = (y_1, y_2, \ldots, y_T)$ where $y_i = t_i - t_e$ is the relative occurrence time of activity a_i. Note that, since we are always predicting times of activities in the future, the values in $\mathbf{y}^*$ are always zero or greater. We label the predicted relative occurrence times $\hat{y}_i$, which form prediction vector $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_T)$. ($\hat{y}_i \in \Re$ and $y_i \in \Re$, so $\hat{\mathbf{y}} \in \Re^T$ and $\mathbf{y}^* \in \Re^T$.) This is shown graphically in Figure 3.1.

Using this definition of activity prediction, we can visualize the ground truth activity occurrence times and predicted times. Figure 3.2 shows example predicted and actual times for a "Bathe" activity. Note that the times approach zero around the time of the activity occurrence, and then abruptly "jump" back up after the activity ends as we look forward to the next occurrence. Based on these times, we can determine the error between the predicted and actual activity times. Figure 3.3 shows the corresponding prediction error for our "Bathe" activity.

In this chapter we defined the activity prediction problem and introduced notation to describe it. This provides a specific framework within which activity prediction methods can be considered. In the next chapter we will describe our proposed methods for predicting activities within this framework.

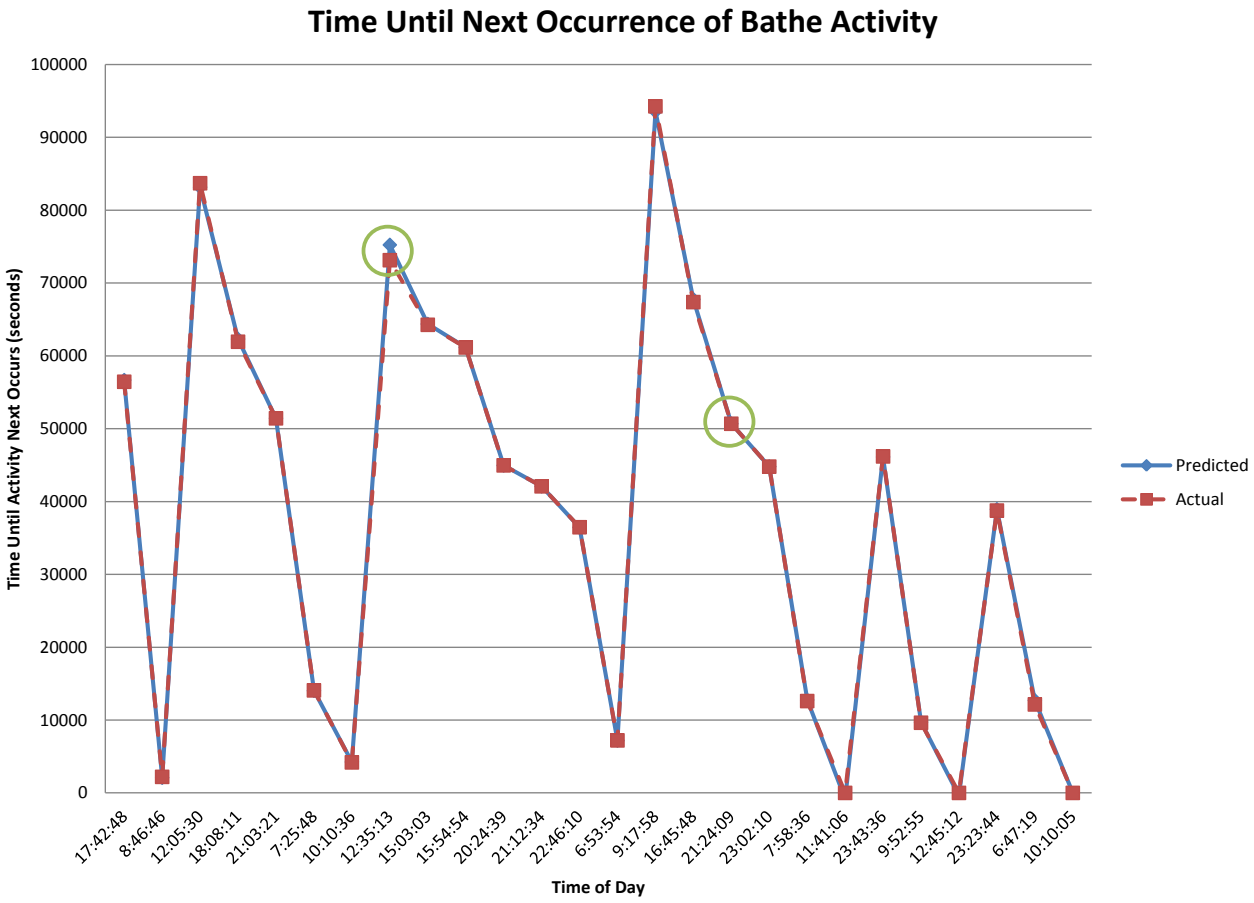


Figure 3.2: Example predicted and actual activity times for activity "Bathe". The predicted elapsed time until an activity will occur reaches zero when an activity occurs, then jumps up after the activity ends. Figure 3.3 shows the corresponding error in predictions, with corresponding points circled.

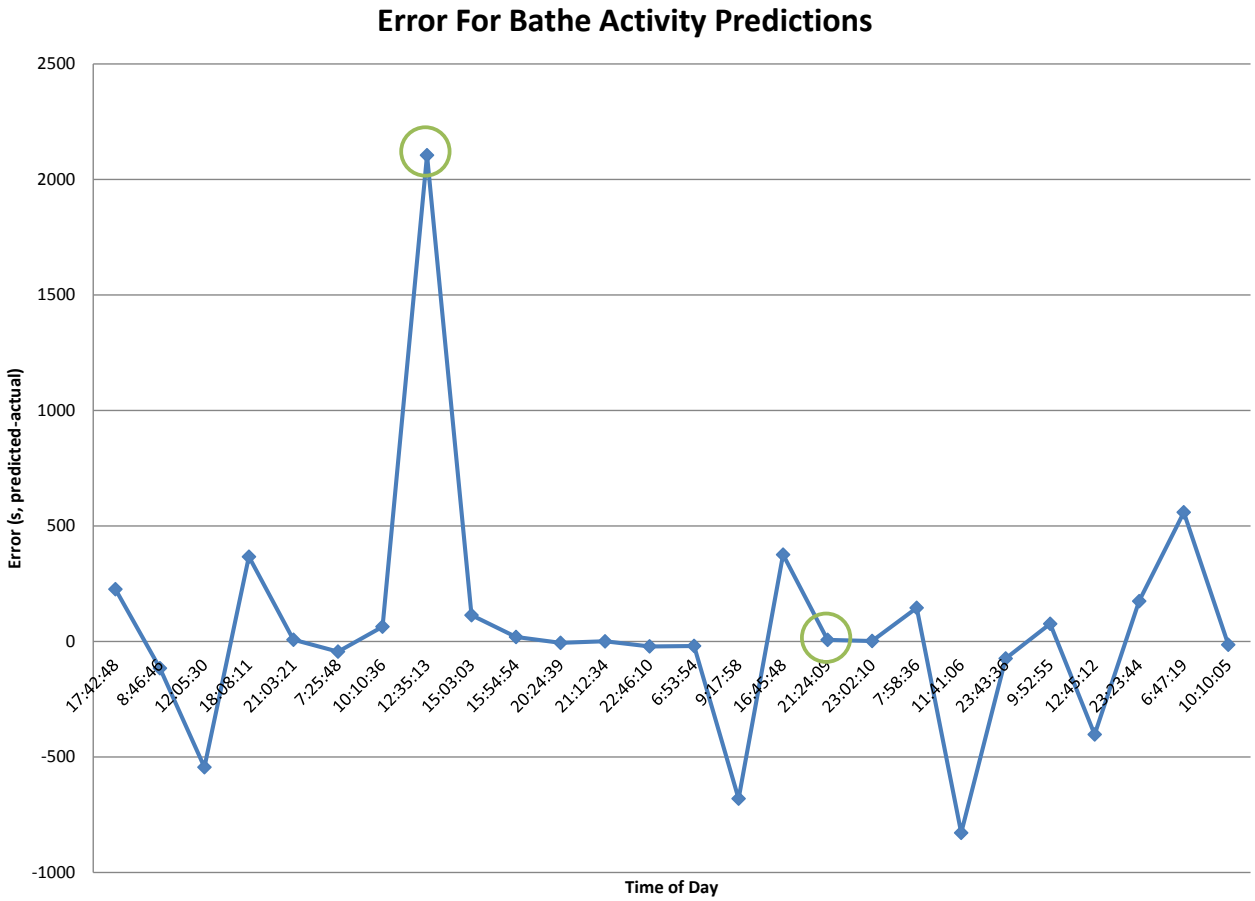


Figure 3.3: Example error in predicted times for prediction in Figure 3.2. Error here is the difference between predicted and actual times. Circles correspond to points in Figure 3.2.

CHAPTER 4. ACTIVITY PREDICTOR

In this chapter we present our activity prediction algorithm, called Activity Predictor (AP). This algorithm uses information drawn from smart environment sensor events to form automated activity predictions. First, we introduce the base AP method, Independent Predictor, and describe the major components. We then describe extensions of the basic algorithm to incorporate further information about the activities and improve performance. These extensions result in the Recurrent Activity Predictor and Two-Pass Activity Predictor algorithms.

4.1 Independent Predictor

Our basic AP algorithm consists of two major components, as shown in Figure 4.1. The first component is the feature extraction component, which uses the sensor data input to generate useful feature values that relate information about the patterns and context in the sensor data. These features are then passed to the regression learner, which uses a regression model to form the numeric predictions. We term this version of the predictor the Independent Predictor (IP), as it uses a separate regression model to predict each activity independently of the others.

In its present form, IP performs feature extraction and outputs predictions for each sensor event in the dataset. In other words, the numeric output of IP is the predicted amount of time between the sensor event and the time when each activity will next occur. These predictions are updated with each new sensor event, and can be used to form predictions as new events are received in a live environment.

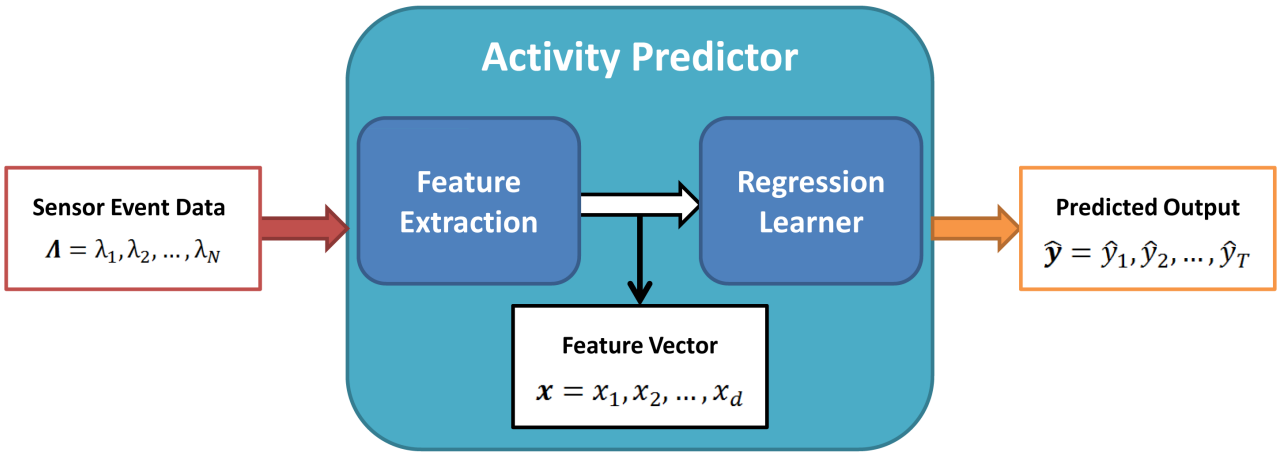


Figure 4.1: Independent Predictor overview. The two major components are the feature extraction and regression learner components, shown as boxes. IP uses sensor events from the smart environment as input and outputs a numeric prediction for each activity time.

Features are only generated once for each sensor event and are then used by all activity models for prediction. A separate regression learner is trained and used for each activity, rather than having a single vector-output model. Thus, each activity is predicted separately, though the forecasted times for all activities are generated for each observed event. Initially, this leads to independent activity predictions, though interdependence can be added as described in Section 4.2.

4.1.1 Feature Extraction

The feature extraction component draws useful contextual information from the sensor event data in order to inform the regression learner. The component takes as input the sequence of sensor events $\Lambda = (\lambda_1, \lambda_2, \ldots, \lambda_N)$. Recall that each event has an associated timestamp,

sensor ID, and sensor message or value. Using the information from these fields, the feature extraction component can generate features related to the timing of sensor events, frequencies of various sensors, and sensor value statistics, among other things. These features are formed into a vector of feature values, known as a feature vector.

Specifically, we have a feature function Φ that, for each sensor event λ_i, computes a d-dimensional feature vector $\Phi(\lambda_i) \in \Re^d$. We label this feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_d)$, such that $\mathbf{x} = \Phi(\lambda_i)$. The feature extraction uses a sliding window of recent sensor event information from which it computes the features for each event. So, in effect, Φ is actually a function of a window of recent sensor events terminating with the specific event λ_i. The size and shape of these windows varies depending on the type of features being generated, as described in following sections.

The features defined by Φ can be chosen using knowledge of the problem domain. They should be designed such that they extract useful contextual information from the raw sensor events in order to aid the regression learner in prediction. For example, for activity prediction, we choose features that provide information about the timing and frequencies of sensor events, as well as statistical value information. Features can also include information about past predictions, as demonstrated in Section 4.2. Since the feature information is in the form of a vector of scalar values, any features that can be represented in this form can be used. Although a model specific to a certain feature set must be learned for regression, the underlying training and use algorithms of the regression learner do not need to be changed in order to test different context features.

A variety of features could be conceived of and used, but we use two primary sets of features. The first, called discrete features, provide information about the temporal patterns in

the sensor event data and the relative frequencies of different sensors within the sliding window. The other set, called sampling features, consists of statistical values derived from a separate sliding window of sampled sensor event values. These features provide information about the values of the sensor events. We describe these two sets of features in the following subsections.

Discrete Features

The discrete features describe the time, location, and frequency of events. They help to establish context for activity prediction from the temporal and spatial distribution of the inhabitant's activities. These features are listed in Table 4.1. The discrete features are generated from a dynamic sliding window of recent sensor events. Features such as *lastSensorEventHours*, *lastSensorEventSeconds*, and *timestamp* provide information about the time of day when the most recent sensor event occurred. *windowDuration* and *timeSincelastSensorEvent* convey information about the temporal distribution of events within the window. The *lastLocation*, *lastSensorID*, and *sensorElTime* features provide information about what sensors have been active recently and give a sense of the location of recent activities. The dominant sensor and *sensorCount* features describe which sensors have been most frequent in the recent window to give a sense of the amount of activity occurring at each sensor. The *laggedTimestamp* features help in understanding the temporal spacing of recent events.

The sliding window of recent events used for generating the discrete features is dynamically changed based on the relationship between sensors and activities. In order to initialize these windows for use, the feature extraction component uses the activity labels on sensor events to calculate activity length frequencies during the training stage. Before generating features from the training data, the feature extraction component first examines the sensor data to determine the most likely window size for each activity. The length of each occurrence

Table 4.1: Discrete features.

Discrete Feature Name	Description
lastSensorEventHours	Hour of the day when the current event occurred
lastSensorEventSeconds	Time since beginning of the day in seconds for the current event
windowDuration	Duration of the window (seconds)
timeSinceLastSensorEvent	Time since the previous sensor event (seconds)
prevDominantSensor1	Most frequent sensor ID in the previous window
prevDominantSensor2	Most frequent sensor ID in the window before that
lastSensorID	Sensor ID of the current event
lastLocation	Sensor ID for the most recent discrete sensor event
sensorCount*	Number of events in the window from each sensor
sensorElTime*	Time (seconds) since last event for each sensor
timeStamp	Time since beginning of the day (seconds) normalized by the total number of seconds in a day
laggedTimestamp*	timeStamp feature value for previous event

*There is one sensorCount and one sensorElTime for each sensor in the smart home. The lag values are created for the previous sensor events (for our experiments, we use 12 lag values).

Table 4.2: Discrete feature activity window length bins. During training, the length of each activity occurrence is binned into one of the following bins (separated for each activity). Then, the most frequent bin is determined for each activity, with the corresponding window size being chosen for that activity.

Occurrence Length (Events)	Window Size (Events)
≤ 5	5
5-10	10
10-20	20
> 20	30

of each activity in the training data is found, and these lengths are binned into one of four bins based on the number of events in the occurrence. The bin sizes are listed in Table 4.2. Next, the most frequent bin for each activity is determined and the resulting window size from Table 4.2 is assigned for that activity. This window size represents the most probable activity occurrence length for that activity.

After the activity window sizes are determined, the algorithm determines which activities are most likely associated with each sensor. This is done by computing the number of times sensor events from each sensor are labeled with a particular activity in the training data. The most frequent activity for each sensor is then determined from these counts.

These activity window and sensor-activity frequencies are computed only once during the training initialization process. After this point, the windows are set for the feature extraction for both training and use of IP; activity labels are not needed when using the trained model.

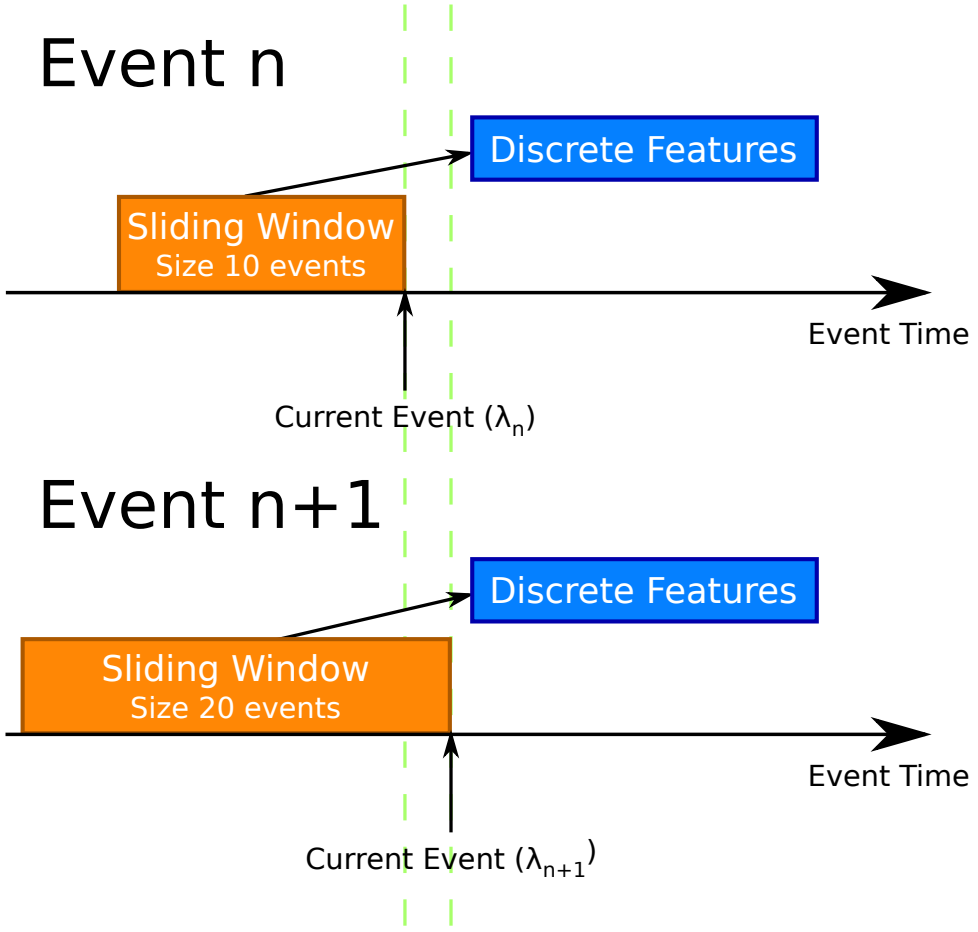


Figure 4.2: Discrete feature generation. The discrete features are generated based on the sliding window which consists of a number of events up to and including the current event. The discrete feature vector generated from this window is associated with the current event. Then, we move to the next event, sliding the window forward one event (and changing its length if needed) and repeat.

Once they are computed, the feature extraction process proceeds to generate the feature values, a process which is performed the same for both training and testing of the IP model. The algorithm waits until reaching a certain number of events in the dataset before generating features in order to allow the maximum sliding window to be full. (In our experiments we

wait until the 30th event.) For each event, the discrete features are computed based on the sliding window and are stored as a feature vector $\mathbf{x}$ associated with that instance as illustrated in Figure 4.2. Note that, in this discussion, each instance refers to a sequence of sensor events leading up to and including the current sensor event. Then, the sliding window is moved forward by one event (and its size adjusted if necessary) and the process is repeated. (The feature vectors can be stored in a dataset when performing offline training and testing or passed directly to the model tree for prediction in online use.)

Sampling Features

In addition to discrete features, IP can also utilize sampling features. While the discrete features provide information about the timing and frequency of events, sampling features provide information about the values provided by the sensor events. These values include the light level, temperature, and other values, as well as the status of the motion and door sensors. By including these values in the features extracted, the IP algorithm is able to discern information related to the duration and change of different sensor states and other useful context. This is especially useful when the change in sensor values over time is important, such as when a temperature sensor near the stove reports increasing temperatures during cooking.

These value-based features are called sampling features due to the way in which they are computed. We borrow from the domain of signal processing, in which an analog value is often sampled at regular intervals to turn it into a digital signal. In our case, we form a "sampling" of the recent values of each sensor at regular time intervals. We consider the "state" of a sensor to be the value of its most recent event. As sensor events are processed, the feature extraction component of IP keeps track of the state of each sensor. At regular time intervals the algorithm stores a vector of values representing the current state of all of the sensors at

that time. This reflects the most recent value received from each sensor when the sample is taken. The algorithm stores a number of these recent sample vectors from which it computes the sampling features. The samples are made with a specific *sampling interval* in time, and the amount of time's worth of samples to keep is the *sample lag*.

The values from the light level, temperature, and battery sensor events are numeric values that are directly used in the sample vectors. For the motion, door, and light switch events, however, we receive binary messages about the sensor state (e.g, ON or OFF). In order to use these in the sample vectors we map the values to binary numeric values (e.g, 1 for ON and 0 for OFF). In this way these binary sensor states can also be included in sampling features.

To illustrate the sampling process, consider the sensor events in Table 4.3. Consider using a sampling interval of one second (i.e, create a sample vector of sensor states at each second). The resulting sample vectors are shown in Table 4.4. The elements in the sample vectors only change at a new sample time if an event from the corresponding sensor occurred during the preceding interval. If the sample lag is four seconds, then any sampling features generated between 06:36:30 and 06:36:31 will use the last four vectors in the table (starting with 06:36:27).

The generated sample vectors are used to calculate the sampling features. The features are created for each sensor event, just like the discrete features, and are included in the overall feature vector $\mathbf{x}$. Sampling features are computed based off the window of recent sample vectors (defined by the sample lag), including the last sample vector before the current event. These features are statistical measures of the values in the sample vectors. All of the sampling features are calculated for each sensor using the sample values for that sensor, meaning there are $34 * NumberOfSensors$ sampling features. The sample features used are described below.

Table 4.3: Example CASAS sensor events for sampling vector demonstration.

Date	Time	Sensor	Message
2012-07-20	06:36:25.8	M002	ON
2012-07-20	06:36:25.8	LS001	27
2012-07-20	06:36:26.9	M001	ON
2012-07-20	06:36:27.1	M002	OFF
2012-07-20	06:36:27.2	D003	OPEN
2012-07-20	06:36:29.7	LS001	36

Many are adapted from those suggested by Cook and Krishnan in [68]. Here, we use v_i to represent the value for the sensor of interest in the ith sample vector in the window, while L represents the number of sample vectors in the window.

- **sensorMaximumValue** The maximum value of the sensor within the window:

$$\text{sensorMaximumValue} = \max_{1 \leq i \leq L} (v_i) \tag{4.1}$$

- **sensorMinimumValue** The minimum value of the sensor within the window:

$$\text{sensorMinimumValue} = \min_{1 \leq i \leq L} (v_i) \tag{4.2}$$

- **sensorSum** The mean of values of the sensor in the window:

$$\text{sensorSum} = \sum_{i=1}^{L} v_i \tag{4.3}$$

Table 4.4: Example sampling vectors for the events in Table 4.3. The sampling interval is one second. Values which change at each interval are in bold.

Sample Time	M001	M002	D003	LS001
06:36:25	0	0	0	0
06:36:26	0	**1**	0	**27**
06:36:27	**1**	1	0	27
06:36:28	1	**0**	**1**	27
06:36:29	1	0	1	27
06:36:30	1	0	1	**36**

- **sensorMean** The mean value of the sensor in the window:

$$\text{sensorMean} = \overline{v} = \frac{1}{L}\sum_{i=1}^{L} v_i \tag{4.4}$$

- **sensorMeanAbsoluteDeviation** The average of the absolute difference between each value and the mean:

$$\text{sensorMeanAbsoluteDeviation} = \frac{1}{L}\sum_{i=1}^{L} |v_i - \overline{v}| \tag{4.5}$$

- **sensorMedianAbsoluteDeviation** The average of the absolute difference between each value and the median.

$$\text{sensorMedianAbsoluteDeviation} = \frac{1}{L}\sum_{i=1}^{L} |v_i - \text{median}(v)| \tag{4.6}$$

  This metric may be more useful than the mean absolute deviation in some cases.

- **sensorStandardDeviation** The standard deviation of the values:

$$\text{sensorStandardDeviation} = \sigma_v = \sqrt{\frac{1}{L}\sum_{i=1}^{L}(v_i - \overline{v})^2} \tag{4.7}$$

- **sensorCoeffVariation** The coefficient of variation of the values, which shows how the standard deviation and mean relate.

$$\text{sensorCoeffVariation} = \frac{\sigma_v}{\overline{v}} \tag{4.8}$$

- **sensorNumZeroCrossings** The number of times the sensor value crosses the median of the values within the window. This provides some indication of the amount of movement within the values.

$$\text{sensorNumZeroCrossings} = |v_i < \text{median}(v) < v_{i+1}| + |v_i > \text{median}(v) > v_{i+1}| \tag{4.9}$$

- **sensor25thPercentile / sensor50thPercentile / sensor75thPercentile** The 25th, 50th, or 75th percentile of the values. These are a value below which the desired percentage of the values fall. They help in understanding the distribution of values.

- **sensorSqSum25thPercentile / sensorSqSum50thPercentile / sensorSqSum75thPercentile** The square sum of values below the given percentile. For example:

$$\text{sensorSqSum25thPercentile} = \sum\{v_i^2 | v_i < \text{sensor25thPercentile}\} \tag{4.10}$$

- **sensorInterQuartileRange** The interquartile range, or the difference between the 75th and 25th percentiles:

$$\text{sensorInterQuartileRange} = \text{sensor75thPercentile} - \text{sensor25thPercentile} \tag{4.11}$$

- **sensorBinCount** There are 10 of these features. The values in the window are placed into 10 bins that are equally spaced across the range of values (sensorMinimumValue to sensorMaximumValue). The features are then the number of values in the corresponding bins.

- **sensorSkewness** An indication of the symmetry of the values about the mean:

$$\text{sensorSkewness} = \frac{\frac{1}{L}\sum_{i=1}^{L}(v_i - \overline{v})^3}{(\frac{1}{L}\sum_{i=1}^{L}(v_i - \overline{v})^2)^{\frac{3}{2}}} \tag{4.12}$$

- **sensorKurtosis** An indication of how "peaked" the distribution of values is:

$$\text{sensorKurtosis} = \frac{\frac{1}{L}\sum_{i=1}^{L}(v_i - \overline{v})^4}{(\frac{1}{L}\sum_{i=1}^{L}(v_i - \overline{v})^2)^3} \tag{4.13}$$

- **sensorSignalEnergy** Sum of the squares of the values, representing the area between the value curve and the time axis:

$$\text{sensorSignalEnergy} = \sum_{i=1}^{L} v_i^2 \tag{4.14}$$

- **sensorLogSignalEnergy** Similar to signal energy, but summing the logarithms of squares instead:

$$\text{sensorLogSignalEnergy} = \sum_{i=1}^{L} \log_{10}(v_i^2) \tag{4.15}$$

- **sensorSignalPower** The average signal energy over the window:

$$\text{sensorSignalPower} = \frac{1}{L}\sum_{i=1}^{L} v_i^2 \tag{4.16}$$

- **sensorPeakToPeak** The amplitude between the maximum and minimum values of the sensor (peak-to-peak):

$$\text{sensorPeakToPeak} = \text{sensorMaximumValue} - \text{sensorMinimumValue} \tag{4.17}$$

- **sensorAvgTimeBtwnPeaks** The average time between two local maximum values (peaks) in the data. These local maxima are defined as points where $v_{i-1} < v_i > v_{i+1}$.

- **sensorNumPeaks** The count of the number of peaks in the values within the window.

We refer again to our previous example. If we have a sampling lag of four seconds and thus use the last four vectors in Table 4.4, the sensorMaximumValue for D003 is 1 while it is 36 for LS001. The sensorMeanValue for D003 is 0.75 while it is 29.25 for LS001.

While these discrete and sampling features provide a useful amount of context for activity prediction, other feature values could be used as well. These feature values should be drawn from knowledge about the particular application in which the activity prediction will be used. The features listed here provide information about the location, frequency, and state of actions within a home, useful for smart environment predictions. However, other features may be useful in different contexts, such as a fast Fourier transform of the values or information from user feedback. When considering which features to use, it is important to take the strengths and weaknesses of those features into account. For example, while the sampling features provide useful information about the state of sensors, they can be computationally expensive to compute. Thus, the requirements of the application should be taken into account when choosing a set of features to use for activity prediction.

4.1.2 Regression Learner

The regression learner is a supervised learning algorithm that creates a model of the activity to be predicted, which can then be used to form predictions. We use a multi-output regression learner for this purpose. A multi-output regression learner is analogous to the multi-label

classification problem [61], except with real-valued output labels instead of binary classifications. Although any multi-output regression learner could be used, we separate the multi-output regression problem by learning a separate regression function for each activity. This technique is inspired by the Binary Relevance classifier [61]. The separate regression functions are each learned and used in a similar and cooperative manner to form a vector of predictions for all activities from each sensor event.

Each regression learner is first trained using a set of training instances. These instances consist of feature vectors $\mathbf{x_i}$ drawn from labeled sensor events and the corresponding number of time units until the activity occurs $\mathbf{y_i^*}$. These input-output pairs $\{\mathbf{x_i}, \mathbf{y_i^*}\}_{i=1}^{N}$ form the set of training instances used to learn the model. The regression learner learns a prediction model in order to minimize a given loss function L over the set of training data. Once the model is learned, it can be used with new feature vectors $\mathbf{x}$ to form predictions for new sensor events.

There are a variety of regression learners available that could be used. One of the simplest options is linear regression, which simply learns a set of linear parameters to generate a prediction. However, this model relies on a linearity assumption that may not be present in the activity data. More advanced methods include logistic regression [69] or support vector machines (e.g. SMOreg) [70, 71], among others. While these methods are able to model many of the complexities in the activity patterns, they also can be computationally expensive to train and use.

For this dissertation, we have chosen to use a model tree based on the M5 algorithm [72, 73]. Model trees are able to model the nonlinear aspects of the activity relationships. Further, although there is some cost associated with training the models, they are able to quickly compute predictions in real-time during use. While multi-output regression models can

be used, we instead use a separate regression model for each activity to predict, inspired by the binary relevance classifier [74].

Model Tree

A model tree is a regression model that relies on a tree-like structure for generating regression output. It is similar to a decision tree. The model tree consists of a hierarchical structure of nodes, as shown in Figure 4.3. Each node references a particular feature x_i (from the available features $\mathbf{x}$) and a corresponding threshold value. While a decision tree has particular class labels at the leaf nodes, however, a model tree leaf node instead contains a multivariate linear regression model. This model is of the form in Equation 4.18, where y is the prediction output, w_i are scalar weights, and a_i are values of feature variables.

$$y = w_0 + w_1 a_1 + w_2 a_2 + \cdots + w_k a_k \tag{4.18}$$

When using the learned model on a new instance, we use the feature values of the instance to traverse the tree. Starting at the root node, we compare the value of the feature specified by that node against the node's threshold value. If the feature value is below the threshold, we move to the first child node; otherwise, we move to the other child node. We continue doing this with the subsequent nodes until a leaf node is reached. We then use the linear model contained in the leaf node with the feature vector for the instance to calculate the scalar prediction output. (Only certain features from the vector are used; these are established during the model training process.) The tree traversal and prediction computation can happen quickly, making the model tree an effective algorithm for real-time prediction.

Model trees are sometimes also called regression trees. However, the term "regression tree" is sometimes used to describe a slightly different model than the one described above. In

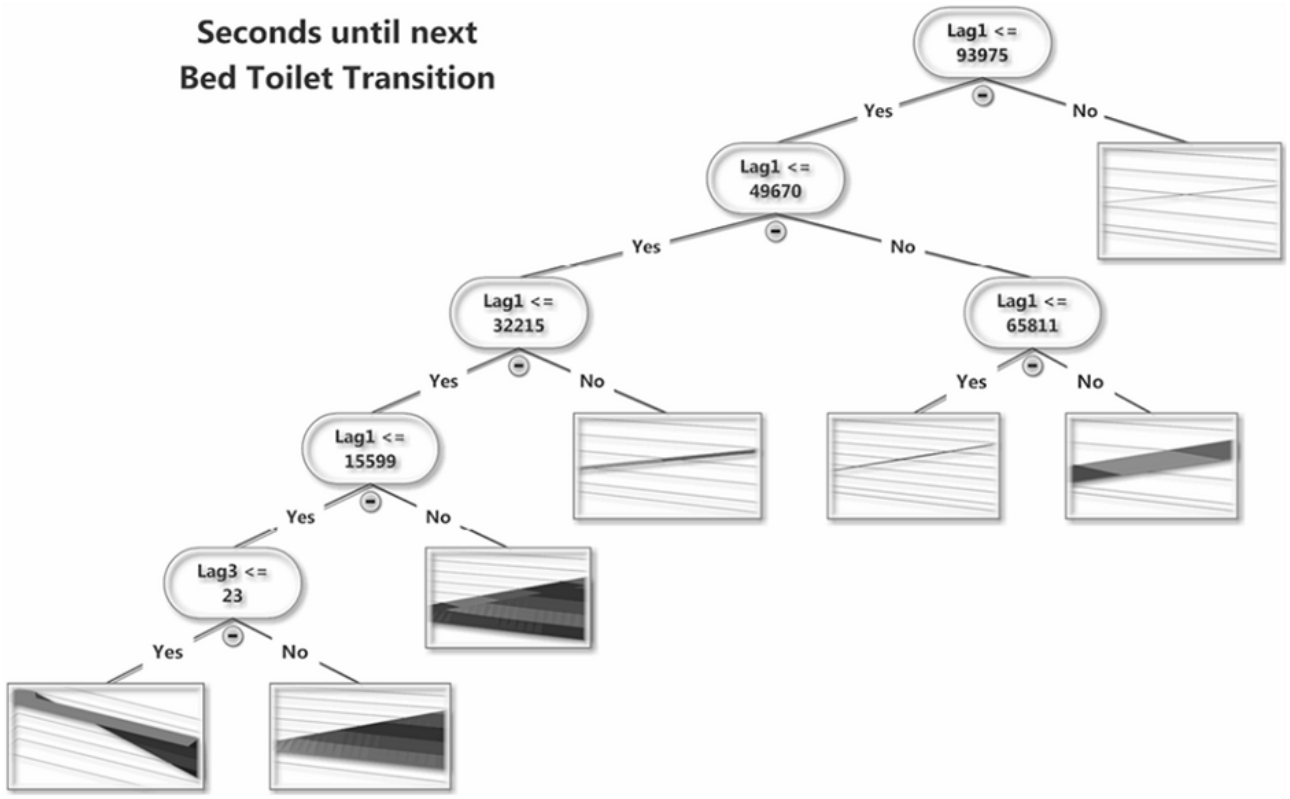


Figure 4.3: Model tree diagram. The tree consists of nodes with split features at each node. It is traversed by comparing the specified feature against a threshold at each node and taking the resulting path. When a leaf node is reached, the linear model it contains (represented by plots here) is used to form a prediction.

that model, the leaf nodes contain constant scalar values rather than linear regression models. These values are usually taken from the training data (e.g, an average of label values at that node). While this may simplify training and use, it also reduces the range and flexibility of the model. In order to be clear, we will use the term "model tree" to denote the tree with linear models that is used in this dissertation.

Training

In order to use the model tree, it must first be trained using example data instances. This training data will consist of pairs of feature vectors and label values $\{\mathbf{x_i}, \mathbf{y_i^*}\}_{i=1}^{N}$. The feature vectors are found using feature extraction from training sensor event data. The labels $\mathbf{y_i^*}$ are calculated using the activity labels on the sensor event data. For each sensor event (and corresponding feature vector), the corresponding labels are the difference in time between the sensor event being processed and the next event with the target activity label. These labels are easily computed from the labeled sensor event data. They are usually found in units of seconds. For example, in Table 3.2, the first event has a label of 0 seconds for the target activity Personal Hygiene (since it is labeled with that activity). The label for the Dress activity is 1.3 seconds and the label for the Toilet activity is 6.5 seconds. Since each activity has a separate model tree, each tree is trained using only one of the elements of the label vector $\mathbf{y_i^*}$ corresponding to that activity.

We denote as Y^* the total set of activity time labels (for the activity of interest) found in the training data. The model is trained starting from the root node. At this node, let the set of labels at that node, Y', be equal to the total set Y^*. At each node, we choose a feature and threshold to split on in order to maximize the gain, defined in Equation 4.19.

$$\text{Gain}(Y', Y'_<, Y'_>) = \sigma(Y') - \frac{|Y'_<|}{|Y'|}\sigma(Y'_<) - \frac{|Y'_>|}{|Y'|}\sigma(Y'_>) \tag{4.19}$$

This equation is dependent on the splitting feature and the chosen threshold value. Here, $Y'_<$ is the set of labels for which the value of the associated splitting feature is less than the threshold value. $Y'_>$ is the set of labels for which the feature value is greater than the threshold. $\sigma(Y')$ is the standard deviation of the values in the label set Y', while $\sigma(Y'_<)$ and $\sigma(Y'_>)$ are the

standard deviation of the values in the two subsets. The gain represents the reduction of error that would occur in the two split sets if we split at a given feature and threshold, where the error is estimated by the deviation in the class labels. The goal is to maximize this reduction in error so that the instances that reach each class node will be similar.

We iterate over all the features and possible splitting thresholds to find the pair that maximizes the gain. This pair is then set as the splitting feature and threshold for the node. The instances associated with the set $Y'_<$ get passed to the first child node (to form Y' for that node), while the instances associated with $Y'_>$ get passed to the second node. The splitting process then continues recursively with the child nodes to build the tree. The splitting terminates when one of the following conditions occurs: all features have been used as splitting attributes on the current path, the number of instances that reach a node is too little (four or fewer), or the standard deviation of the remaining instance labels Y' is smaller than 5% of the standard deviation of the original set Y^*. When this occurs, we form a leaf node.

At each node in the tree, we form a multivariate linear regression model. This model is learned using the set of training instances that would reach that node following the splitting in the nodes above it. The model uses only the features that are used as splitting features in the node itself or the nodes below it in the tree. In this way, it forms a localized regression model for instances that are similar to the ones at the node and would reach it following the same path.

After the full model tree is formed, it can be pruned to reduce its complexity and improve prediction performance. Pruning starts at the leaf nodes and proceeds in a bottom-up fashion. At each node, we use the regression model at that node to form predictions $\hat{y}_m$ for all training instances in the subset that reached that node. We then compute the root mean squared error

(RMSE) of the predictions compared to the training labels, as shown in Equation 4.20.

$$\text{RMSE} = \sqrt{\frac{\sum(\hat{y}_m - y_m^*)^2}{N}} \tag{4.20}$$

The summation is performed over the N predictions $\hat{y}_m$ and labels y_m^* in the set.

The instances are also predicted using the node's subtree (with each instance traversing the nodes) and the RMSE of these results is computed. If the RMSE of the node is lower than the RMSE of its subtree, the subtree is pruned and the node becomes a new leaf node. In this way, we reduce the depth of the tree while also improving its performance.

Finally, a smoothing process can be used to improve the performance of the predictions, especially when the number of instances used to construct a node's model is small [72]. This smoothing combines the model in each node with the models of the nodes in the path above it. We denote S as the node of interest, and $C(S)$ is the regression value that will be output for that node. To perform smoothing, we adjust the model such that the regression output becomes as shown in Equation 4.21.

$$C(S) = \frac{n_i C(S_i) + kM(S)}{n_i + k} \tag{4.21}$$

Here S_i is the ith node that was traversed to reach the current one, and n_i is the number of training instances that reached that node. $M(S)$ is the original output of the linear model at node S, while k is a smoothing constant (15 in our case).

4.2 Recurrent Activity Predictor

The IP predictor uses independent regression models to separately predict each activity. That is, the regression learner is only provided features directly from the sensor data, with no

information about predictions from other activities (or even previous predictions of the same activity). While these features are able to provide a basic context for prediction, they can be improved by using joint activity models that can utilize information about the joint relationships between activities. This allows the algorithm to use the context of recent predictions from all activities to inform the prediction of individual activities.

4.2.1 Joint Activity Prediction

Recall that the IP utilizes a feature function Φ to extract context features from each event λ_i. These discrete and sampling features are drawn directly from the sensor events themselves. They do not provide any information about different activities and the relationships between them. However, many activities that take a user may perform are interrelated. For example, in a smart environment, a resident will often cook breakfast, then eat the prepared meal, and finally clean the dishes afterward in a regular and repeated pattern. Similar relationships exist between bathing, dressing, and grooming activities as well as others. If our algorithm can learn and reason about these activity relationships performance can be increased. Therefore, we wish to incorporate the activity context into our algorithm in order to create joint interdependent prediction models for all activities.

In order to include this information we utilize the framework of imitation learning. In traditional imitation learning, an expert generates training data which the learner uses to imitate the behavior of the expert in a generalizable way. For our application the expert information is drawn from the labeled sensor event data used for training the regression learner. The labeled data is used during the training process to provide the expert (ground-truth)

behavior which the learner should follow. When the learned model is used, the expert data is replaced with values generated by the learned model. We wish to find a set of useful contextual values that can be used in this way to incorporate information about activity relationships using this imitation learning method.

Formally, we wish to include additional feature values in our feature extraction component that provide contextual information about recent activity predictions. Let us define $\Phi_{independent}(\lambda_i)$ to be our feature function $\Phi(\lambda_i)$ used to generate the discrete and sampling features in IP. We wish to incorporate a second feature function $\Phi_{joint}(\lambda_i)$ that provides additional features consisting of recent activity predictions made by the predictor. The two feature functions combine to form a new overall feature function $\Phi_{total}(\lambda_i) = \{\Phi_{independent}(\lambda_i), \Phi_{joint}(\lambda_i)\}$ that consists of both sensor and activity context features. This new feature function is utilized by the feature extraction component of AP. In this way we provide information about activity relationships by allowing the model to learn the relationships between prediction times for several activities and the activity it is attempting to predict. Since this context is simply included as new feature values in the feature vectors $\mathbf{x}$, the regression learner does not need to use any new algorithm to learn and use the activity context. Instead, it simply learns the activity relationships from the new features provided to it.

4.2.2 Joint Prediction Through Recurrence

Our joint activity predictor, called the Recurrent Activity Predictor (RAP), utilizes predictions from the previous event (lagged predictions) as the activity features generated by feature extraction. Formally, the feature function $\Phi_{joint}(\lambda_i)$ produces as features the prediction

for each activity from the previous event. These values are known for each event as they are generated for the event before it. Since the events are not spaced evenly in time, the model may have difficulties properly incorporating these lags as they may be more or less "fresh" for each event. To counter this, we adjust the lags by subtracting the time interval between the previous event and the current one. In this way, the lag values represent the time from the *current* event time until the next activity occurrence based on the predictions from the previous event. Since we want the activity prediction values to be non-negative (only pointing forward to future activity occurrences), we limit the lag values to be non-negative as well. Similarly, we limit RAP to only predict non-negative values (values less than zero are set equal to zero). Note that we still train a separate regression leaner for each activity, but they are now able to utilize context from the other activities (and past predictions for their own activity) through the lag features.

During training the lag values are created from the labeled sensor event data. Since the adjusted lag is used, these values are simply the activity occurrence time labels for each event, equivalent to the ground truth adjusted lag. These values represent the "expert" the learner is trying to model. During testing or use of the learned model, these values are provided from the actual predictions made by the model for each event.

When testing with pre-recorded datasets (e.g, a validation test), we can initialize the lag values on the first event using the ground truth label values as during training. This is because with the pre-recorded data the time until the next occurrence of each activity at the start is already known. When using the learned model in a live environment, however, we must consider how the lag values are initialized. Since the lag value represents the adjusted activity occurrence time from the previous event, we need to know when these activities will occur in

the future. Thus, when we start using the model we do not know what the lag values should be. However, we can use an activity recognition algorithm to determine what activities occur as sensor events are observed. The lag values can be fully initialized once all of the activities are detected to occur. Once this occurs, we can initialize the lag values for the original event (before the first activity occurs), using the time from that event to each of the occurrences for the lags. From this point, we can use the observed sensor events between that initial event and the current one with the RAP model to form forward predictions until we "catch up" to the current time.

Alternatively, an IP model could be trained alongside the RAP. This model could then be used to form predictions for the first online event to initialize the lags, with the recurrent model used for subsequent events. While this initialization may not be as accurate as waiting for all activities to occur, it avoids the delays caused by waiting for the activities and allows prediction to start immediately.

4.3 Two-Pass Activity Predictor

4.3.1 RAP Training Issues

The RAP algorithm can be susceptible to certain problems due to the way in which the regression learner is trained. During training the ground-truth activity labels are used to generate the adjusted lag features. Since these values are "perfect" within the context of the data, the lag values are equal to the actual time to be predicted for each event. In other words, the training lag values provide as input to the learner the exact same value we wish to have it output for each event. The model tree learner will see this relationship and establish a model

where it simply outputs the adjusted lag value for the activity as the prediction for each event. Since the lag values are perfect in training, this is the optimal model to learn based on the training data, and would result in no errors on the training dataset.

However, this model does not work as well when tested or used in an online setting. The lag values are initialized either from the actual activity labels (in pre-recorded datasets) or after all activities are first observed (in a live setting). In this way, the initial lag values are also "perfect" like the training data, and the model should achieve a perfect prediction on the first event by outputting the lag value. Since the time between each event is known, the feature extraction component can accurately create the adjusted lag value for the next event, which will also be perfect. This continues and provides great results for the first events.

The problem arises when the activity occurs. At this point, the prediction output will go to zero (meaning the activity is occurring on that event). This will continue during the activity occurrence, which is fine. However, when the activity ends, the actual activity time will become nonzero to point to the next activity occurrence. RAP, however, continues to output the adjusted lag values as the new predictions. Since the previous lag value is zero (from when the activity was occurring), RAP will again output zero after the activity ends. This will continue through the rest of the testing set, as RAP essentially gets "stuck" predicting zero as the time for the next activity occurrence through its feedback loop. As a result, the predicted values become uninformative and the error will increase. So, while RAP can perform perfect predictions for all events up to and during the first occurrence of the activity, it becomes ineffective after that point. This problem stems from the simple "input-output" model learned by the model tree in training with the perfect lag values.

Another issue faced with RAP is that only one set of lag values (from the previous event)

can be used. We can add additional lag values from events before the previous one. If we adjust these lag values as we do for the first lags and still use the training labels to do so, we find that all of the lag values for each activity will be the same. That is, since each lag value is a perfectly-adjusted time value, all of the lags will provide the same information to the regression learner. Since the learner is attempting to optimize its model, it only chooses one of the lag values to incorporate in the model. The others are ignored since they provide redundant information.

4.3.2 Using a Two-Pass Method

As an alternative to these options, we present a two-pass predictor that uses a combination of independent and recurrent predictors to address many of the problems noted above. We denote this algorithm the Two-Pass Activity Predictor (TPAP). TPAP uses an independent predictor followed by a recurrent predictor, as shown in Figure 4.4. The independent predictor provides prediction labels for the sensor events, which the recurrent predictor then uses as lag values to form the final prediction. While this requires learning two separate models, it allows us to avoid the training issues found in the original RAP and also to include additional lag features for improved performance.

During training, the independent predictor is trained using the labeled sensor events with feature extraction as before. It then makes predictions for every event in the training set, and these predictions are attached to their associated events. The second-stage feature extraction then uses these predictions, along with the original features, to set up the features for the recurrent predictor. Specifically, it uses the predictions made by the independent predictor to

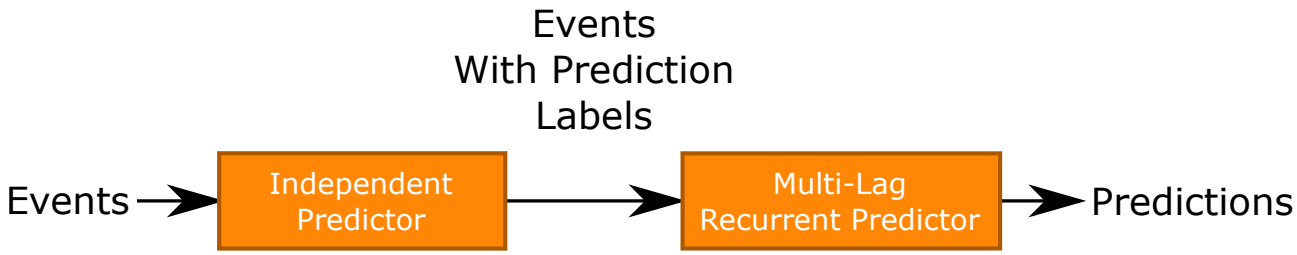


Figure 4.4: Two-Pass Activity Predictor predictor block diagram. The independent predictor is used to form prediction labels for the recurrent predictor to use for lag values.

create a number of lag prediction features for each activity. For each event, these lags consist of the prediction for that event, as well as the (non-adjusted) predictions for a fixed number of previous events. The lags are created for each activity in the set, so that all activity models have access to recent predictions from all activities. The recurrent predictor is then passed these lag features along with the original features to learn a new model. In this way, the recurrent predictor is able to use multiple lag values to improve predictions. Since these lag values are not all the same (due to the variability in the independent predictions), the recurrent model can be trained to use multiple lags. It will also not rely on only the lag prediction to form the prediction since the training lag values it sees are not perfect. Thus, it addresses the two major problems with our original RAP algorithm.

During testing or use of the method the process is similar. The independent predictor is used to form predictions for all events to be tested. These predictions are then turned into lags and are passed to the recurrent predictor, which makes the final predictions for each event. If lags from previous events are used (beyond the independent prediction for the current event), initial events will not have all of the lag values initialized. Thus, the recurrent predictor will need to wait until enough events have occurred before it forms its first predictions. However in

the interim the independent predictions can be used instead.

TPAP is similar to our previous AP algorithms, but has some notable changes that increase its utility. Although we must train both an independent and a recurrent model for each activity, this increases the flexibility of the model. We can use the independent predictor's outputs in conjunction with the recurrent predictions if desired, such as in an ensemble predictor. This method also addresses the problems faced with training the RAP predictor without needing to change the training data itself. Further, TPAP allows a flexible number of lags to be used, depending on the characteristics of the dataset we are predicting. It is also relatively simple, building on our previous methods without requiring much additional computational change in order to achieve a more robust model.

The TPAP model is also similar to the use of an off-trajectory training approach. In such an approach, a learner is trained in order to improve its ability to adapt to prediction errors. When using lag values, this can be done by training the learner using its own predictions as part of the training data, allowing it to observe erroneous lag values and learn how to correct from them. TPAP follows a similar approach in the training of its second-stage recurrent predictor. Since the recurrent predictor is trained using multiple lag values, all of which are generated by the independent predictor, it learns how to correct from erroneous predictions from the independent predictor. While the recurrent predictor is not provided with its own previous outputs for training, this adaptation could be made in order to allow an off-trajectory approach.

4.3.3 Other Regression Learners

One benefit of the TPAP method is the ability to vary the regression learners used to improve performance. We use WEKA [75] to implement the regression learners in this method. We use the same M5 model tree algorithm as before, meaning that the models are constructed in the same way as described previously. However, this also provides flexibility for testing new types of learners that may yield higher performance. For example, the model tree learners could be replaced with ensemble methods or a boosting method such as Adaboost [76]. It is likely that these methods can improve performance over the model tree. They can also be rapidly prototyped using WEKA and different learners can be used for each stage of the predictor if desired. These are ideas that we plan to examine in future experiments.

In this chapter we have presented a number of versions of our AP algorithm. The IP algorithm uses independent regression learners to separately form predictions for each activity. The RAP method utilizes information about the relationships between activities found in lagged prediction values to increase performance. Finally, the TPAP method jointly utilizes IP and RAP predictors to address potential problems with the other predictors. Each of these methods is capable of predicting human activities through observed sensor events. The variations in each are useful for adapting to different contexts with varying complexity. In the next chapter we describe various evaluation metrics which we can use to verify the effectiveness of our proposed methods.

CHAPTER 5. EVALUATION METRICS

In order to properly determine the effectiveness of an activity prediction approach, it is important to consider what metrics are being used to evaluate it. Different metrics can tend to emphasize certain aspects of a method's performance, so this must be taken into consideration. The appropriateness of a single metric can even change based on the application and data being evaluated. It may often be necessary to use several metrics to fully evaluate and compare results. In this chapter we discuss various evaluation metrics, their properties, and their applicability for activity prediction evaluation.

There are two main ways in which activity prediction can be evaluated. The first takes the view of activity prediction as a classification problem. This view is often used in the context of sequence activity prediction, where the number of incorrect predictions can be used to compute a classification accuracy. While a similar approach can be taken with numeric activity prediction (where any non-zero error is a misclassification), this fails to take into account other considerations of time-based predictions. For example, while having a perfect (zero-error) prediction may be most desirable, it is usually tolerable to have predictions with error smaller than a certain threshold (e.g, five minutes). When this is the case, we are usually more interested in the relative magnitude of the error, which is the second approach to evaluation. One option, used by [19], is to apply metrics such as precision-recall curves based on whether an event occurs within a specified time interval. However, we assess activity prediction by evaluating the amount of error present in the results. While this may be more complicated, it allows us to analyze the error in light of the requirements of the application in which we will use the

predictions.

In order to concisely describe evaluation metrics in this chapter, we will use the following notation:

- y_m^* represents a ground-truth value (label) from the set Y^* of all ground-truth values we wish to compare against
- $\hat{y}_m$ represents the predicted values from the set $\hat{Y}$ of all predictions we are evaluating
- N represents the number of instances (ground-truth value and prediction pairs) being evaluated

Most of the metrics listed in this chapter can be utilized on various subsets of the total results of an experiment. For example, we may wish to evaluate a metric separately for each activity being predicted, or for results at different test horizons. We can then macro-average the results across multiple subsets to form overall metrics for a certain dataset. This is the approach we will often take in the experiments in this dissertation.

5.1 Unnormalized Metrics

Most common metrics are defined in such a way that they can be used for direct comparison between different datasets (usually in the same units as the error itself). They are not normalized relative to different datasets, allowing their meaning to usually be intuitively determined. Most of these metrics are based on averaging of different measures of error.

Mean absolute error (MAE), defined in Equation 5.1, measures the average absolute error between the predicted and ground-truth values. The MAE does not emphasize any particular

aspect of the error, but rather evaluates all errors in an average sense [77]. Zero MAE means there is no error, and MAE has no upper bound.

$$\text{MAE} = \frac{\sum |\hat{y}_m - y_m^*|}{N} \tag{5.1}$$

Another common measure is root mean squared error (RMSE), defined in Equation 5.2. It uses the square of the error, with the root used to return the measure to the same units as the error. RMSE tends to emphasize the largest errors due to the square, so it may be strongly affected by only a few instances of large error [77, 78]. Like MAE, zero RMSE indicates perfect performance and RMSE has no upper bound.

$$\text{RMSE} = \sqrt{\frac{\sum (\hat{y}_m - y_m^*)^2}{N}} \tag{5.2}$$

If the sign of the errors is of interest, mean signed deviation (MSD; also called average error) can be used. MSD is similar to MAE, except that it allows the sign of the error to be taken into account. This allows us to determine whether the predictions are, on average, above or below their respective ground-truth values. MSD is defined in Equation 5.3. An MSD of zero indicates no error, but MSD is unbounded in both the positive and negative direction.

$$\text{MSD} = \frac{\sum \hat{y}_m - y_m^*}{N} \tag{5.3}$$

As these metrics use an average of the error values, they may be affected by outliers. If this is a concern, a corresponding version of each of these metrics is available which uses the median of error values rather than the mean.

5.2 Normalized Metrics

While the unnormalized metrics are widely-used, they do not take into account the differences in scale between datasets. For example, consider the prediction of a "Toileting" activity and a "Housekeeping" activity. The Toileting activity may occur multiple times a day, while the Housekeeping activity could occur as infrequently as once per month. Using an unnormalized metric, we may find that the average error for Toileting is around a minute, while it is a few hours for Housekeeping. This would appear to indicate that the Housekeeping predictions are much worse than those for Toileting. While this is true in an absolute sense, it may not be the case in a relative sense given the differences between the activities. While an error of hours may be unacceptable for Toileting, this may not be the case for Housekeeping since the times between occurrences of that activity are so large. In order to address this and allow comparison between different datasets and contexts, it may be desired to use a normalized error measure. Usually, these measures normalize a common unnormalized metric by some aspect of the error values or the dataset itself.

A common normalized metric is the mean absolute percentage error (MAPE), shown in Equation 5.4. MAPE normalizes each error by the corresponding ground-truth value, meaning that it is an "average of normalized errors" value. This is useful as it tends to de-emphasize large errors due to the normalization. However, MAPE can be greatly affected in cases where the ground-truth value y_m^* is at or near zero. For these instances, even small errors will be normalized to very large (possibly infinite) values, leading to distorted MAPE values. While MAPE has no upper bound, a value of zero indicates perfect prediction, while a value of 1

indicates the error is as large as the ground-truth values themselves.

$$\text{MAPE} = \frac{1}{N} \sum \left| \frac{\hat{y}_m - y_m^*}{y_m^*} \right| \tag{5.4}$$

Other normalized measures utilize the unnormalized metrics listed above. Those metrics can be normalized by the range of the ground-truth values, leading to the range-normalized MAE and RMSE (RangeNMAE and RangeNRMSE), defined in Equations 5.5 and 5.6.

$$\text{RangeNMAE} = \frac{\text{MAE}}{\max(y_m^*) - \min(y_m^*)} \tag{5.5}$$

$$\text{RangeNRMSE} = \frac{\text{RMSE}}{\max(y_m^*) - \min(y_m^*)} \tag{5.6}$$

Another option is to normalize by the mean of the ground-truth values to determine the mean-normalized MAE and RMSE (MeanNMAE and MeanNRMSE), defined in Equations 5.7 and 5.8.

$$\text{MeanNMAE} = \frac{\text{MAE}}{1/N \sum y_m^*} \tag{5.7}$$

$$\text{MeanNRMSE} = \frac{\text{RMSE}}{1/N \sum y_m^*} \tag{5.8}$$

All the range-normalized and mean-normalized metrics have no upper bounds, though a value of 1 indicates that the error is as large as the normalizing factor. Values of zero indicate no error.

While normalized metrics can be useful for comparing data across datasets, the normalization process causes them to lose their relationship with common units. This can make them difficult to interpret, since we now only have a unitless value rather than a value in seconds or some other units as with unnormalized metrics. Normalized metrics also tend to be less-used in the literature. Thus, when deciding to use normalized or unnormalized metrics, it is important to consider whether the normalization is needed based on the application and other concerns.

5.3 Other Metrics

Since the above metrics are based on the average error of results, they do not provide much information about the form and other characteristics of the errors. They also may be subject to distortion by outlier error values. In order to understand the distribution of the errors and to detect outliers, it is often useful to include other error metrics.

A common metric is Pearson's r, or the correlation coefficient between the predicted and actual values. It is defined in Equation 5.9, where $\overline{\hat{y}}$ is the average predicted value and $\overline{y^*}$ the average ground-truth value. r provides a measure of the relationship between the actual and predicted values, and ranges from -1 to 1. A value of zero indicates that the predicted and actual values have no relationship to each other (no dependence), indicating bad predictions. A value of -1 indicates an inverse correlation, while a value of 1 indicates the predictions are highly correlated to the corresponding label values. Usually values of r near 1 are desired. The correlation information allows us to observe whether the predictions are properly changing in a manner similar to the actual values.

$$r = \frac{\sum(\hat{y}_m - \overline{\hat{y}})(y^*_m - \overline{y^*})}{\sqrt{\sum(\hat{y}_m - \overline{\hat{y}})}\sqrt{\sum(y^*_m - \overline{y^*})}} \tag{5.9}$$

Another metric is the number of sign changes (NSC) [77]. The NSC is a count of the number of times that the sign of the error changes in the sequence. This provides information about whether the predictor is consistently over- or under-predicting the values. An NSC value of zero indicates that the error is consistently of the same sign, though it does not necessarily indicate a perfect prediction.

The absolute maximum error AME, defined in Equation 5.10, is simply the largest absolute error value in the sequence [77]. It has no upper bound and a value of zero indicates

perfect prediction. AME may be useful if the error cannot be allowed to go above a certain threshold. For example, if an inhabitant must be reminded to take their medicine within an hour interval, we may want to avoid errors greater than one half-hour. AME would allow us to ensure the largest error is within this limit, although it would not tell us anything about the distribution of the other error values.

$$\text{AME} = \max(|\hat{y}_m - y^*_m|) \tag{5.10}$$

Another metric that may be used is the peak difference (PDIFF) [77], defined in Equation 5.11. It simply measures the difference between the maximum predicted value and the maximum ground-truth value. It does not account for the correlation between the peak values or the results as a whole. However, it provides some information as to whether the predicted values achieve the same range as the ground-truth values. PDIFF is unbounded, though a PDIFF of zero would indicate that the predicted and actual values reach the same upper limit.

$$\text{PDIFF} = \max(\hat{y}_m) - \max(y^*_m) \tag{5.11}$$

Others have defined additional metrics for measuring error. Torgo and Ribeiro [79] adapted precision and recall metrics to be usable for regression. Others have used standard deviation and normalized variations of it as a metric [80].

We introduce a new metric, the error threshold fraction (ETF), defined in Equation 5.12. ETF is defined for a non-negative scalar threshold value v. It quantifies the fraction of errors (out of all errors in the sequence) that are smaller than the threshold in magnitude. For a given threshold v, an ETF of 1 indicates all errors are below the threshold, while a value of zero indicates all the errors are above the threshold. $\lim_{v\to\infty}\text{ETF}(v) = 1$, and we can vary v to see how the errors are distributed. If the ETF does not approach 1 until v is large, this

indicates that there are a significant number of large-error outliers. ETF(0) indicates what fraction of the predictions have zero error.

$$\text{ETF}(v) = \frac{||\hat{y}_m - y_m^*| \leq v|}{N} \tag{5.12}$$

5.4 Ground-Truth Considerations

When considering the activity prediction problem, there are some issues to consider related to the activity labels which may affect prediction accuracy. Recall that the data used for activity prediction training and testing relies on activity labels generated by human annotators or an activity recognition algorithm. In the case of human annotation, the activity labels usually have some inherent error. This is due to the inability of annotators to perfectly observe and interpret inhabitant activities and create perfect labels. With activity recognition algorithms, there may be errors in the model representation that leads to errors in the labels. Activity recognition algorithms are also subject to inaccuracies introduced in the human-annotated data used to train them.

Due to these inaccuracies in the "ground-truth" data, it is important to consider how the prediction performance may be affected and to quantify this effect. We should not expect the activity predictor to have perfect accuracy if there is error or disagreement within the data used to train it. One method that is frequently used to address these issues is Cohen's kappa [81]. It allows us to measure the inter-annotator reliability of labels from multiple annotators (human and machine). Many classification metrics can be $\kappa{-}normalized$ to emulate the performance we would expect from a perfect predictor if these discrepancies are taken into account. While these normalizations have not been as widely applied for regression metrics, we can use information

about inter-annotator reliability to inform our interpretation of prediction results. Care should be taken when analyzing prediction results to understand the effects of label reliability on prediction performance.

In order to understand and compare activity prediction results, it is important to carefully consider the evaluation metrics that are used. In this chapter we have presented a number of metrics for analyzing regression problems. Many of these metrics quantify the average or normalized average error with different emphases. Others allow us to understand the distribution of errors in the dataset in order to identify outliers and other issues. We note that, given the complexity of the activity prediction problem and inherent issues with labeled data, it is often required to use multiple evaluation metrics to fully understand the performance of a particular algorithm. In the following chapters we apply these evaluation metrics in order to understand the performance of our AP algorithms.

CHAPTER 6. CASAS SMART HOME SYSTEM

In order to verify the performance of our AP methods we wish to test them using data collected from actual smart environments. The experiments described in this dissertation were carried out using data from the CASAS smart home system. The CASAS smart home system, developed at Washington State University, consists of several smart environment testbeds installed in participants' homes. Each testbed is equipped with various sensors to monitor inhabitants' daily activities over time, as shown in Figure 6.1. These sensors generate events which are recorded in a sensor event database. The recorded sensor events and accompanying activity labels are utilized in the evaluation of activity prediction algorithms described later in this dissertation. This chapter describes the system architecture of the CASAS smart home systems and describes the testbeds used in this dissertation.

6.1 CASAS Sensors

Each CASAS smart home testbed is equipped with a number of sensors used to monitor inhabitants. These sensors are designed to work passively with minimal invasion of the inhabitants' privacy. Some of the sensors are wired-type sensors, though most use wireless ZigBee communication. The ZigBee wireless mesh is used to connect the wireless sensors with the CASAS middleware [82], as shown in Figure 6.2.

Figure 6.1: A CASAS smart home testbed. Wireless ZigBee passive infrared motion detectors are attached to the ceiling, including top center.

6.1.1 Passive Infrared Motion and Light Level Detector

Passive infrared (PIR) motion detectors use infrared light to detect motion occurring within their field of view. These sensors report a simple "ON" message when motion is detected, followed by an "OFF" message a short time after movement stops. A PIR motion detector is shown in Figure 6.3.

Motion detectors are divided into two types based on the same Control4^{TM}wireless motion sensor:

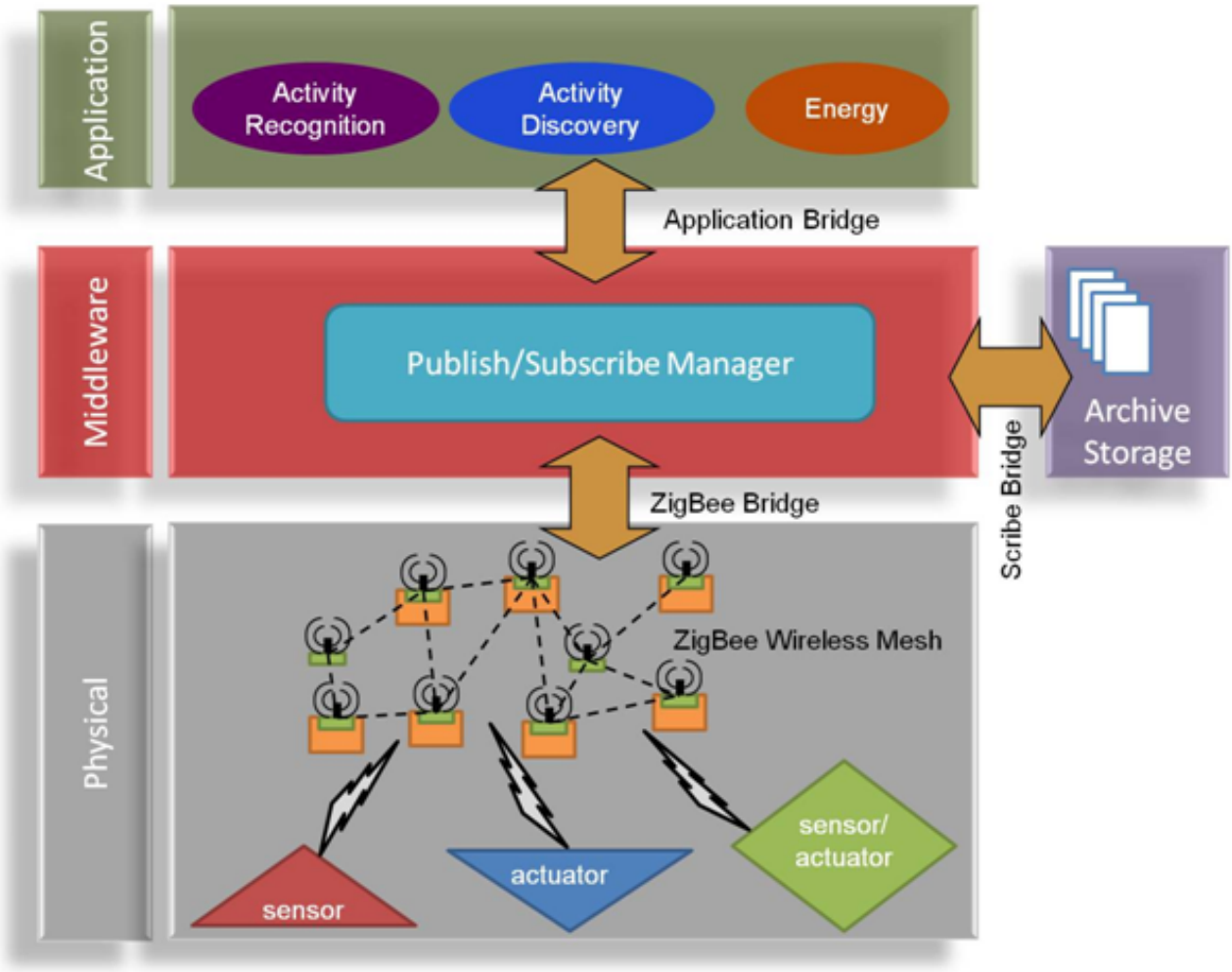


Figure 6.2: CASAS system component diagram, as detailed in [82]. The sensors form the physical layer, which communicates with the middleware layer through a ZigBee wireless mesh. The middleware layer communicates with smart home applications and sensor event storage.

Figure 6.3: CASAS passive infrared motion sensor. These sensors detect motion within their field of view.

- **Narrow-Field (Normal) Motion Detectors** These sensors have a narrow field of view covering only a few feet. They are usually placed on the ceiling in order to detect motion in a specific portion of a room. Often many of these sensors will be placed throughout a room's ceiling in order to detect inhabitants' motion through the room. Motion detectors of this type are shown on the ceiling of the room in Figure 6.1.

- **Wide-Field (Area) Motion Detectors** These sensors have a much wider field of view. They are usually placed on walls in order to detect any motion within a room and provide indication of an inhabitant's presence within the room. They generally cannot detect an inhabitant's specific location within the room.

In addition to providing ON/OFF motion events, PIR motion detectors also provide numeric indications of other parameters. They are equipped with light level sensors that observe the ambient light level within their field of view and report this as a numeric value as levels change. This allows the CASAS system to observe changes in light as an inhabitant

Figure 6.4: A CASAS door sensor, which detects whether a door is open or closed.

interacts with the space. The PIR motion detectors also report their battery charge and voltage at regular intervals.

6.1.2 Magnetic Door Sensors

Magnetic door sensors use a magnetic switch to detect when doors are open or closed. They are usually mounted on doors throughout a testbed. They send simple "OPEN" or "CLOSE" events when the door is opened or closed. These door sensors are connected wirelessly and can be run on either battery or mains power. Figure 6.4 shows a door sensor in use.

6.1.3 Temperature Sensors

Temperature sensors report a numeric value for the temperature within a room. They are usually placed on ceilings, though also can be placed in other areas of interest such as near a stove. The temperature sensors are wireless and send an event when the temperature changes.

The temperature sensors use the same wireless module as the door sensors, so temperature readings can be sent from door sensor locations as well.

6.1.4 Light Switch Sensors

Light switch sensors are installed in light switches throughout a testbed. They wirelessly report when the switch is turned on or off with simple "ON" or "OFF" events. These sensors provide insight into inhabitants' presence and need for lighting in each room.

6.2 CASAS Middleware

The CASAS middleware layer [82], as shown in Figure 6.2, handles the collection and dissemination of sensor events to other components. It uses a publish/subscribe manager to establish connections among parts of the system. This architecture allows applications running on the application layer to receive sensor events and send commands to the middleware in real time.

The middleware also allows connection to the Scribe bridge for archiving sensor events. Each sensor event generated by a testbed's sensors is stored in a database, along with a timestamp indicating when the event occurred. Example sensor events from the CASAS system have been shown in Tables 3.1, 3.2, and 4.3.

For most of the experiments in this dissertation, we used archived sensor event data. However, our activity prediction algorithms could also be used in real time by connecting to the CASAS middleware interface.

6.3 Activity Labeling

Activity labeling is performed in order to determine which inhabitant activity relates with each sensor event. This associates each sensor event with a particular activity that was being performed when that event occurred. Our activity prediction algorithm uses these activity labels to learn the inhabitant's activity patterns.

6.3.1 Human Annotation

There are two ways in which we perform activity labeling. The first method is human annotation. Human annotators look at the floorplan and sensor layout of a testbed, interview inhabitants to ascertain their daily routines, and visualize the sensor event sequence in the PyViz visualization tool [83]. The annotator interprets which activity is occurring during each sensor event and labels the event appropriately. Multiple individuals annotate activities and interannotator agreement is at the $\kappa = 80\%$ level for the activities evaluated in this dissertation.

Human annotation may be able to account for nuances in the sensor events that may go unnoticed by automated methods. However, human annotations tend to be inconsistent across different annotators and even within the same annotator's labels. It is also difficult to obtain human annotations on sensor events in a live environment in real time.

6.3.2 Activity Recognition

The second activity labeling method uses an activity recognition algorithm. Activity recognition algorithms can automate the activity labeling process, and can even be used with

activity discovery algorithms to model all of an inhabitant's behaviors [84]. The goal of activity recognition is to map a sequence of sensor events $\mathbf{x} = e_1, e_2, \ldots, e_n$ to an activity label a from a set of predefined activity labels $a \in A$. This can be viewed as a machine learning problem. Starting with a feature vector $\mathbf{X}$ that represents the event sequence $\mathbf{x}$, we wish to learn a function h that maps $\mathbf{X}$ onto an activity label in A: $h : \mathbf{X} \rightarrow A$. This function is learned from collected training data, usually labeled by human annotators. The learned function can then be used to label sequences of events with the appropriate activity labels. This results in activity labels being assigned to each event indicating which activity was occurring during the event.

Some aspects of the activity recognition problem require special attention compared to other machine learning problems. First, the sequential nature of the sensor events must be accounted for, as the ordering of inhabitants' actions is important for distinguishing among activities. Second, there are often unclear boundaries between separate activities in an event sequence. Activities will even overlap and interweave with each other. These factors must be taken into account, leading to additional data processing being required.

There are many approaches to supervised activity recognition that have been explored by our group and others [61, 62, 85–98]. They have been used with many different sensor modalities, such as environmental [99–101], wearable [102–105], object [40, 106, 107], video [108–111], and smart phones [112, 113]. Some methods are designed for specific purposes, such as energy conservation [114] or monitoring inhabitant wellness [39, 115]. Transductive techniques include template matching using a kNN classifier with fixed or variable (with dynamic time warping) sensor window size [116]. Generative techniques, including Naïve Bayes classifiers, Markov models, and dynamic Bayes networks, can perform well if there is a large amount of labeled training

data available [99,117–123]. When training data is more limited, discriminative methods which learn the boundary between different activity classes may offer improved performance. These methods include decision trees, boosting and bagging meta classifiers, support vector machines, and probabilistic graph models such as conditional random field models [99,124–126]. Ensemble approaches, such as boosting, can also be used to combine these basic methods [120,127,128].

Many activity recognition algorithms are designed only for specific situations with only a single inhabitant and pre-segmented activities. Our group's recent work allows generalization to consider activity models for multiple inhabitants [122,129]. Our activity recognition algorithm, AR, has achieved over 95% accuracy for 30 activities in 20 smart homes. While many other approaches require offline data segmentation of activities [106, 130–136], AR avoids this by extracting features from a sliding window of events that moves over the data [137]. The window's size is adjusted automatically based on the most likely activities associated with each sensor event and their corresponding durations.

It is difficult to recognize activities that overlap or are performed in parallel. One approach is to train a model on each separate combination of activities, but this leads to an exponential number of combinations that must be considered. Other approaches include training for single-activity situations with coupled hidden Markov models or factored conditional random fields [138–151], but this requires a substantial amount of training data to be available. Instead, AR uses a weighting scheme for events in the sliding window based on the relationship between sensor occurrences. This weighting is based on the empirical probability of two sensors occurring sequentially within the window sequence (used as mutual information in [137]). If we allow S_i and S_j to represent the ith and jth sensors in the set of sensors being used, then their weighting is defined as in Equation 6.1, where N is the sliding window size. Here, s_k is the sensor for the

kth event in the window; s_{k+1} is the sensor for the event after that. $\delta(s_k, S_i) = 1$ if $s_k = S_i$ (if the kth event is from sensor S_i) and 0 otherwise; the other term is defined similarly. The summation will only be increased if S_i and S_j occur sequentially in the sensor sequence. The use of this weighting allows AR to focus on sensor events that are closely related and thus filter out sensor events that are not directly relevant to the activity of interest.

$$\mathrm{W}(i,j) = \frac{1}{N}\sum_{k=1}^{N-1}\delta(s_k, S_i)\delta(s_{k+1}, S_j) \tag{6.1}$$

AR generates a set of feature values from the sliding window of sensor events which relate information such as the time of day and the most frequent recent sensors, similar to our IP algorithm. These features are then passed to a C4.5 decision tree model [152]. During training, the decision tree creates a tree structure, choosing features at each node so as to maximize the information gain. When used for activity labeling, the tree is traversed until a leaf node is reached, at which point an appropriate activity class label is found.

While AR can be used to automate activity labeling, its performance is dependent on the accuracy of the learned model. However, AR tends to be more consistent in its annotations compared to human annotators. It also can be used to label large sensor event datasets at much lower cost than with human annotators. Our AR algorithm is used throughout this dissertation to generate activity labels to support the activity prediction methods. Details of the experiments in the next chapter will indicate the manner in which AR is used for both training and testing datasets.

6.4 Smart Home Testbeds

We use sensor data from a number of CASAS testbeds for the analyses in this dissertation[1]. Each testbed has one or more inhabitants who went about their daily routines as sensor data was collected. Following is a description of the datasets in each group.

6.4.1 "Bosch" Testbeds

This group consists of three CASAS apartment testbeds, each housing one older adult over age 65. The inhabitants performed their daily routines for at least six months while data was collected. The layout of these apartments is shown in Figure 6.5.

These testbeds provide basic datasets for testing activity prediction algorithms. They contain only motion and door sensors, simplifying the types of sensors to be examined. Each testbed only houses one resident, reducing the complexity of activities compared to having multiple residents. These testbeds are used in the experiments described in Section 7.2.1.

6.4.2 "Horizon House" Testbeds

This group consists of 25 CASAS apartment testbeds. Each apartment houses one or two older adults over age 73. Each testbed includes at least one bedroom, a kitchen, a dining area, and at least one bathroom. These testbeds utilize not only motion and door sensors, but also light switch, light level, and temperature sensors. Information and a layout for each testbed

[1]Many of these datasets are available at http://casas.wsu.edu

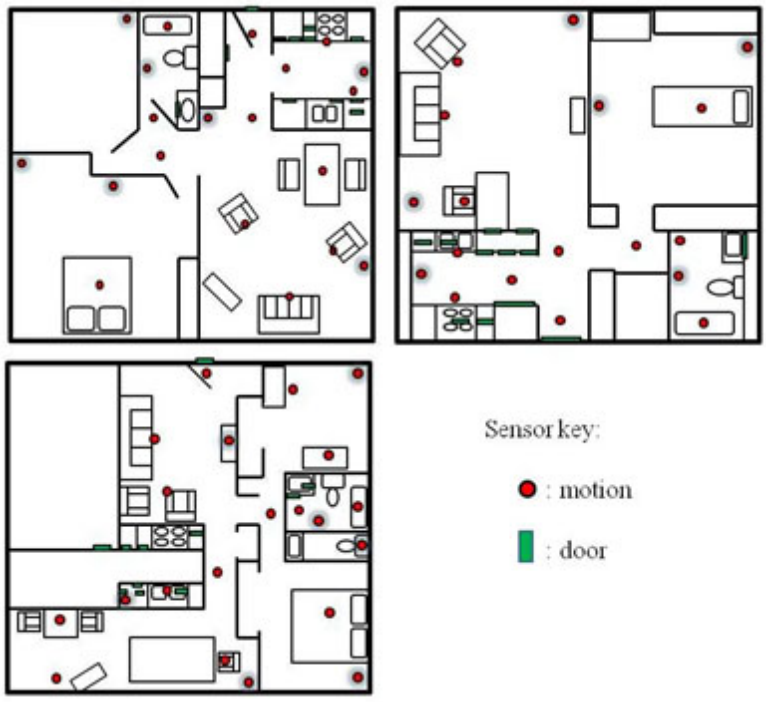


Figure 6.5: Bosch apartment testbed layouts. Each apartment houses one older adult. Only motion and door sensors were used in these testbeds.

are found in the appendix.

Two separate groups of datasets from the Horizon House testbeds are used, representing different time periods and events. One was labeled by human annotators, while the other was labeled using AR. The diversity of these datasets, including the presence of more than one resident in some apartments, allows for examining activity prediction performance in more difficult situations. These datasets are used in many of the experiments in Chapter 7.

6.4.3 "Navan" Testbed

The Navan testbed consists of an apartment with one resident. It is equipped with motion, door, light level, and temperature sensors. The testbed also has thermostat and power metering sensors. The sensor layout for this testbed is shown in Figure 6.6. Data from this testbed is utilized for the prompting application trials in Chapter 8. Details about the data used are described in that chapter.

6.4.4 "Kyoto" Testbed

The Kyoto testbed houses two residents in a two-level apartment. However, it is also used as an experimental testbed for controlled tests of the CASAS system with various participants. These experiments occur on weekdays between 9:00 am to 4:00 pm. In order to capture only data from the residents' behaviors, sensor events that fall into the experiment time periods are removed.

This testbed is equipped with motion, door, light switch, light level, and temperature sensors. It also has item sensors to detect use of specific items and water flow sensors. The testbed sensor layout is shown in Figure 6.7. Data from the Kyoto testbed is used in Chapter 8 and are described further there.

We use the data collected from these CASAS testbeds to demonstrate the AP methods described in this dissertation. These datasets provide a convenient testing environment for our AP methods in a real-world setting. By using data collected from observation of actual inhabitant activities, we are able to understand how well the methods perform in a variety

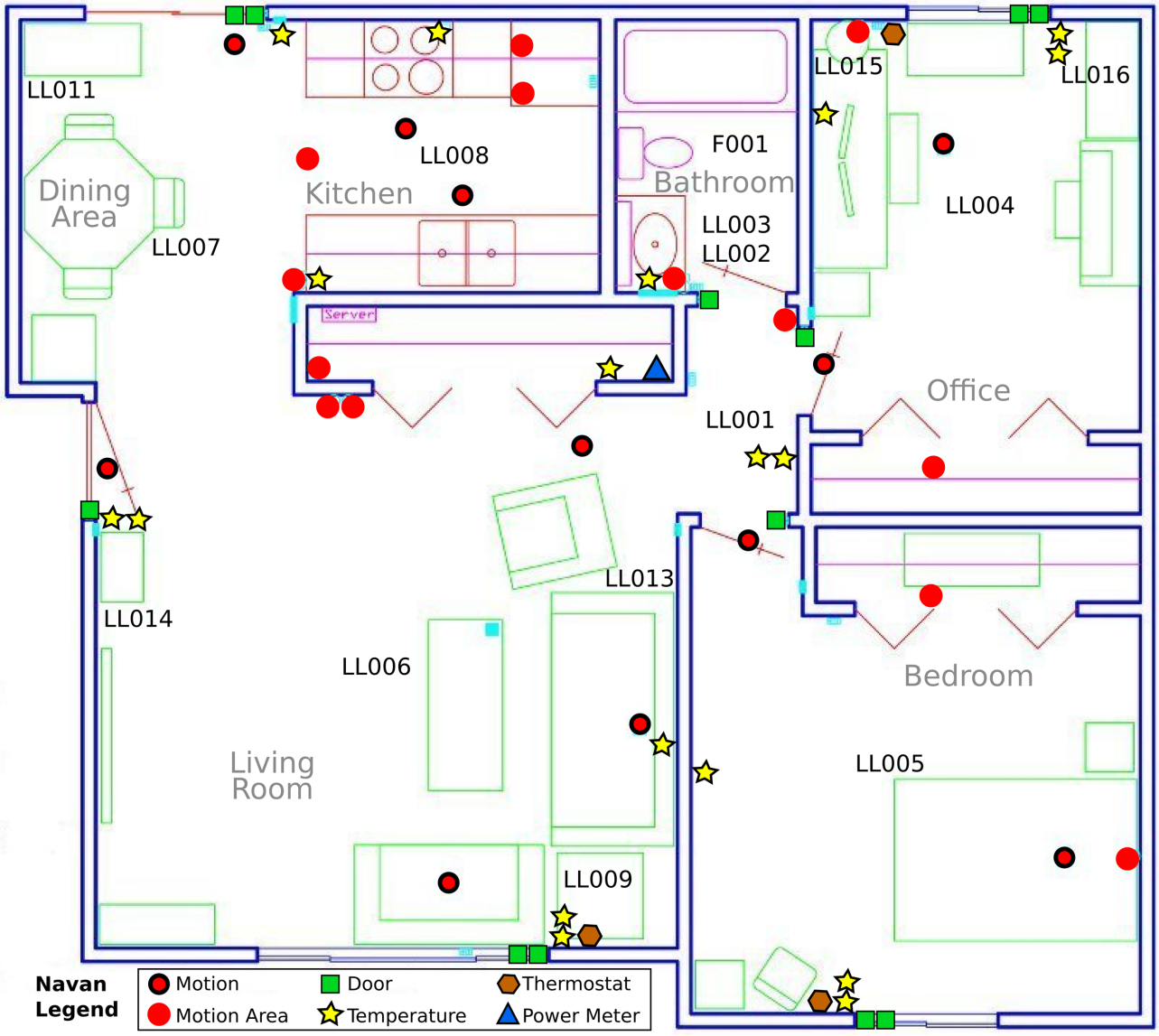


Figure 6.6: The Navan testbed layout. LL = light level sensors.

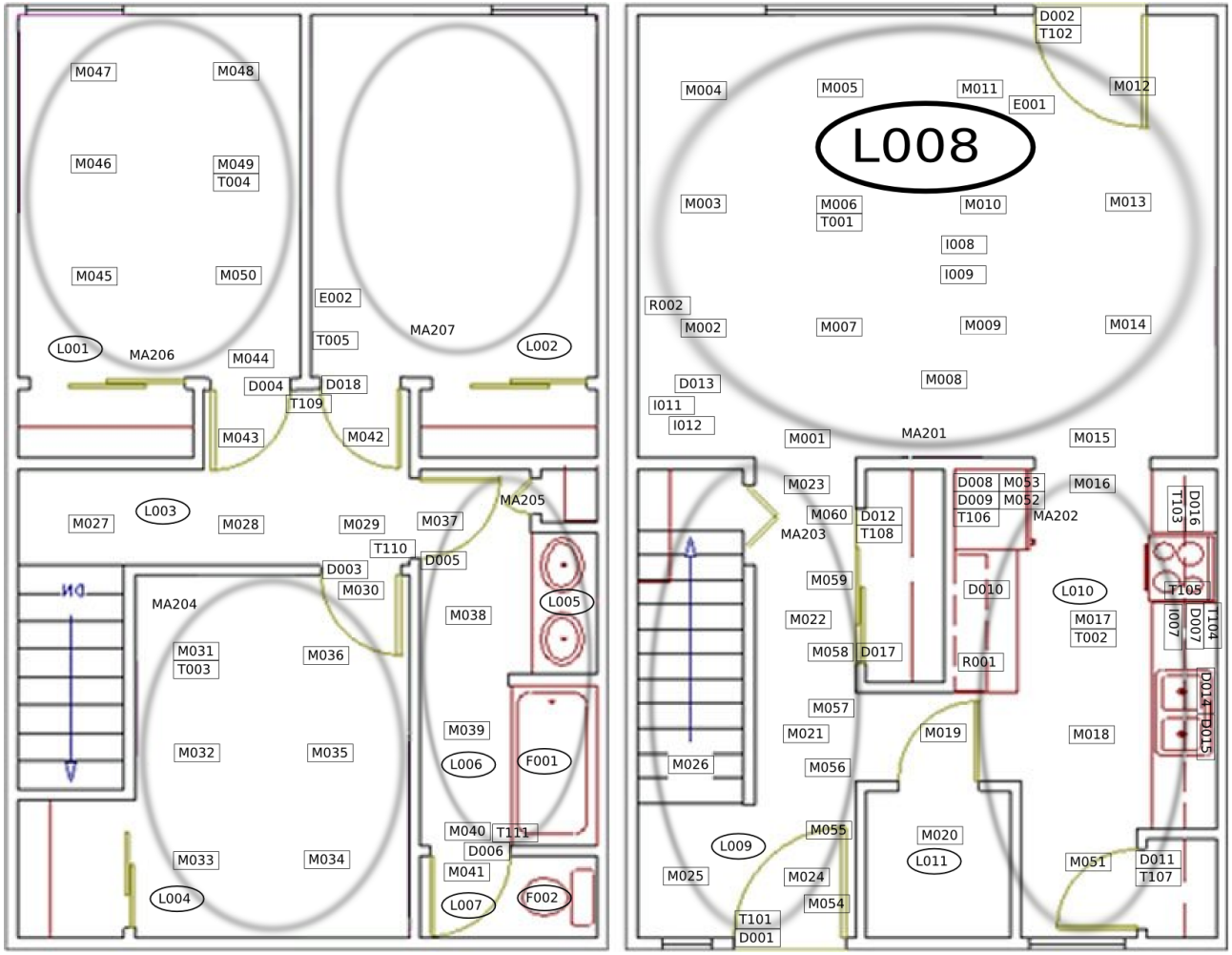


Figure 6.7: The Kyoto testbed layout. Legend: M = motion sensor, MA = motion area sensor, D = door sensor, L = light switch, LS = light level sensor, T = temperature sensor, I = item sensor, F = flow sensor.

of contexts. In the next chapter we describe a number of experiments to validate our AP methods.

CHAPTER 7. EXPERIMENTAL RESULTS

In this chapter we present a number of experiments with which we evaluate our AP methods. Using the CASAS sensor data and the evaluation metrics described earlier we are able to test the performance of our methods in a number of situations. This allows us to understand how the algorithms perform in different contexts and situations, while also determining the strengths and weaknesses of each model. The goal of these experiments is to validate our activity prediction hypothesis with our AP algorithms. We begin by introducing the sliding window validation method used for most of these experiments before describing each experiment and its results.

7.1 Sliding Window Validation

In order to evaluate the performance of our AP algorithms, a form of validation must be used. In a classification problem this would typically take the form of a k-fold cross-validation. In this form of validation, the data is divided into k folds, each consisting of randomly-chosen data points from the overall set. The regression model is trained on all but one fold and the model is tested on the remaining fold. This is repeated for all k folds and the results are averaged to obtain the overall performance.

However, normal k-fold cross-validation cannot be directly applied to the activity prediction problem due to the sequential nature of the data. It is important that the training and testing maintain the sequential ordering of the events, but a random fold selection would disrupt this. Instead, we use an adaptation of the cross-validation approach called *sliding window*

validation. In sliding window validation, we choose fixed training and test window sizes. Then, starting from the beginning of the dataset, we move the training and test windows through the data. The AP model is trained using the events in the training window. The learned model is then used to form predictions for the events in the test window, which follows immediately after the training window. The two windows are then moved forward by a specific number of events w_{skip} and the process is repeated, as shown in Figure 7.1. This process is repeated through the rest of the data.

Since the prediction labels for training are generated by finding the time from each event to the next event with the activity label, we stop the sliding window at the last event labeled with the activity of interest. Beyond this point, we do not know when the next occurrence of the activity will be (since it is not in our labeled data). Thus, we cannot compute the ground-truth prediction labels beyond this point. While this means that the entire dataset is not used for all activities, the number of windows used for each activity is approximately the same and most of the dataset is covered.

In this way, we form multiple separate validation windows from the whole dataset. Similar to k-fold validation, we can then average the results from all of the test windows to obtain an overall performance measure. While this method does introduce some dependency relationships amongst the validation windows, it is a convenient application of the k-fold validation to our temporal data. It also allows us to test multiple learned models from the same dataset. Note that the sliding window validation is run separately for each activity, and then the results can be averaged across all activities.

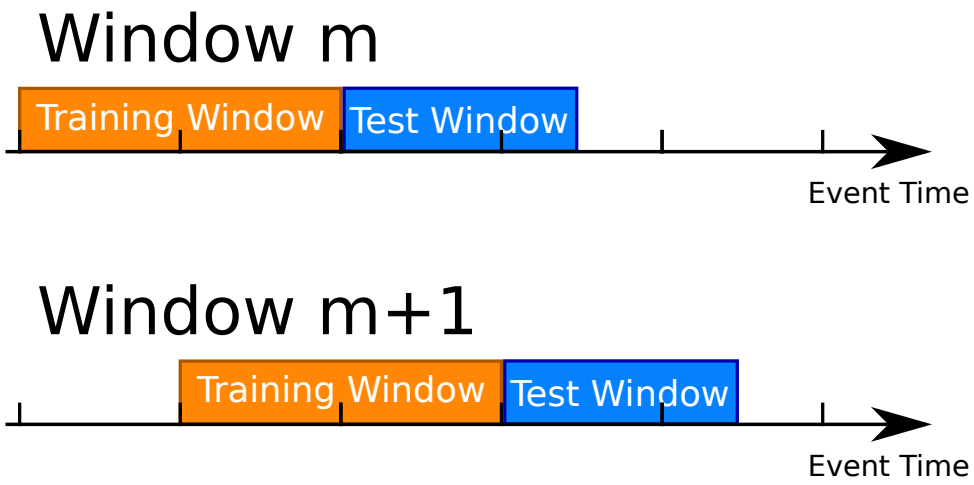


Figure 7.1: Sliding window validation. At each step, the model is trained on the events in the training window and tested on the events in the test window. Then, the windows are moved forward by a certain number of events and the process is repeated until the whole dataset is used.

7.2 IP Evaluation

7.2.1 Basic Method Evaluation

Setup

The IP predictor is our basic AP algorithm. In this experiment we wish to evaluate the effectiveness of the IP algorithm to ensure the validity of our underlying model. This will allow us to determine if using a regression learner with contextual features can provide effective automated prompting. To that affect, we test IP using discrete features in an experiment using the Bosch datasets described in Section 6.4.1. These datasets were labeled by human annotators. Each sensor event is labeled with one of 12 ADL labels, with events not belonging to any particular activity labeled as an "other" activity. The "other" category is not modeled

by AR or AP and thus is not included in the performance evaluation. The activities chosen represent a core group of ADLs that represent important aspects of daily function, such as cooking, eating, and taking medication. Tables 7.1-7.3 show the statistics for each dataset, including the distribution of activity events and occurrences in each dataset. The B1 and B2 datasets each have housekeeping activities but the residents did not work, while the B3 dataset contains work activity but no housekeeping.

We used a sliding window validation for this experiment. In order to understand how the amount of training data affected the performance of IP, we varied the size of the training window. Specifically, we collected results when the training window size was 1,000, 2,000, and 5,000 events. These lengths were chosen to ensure a sufficient number of windows were used in the sliding window validation. (Note that the average day is approximately 2,500-10,000 events in length.) For these initial tests, we used a test window size of one event. That is, for each training window, we tested the performance on the next event after the window. We also moved the sliding windows forward by $w_{skip} = 1,000$ events each time.

We compared IP against a standard linear regression (LR) algorithm. We used the same feature extraction for both our model tree learner and a LR learner. LR provides a baseline comparison as it is a commonly-used regression technique but it does not model the nonlinearity of the data. Thus, using LR as a baseline allows us to understand if the nonlinear properties of the model tree are useful for predicting activity times.

We utilize the MAE, RangeNRMSE, and Pearson's r as evaluation metrics for this experiment. The MAE provides us with a real-units measure of the error. The normalized measure allows us to compare the results between different activities and the different datasets. The correlation demonstrates the amount of agreement between the predicted and actual values in

Table 7.1: B1 dataset statistics.

Jul. 7, 2009 - Feb. 3, 2010 658,811 Events		
Activity	Events	Occurrences
Bathing	7,198	89
Bed to Toilet Transition	4,170	136
Eating	28,771	599
Enter Home	3,711	586
Housekeeping	3,280	65
Leave Home	4,305	578
Meal Preparation	101,820	921
Personal Hygiene	39,190	1,181
Sleeping (in bed)	33,213	336
Sleeping (not in bed)	39,934	737
Take Medicine	15,388	664
Work	0	0

Table 7.2: B2 dataset statistics.

Jun. 15, 2009 - Feb. 4, 2010 572,254 Events		
Activity	Events	Occurrences
Bathing	16,295	75
Bed to Toilet Transition	14,641	391
Eating	24,417	416
Enter Home	2,440	463
Housekeeping	12,971	255
Leave Home	2,476	460
Meal Preparation	55,240	615
Personal Hygiene	42,704	1,285
Sleeping (in bed)	10,477	407
Sleeping (not in bed)	16,996	215
Take Medicine	22,524	170
Work	0	0

Table 7.3: B3 dataset statistics.

Aug. 11, 2009 - Feb. 4, 2010 518,759 Events		
Activity	Events	Occurrences
Bathing	5,151	52
Bed to Toilet Transition	4,346	120
Eating	39,453	477
Enter Home	996	179
Housekeeping	0	0
Leave Home	1,246	212
Meal Preparation	44,842	638
Personal Hygiene	37,237	711
Sleeping (in bed)	20,693	306
Sleeping (not in bed)	8,207	138
Take Medicine	1,248	68
Work	108,763	500

Table 7.4: Overall results for discrete feature tests with IP. MT = model tree, LR = linear regression. Results are macro-averages of the average for each dataset. MAE values are in seconds. Best values are in bold.

Test	Average MAE	Average RangeNRMSE	Average r
MT - 1000	**728.8**	**0.0089**	**0.9935**
MT - 2000	1,047.2	0.0104	0.9908
MT - 5000	1,436.0	0.0137	0.9864
LR - 1000	7,345.4	0.0586	0.9334
LR - 2000	15,060.7	0.1295	0.7605
LR - 5000	9,523.3	0.0452	0.9400

each group.

Results

The overall results for this experiment are shown in Table 7.4. The model tree performs better than the LR at all window sizes. It performs the best for a window size of 1,000 events. In this case, the average error is about 725 seconds (about 12 minutes). Even at the larger window sizes, the model tree has average error under 25 minutes. These error values are likely to be acceptable for many prediction tasks, indicating that our IP algorithm is able to be a strong predictor. This is further supported by the high correlation r values near 1.

The lowest average error for LR (for the 1,000-event window) is over two hours. In the worst case, the error is over 4 hours. These errors are likely to be unacceptable for many

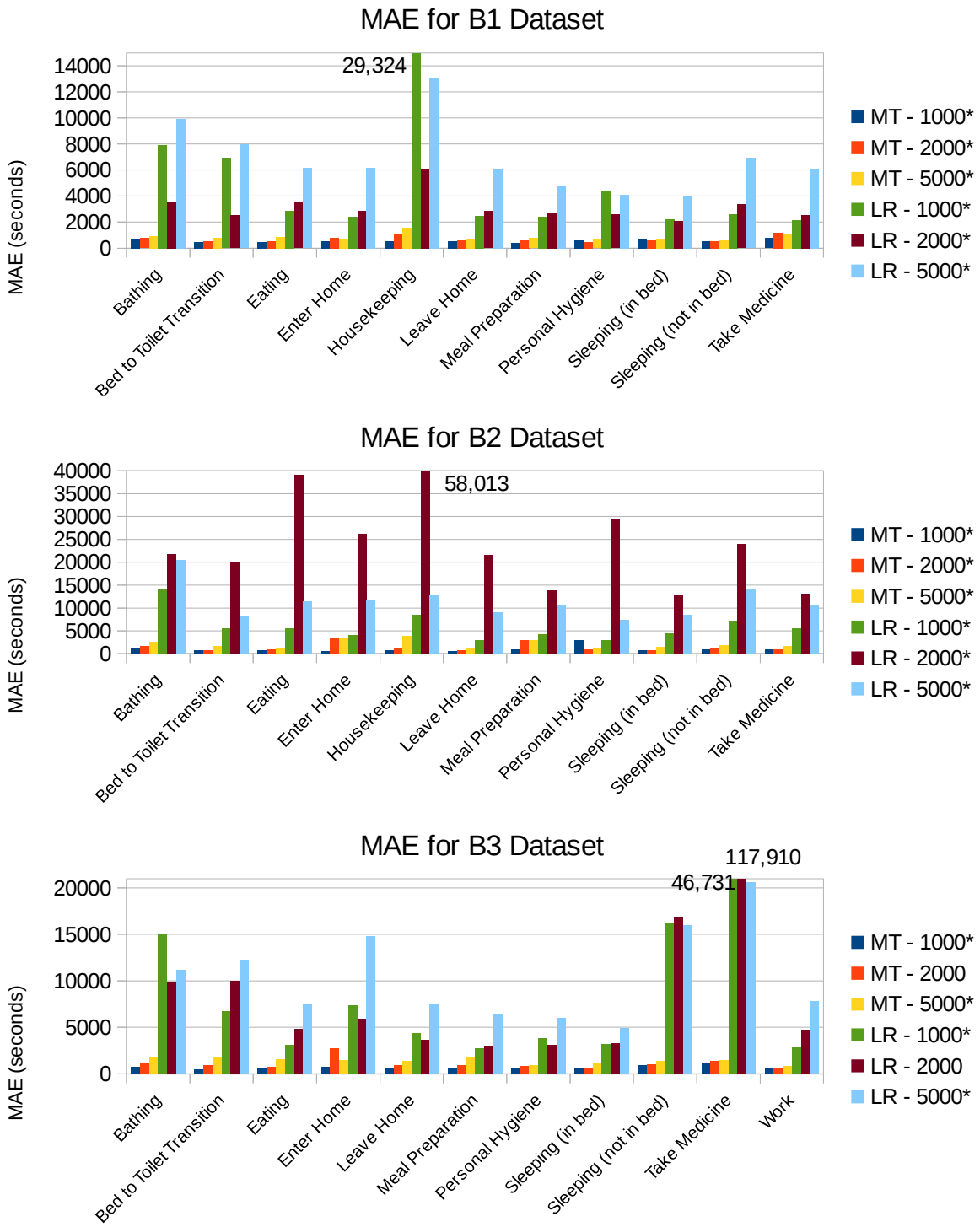


Figure 7.2: Discrete feature IP MAE results by window size. MT = model tree, LR = linear regression. *Results are significant ($p < 0.05$).

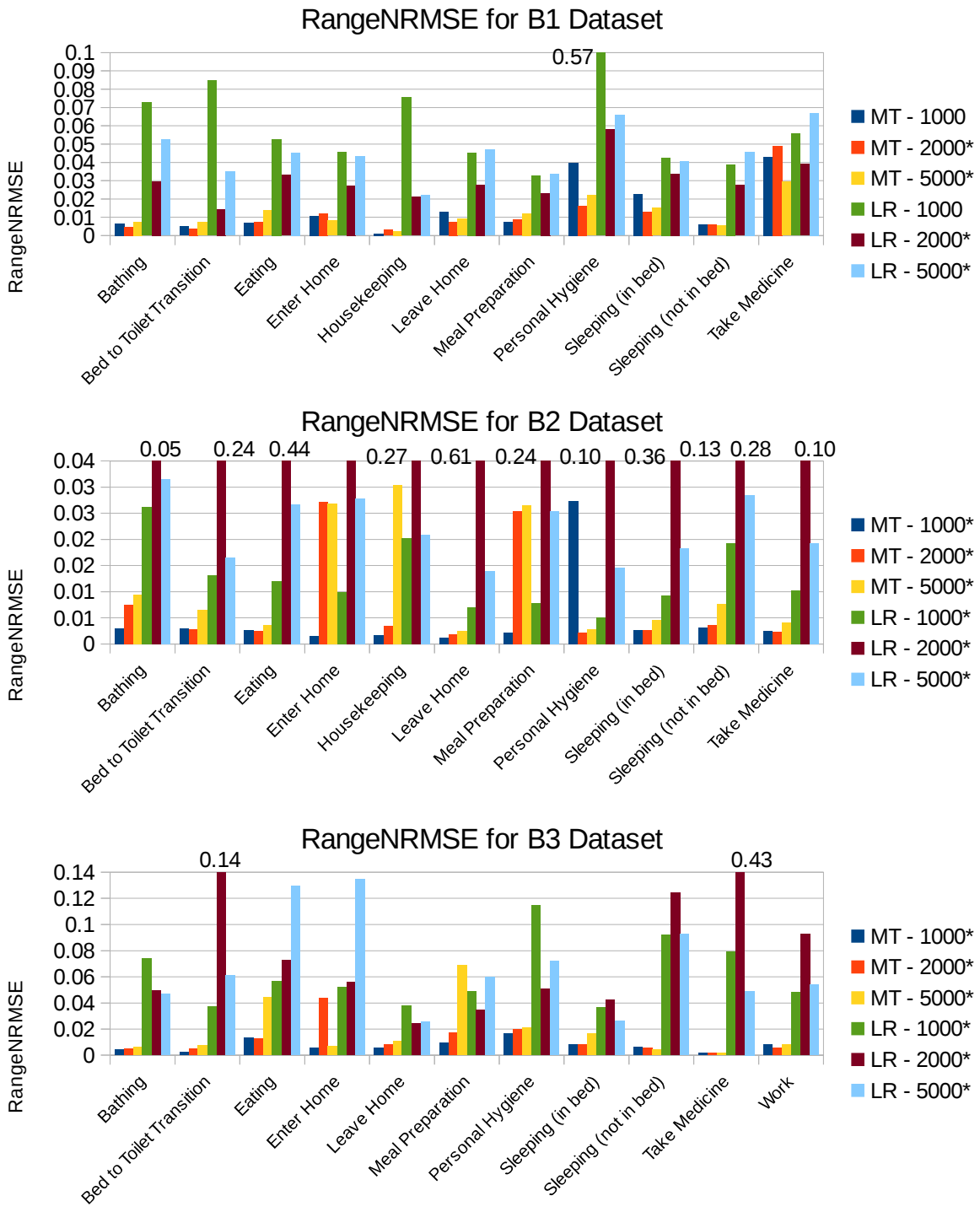


Figure 7.3: Discrete feature IP RangeNRMSE results by window size. *Results are significant ($p < 0.05$).

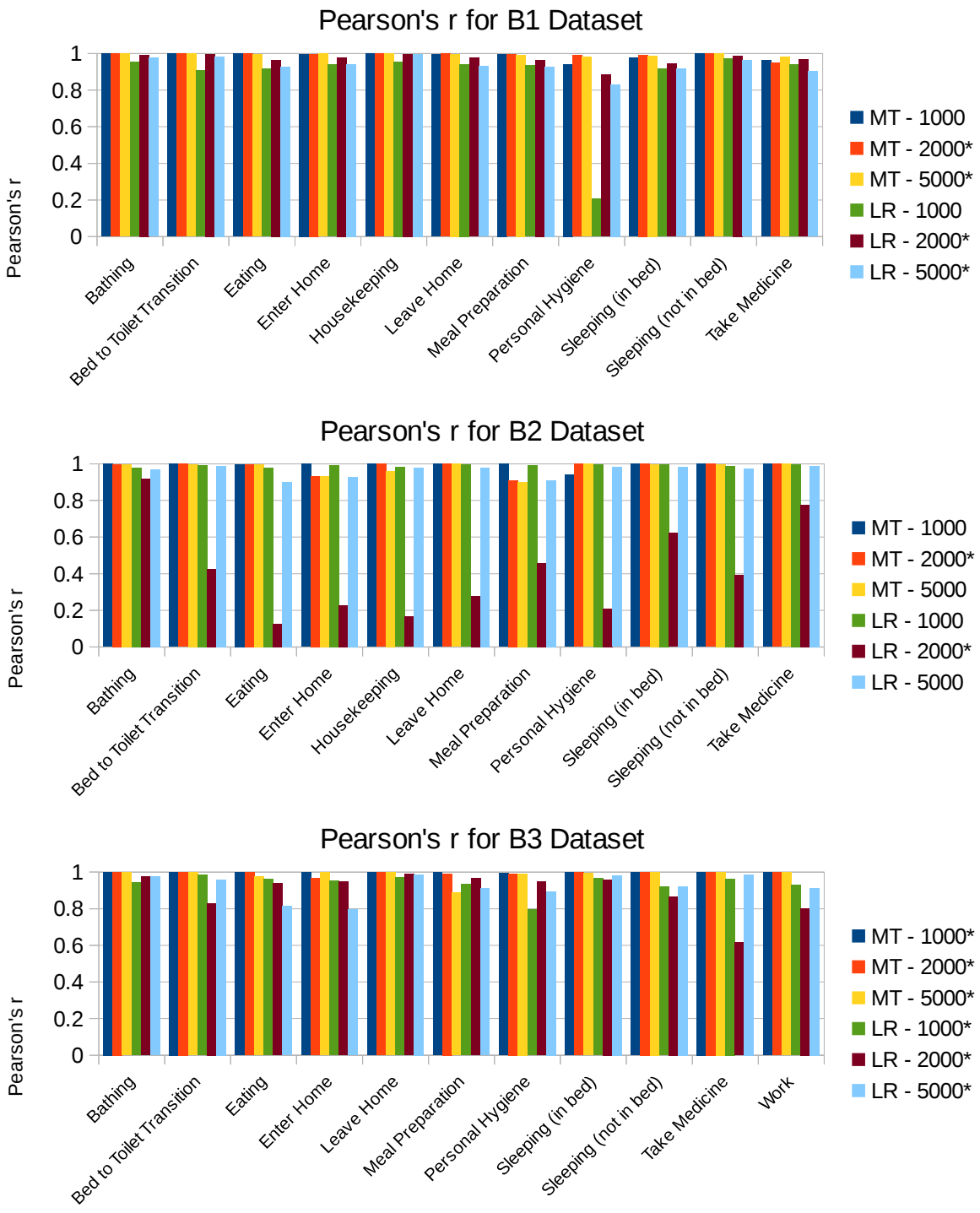


Figure 7.4: Discrete feature IP Pearson's r results by window size. *Results are significant ($p < 0.05$).

activity prediction needs. Thus, it appears that the LR model is unable to properly account for the nonlinear aspects of the activity data.

Plots of the MAE results for these experiments are shown in Figure 7.2. The predictions are especially strong for the B1 dataset, where the average error for a size-1,000 window is around 600 seconds (10 minutes). While the errors are higher for the B2 and B3 datasets, they are still within 15-30 minutes. These results indicate that the IP algorithm is able to predict activities with a good degree of accuracy. For many uses, such as energy management or certain forms of prompting, errors of 30 minutes or less would be acceptable.

The model tree performs better than the linear regression in almost all cases. This is especially true for the B2 dataset, where the linear regression predictions for the 2,000-event window average 25,400 seconds (seven hours). LR seems to have difficulty modeling the Housekeeping activity for B1 and B2. This may be due to irregularity in the housekeeping activities which make them difficult to model with a linear approach. LR also has much difficulty modeling the Take Medicine activity in B3, having an average error of over 32 hours for the 2,000-event window. It is possible this is due to irregularity or lack of frequency for the resident taking medicine, as there are not many occurrences of this activity in the dataset. Overall, the model tree used in our IP algorithm performs much better than the linear approach, indicating the importance of modeling the nonlinear aspects of the data.

Figure 7.3 shows the RangeNRMSE results. As with the MAE results, the normalized RMSE also shows a large improvement using the model tree over LR in most cases. Significant exceptions to this include the Take Medicine activity for B1. Here, the model tree does about as poorly as LR. The performance of the model tree was also relatively poor for these predictions in the MAE sense, indicating that the B1 resident's medicine-taking habits are difficult to

model even with a nonlinear approach.

The Enter Home and Meal Preparation activities for B2 also seem to have relatively high error. Note, however, that the RangeNRMSE results show more equal error between the model tree and LR than the MAE results. This may be due to the fact that the RMSE highlights the larger errors in the results which do not hold as much weight in MAE. These activities may have a few large errors which affect the RangeNRMSE, but most of the errors are likely smaller. It is also possible that the Enter Home activity has higher error in all the datasets due to the lack of sensor data preceding it. Since this activity involves the resident returning from outside (where there are no sensors), there is no motion or door data generated until the activity begins, making it difficult to predict.

The effects of the normalization are especially apparent for the Housekeeping activity. With the MAE results, both B1 and B2 had very high error for the LR model with this activity. However, the RangeNRMSE value for B1 Housekeeping is actually closer to the average, while it is still high for the B2 dataset. Likely this is due to differences in timing for this activity in the two different testbeds. In B1 the time between Housekeeping occurrences is likely greater (supported by the low number of occurrences in the dataset), which helps lower its normalized error.

The correlation results are shown in Figure 7.4. The r values for the model tree are almost always over 0.99. This indicates that there is very high correlation between the predicted and actual values. Again, the Enter Home activity provides some difficulty, with a lower correlation value in the B2 dataset. Overall, however, this shows a strong ability of the model tree and IP to form accurate predictions.

As with the other metrics, LR has lower performance in this aspect as well. While it has

some high correlation values for certain activities, it has less correlation for many others. This is especially the case for some of the window sizes for Personal Hygiene. In some ways this is surprising, as Personal Hygiene is the activity with the most occurrences in each dataset. The difficulty of performance here may be due to a variety of different sub-activities being labeled as Personal Hygiene. These activities may include grooming, brushing teeth, and other tasks which are similar but may occur at different times. The human annotators may have also been inconsistent with their labeling of this activity due to its variety, leading to higher variation in the data.

It is interesting to note that the best performance in many cases comes from the 1,000-event window, the smallest window size used. This may be due to the fact that the sliding window validation tests the next event after the training set. It is likely this event is closely related to the training window data and having a smaller training window helps the model "focus" on the prediction of this event better. As a result, the prediction for the test point is more accurate in this case. This is something we are continuing to investigate and will also be discussed further in later sections.

Table 7.5 shows the statistical significance of the results. The p-values are calculated using a two-tailed paired t-test comparison of the model tree and LR results for each window size and dataset. In all but one case the results are significant (p-value < 0.05). The differences for B3 with 2,000-event windows are less significant, which may be due to the large variance in the error values for that case.

From these results, we determine that a combination of contextual features and regression learners can be used to form accurate activity predictions. IP is able to form predictions with an acceptable level of error. The model tree used in IP also performs well compared

Table 7.5: Significance results for discrete features. The p-values are computed using a two-tailed paired t-test between the model tree and LR MAE results for the activities. Significant p-values < 0.05 are marked.

B1		B2		B3	
Window Size	p-Value	Window Size	p-Value	Window Size	p-Value
1000	0.0491*	1000	0.0006*	1000	0.0342*
2000	0.0001*	2000	0.0001*	2000	0.1567
5000	0.0001*	5000	0.0001*	5000	0.0001*

to the LR baseline, having much lower error compared to the linear approach. Thus, it is important to model the nonlinearity of the activities in prediction. These results also indicate that the discrete features used provide a significant amount of information about the underlying activities represented in the sensor data. They allow the IP algorithm to form relatively strong activity predictions in an automated fashion. Thus, we conclude that IP with discrete features provides a strong predictor that can be used as a basis for comparing our other methods.

7.2.2 Sampling Feature Improvements

Setup

In this experiment we wish to see how the sampling features can be used to improve the performance of the IP algorithm compared to the use of discrete features alone. We wish to see if the state-based information provided through the sampling features is useful for activity

prediction. We use the Horizon House testbeds described in Section 6.4.2 for these experiments. For these experiments, the datasets were labeled by human annotators, with each dataset being annotated by multiple people to maximize label accuracy. In total, there are 118 distinct activity labels used amongst the 25 datasets. However, there are 30 core activities that are each present in 19 or more of the 25 datasets which represent the most important aspects of daily living (e.g. Eating, Cooking, Sleeping, etc). We assess only these 30 core activities in order to compare results across multiple datasets and to ensure a sufficient number of instances for each activity. Table 7.7 shows an overview of each of the datasets.

For these experiments we again used the sliding window validation approach used previously. For simplicity, we used a single training window size of 2,000 events. As before, the test window size was one event and $w_{skip} = 1,000$.

Sensor Comparison

When using sampling features we may utilize additional sensor types that were not used in the previous experiment. While the Bosch datasets used previously only had "discrete" sensor events (e.g, motion and door sensors), this is not the case for the Horizon House datasets. These datasets also contain "sampling" sensor events, where numeric continuous values are reported (e.g, light level, temperature, and battery sensors). We want to determine what affect the inclusion of these additional sampling sensors have on IP performance. In this way we can see whether performance increases when using sampling features are due to additional sensors or to the features themselves. Thus, we compare results of the sliding window validation for two sets of the Horizon House data: one with only the discrete sensors present, and a second with all sensors included. In both cases, only discrete features are used.

Table 7.6 shows the overall results for these tests. The average MAE is 1,173 seconds

Table 7.6: Overall results comparing discrete vs. all sensor types. These values represent macro-averages over the results for all activities in all datasets. Best results are in bold. Note that the error for HH125 was equal in both cases as that dataset has no sampling sensor events.

Test	Discrete Sensors	All Sensors
Average MAE	1,173.3	**749.2**
Average RMSE	8,858.2	**6,017.8**
Number of Datasets Where Test Has Lowest Average MAE	5	**19**
Number of Datasets Where Test Has Lowest Average RMSE	**13**	11

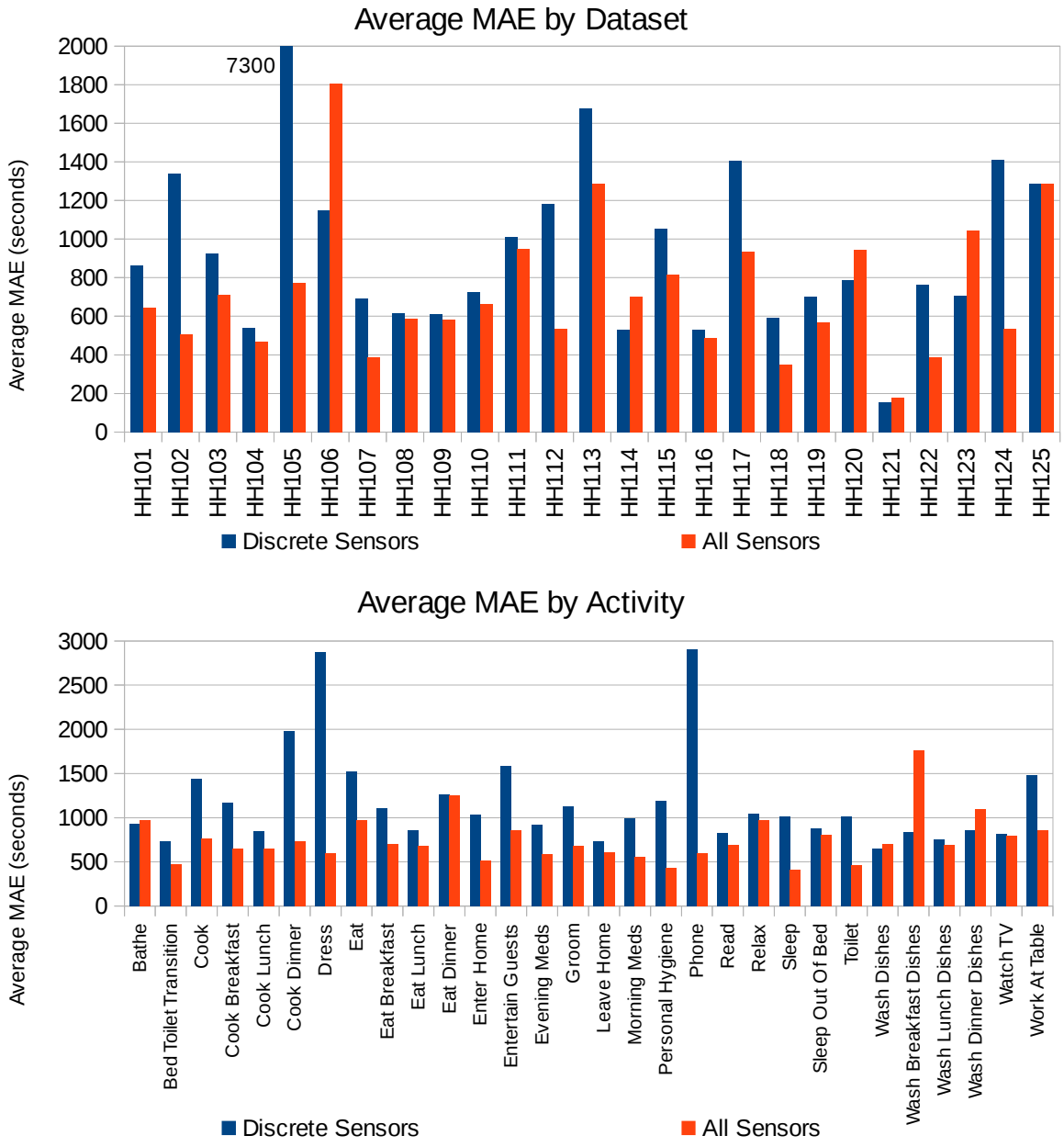


Figure 7.5: Sensor type comparison MAE results. Top plot shows the average MAE for each dataset, while the bottom plot shows the average MAE for each activity (across all datasets). All tests are with discrete features only and 2,000-event training windows. Dataset results are not statistically significant ($p < 0.05$), but activity results are.

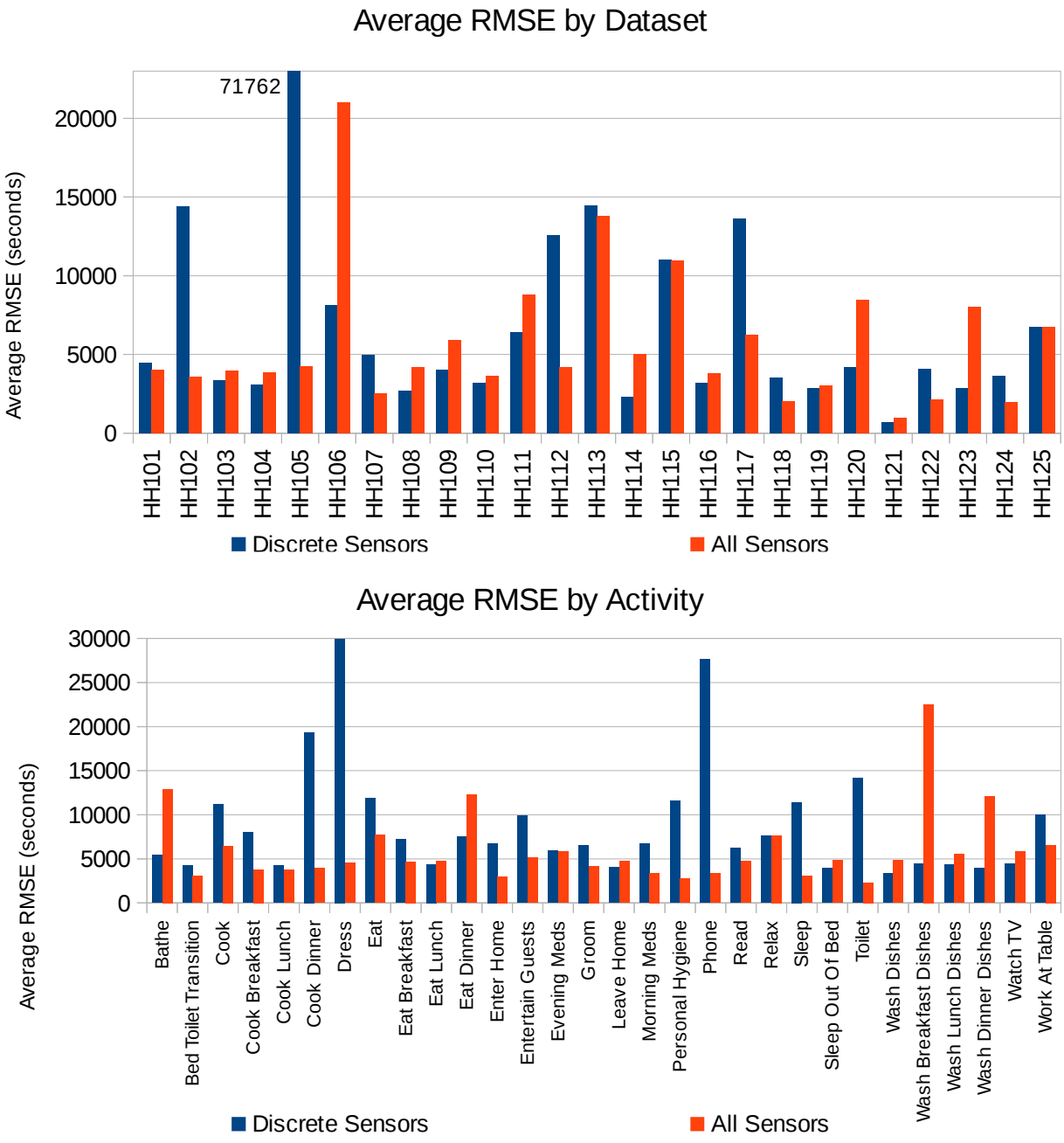


Figure 7.6: Sensor type comparison RMSE results. Top plot shows the average RMSE for each dataset, while the bottom plot shows the average RMSE for each activity (across all datasets). All tests are with discrete features only and 2,000-event training windows. Results are not statistically significant ($p < 0.05$).

Table 7.7: Horizon House datasets for sampling features.

Dataset	Residents	Timespan	Events	Labeled Activities
HH101	1	2 months	326,066	30
HH102	1	2 months	413,142	30
HH103	1	2 months	167,183	29
HH104	1	2 months	484,931	32
HH105	1	2 months	225,820	30
HH106	1	2 months	263,083	33
HH107	2	1 month	295,165	27
HH108	1	2 months	362,164	31
HH109	1	2 months	569,331	32
HH110	1	1 month	138,331	25
HH111	1	2 months	356,496	33
HH112	1	3.5 months	669,330	30
HH113	1	17 months	3,250,290	32
HH114	1	1 month	194,345	28
HH115	1	10.5 months	2,169,339	64
HH116	1	2 months	507,995	32
HH117	1	11.5 months	1,118,020	37
HH118	1	1 month	254,112	30
HH119	1	1 month	140,711	28

Table 7.7: Horizon House datasets for sampling features.

Dataset	Residents	Timespan	Events	Labeled Activities
HH120	1	2 months	300,037	32
HH121	2	0.5 months	170,930	75
HH122	1	1 month	202,112	32
HH123	1	1 month	154,068	30
HH124	1	2 months	76,024	21
HH125	1	2 months	216,255	32

(almost 20 minutes) when only discrete sensors are included. By including events from the sampling-type sensors, the error decreases to 749 seconds (about 12.5 minutes). This indicates that the inclusion of other sensor types causes the error to be reduced by about half, at least in terms of absolute error. The reduction in error is not as significant when examining the RMSE results, where the error drops by only about 32%. Recall the RMSE emphasizes larger errors more than the MAE. It is likely that the additional sensors mostly help to reduce the smaller prediction errors, but do not have as much effect on larger errors. Overall, however, the addition of more sensor types helps to improve the overall performance of the IP predictions in a noticeable way.

Figure 7.5 shows plots of the average MAE both by dataset and by activity. In 19 of the 25 datasets the inclusion of more sensors leads to a noticeable reduction in error. This is especially true for the HH105 dataset, where the MAE with only discrete sensors is over two

hours, but drops to about 12 minutes with all sensors included. On the other hand, five of the datasets had higher MAE when the extra sensors were included. Often this increase in error is small (e.g, HH21), indicating that the increase may be due mostly to different dynamics in the extra events for which IP is forming predictions. For HH106, however, the increase in error is much greater (about 650 seconds). It is possible that the placement of the sampling sensors and the inhabitant's habits in some datasets may be such that including them actually makes hinders the ability of the model tree to predict accurately. This could also be due to the additional events being added having more difficult activity times to predict, such as those that are not experienced in the training windows. These differences highlight the need to consider different parameter approaches for each activity prediction situation in order to best match the environment. It should be noted that the HH125 dataset contains no sampling sensors, so there is no change in performance between the two cases for that dataset.

Inclusion of additional sensors also has a varying effect on performance for different activities. The most improvement is seen in activities such as Cook Dinner, Dress, and Phone. It is possible that the additional sensors provide improved coverage of areas and situations where these activities occur. For example, since temperature sensors report values more frequently when the temperature changes more rapidly, it is possible that IP observes that many temperature events caused by stove use during cooking can be used to improve performance. Similar situations may occur with lighting sensors. It should also be noted that the MAE increases noticeably for the dish washing activities, which may be due to the additional sensors making it more difficult to predict this activity. Overall, however, the sampling sensors lead to improvements for most activities.

Figure 7.6 shows the RMSE results. As with the MAE results, the addition of more

sensors improves performance in many cases, although the results are more mixed. The HH105 dataset still has much improvement with the additional sensors, while the HH106 and HH123 datasets show notable decrease in performance. Comparing activities, the improvement for all of the cooking activities is noticeable. The performance decrease for the dish-washing activities is more noticeable as well. The dish-washing activities with all sensors may have a few larger errors that cause the RMSE to rise more markedly than the MAE. It seems as if many of the activities which have higher RMSE when more sensors are added would benefit from efforts to reduce the larger errors to lower the overall error rate.

The p-value from a paired t-test between the MAE results for the different activities is 0.0009, which is significant. Thus, the inclusion of additional sensor types is useful for better-informing the predictor in most cases. However, this is not universally true, as the performance can decrease if the additional sensors are not able to provide useful new contextual information.

There is also an additional aspect of the sensor types used that we find notable. Recall that many of the battery-operated wireless sensors report their battery voltages and capacity percentages at regular intervals. Normally, these events only come at certain intervals, without any direct link to inhabitant activities. This may lead to the conclusion that the battery-related events would not be helpful for activity prediction. However, we briefly tested what effect is had if these events are removed, and found that the performance actually decreased. Although the decrease was not particularly large, it is an interesting anecdotal observation that IP is able to use the seemingly unhelpful battery information to improve performance.

Sampling Feature Inclusion

We now consider the effects of including sampling features in addition to the discrete features previously used. Since we found that performance generally increases using additional

sensors, we are now interested to see how much more improvement the sampling features themselves can contribute. This allows us to understand if the state-based sampling features are useful for activity prediction. For this comparison, we use IP with the same Horizon House datasets with all sensor types included. (This is the same collection of data used for the "all sensors" comparisons in the previous section.) We again use a window size of 2,000 events and test on the next event. w_{skip} is again 1,000 events. This time, we use only discrete features for one set of results, but utilize the sampling features previously described for the other. The sampling interval is set at one second, while five seconds of sample lag are used. Larger sampling intervals and lags can be used, but this can lead to increased computation time and complexity for this many datasets.

Table 7.8 shows the overall results of these tests. The average MAE is improved by about 14 seconds with the inclusion of the sampling features. While this improvement is not much, the average RMSE improves by about 722 seconds (about 12 minutes) to 5,296 seconds. The previous section indicated that adding more sensor types mostly reduces smaller errors (reflected by the greater MAE reduction). Here, however, the RMSE reduction is greater, indicating that adding the sampling features most strongly reduces larger errors emphasized by the RMSE. While only 11 of the datasets have improved MAE results with sampling features, 16 show improvement when RMSE is considered. Thus, while adding sampling features does reduce the overall error in both an MAE and RMSE sense, the effect is not consistent and is mostly related to reducing large-magnitude errors.

Figure 7.7 shows plots of the MAE results. Although many of the datasets have higher error with sampling features added, the difference is usually small (usually less than a minute). Thus, there is not much reduction in performance that occurs when the sampling features are

Table 7.8: Overall results comparing discrete vs. all features with IP. These values represent macro-averages over the results for all activities in all datasets.

Test	Discrete Features	Discrete + Sampling Features
Average MAE	749.2	**735.1**
Average RMSE	6017.8	**5295.5**
Number of Datasets Where Test Has Lowest Average MAE	**14**	11
Number of Datasets Where Test Has Lowest Average RMSE	9	**16**

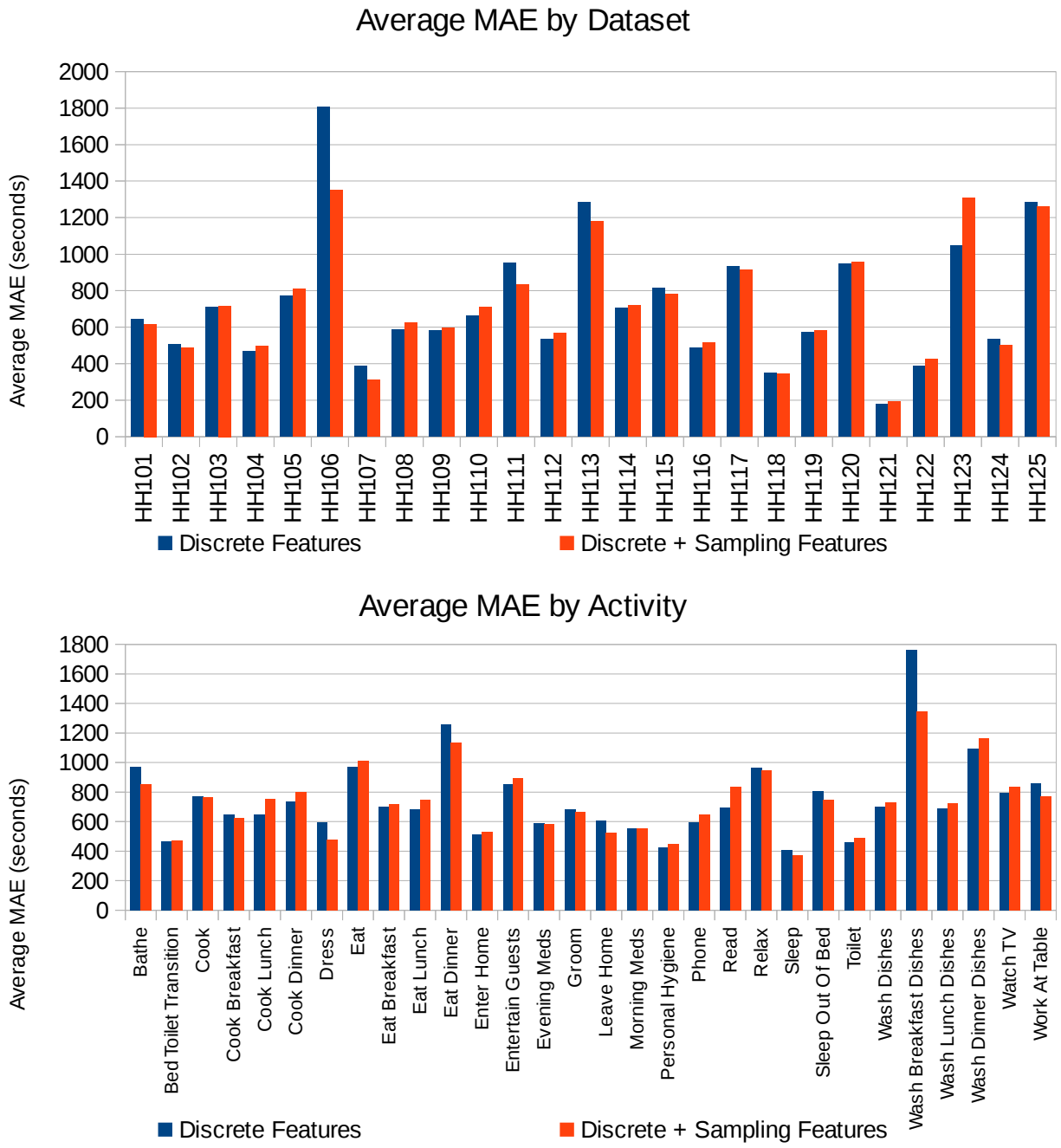


Figure 7.7: Features comparison MAE results. Top plot shows the average MAE for each dataset, while the bottom plot shows the average MAE for each activity (across all datasets). All tests are with all sensor types and 2,000-event training windows. Results are not statistically significant ($p < 0.05$).

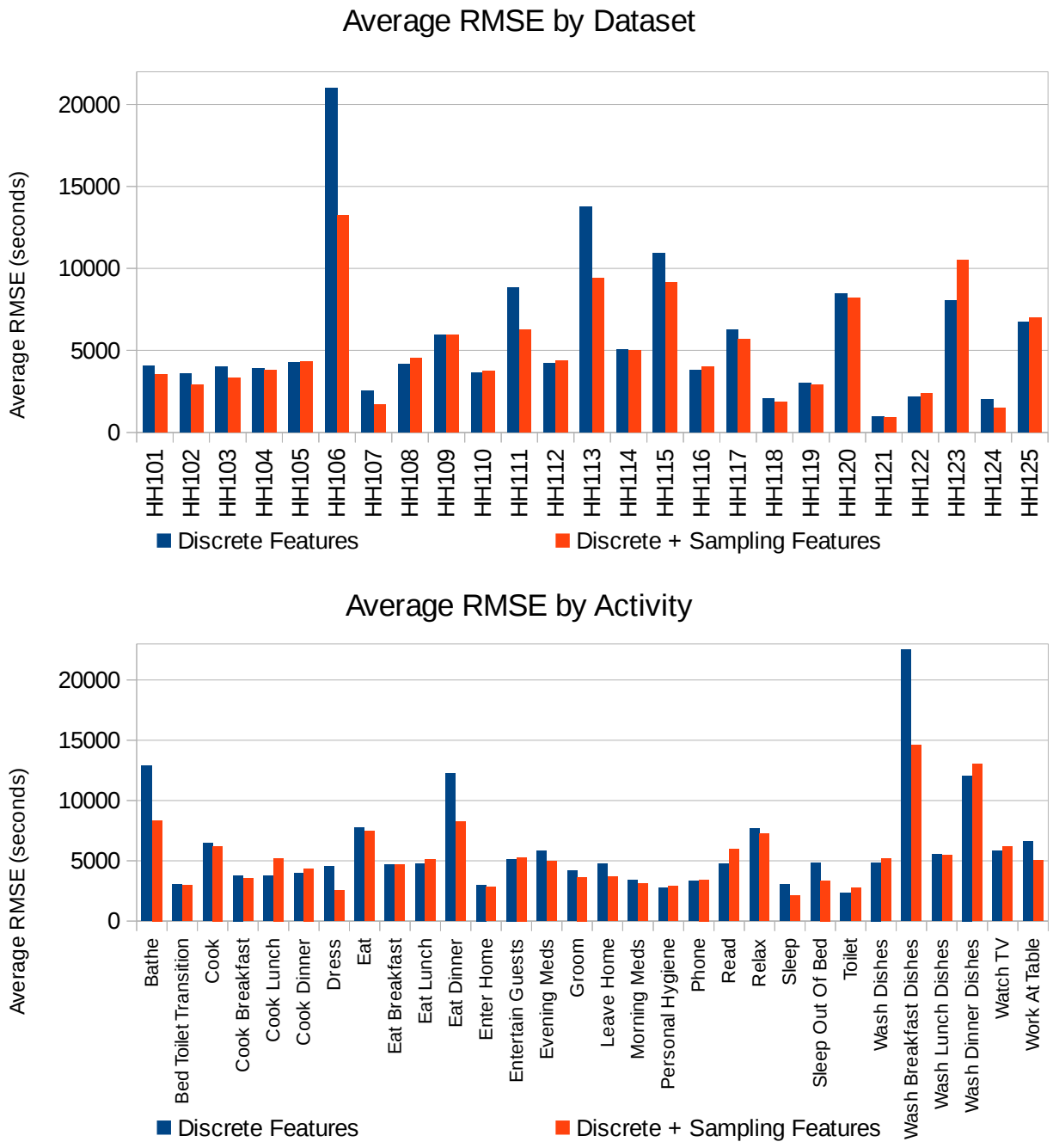


Figure 7.8: Features comparison RMSE results. Top plot shows the average RMSE for each dataset, while the bottom plot shows the average RMSE for each activity (across all datasets). All tests are with all sensor types and 2,000-event training windows. Results are not statistically significant ($p < 0.05$).

added. Only HH123 shows a relatively large error increase of about 260 seconds. When the error is reduced, however, the reduction can be notable. With the HH106 dataset, for example, adding sampling features reduces the error by almost 8 minutes. Note that the HH106 dataset showed an increase in error when adding sampling-type sensors, so this reduction in error with the sampling features is of interest (though not as great as the previous error increase).

The activity errors show a similar trend. In most cases, the error difference between using only discrete sensors and using all sensors is relatively small. The most significant difference is with the Wash Breakfast Dishes activity, which decreases by about 7 minutes. This may indicate that the inclusion of the value information gained from the sampling features is useful in predicting the dish washing activity in the morning. Perhaps this is due to increased context related to the use of light and changes in temperature within the home as the inhabitant performs their morning routine.

Figure 7.8 shows RMSE plots for these tests. Here, the decrease in error with the addition of sampling features is more marked. As with the MAE, the HH106 dataset gains noticeably improved performance, reducing the RMSE by over two hours. The HH111, HH113, and HH115 datasets also show a strong improvement in performance. For the other datasets the improvement is less, but the only dataset showing a strong increase in error is HH123. This dataset has an RMSE increase of about 42 minutes.

The activity RMSE results show a similar pattern. The majority of the activities have improved RMSE performance with sampling features, especially Bathe, Eat Dinner, and Wash Breakfast Dishes. These activities may be helped by the addition of knowledge about the sensor states from the sampling features. For other activities, such as Bed-Toilet Transition or Enter Home, the improvement is less, indicating that sensor values are not as useful for predicting

these activities. It is interesting to note that the Sleep activity has particularly low error, which may be the result of relating light sensor values throughout the day with upcoming Sleep occurrences.

The p-value from a paired t-test for the MAE activity values is 0.51, which is not significant. Comparing the RMSE results, the p-value is 0.052, which is not particularly significant, though more so than for MAE. The low significance is likely due to the small differences between the sets of results compared to the relatively large error differences between activities. The effect gained from adding more sensor types is more significant than adding the sampling features, although the sampling features may still be helpful.

Again, the improvements in the RMSE results indicate that much of the error reduction from adding more features is found in reducing larger errors. This may be a significant improvement for applications where very large errors can be particularly costly, such as cases where action may be taken to prepare for an activity that is predicted with large error. Even in cases where the MAE or RMSE error is greater, the difference is usually small. Thus, it appears that using sampling features can increase the ability of the regression learner to predict activities with fewer large errors.

Correlation with Sensor Data Statistics

Next, we probe deeper to determine what the relationship is, if any, between the characteristics of activities and the ability of IP to predict them. We are interested in knowing if we can expect the performance of IP to vary dramatically based on the type and composition of the data used for predictions. This is useful in understanding whether our regression learner prediction techniques are susceptible to major changes in performance as dataset characteristics change. Ideally, the performance would not change much across different datasets. To this end,

we correlate the MAE results from the previous section when using all sensors and all features with various statistical measures of the Horizon House data. In particular, we compare against the following parameters:

- **Number of activity occurrences per day** The average number of occurrences of a particular activity per day throughout the dataset.
- **Mean start / end times of activities** The average start or end time of all occurrences of an activity throughout the dataset. These averages are calculated using a circular average in order to account for the cyclical nature of daily times. For example, start times of 23:00 and 01:00 would average to 00:00 with a circular average.
- **Variance of the start / end times of activities** The variance in the start / end times of occurrences of an activity. These variances are based on a normalized circular scale similar to the circular mean.
- **Mean number of total events per day** This is an average of the number of events per day, across all activities. It is computed separately for each dataset.

In the case of the activity-related statistics, we linearly correlate each statistic with the MAE results across all activities in each dataset. We then average the correlation for each dataset. For the last statistic, we find an overall correlation between the mean number of events and the overall average MAE across all datasets.

These results are shown in Table 7.9. The number of occurrences per day has a moderately negative correlation with the error. This indicates that activities that have more occurrences per day will have slightly lower error and thus increased performance. This seems sensible as more activity occurrences leads to both more interesting instances to train from as well as

reduced time between occurrences that must be predicted. MAE is slightly positively correlated with the start and end times of activities, indicating that activities earlier in the day are slightly easier to predict. This may be due to more strongly fixed morning routines of the inhabitants compared to the evenings, which may vary with different activities throughout the week.

Table 7.9: Correlation of MAE results with dataset statistics. Results for the first five statistics are averages across datasets of the linear correlation of activities in each dataset. The last statistic is a linear correlation of all datasets.

Statistic	Correlation
Occurrences/Day	-0.213
Mean Start Time	0.145
Mean End Time	0.150
Start Time Variance	-0.131
End Time Variance	-0.126
Dataset Events/Day	-0.547

Interestingly, the MAE is negatively correlated with the variance in activity start and end times. In other words, the performance improves somewhat when the variability in the activity occurrence times increases. However, this effect is only slight and not strongly correlated.

Finally, the strongest correlation is a negative correlation between the MAE and the number of events per day in each dataset. When there are more events per day the error decreases and performance improves. Similar to the correlation with activity occurrences, this also seems sensible given that there are more training instances to learn from for each day's

activity patterns, leading to improved prediction models.

These results indicate that the greatest contributing factor to performance seems to be having a high number of events and occurrences of activities per day. There is less effect from the actual start and end times of activity occurrences. With the exception of the number of events per day, however, none of these correlations are particularly strong. Therefore it is likely that IP will perform well when used with data with varying characteristics and can be applied across multiple applications and scenarios. Even when the activities in those datasets may vary more significantly, the resulting impact on error will likely be small.

7.2.3 Regression Learner Comparison

We are also interested in determining the effectiveness of the model tree regression learner used in IP compared to other approaches for activity prediction. A variety of regression learners could be used in activity prediction and we are interested in seeing if the model tree is a strong model for this purpose. Using the previous Horizon House data, we compare the model tree against two popular learners: a linear regression model and a support vector machine (SVM). The linear regression was used in our experiments with the discrete features in Section 7.2.1. It again provides a baseline comparison using a basic linear model. The SVM provides a more complex regression model that should be able to predict activities using a nonlinear model, similar to our model tree.

We use the feature extraction component of IP to generate the same features for all three regression learners. All features (both discrete and sampling features) were used for these experiments. As before, we trained and tested all three regression learners using a sliding

window validation. The same windows were used for all three models. We then compared the predicted times from each regression learner with the ground-truth labels for each event.

We use the WEKA data mining software [75] to implement the LR and SVM regression learners. LR was used without attribute elimination, so all of the features were available to the model. The SVM was trained using sequential minimal optimization [70] with a linear kernel and attribute normalization.

One of the drawbacks of the SVM model was observed while attempting to run the comparison experiments. We had originally used a window size of 2,000 events, as with previous tests. The model tree and linear regression learners were able to complete the training and testing with this window size in fewer than 12 hours. The SVM model, however, had difficulty with the amount of data. After running for 10 hours it was only able to complete one or two sliding windows out of the hundreds needed. We believe the SVM model had difficulty learning a model due to the high number of data attributes and data instances in each training window. (With all of the sample features included, there are over 2,000 features for each data instance.) This indicates that the SVM learner may not be optimal for the activity prediction problem due to its training time on the large training datasets used. In order to allow the SVM model to finish within reasonable time, we reduced the window size to 500 events, moving the window forward by 500 events each time.

Table 7.10 shows the overall results of these experiments. The LR model has a very high average MAE and RMSE on the order of 10^{11} to 10^{12} seconds. Much of these large error results can be attributed to a few instances where the error is very large, causing the averaged values to be very large. Most of the errors for LR are much smaller. However, these very large errors indicate that the LR model may have trouble dealing with nonlinear aspects of

Table 7.10: Overall results comparing different regression learners. MT = model tree, LR = linear regression, and SVM = support vector machine. These values represent macro-averages over the results for all activities in all datasets.

Regression Model	MT	LR	SVM
Average MAE	**522.0**	$1.84 * 10^{11}$	530.4
Average RMSE	**5810.3**	$5.15 * 10^{12}$	10,062.1
Number of Datasets Where Test Has Lowest Average MAE	8	0	**17**
Number of Datasets Where Test Has Lowest Average RMSE	**13**	0	12

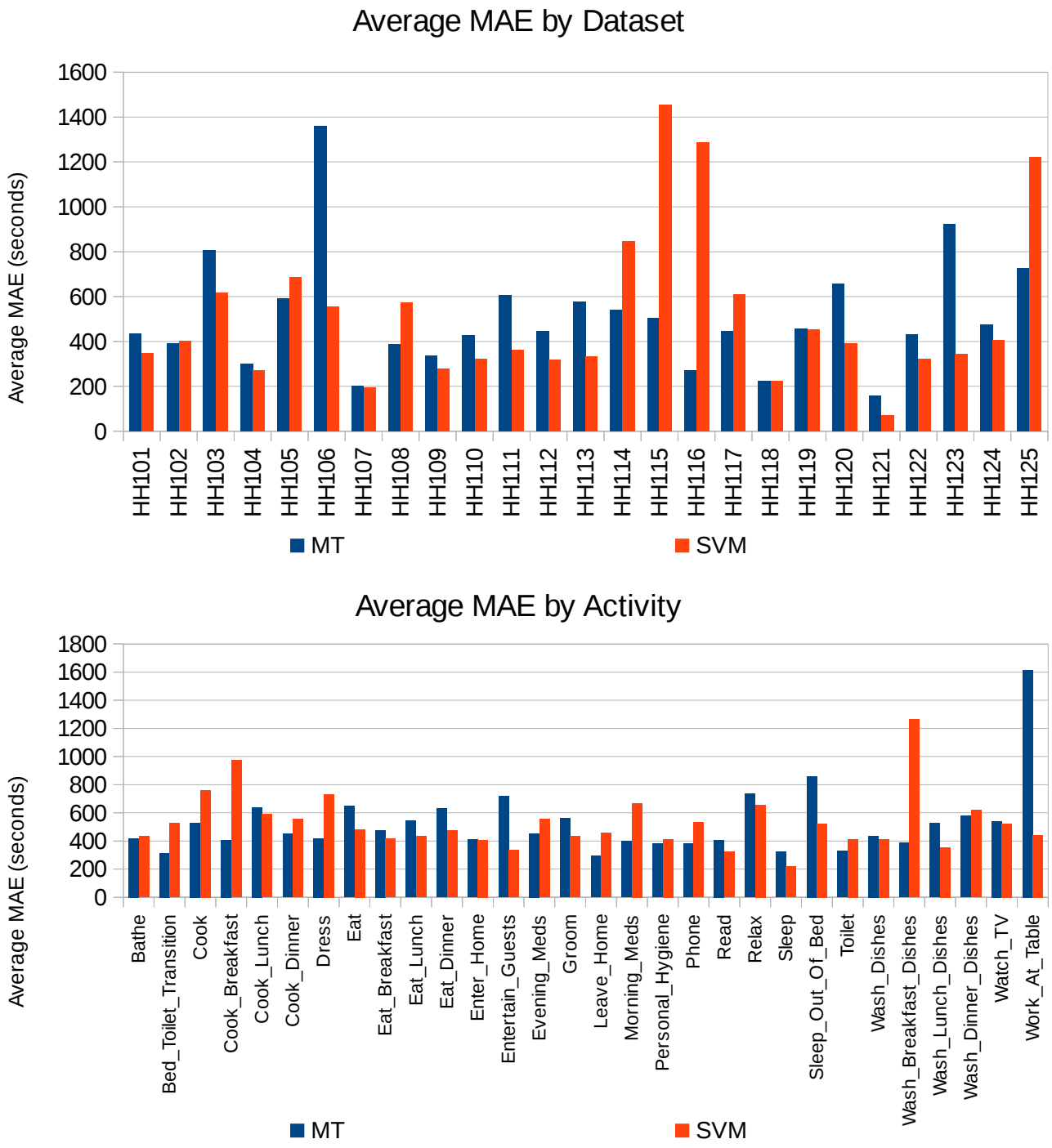


Figure 7.9: Regression learner comparison MAE results. MT = model tree and SVM = support vector machine. All tests are with all sensor types and 500-event training windows. Results are not statistically significant ($p < 0.05$).

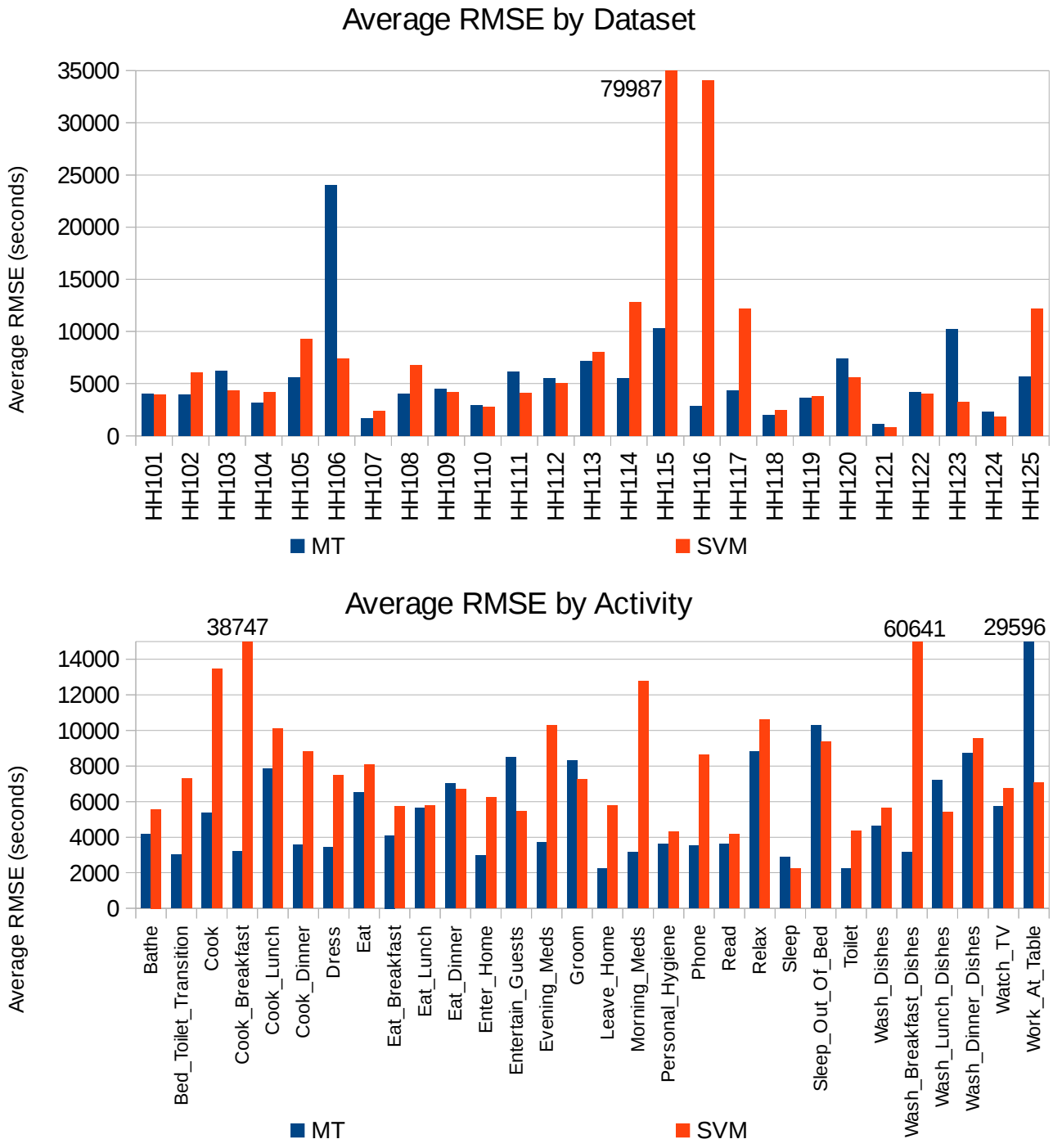


Figure 7.10: Regression learner comparison RMSE results. MT = model tree and SVM = support vector machine. All tests are with all sensor types and 500-event training windows. Results are not statistically significant ($p < 0.05$).

the activities that are important in multiple cases. They also suggest that it may be useful in activity prediction to detect and minimize predictions with large errors. For example, a smoothing operation may be useful to avoid large error spikes.

The LR model was also impacted by the small training window size and correlation between feature values. With only 500 training events in each window, the values in all of the training feature vectors are often similar to each other. This may cause the learned model to over-fit the training examples, leading to reduced performance on test events which have different feature values. Collinearity between features can also decrease the LR model performance. Feature vectors from small training windows may have many collinear features which were not removed in this experiment, leading to larger error. The use of feature reduction techniques and larger training window sizes may improve LR performance, though the inability of LR to form nonlinear models would still limit performance.

The SVM and model tree learners have considerably better performance. The MAE for the model tree is 522 seconds, while it is about 530 seconds for the SVM. This is about a nine-minute average error. While the model tree has slightly lower overall MAE, the SVM had lower MAE on 17 of the datasets (the model tree was lowest for 8 datasets, while the LR was not lowest for any datasets). However, we note that the RMSE for the model tree is 5810 seconds (about 1.5 hours), while it is 10,062 seconds (2.75 hours) for the SVM. The model tree has only about half the RMSE as the SVM. Since MAE emphasizes all errors equally but RMSE weights larger errors more heavily, this indicates that the SVM model produces more notably large errors while performing approximately the same as the model tree in general.

Figure 7.9 shows the MAE results. In many cases, the error for the model tree and the SVM is nearly the same. Noticeable exceptions include HH115, HH116, and HH125, where the

SVM performs about twice as poorly or worse compared to the model tree. The model tree has much greater error for HH106 and HH123. Recall that the model tree has had relatively poor performance for HH106 in many of the previous experiments as well, indicating that the tree structure may have difficulty modeling the particular habits of that testbed's inhabitant. For many of the datasets the performance with the SVM is better. However, in the cases where the model tree performs better, it often does so by a strong margin.

From the activity plot, we see that the two models perform best on an approximately equal number of activities. The SVM appears to have trouble with some morning activities, such as Cook Breakfast, Morning Meds, and Wash Breakfast Dishes. The model tree has improved performance for these activities. It is possible that the model tree is able to model variability in morning routines that the SVM has difficulty with. The model tree performs poorly relative to the SVM for activities such as Entertain Guests, Sleep Out of Bed, and Work At Table. These are activities that may occur at different times, locations, and contexts during the day. (For example, the inhabitant may entertain guests at much different times on different days.) The model tree might have difficulty grouping similar training data for these activities and thus has reduced performance, while the SVM is able to distinguish these contexts more accurately. However, the two models are roughly equal in performance across most other activities.

Figure 7.10 shows the RMSE results for these experiments. For many of the datasets, the model tree and SVM have similar RMSE values. As with other metrics, the model tree has much higher RMSE for the HH106 dataset than the SVM. However, the SVM has very poor RMSE performance for the HH115 and HH116 datasets, with average RMSE values of about 80,000 and 34,000 seconds, respectively. This is an error of almost a day in some cases. Comparatively, the RMSE for the model tree is only around 10,000 seconds for HH115 and less

than 3,000 seconds for HH116. It appears that the SVM has a significant amount of large-error predictions for these datasets which cause its performance to be much lower. Thus, when large-magnitude errors should be avoided, the model tree appears to be a more effective regression learner.

Similar results can be seen in the activity RMSE data. The SVM again has difficulty with morning activities, where it obtains large RMSE errors. While the model tree again has higher RMSE for many of the activities where it has high MAE, the only very large error is for the Work At Table activity. Thus, it appears that many of the errors that cause higher MAE for these activities are smaller errors not weighted as heavily in the RMSE measure. It should also be noted that both models have relatively low RMSE for predicting the Sleep (in bed) activity, indicating that this is easy to predict using the nonlinear models.

While the SVM has improved MAE performance on some datasets, the model tree overall performs slightly better. It also shows a much greater improvement in RMSE measures compared to the SVM. Thus, the model tree seems able to perform prediction without as many large erroneous predictions. This may be useful in many cases, where we wish to avoid mistakenly acting on predictions when they can be very far from the actual activity times. While the LR model is easy to train, it is unable to handle certain aspects of the activity patterns, resulting in large error. Thus a nonlinear model is needed to predict activities more accurately.

The SVM model can obtain relatively strong performance over the model tree in some cases. However, its inability to handle larger and more complex data limits its application to activity prediction. Anecdotal observations suggest that the SVM is also slower to train than the model tree, suggesting the latter is a useful model for its performance and ease of training. It is also possible that the small training window size leads to a poor model tree structure, as

there may not be enough data instances to properly generate a full model tree and its nodes for more accurate prediction. However, we will often want to use much more training data than just 500 instances regardless of the model used in order to achieve better performance across the variety of daily activity patterns for each inhabitant. The model tree provides a strong prediction component for IP with lower complexity compared to SVM but better performance, especially compared to the linear model. This indicates that the tree structure is effective at modeling activity patterns for prediction.

7.3 RAP Evaluation

7.3.1 Setup

We now evaluate the performance of the RAP method compared to that of IP and other baselines. We wish to know whether the inclusion of joint activity information allows us to improve predictive performance. In this way we can analyze whether relationships between activities can be effectively used to perform predictions in support of our hypothesis. We also wish to evaluate the performance improvement that may be achieved through the use of imitation learning techniques.

For these tests we use the Horizon House testbeds as we did for the sampling feature tests for IP. However, we only utilize 24 for the 25 datasets (HH124 is excluded due to the lack of data from that testbed). Using the initial human-annotated data described in Section 7.2.2 we train AR to learn a generalized model from the data for all 24 datasets. AR achieves 96% classification accuracy on these datasets under a 10-fold cross validation of the labeled data. The AR model is then used to label additional data from these testbeds which are then used

for these tests. This is done to avoid the irregularity and inter-annotator reliability problems in human-annotated data. The statistics for these AR-labeled data are shown in Table 7.12. We have condensed many of the 30 core activities used in previous experiments into 11 main activities (e.g, all of the variations of eating are now in the Eat activity). Table 7.11 shows the distribution of the activities (across all 24 datasets).

We again use the sliding window validation approach for these tests. While the training window size is 2,000 events, we increase the test window size to 5,000 events. That is, we now test on 5,000 events after each training window. This is done in order to mitigate the effects of the close proximity of each training and testing window. By adding more test events that move further from the training window we can better evaluate how well RAP will do with new data different than the initial training set. This also allows us to see how performance changes as the data changes. As before, w_{skip} is 1,000 events.

Algorithms

For this evaluation we compare the following four algorithms:

- **Independent Predictor** The IP provides an informed baseline for comparison.
- **Recurrent Activity Predictor** The RAP includes activity relationship features which we hope will improve predictive performance.
- **Gaussian** As an uninformed baseline, we utilize a Gaussian predictor. This predictor does not rely on a complex underlying model for generating prediction times. Rather, it models the activity occurrence times for the activity of interest from the training window as a Gaussian distribution. The Gaussian predictor then samples from this distribution in order to generate a prediction.

Table 7.11: Activity class distributions in datasets used for testing recurrent predictors. The number of events is a total from across all datasets.

Activity	Sensor Events
Bathe	208,119
Bed-Toilet Transition	174,047
Cook	2,614,836
Eat	585,377
Enter Home	174,486
Leave Home	311,164
Personal Hygiene	1,916,646
Relax	2,031,609
Sleep	732,785
Wash Dishes	1,139,057
Work	2,028,419

Table 7.12: Statistics for AR-labeled datasets used for testing recurrent predictors.

Dataset	Residents	Time Span	Sensors	Sensor Events
HH101	1	2 months	36	219,784
HH102	1	2 months	54	280,318
HH103	1	2 months	26	112,169
HH104	1	2 months	66	344,160
HH105	1	2 months	60	146,395
HH106	1	2 months	60	201,735
HH107	2	1 month	54	199,383
HH108	1	2 months	54	284,677
HH109	1	2 months	44	399,135
HH110	1	1 month	38	98,358
HH111	1	2 months	54	219,477
HH112	1	4 months	40	468,477
HH113	1	12 months	58	1,643,113
HH114	1	1 month	32	133,874
HH115	1	10 months	40	1,591,442
HH116	1	2 months	38	386,887
HH117	1	12 months	32	767,050
HH118	1	1 month	46	178,493
HH119	1	1 month	36	92,000

Table 7.12: Statistics for AR-labeled datasets used for testing recurrent predictors.

Dataset	Residents	Time Span	Sensors	Sensor Events
HH120	1	2 months	40	217,829
HH121	2	10 months	62	3,361,406
HH122	1	2 months	56	247,434
HH123	1	1 month	32	106,836
HH125	1	2 months	34	216,245

- **Oracle** The Oracle predictor provides a best-performance limit to the performance of RAP. It is based on the idea that the DAgger algorithm [60] could be used to improve the predictive ability of the RAP model. The Oracle predictor employs the same features as RAP, except that the lag features Φ_{joint} are based on the true activity times, rather than predictions from the model. This mimics what could be obtained through optimal imitation learning of the lag values, as could be provided by the iterative learning of DAgger. Oracle provides an upper limit for the performance limits that should be achievable through RAP.

7.3.2 Results

Table 7.13 shows the overall results for each algorithm. The RAP method shows a very noticeable improvement over IP in RMSE, reducing the error by almost eight-fold from 175,502

seconds to 22,337 seconds. RAP's improvement over IP is less dramatic for the MAE results, but is still significant (reduced by half). The RMSE is especially affected by a few large IP errors that are greater than one day, causing it to rise dramatically compared to MAE. This emphasizes how MAE is a better measure for determining overall error performance.

Table 7.13: Overall results comparing recurrent and other predictors. A one-way ANOVA of the results indicates that the results are significant ($p < 0.05$).

Method	MAE	RMSE
Gaussian	12,134.30	23,142.38
IP	19,050.85	173,501.54
RAP	8,433.38	22,337.32
Oracle	2,686.47	12,758.97

Figure 7.11 shows the MAE results for each activity. RAP performs better than IP for all activities. Even the Gaussian learner performs better than IP. This demonstrates some of the variability exhibited by IP, which can be subject to large prediction swings. Unlike RAP, IP has no lag values to help condition new predictions, so it can generate very erroneous predictions. We believe that the Gaussian predictor outperforms the IP predictor because it generates predictions close to the mean value of the training labels. Since the test events are still relatively close to the training windows the activity times do not change greatly, so the random Gaussian predictions are still closer than the widely-varying IP predictions.

RAP performs noticeably better for activities such as Cook, Eat, and Wash Dishes. This

may be because these activities are often closely related to each other and performed in order. RAP is able to use the context of the related activities from its lag features to improve prediction. RAP also performs well against IP for the Personal Hygiene, Relax, and Sleep activities. These activities can have variability in their start and end times, and (in the case of the first two) may occur in different places throughout the home. The sensor-based features alone do not provide enough context for these activities, but RAP is able to discover the relationships between these activities and others to more accurately predict them. RAP, IP, and the Gaussian predictor all have poor performance for the Bathe and Bed-Toilet Transition activities. This may be due to variability in these activities and their relative sparseness in the dataset, making them harder to model and predict. It may also be noted that RAP does have some trouble predicting the Enter Home activity, likely because there are not a lot of events preceding the inhabitant's return home. Overall, RAP outperforms both the IP and Gaussian predictors with significant improvement.

Figure 7.12 shows the ETF values at varying thresholds. The IP model has only about 5% of its errors below one second. RAP increases this to 18% of errors less than one second, a notable improvement. About 55% of RAP errors are less than 15 minutes, compared to only 40% of errors with the independent predictor. Both methods have about 99% of errors smaller than 24 hours. These results indicate that RAP can predict more often with smaller error compared to the independent case while still having a majority of its errors less than one hour. This level of performance is useful for many activity prediction problems.

The Gaussian predictor has less than 1% of its errors under 30 seconds. In this regard it is outperformed by both IP and RAP. It is interesting to note that, although the Gaussian model has lower overall error than IP, it has a larger number of errors over an hour. This is

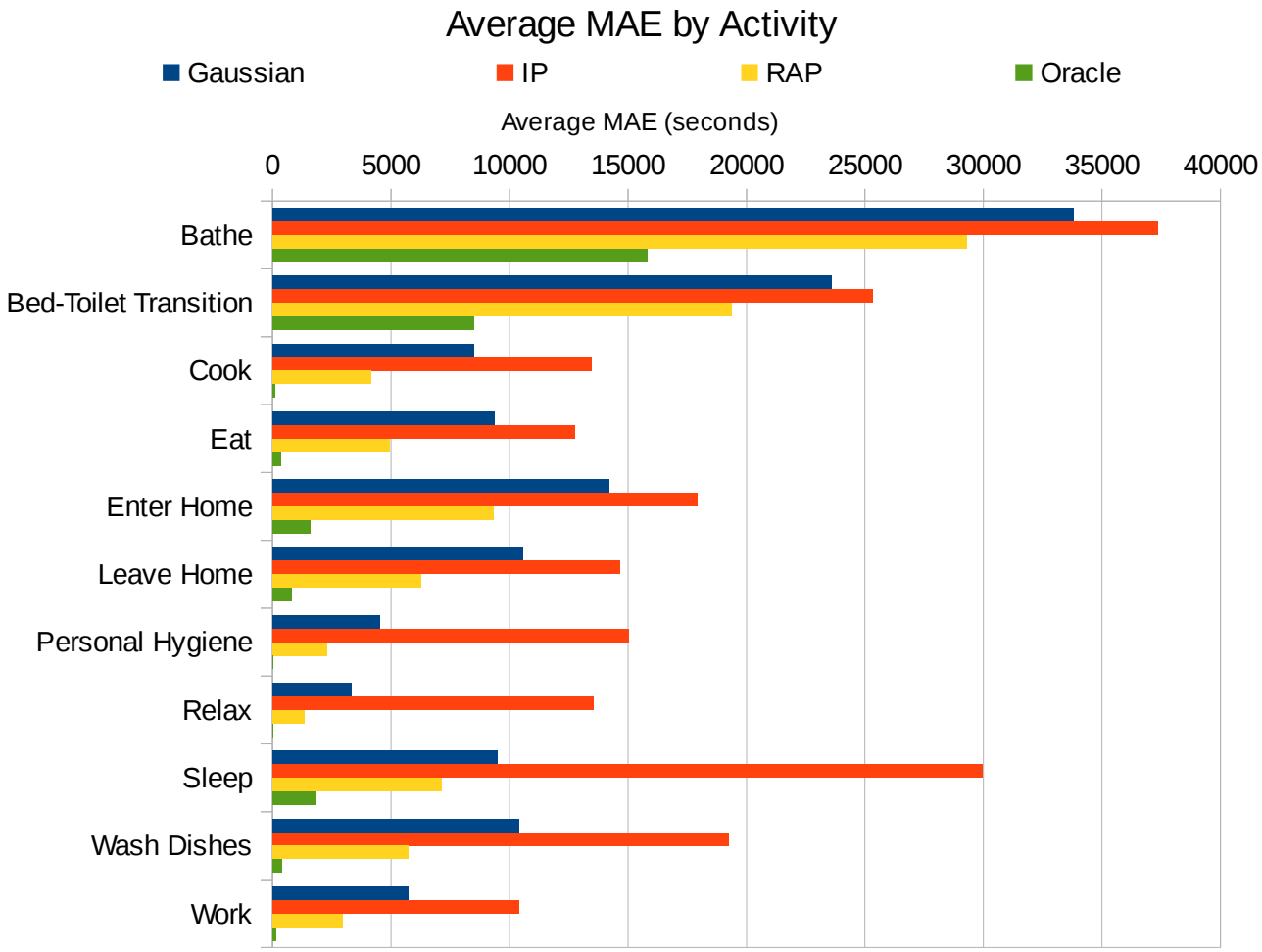


Figure 7.11: Recurrent predictor MAE by activity.

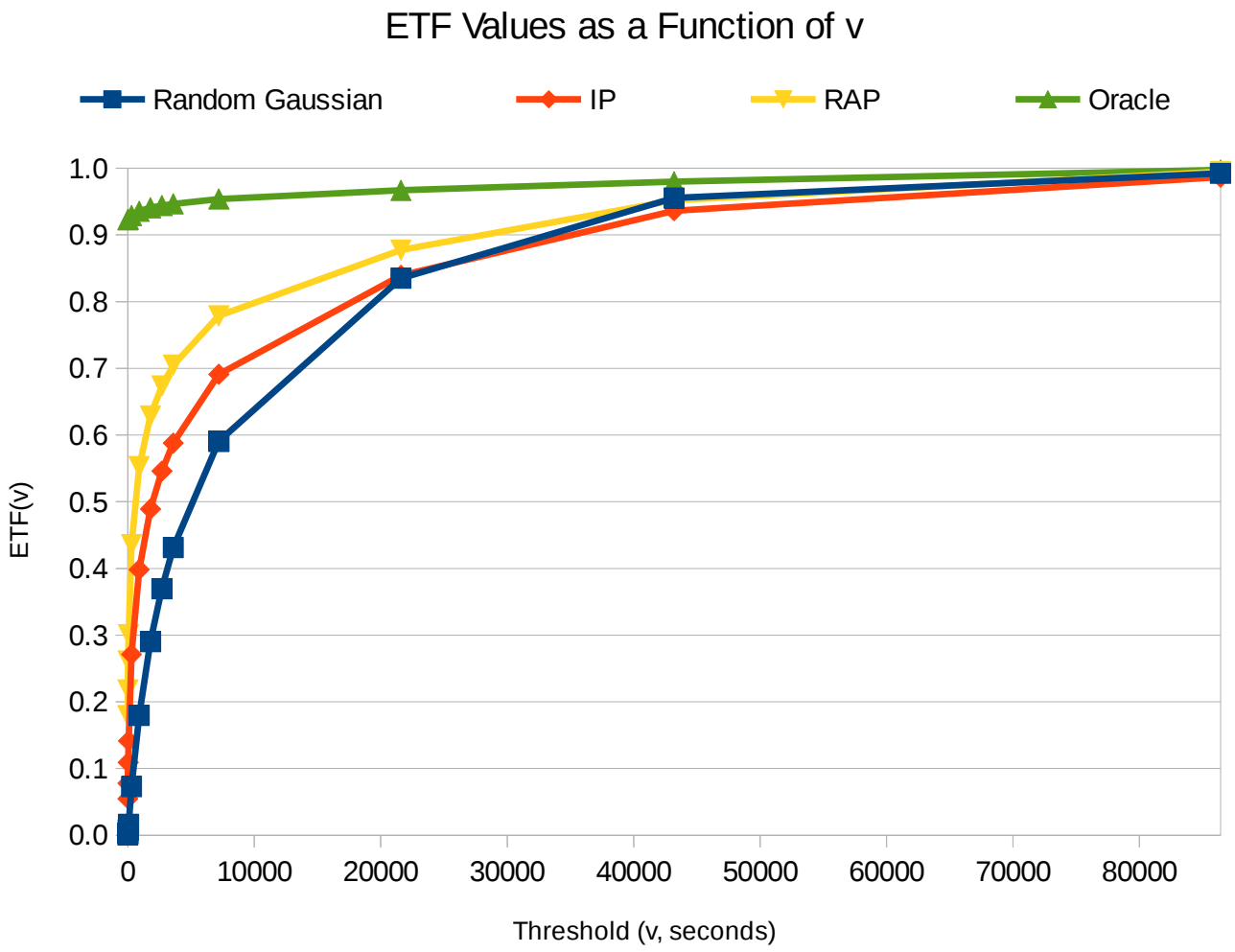


Figure 7.12: Recurrent predictor ETF values. Threshold values range from one second up to one day.

because the Gaussian predictor takes a "middle-of-the-road" approach, neither predicting with very low or very high error. The IP model is more subject to swings in the predictions that cause large errors, but it is still able to outperform the Gaussian model in terms of the error threshold. It is possible the IP predictor could be used with a limitation threshold above which certain predictions can be ignored.

The Oracle predictor achieves about a three-fold reduction in MAE compared to RAP, but only reduces the RMSE by about half (Table 7.13). This indicates that, although the Oracle performs better than RAP, this improvement is mostly in reducing the number of medium-magnitude errors that are not emphasized by the RMSE. This is also indicated in the ETF values in Figure 7.12. Over 90% of the errors for Oracle are less than one second, compared to the 18% for RAP. However, both predictors have around 5% of errors larger than 43,200 seconds (12 hours), with only slightly fewer under Oracle. Thus it appears that the majority of the improvement that is gained from the improved fidelity of lag values is in reducing the magnitude of most errors, increasing the number of near-perfect predictions. However, there are still some large outlier errors that are encountered with the Oracle.

In Figure 7.11 we see that Oracle performs better than the other solutions on all activities. In fact, Oracle has almost no error for activities such as Cook, Personal Hygiene, and Relax. This indicates that RAP would be able to achieve lower error rates, especially on these activities, by learning to recover from errors with DAgger.

We note that the ETF curves in Figure 7.12 resemble a standard ROC curve used in classification problems. In this case, the discrimination threshold is the time threshold on prediction error, v. With common ROC curves, the area under the ROC curve (AUC) can be used as a numeric measure of performance. We thus utilize the area under the ETF curve

Table 7.14: Area under the ETF curve (AUETF) values for each predictor. A value of 1.0 is a perfect predictor (no error).

Method	AUETF
Gaussian	0.8619
IP	0.8757
RAP	0.9091
Oracle	0.9972

(AUETF) in an analogous fashion. A perfect predictor would have an AUETF of 1.0. We list the values of AUETF for each predictor in Table 7.14. As with the ETF plots themselves, RAP outperforms IP, while both RAP and IP outperform the Gaussian baseline. The AUETF value for Oracle is particularly high, indicating that RAP can become a very strong predictor with the help of a method such as DAgger.

We are also interested in how the performance of each predictor changes as we move away from the training data. Figure 7.13 shows the average MAE based on how far the test event horizon was from the training window. For all predictors except the Gaussian baseline the errors are very low just after the training window (about 8 minutes for IP and 30 seconds for RAP and Oracle). The MAE slowly increases as the horizon moves away from the training window since the events gradually have different properties than those in the training window. However, IP has a very sharp initial increase in error. It also exhibits a much more variable curve, with some jumps of over 10,000 seconds or more. The curves for RAP and Oracle are

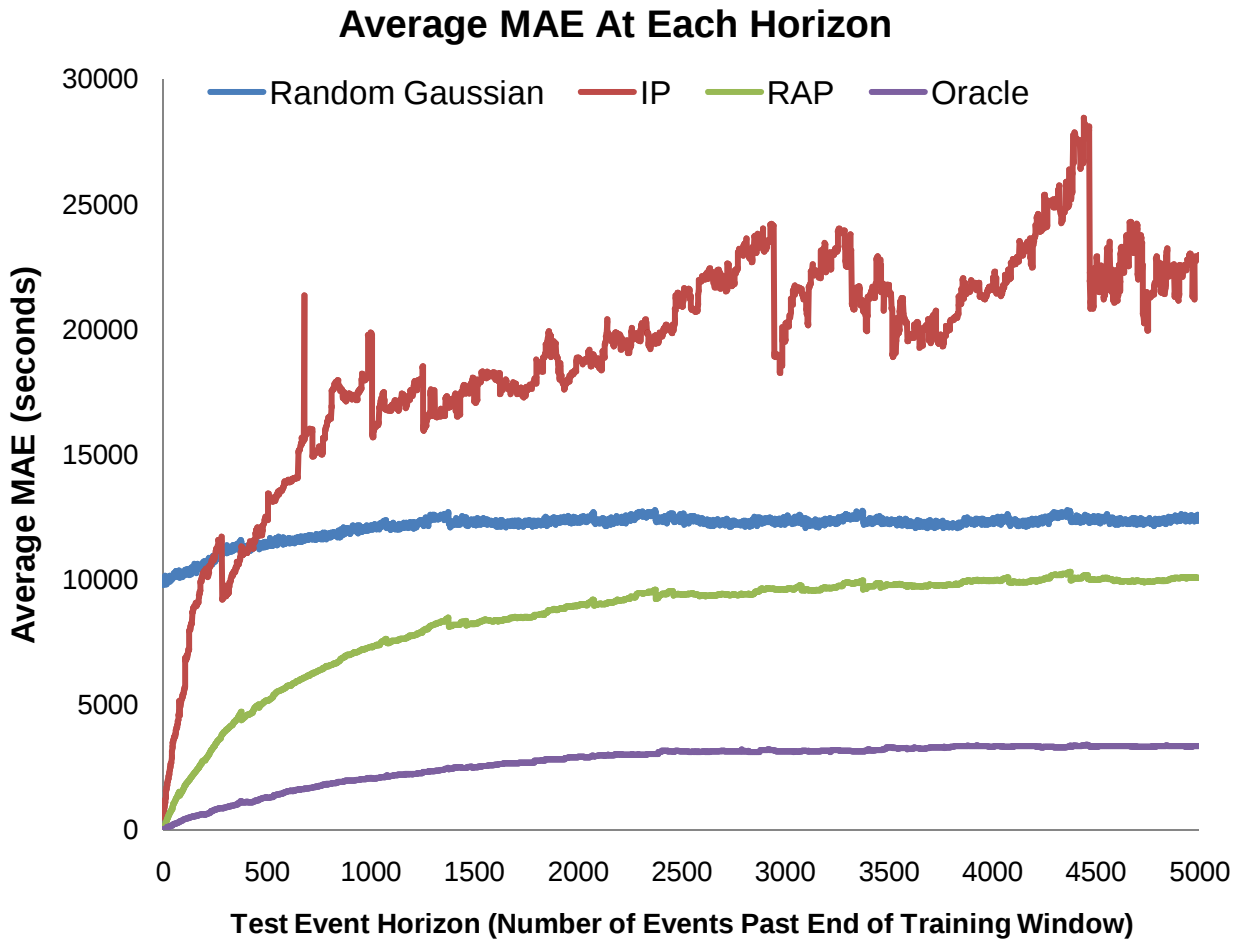


Figure 7.13: Recurrent predictor MAE by horizon. The test horizon indicates how far (in number of events) the test event is from the end of the training window. MAE values are averaged over all activities and datasets at each test horizon.

much smoother and do not increase as sharply. This is indicative of the smoothing properties of RAP, which can use previous predictions to reduce prediction variability and avoid particularly large jumps in error.

For all four predictors the error tends to increase as the horizon moves away from the training window. While the variability of the IP curve makes it difficult to determine the overall trend, it appears that it settles at an error of around 21,000 seconds (5.8 hours) after about 4,500 events. RAP and Oracle, on the other hand, achieve much lower steady-state errors sooner than IP. RAP settles at around 2.5 hours after about 2,000 events, while Oracle has a lower error at about 1 hour after 2,000 events. These errors may be reasonable for some applications, but we suspect they can be lowered further with additional training data in each window. 2,000 events is relatively small compared to the size of the datasets. While this window size was chosen to provide a sufficient number of test windows for our validation, it seems likely that using larger training sets can improve predictive performance by allowing more of the residents' activities to be observed. This is supported by the results for our prompting applications described in Chapter 8, where errors of less than an hour are achieved even at two weeks after training. For those predictors, the training data consisted of more than a month's worth of activities.

This experiment demonstrates the usefulness of activity relationship information for activity prediction. By incorporating this information through the lag values as in RAP, an activity predictor is able to noticeably improve its performance. Thus, activity relationships are an important source of context for prediction. The strong performance of the Oracle predictor suggests that RAP can be improved even further with the use of algorithms such as DAgger which allow it to better correct from errors. RAP is a strong predictor and useful for many situations, though it is open to further improvement with the TPAP method.

7.4 TPAP Evaluation

In this experiment we hope to determine if the TPAP algorithm can be used to address some of the issues found with the RAP predictor. We also evaluate TPAP using three different lag sizes in order to understand how the number of lag predictions affects performance. In this way we can understand if using a combination of learners can improve predictive capability. For these experiments, we used the same Horizon House AR-labeled data used in Section 7.3. WEKA [75] was used for the independent and recurrent model trees in these experiments to allow more rapid testing and changes. However, the M5 model tree regression learner was still used, so the model is equivalent to that used previously.

Instead of using a sliding window validation as previously, we now use a holdout validation method. In this method, a certain percentage of the dataset is "held out" to be used as test data. The model is trained on the remaining data, and then tested on the held out set. The performance of both the independent and recurrent predictors were measured using this method. In both cases, 67% of the data was used for training while 33% was used to test. The test points are chosen at random from the dataset. However, this choosing happens after all of the features are generated for each stage. Since the features are already created for each instance at this point, the temporal ordering of the events is no longer critical as the regression learners treat each instance separately. We also use only 75,000 events from the beginning of each dataset to reduce the run time of the tests with the Weka platform.

We compare the two-pass method using 0, 1, or 12 lags. We also use the performance of the independent predictor as a baseline. The lag-n predictors use both stages of the two-pass method, while the independent results are only from the first-stage independent predictor.

Table 7.15: Overall results for TPAP. The number n for the Lag-n predictors indicates the number of lagged predictions from previous events that are used for that predictor. Each predictor was tested on a 33% holdout from 75,000 instances in each dataset. Best results are in bold.

Method	Average MAE	Average RMSE	Average r
Independent	721.7	2,492.4	0.9815
Lag-0	637.3	7,621.3	0.9840
Lag-1	642.0	7,628.5	0.9841
Lag-12	**567.9**	**1,927.9**	**0.9873**

In the zero-lag case, only the prediction for the same event is added before the recurrent stage. In the lag-1 or lag-12 cases, we additionally add predictions from the 1 previous event or 12 previous events, respectively. So, in the lag-12 case, each feature vector passed to the recurrent stage includes the predictions for that event along with the 12 events preceding it. By comparing varying numbers of lags we can observe how the amount of previous history affects the performance of the algorithm. Discrete features are also used for both the independent and recurrent predictors.

Table 7.15 shows the results of these experiments. We measure the MAE, RMSE, and Pearson's r for each test. In terms of both MAE and the correlation statistic, the two-pass results outperform the independent model. The independent model has MAE of around 722 seconds (12 minutes), which is reduced down to 568 seconds (9.5 minutes) using the lag-12 predictor. While the MAE and correlation results are better for the lag predictors, this is not

necessarily the case with RMSE. While the RMSE is about 2,500 seconds for the independent predictor, it jumps up to around 7,600 seconds for the lag-0 and lag-1 predictors. For the lag-12 predictor the RMSE is reduced to about 1,900 seconds. This indicates that the introduction of the two-pass model can lead to an increase in the size of large errors. Adding more lags helps to reduce this variability by allowing for the predictions to be conditioned on more previous values. The correlation also increases as the number of lags increases.

It is also interesting to note that the MAE and RMSE actually increase from the lag-0 to lag-1 predictors. However, this increase is very slight, indicating that there is not much difference between the two predictors in terms of performance. Therefore it appears that the inclusion of more than one lag is needed before the most benefits can be seen.

Figure 7.14 shows the MAE for each activity. The independent predictor has the highest MAE for almost all of the activities, with the exception of Eat. For this activity, the lag-0 and lag-1 predictors have a very large MAE. For most other activities, the MAE for lag-0, lag-1, and lag-12 are nearly the same. Thus, it appears that, for most activities, adding more lag values does not necessarily help improve performance. However, at least for some activities, the additional lags can improve performance and reduce the number of large errors. The results are found to be not statistically significant through a one-way ANOVA.

This is further suggested by the RMSE results in Figure 7.15. Again, the lag-12 predictor does show improvement over the other lag predictors for Eat, as well as Leave Home. Since the RMSE for the lag-0 and lag-1 predictors is so high for these activities, it is likely that most of the error experienced is concentrated in a few larger errors. It is also notable that the RMSE for the Enter Home activity is much higher with the independent predictor. Recall that this activity is often not preceded by many sensor events. The lag predictors are likely able to use

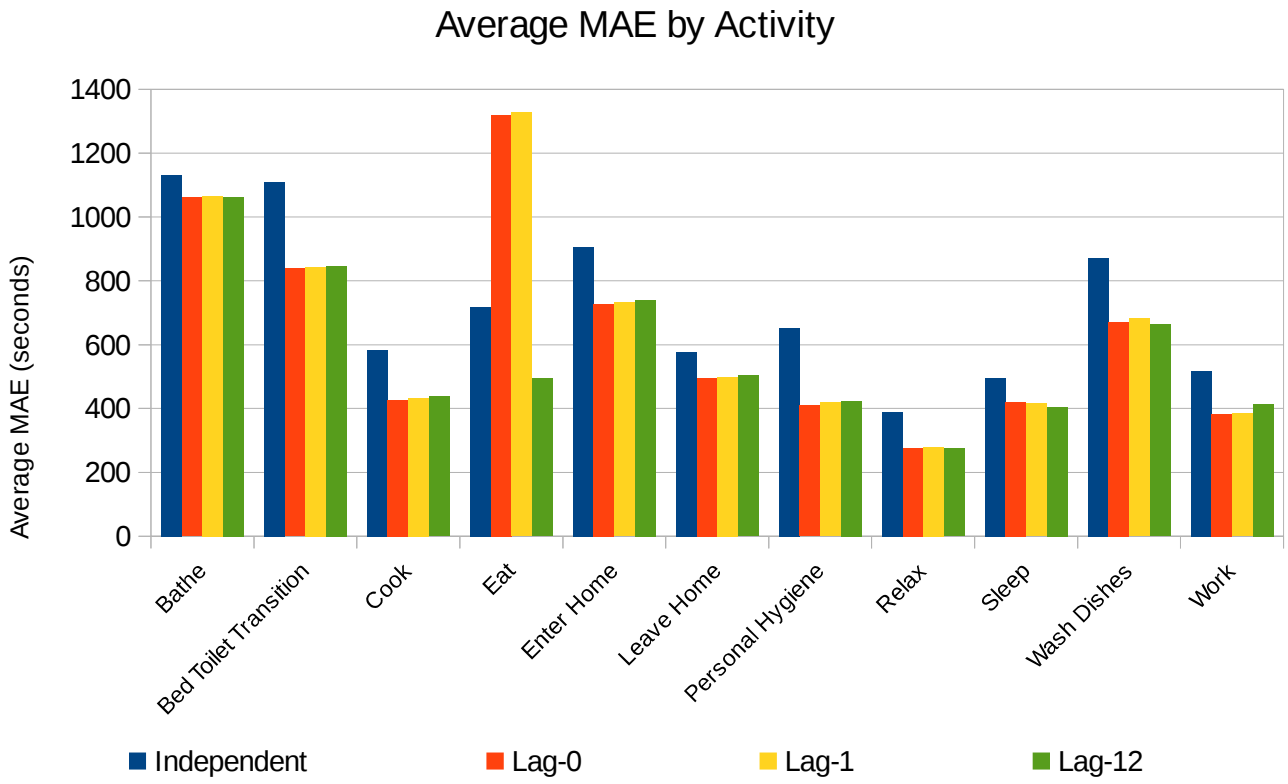


Figure 7.14: Two-pass MAE results by activity.

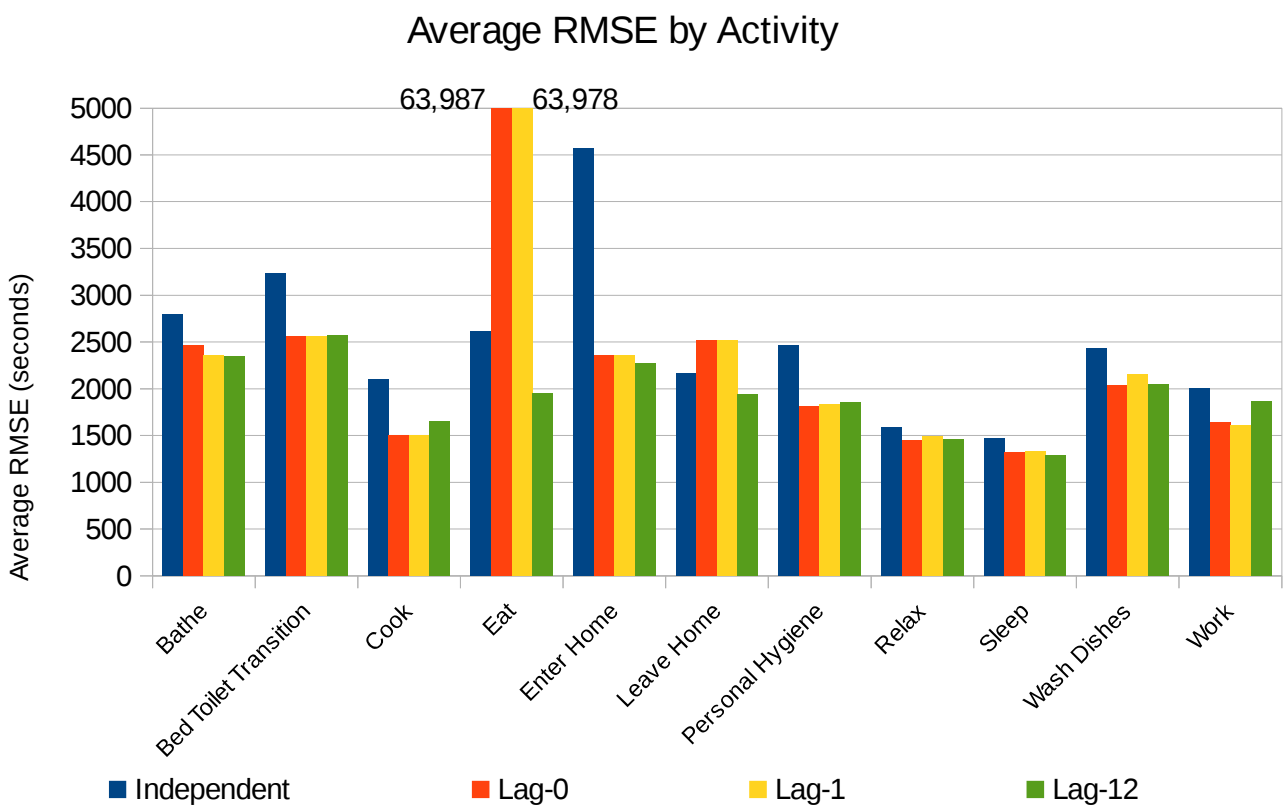


Figure 7.15: Two-pass RMSE results by activity.

the predictions from events before the resident left home in order to better predict when they will return.

While examining the results of these experiments we noted some common causes of errors. The independent predictor had large errors where the time between sensor events was very large. In these cases, the gap in the timeStamp features tends to cause the model to output an incorrect prediction. Large errors are also caused by a transition between different leaf nodes in the model tree. At these points, the splitting features in the tree vary enough that the model will use a different node for a few events and produce large-error predictions before moving back to a more reasonable state. It is likely that the lack of both of these types of situations in the training data may be why the predictor has trouble handling them.

For the recurrent predictor (and TPAP overall), the largest errors are due to the independent model's errors. When the independent model has one of the problems noted above, it will cause the lag values to become erroneous as well, reducing the performance of the overall predictor. The recurrent predictor is also susceptible to large gaps in the event times, as it relies on the lag values without adjustments for spacing between events. Although we tried using adjusted lag values to reduce this problem, we found that the performance did not noticeably increase.

Overall the two-pass method performs well, indicating a combination of predictors and more history information can further enhance prediction. However, there are still some open questions regarding its effectiveness and how it reacts to variations in parameters. It is possible that adding more lag features can improve the performance further, as supported by the overall results. However, there are some activities where the lag-12 predictor performs worse. Thus, we plan to continue investigating this prediction method to determine how it can be most effectively used in further experiments.

In this chapter we described a number of experiments to validate our AP algorithms. We wanted to understand how the various features of each algorithm improve performance, as well as how they are affected by different variations in the data. Our basic activity prediction premise was found to be a strong baseline performance which shows lower error than a linear regression or SVM model. We also found that the sampling features help improve performance, though not as dramatically as the addition of new sensor types. The inclusion of additional contextual information about activity relationships is beneficial in both reducing error and smoothing predictions. This can be further enhanced using a two-pass method that combines multiple predictors. Overall, out AP algorithms are found to be effective at forming activity predictions

in an automated fashion, supporting our hypothesis. In the next chapter we describe pilot tests of an activity prompting application using our AP algorithms in a real-world context.

CHAPTER 8. PROMPTING APPLICATION

In order to test the utility of our AP methods in a real-world application scenario, we have deployed a mobile-based activity prompting application called CAFE (CASAS Activity Forecasting Environment). This application provides reminder prompts to users based on predictions made by an AP algorithm. Prompts can provide important assistance to smart environment inhabitants to help remind them to perform needed ADLs. For example, a prompt could be used to remind the inhabitant to take their medications before dinner each evening. Prompting systems can help increase independence for cognitively-impaired individuals and increase adherence to medical interventions [153, 154]. Since these prompts are best delivered in the context of inhabitants' daily activities [155], we use an AP algorithm to predict when activities will occur and prompt appropriately. In this chapter we describe two pilot studies of the CAFE app, using two different prediction methods, along with their results.

8.1 CAFE Prompting App

The CAFE prompting app is an iOS app designed to run on mobile devices. The app periodically queries a server to get the latest predicted times for activities of interest. These predictions are generated every 15 minutes using one of our AP methods, using the latest sensor events from the smart environment. The app schedules alerts to prompt the user to perform the activities at the predicted time, as shown in Figure 8.1. When the user responds to a prompt, they are presented with multiple responses to indicate whether they are performing the activity, plan to perform it, or already performed it. Their response is logged to the server

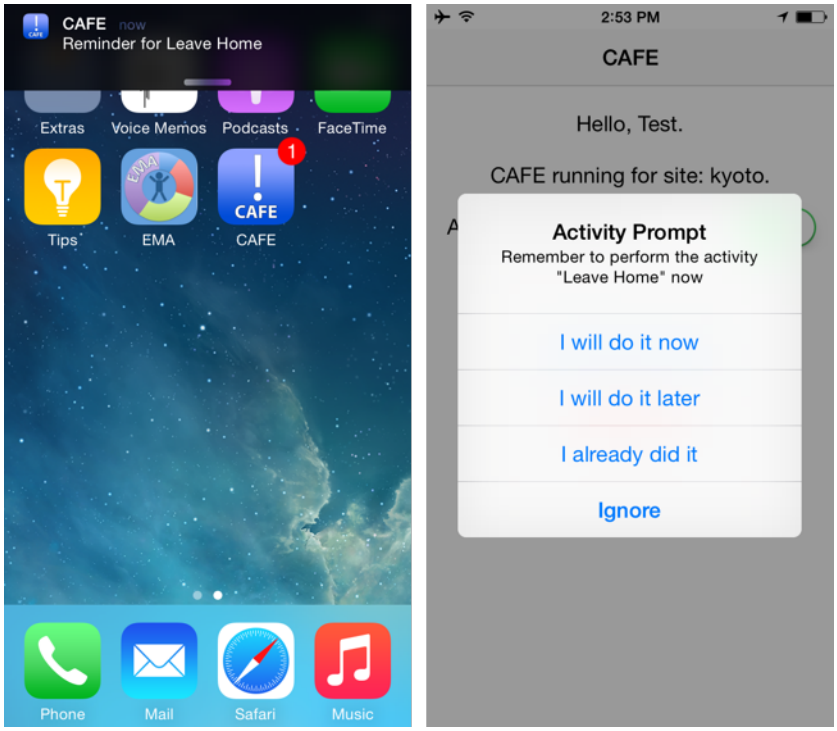


Figure 8.1: Interface for the CAFE prompting app. The screen on the left shows a prompt notification, while the right screen shows the user response screen.

for later analysis.

Due to the restrictions of the iOS platform the retrieval of new predictions from the server is scheduled by the iOS process manager. The update is not guaranteed to happen at a specific interval, although it tends to occur approximately every 15-45 minutes. Combined with the 15 minute update interval of the server-side predictions, this can lead to 30 minute or longer delays in fetching new predictions from the server. While any older predictions are used and displayed in the interim, these predictions may not be as accurate as those generated from newer sensor events closer to the activity occurrence. Thus, there is some reduction in performance introduced in the system delays inherent to the app.

The app was deployed in two separate pilot studies, each featuring the same two participants. One participant lived in the Navan testbed, while the other lived in the Kyoto testbed (both described in Section 6.4). The Navan participant was the sole resident of that home, while the Kyoto participant shared the home with another resident (who did not partake in the study). For each testbed only the motion and door sensors were used and were retrieved for the predictor at the 15-minute intervals. Since the Kyoto testbed houses experiments with guest participants during the 9:00am to 4:00pm timeframe when the participant is away, these events are not used in either training or use of the models. Participants were prompted for seven activities: Bathe, Cook, Eat, Leave Home, Relax, Sleep, and Work.

8.2 Independent Predictor Prompter

For the first pilot test we evaluated CAFE using IP to generate activity predictions. The predictor for Kyoto was trained on two months' worth of data for that testbed, while the one

Table 8.1: Data statistics for first CAFE app deployment with the IP algorithm.

Testbed	Residents	Time Span	Sensor Events
kyoto	2	2 weeks	147,919
navan	1	2 weeks	57,241

for Navan was trained on four months of data. This training data had been labeled by AR using a generalized model trained from the Horizon House datasets. Only the discrete features were used for the independent predictor. The pilot lasted two weeks, over which time the data generated by each testbed is as shown in Table 8.1. While the predictions were being generated and used by the app, AR was used to label the inhabitants' activities with the same generalized model used to generate the training data. This labeled data is used as a comparison to determine the effectiveness of the activity prompts.

The participants provided 112 responses, which were fairly equally divided between "I will do it now", "I already did it", and "I will do it later". Since the participants had to open the mobile device to respond to the prompt, there was sometimes a delay between when the actual prompt occurred and the participant's response. Thus, while the resident may have been actually performing the activity when the prompt occurred, they may not have seen the prompt until later and thus replied "I already did it" or "I will do it later" instead.

Using the AR-labeled sensor data for comparison, we are able to compute MAE and normalized MAE measures as shown in Table 8.2. These data are based on the predictions made for prompts where responses were collected. Since each activity occurred at least once per day, the MAE is normalized by 43,200 seconds, or half a day. The MAE value is 2,925

Table 8.2: First CAFE app deployment results, averaged over all collected responses. MAE value in seconds. Normalized MAE is computed using 43,200 seconds as the normalization factor. κ-normalized ETF was adjusted using the accuracy of the AR algorithm.

MAE	Normalized MAE	ETF(1800 s)	κ-Normalized ETF
2,925	0.07	0.64	0.72

seconds, or about 48 minutes. Given the inherent delays in the app infrastructure that can lead new predictions to be delayed by 30 minutes or more, we find this to be a reasonable error. We also compute the ETF value using a threshold of 30 minutes (the infrastructure delay), as shown in the table. 64% of the prediction errors are less than 30 minutes, indicating that the prediction errors were largely smaller.

We note that these error levels were achieved even after 2 weeks of sensor activity and with events occurring many months after the training data. Compare this with the horizon-based results from Section 7.3, where the error rose much more rapidly. We believe that the use of extra training data for the prompting deployment allows the error rate to be reduced over time compared to the smaller window used for the previous sliding window validations.

While the participants were taking part in this CAFE pilot study, they were also using a separate app named EMA (Ecological Momentary Assessment). This is an established method to obtain participant information "in the moment" in order to collect more accurate information about their activities [156]. The app queried participants every 15 minutes asking what activity they were performing. The stored responses can be used to validate our AR algorithm. Using these responses, we found that AR had about 92% accuracy on the seven activities being

prompted. This leads to an adjusted κ-normalized ETF value of 0.72, as listed in Table 8.2.

We also used the AR-labeled data to compare the participant's prompt responses to their actual behavior. We observe that, when the participant responded to a prompt with "I will do it now", they always initiated the activity within the next 20 minutes. This indicates that the prompts were effective in influencing and predicting the inhabitants' behavior in these circumstances.

8.3 Two-Pass Prompter

The second pilot test utilized the TPAP algorithm for generating the activity predictions. The lag-12 predictor was used for the second stage with discrete features for both stages. An AR model was trained separately for each testbed to generate training data. For Kyoto the model was trained on human-annotated data covering a five-year span from multiple datasets. However, the human-annotated data was for residents other than those currently living at this site. In addition, Kyoto houses two residents which complicates activity recognition and prediction. Navan was trained on data spanning two years from only that testbed. The AR models were then used to label more recent sensor data for each testbed: two months' worth for Kyoto and three months for Navan. This AR-labeled data was then used to learn a TPAP model for each testbed. Four weeks' worth of data were collected for this pilot test, with the statistics shown in Table 8.3. As before, AR was used alongside TPAP in order to provide activity labels for each event.

In the previous pilot study the participants would often get multiple repeated prompts for the same event within a few minutes of each other. This was due to the fact that the activity

Table 8.3: Data statistics for second CAFE app deployment with TPAP predictor.

Testbed	Residents	Time Span	Sensor Events
kyoto	2	4 weeks	185,560
navan	1	4 weeks	154,997

predictions often shift slightly with each new prediction computation. When the participants responded to one prompt the predictions were instantly updated and so a second prompt might be scheduled shortly after the first one. The participants requested that a minimum timing between prompts be enforced so that they were less bothered by the app. For this second pilot test a minimum time between prompts for the same activity of 15 minutes was enforced.

This study also added a response option of "I am doing this now (or in the last half hour)". We designed this option to capture the instances when the user was prompted for an activity while they were performing it, but did not respond to the prompt until after they finished. Users could respond to a prompt with "I am doing this now (or in the last half hour)", "I will do it now", or "I will do it later".

98 prompt responses were received during this pilot test. The distribution of responses favored "I will do it later", as shown in Figure 8.2. Figure 8.3 shows the total number and distribution of responses for each activity. The Bathe and Leave Home activities were the most prompted by a significant margin. For both of these activities, a large number of the participant responses were "I will do it later", indicating that the predictions were such that prompts were presented too early. One of the participants relayed that they felt they were only being prompted for the Bathe activity for long periods of time. Examining the predictions

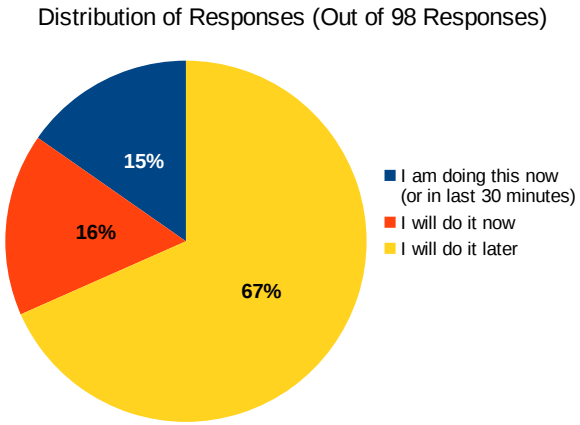


Figure 8.2: Distribution of responses for second CAFE app deployment.

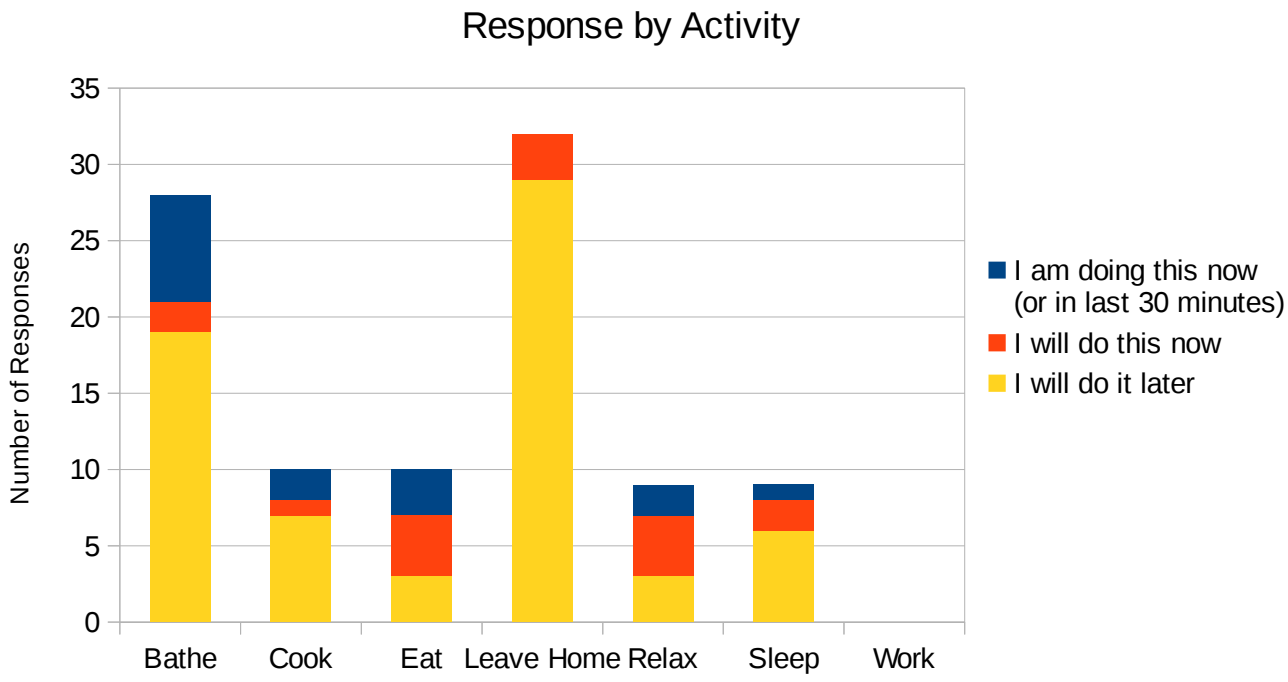


Figure 8.3: Responses by prompt activity for second CAFE app deployment. No prompts were sent for the Work activity.

indicated that AP would sometimes keep moving the predicted time for many of the activities by a few minutes with each prediction iteration. This had the effect that only a few of the activities (mostly Bathe) would be prompted during some time periods. This is something we intend to investigate further with future studies.

We also note that there was often a significant delay between when a prompt was issued and the inhabitant responded. This is shown in Table 8.4. While often the response delay was less than a minute, there were multiple times when the delay was a half hour or even more. The average delay was approximately a half hour. Since the inhabitant responses reflected the inhabitants' activities when they *responded* to the prompt rather than *received* it, the prompted activity may have already occurred by the time of their response.

Also shown in Table 8.4 is the MAE and normalized MAE for the prompted activity times. These values were calculated using the AR-labeled data as in the last study, with the same normalization factor of a half day for the normalized MAE. The MAE was higher for this pilot by about 2,000 seconds (about a half hour). This increase in error may be due to differences in the inhabitants' activities that made them more difficult to predict. It is also possible that the AR models are less accurate in this case. We note that the larger training windows with widely varying patterns (including multiple different sets of residents in at least one of the testbeds) may have reduced the accuracy of AR. The EMA app was not a part of this study, so ground-truth annotations cannot be used to analyze the accuracy of the AR models. The ETF value with a 30-minute threshold for this test was 0.56, indicating a few more large errors compared to the previous pilot, although the number is similar.

The pilot studies presented here have allowed us to gain a better understanding of the utility of the AP algorithm and some of the challenges of using it in real applications. We found

Table 8.4: Second CAFE app deployment results, averaged over all collected responses. MAE value in seconds. Normalized MAE is computed using 43,200 seconds as the normalization factor. The average response delay is the average number of seconds between when a prompt is issued and the participant responds.

MAE	Normalized MAE	ETF(1800 s)	Average Response Delay
4,282.5	0.10	0.56	1,508

the resident feedback regarding the prompting to be especially interesting. In addition to the feedback mentioned previously, the inhabitants also suggested that the app sometimes created a modification in behavior. One participant mentioned he was debating between getting groceries and watching television when he received a CAFE prompt for Leave Home. He immediately left and went to perform his errands. At another time a participant started working earlier than planned due to a prompt. These instances are representative of the challenges in intervention design and prompting that are important to consider in future work.

These pilot studies have indicated that AP can be usefully applied for behavior prompting in a real-world context. We hope to continue developing the CAFE prompting app alongside our AP algorithms to improve both performance and usability. Future plans include deploying the app with older adults in wider studies. By learning how users interact with and are affected by the capabilities of activity prediction algorithms we can continue to improve and grow in understanding of the activity prediction problem and solutions.

CHAPTER 9. CONCLUSION

9.1 Summary and Conclusions

In this dissertation we have addressed the challenge of predicting human activities. This is an important area whose importance will only continue to increase as the number of activity-aware services continues to grow. Our work focuses on the ability to predict inhabitant activities within a smart home environment by drawing on sensor observations of daily life. While others have explored this area in the past, our work presents a novel solution to this problem with an automated method to model complex activity patterns in order to generate numeric predictions of activity times. We hypothesized that these activity times can be predicted using machine learning and time series techniques and information found in sensor data. To provide evidence supporting our hypothesis, we implemented and evaluated several Activity Predictor methods.

We have found that using feature extraction to draw useful contextual features from the raw sensor event data can provide a useful basis for activity prediction. The discrete and sampling features used provide information about the temporal nature and state values of activities, both of which have been shown to be useful for informing predictors. The ability of a regression learner to form accurate predictions of activities has also been demonstrated. Using features drawn directly from the sensor data, our IP predictor is able to form predictions under 15 minutes on average. The nonlinear model tree used in our algorithms can model the activity information effectively, without added complexity. Therefore, our underlying hypothesis that

numeric activity times can be accurately predicted from sensor data is valid.

We have also found that the use of activity relationship information is useful for predicting activities and that it can be easily encoded as lagged prediction values. Our RAP implementation of this enhancement is shown to have increased performance, while also reducing the variability of the predicted times. By comparing performance against an Oracle predictor we have also found that RAP can be further enhanced with the help of other published training techniques. We also have found that a combination of two separate predictors, such as in our TPAP approach, can allow for further improvements in performance while addressing problems due to training. The power in this method includes its ability to be customized using different regression learners for even greater results.

We have also discussed and used a variety of evaluation metrics and methods throughout our work. The challenge of interpreting algorithm performance through several evaluation metrics has been discussed. Our results have indicated firsthand how various metrics can highlight different aspects of prediction error and the importance of using multiple metrics to fully understand what is occurring. We have also deployed two pilot tests of our CAFE prompting app. From these tests we have observed the ability of AP to form useful and helpful predictions for real-world applications, but we have also discovered new challenges for the future.

Our results have supported our hypothesis on automated activity prediction from sensor data. The methods presented here draw their feature context information from the sensor data and their own predictions, without the need for user input. By using nonlinear model trees, it is possible to model the complexities of activity relationships in a concise way with ample performance. This has been demonstrated in the low error results observed in both our

validation and prompting tests.

9.2 Future Work

While the work detailed in this dissertation reflects strong progress in the activity prediction field, there are still many challenges and issues that will need to be addressed by future works. We are particularly interested in using a variety of different models and parameters in our two-pass prediction method to further test its performance and see how it can be improved. We feel that the use of boosting and other methods can help reduce errors even further without severely increasing complexity or runtime performance. The effects of training dataset size and differences between training and test data on performance are also of interest as questions related to these issues have only partially been resolved in our current work.

We are also interested in introducing additional enhancements to our AP framework to further support automated activity prediction. We hope to develop a system for incremental updating of the regression model over time in order to adapt to changes in inhabitant activities. Further work will include addressing the challenge of predicting activity durations and reducing the errors surrounding the issue of the "jump" in activity times. Our future work will also explore the development and use of new error metrics and the application of established metrics to address the challenges of prediction evaluation. There are still many unanswered questions in the area of evaluation which we hope to address.

One of the major challenges for future research in this area lies in being able to adapt our methods to different situations and problems. Consideration should be made for how to adapt to new users, activities, and environments. For example, it is important to determine

the ability of our activity prediction models to accurately predict new users' activities based on the habits of current users. This might be done through the user of transfer learning and incremental update methods. Other adaptations that we plan to consider include adapting to new types of prediction (e.g, energy prediction, traffic forecasting, etc.) and incorporating user feedback to better inform the predictor and allow it to adapt. These adaptations pose many interesting questions which we hope to address in our future work.

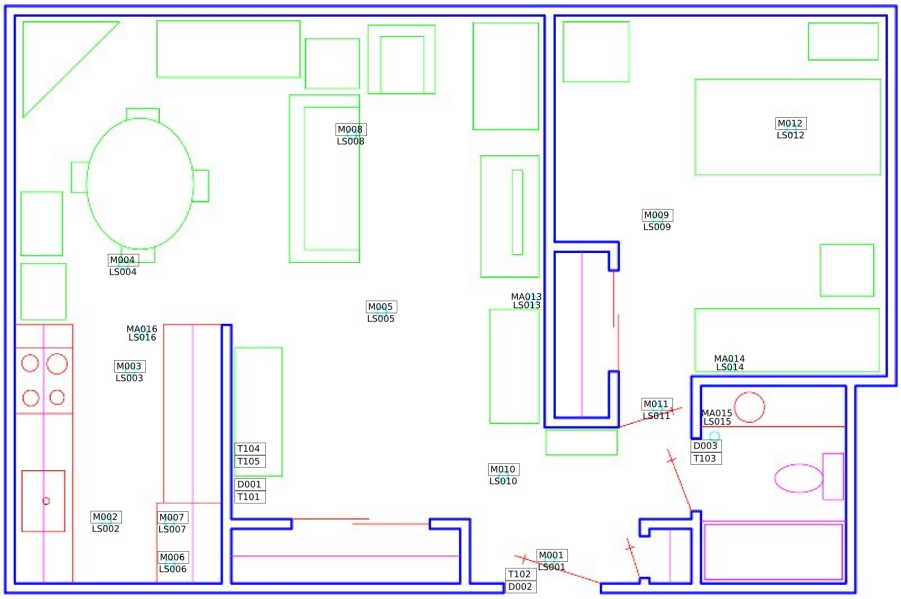
We also plan to further develop and deploy our CAFE prompting software. We plan to revise and improve the user experience to both reduce the burden placed on the user and to improve prompt effectiveness. Future deployments are planned to include more participants, including many older adults for whom activity prompts can be most useful.

The work in this dissertation has provided first steps toward new methods of activity prediction. Our AP framework is able to predict activities with reasonable accuracy and limited need for human setup or interaction. Although we have shown a number of improvements and successes for activity prediction in this work, there are still many new challenges to explore in this area. Our hope is that the continued development of prediction technologies will lead to greater health and independence for older adults and others in their daily lives. There is a need in many fields for improved activity predictions based on automated observations, and our work with automated contextual prediction provides a first step in this direction.

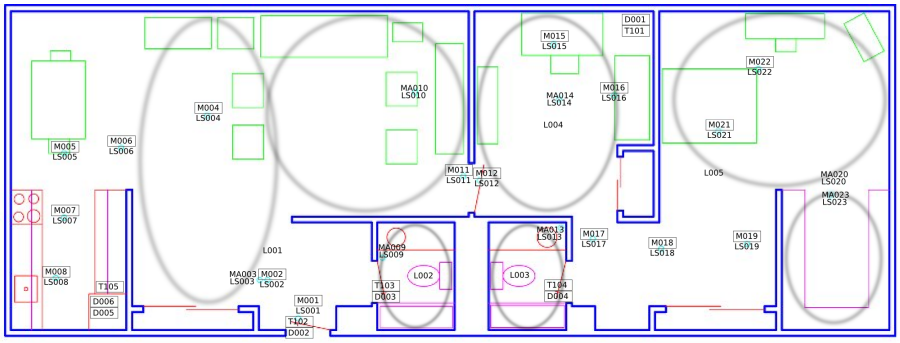
Appendix: Horizon House Testbeds

Horizon House testbeds. This table contains information about the number of residents and sensors in each testbed.

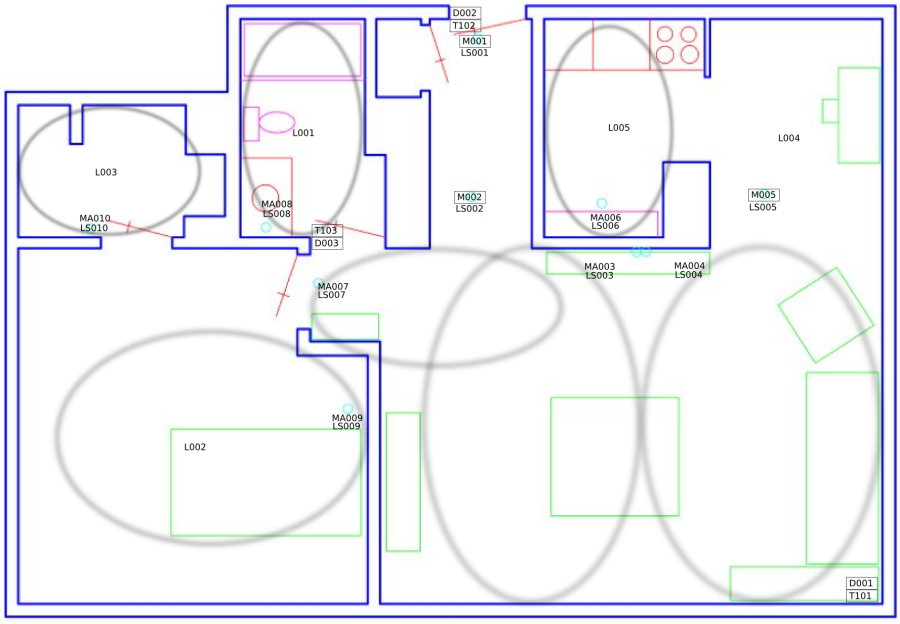
Dataset	Residents	Sensors	Dataset	Residents	Sensors
HH101	1	19	HH114	1	17
HH102	1	32	HH115	1	20
HH103	1	13	HH116	1	19
HH104	1	33	HH117	1	16
HH105	1	26	HH118	1	25
HH106	1	35	HH119	1	18
HH107	2	32	HH120	1	20
HH108	1	29	HH121	2	32
HH109	1	26	HH122	1	28
HH110	1	22	HH123	1	20
HH111	1	29	HH124	1	19
HH112	1	20	HH125	1	34
HH113	1	29			


M008
LS008
M012
LS012
M009
LS009
M004
LS004
M005
LS005
MA013
LS013
MA016
LS016
M003
LS003
MA014
LS014
M011
LS011
MA015
LS015
D003
T103
T104
T105
D001
T101
M010
LS010
M002
LS002
M007
LS007
M006
LS006
M001
LS001
T102
D002

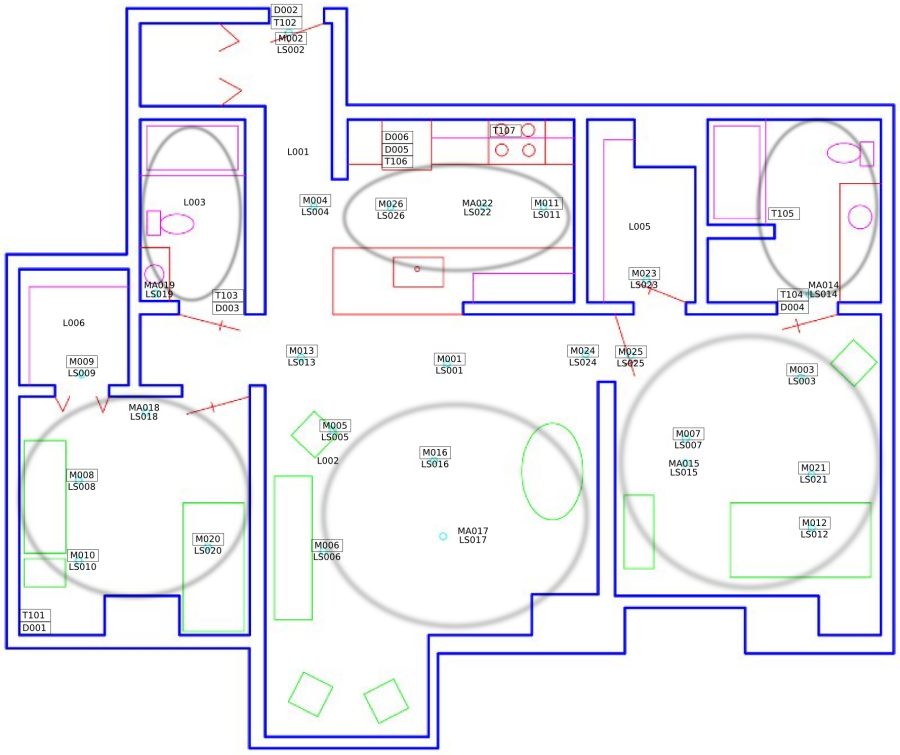
HH101


D001
T101
M015
LS015
M022
LS022
MA010
LS010
MA014
LS014
M016
LS016
M004
LS004
L004
M021
LS021
M005
LS005
M006
LS006
M011
LS011
M012
LS012
L005
MA020
LS020
MA023
LS023
M007
LS007
M013
LS013
M017
LS017
M018
LS018
M019
LS019
L001
MA009
LS009
M008
LS008
MA003
LS003
M002
LS002
L002
L003
T103
D003
T104
D004
T105
D006
D005
M001
LS001
T102
D002

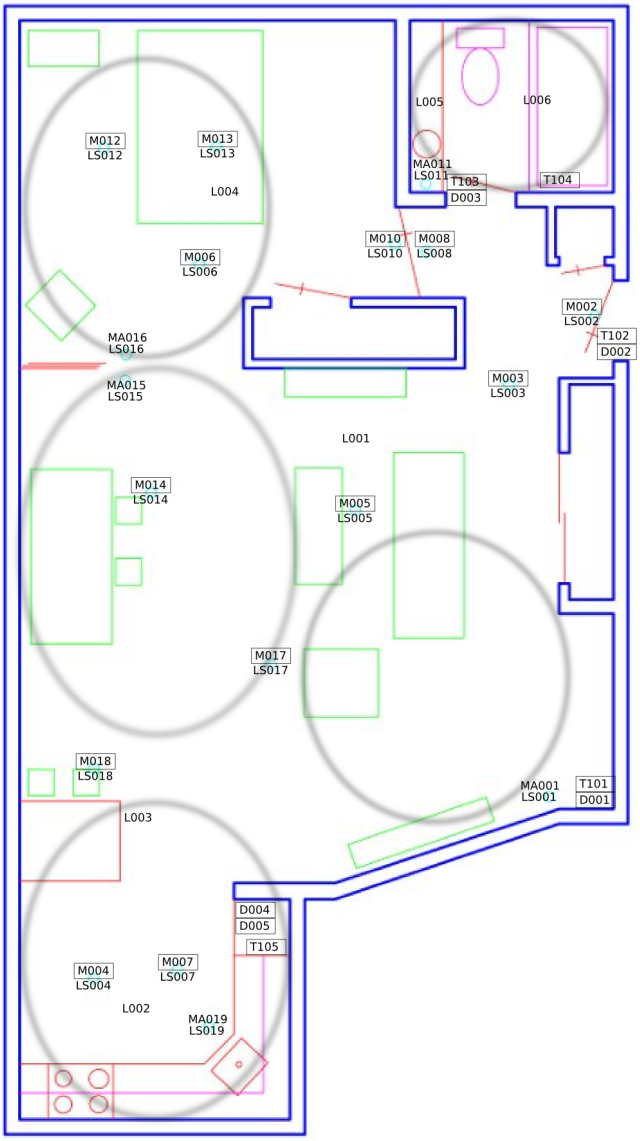
HH102


D002
T102
M001
LS001
L001
L003
L005
L004
M002
LS002
M005
LS005
MA008
LS008
MA010
LS010
T103
D003
MA006
LS006
MA003
LS003
MA004
LS004
MA007
LS007
MA009
LS009
L002
D001
T101

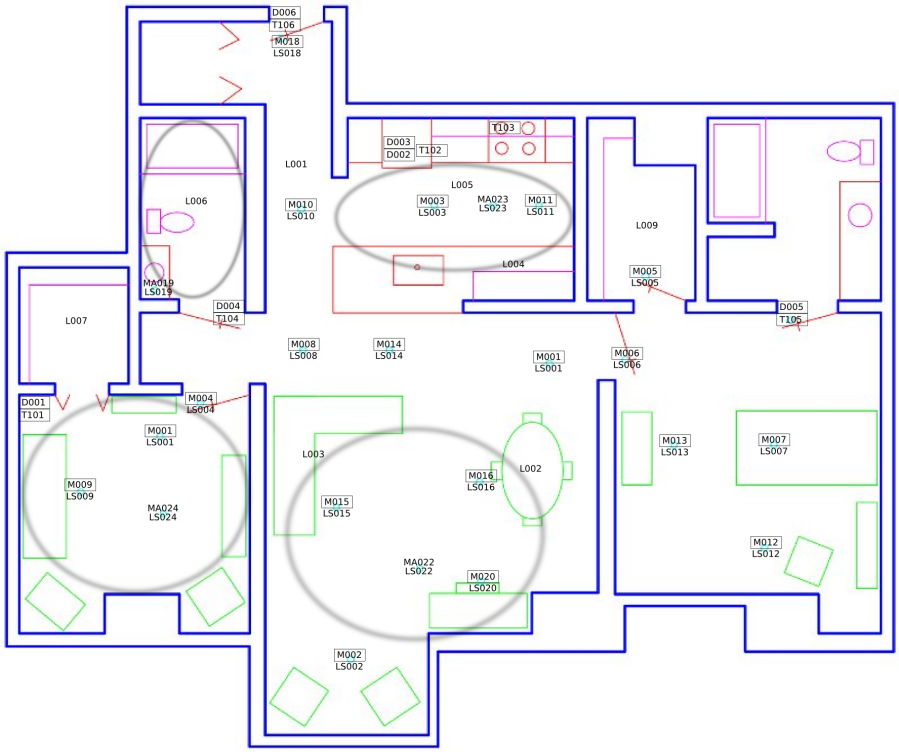
HH103


D002
T102
M002
LS002
D006
D005
T106
T107
L001
L003
M004
LS004
M026
LS026
MA022
LS022
M011
LS011
L005
T105
MA019
LS019
T103
D003
M023
LS023
MA014
LS014
T104
D004
L006
M009
LS009
M013
LS013
M001
LS001
M024
LS024
M025
LS025
M003
LS003
MA018
LS018
M005
LS005
M007
LS007
M016
LS016
L002
MA015
LS015
M021
LS021
M008
LS008
M020
LS020
MA017
LS017
M012
LS012
M006
LS006
M010
LS010
T101
D001

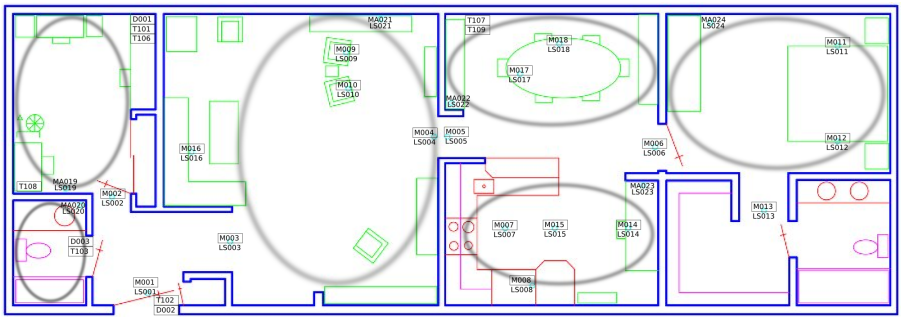
HH104


L005
L006
M012
LS012
M013
LS013
MA011
LS011
T103
D003
T104
L004
M010
LS010
M008
LS008
M006
LS006
M002
LS002
T102
D002
MA016
LS016
MA015
LS015
M003
LS003
L001
M014
LS014
M005
LS005
M017
LS017
M018
LS018
MA001
LS001
T101
D001
L003
D004
D005
T105
M004
LS004
M007
LS007
L002
MA019
LS019

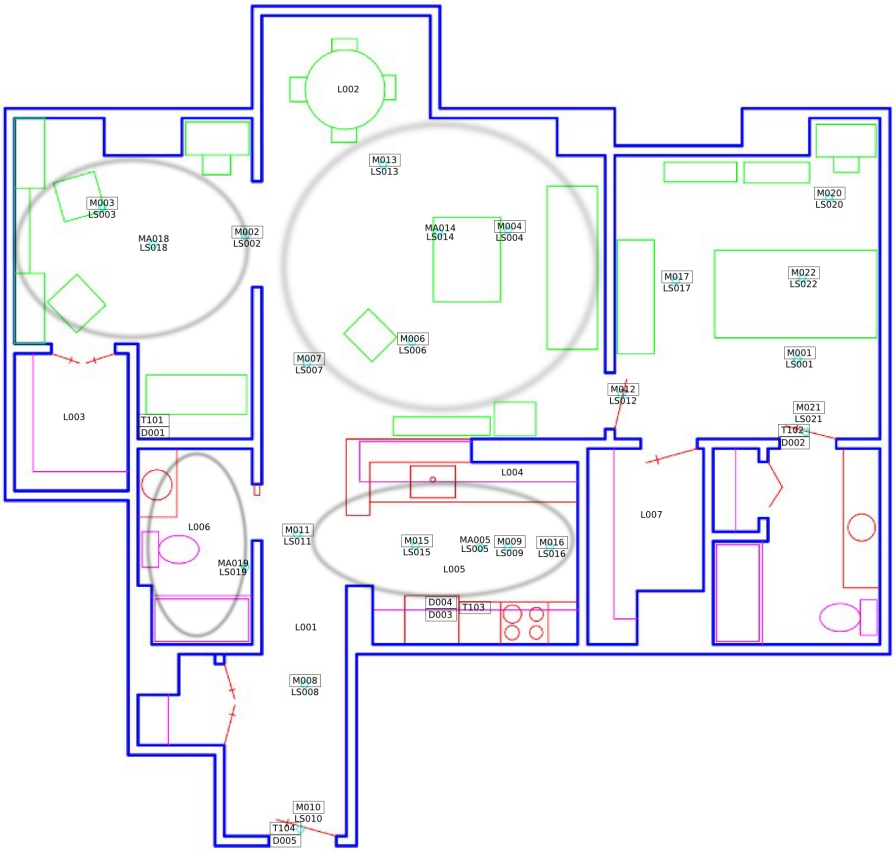
HH105


D006
T106
M018
LS018
T103
D003
D002
T102
L001
L005
L006
M010
LS010
M003
LS003
MA023
LS023
M011
LS011
L009
L004
M005
LS005
MA019
LS019
D004
T104
L007
D005
T105
M008
LS008
M014
LS014
M001
LS001
M006
LS006
D001
T101
M004
LS004
M001
LS001
L003
L002
M013
LS013
M007
LS007
M009
LS009
MA024
LS024
M016
LS016
M015
LS015
M012
LS012
MA022
LS022
M020
LS020
M002
LS002

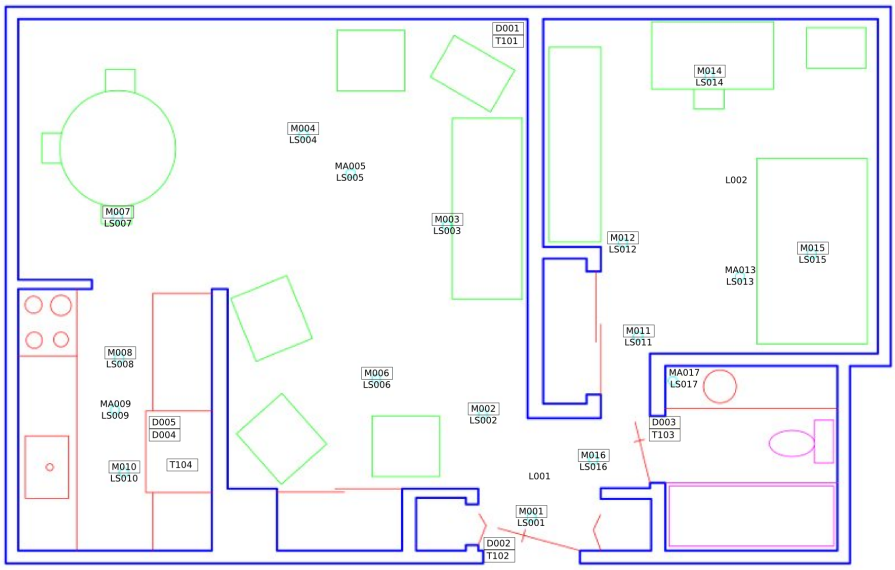
HH106


D001
T101
T106
MA021
LS021
T107
T109
MA024
LS024
M009
LS009
M018
LS018
M011
LS011
M017
LS017
M010
LS010
MA022
LS022
M004
LS004
M005
LS005
M016
LS016
M006
LS006
M012
LS012
T108
MA019
LS019
M002
LS002
MA023
LS023
MA020
LS020
M013
LS013
M007
LS007
M015
LS015
M014
LS014
D003
T103
M003
LS003
M008
LS008
M001
LS001
T102
D002

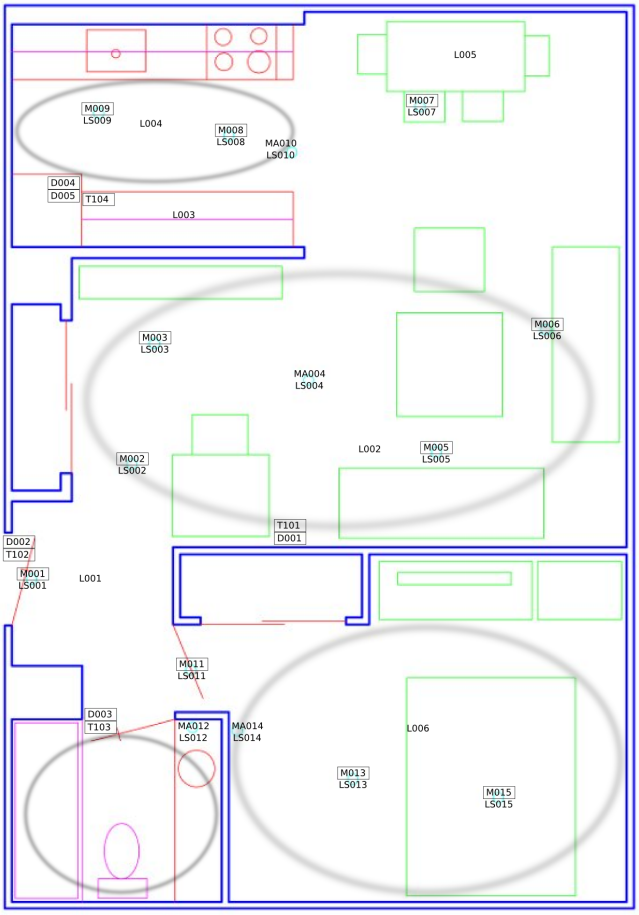
HH107


L002
M013
LS013
M003
LS003
MA018
LS018
M002
LS002
MA014
LS014
M004
LS004
M020
LS020
M017
LS017
M022
LS022
M006
LS006
M007
LS007
M001
LS001
M012
LS012
L003
T101
D001
M021
LS021
T102
D002
L004
L007
L006
M011
LS011
M015
LS015
MA005
LS005
M009
LS009
M016
LS016
MA019
LS019
L005
D004
D003
T103
L001
M008
LS008
M010
LS010
T104
D005

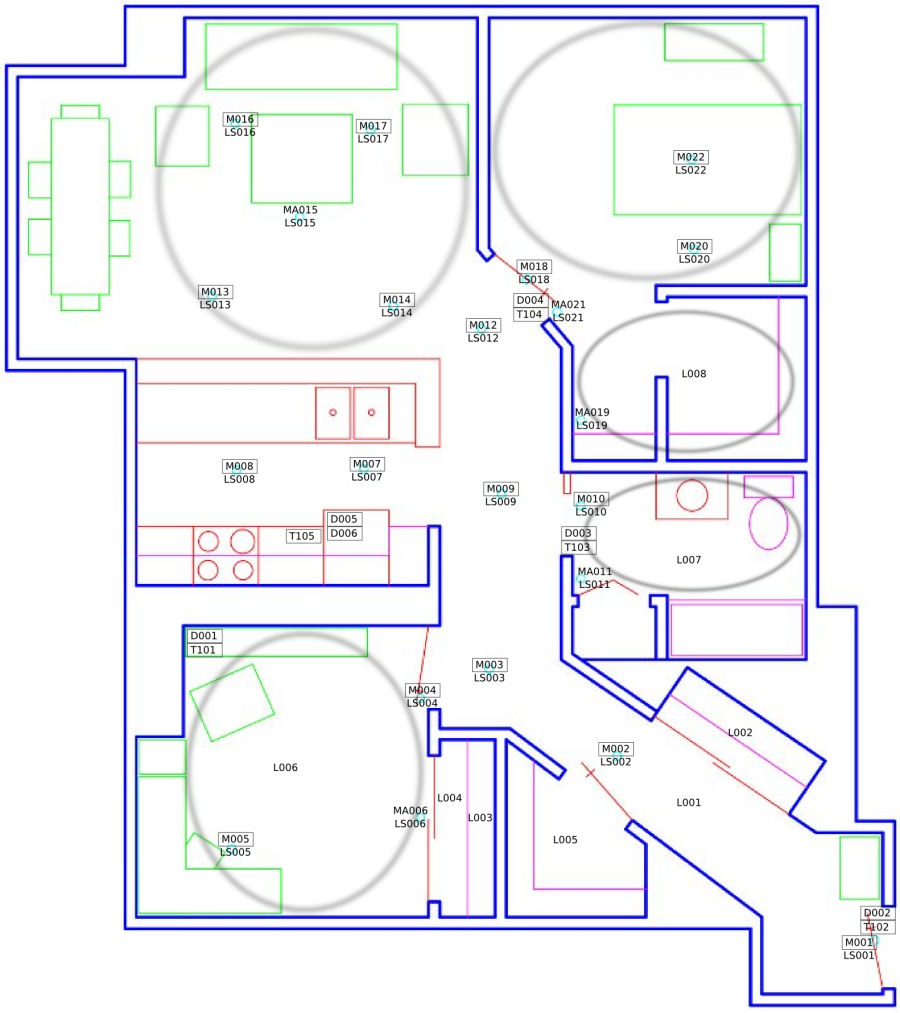
HH108


D001
T101
M014
LS014
M004
LS004
MA005
LS005
L002
M007
LS007
M003
LS003
M012
LS012
M015
LS015
MA013
LS013
M011
LS011
M008
LS008
MA017
LS017
M006
LS006
MA009
LS009
M002
LS002
D005
D004
D003
T103
M010
LS010
T104
M016
LS016
L001
M001
LS001
D002
T102

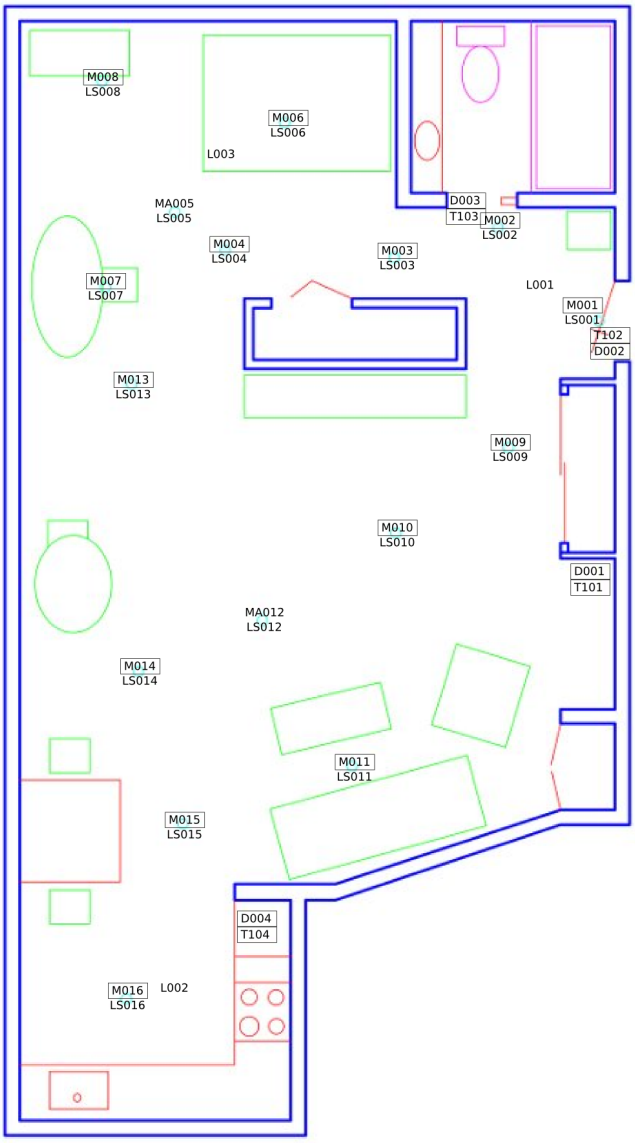
HH109


L005
M007
LS007
M009
LS009
L004
M008
LS008
MA010
LS010
D004
D005
T104
L003
M003
LS003
M006
LS006
MA004
LS004
L002
M005
LS005
M002
LS002
T101
D001
D002
T102
M001
LS001
L001
M011
LS011
D003
T103
MA012
LS012
MA014
LS014
L006
M013
LS013
M015
LS015

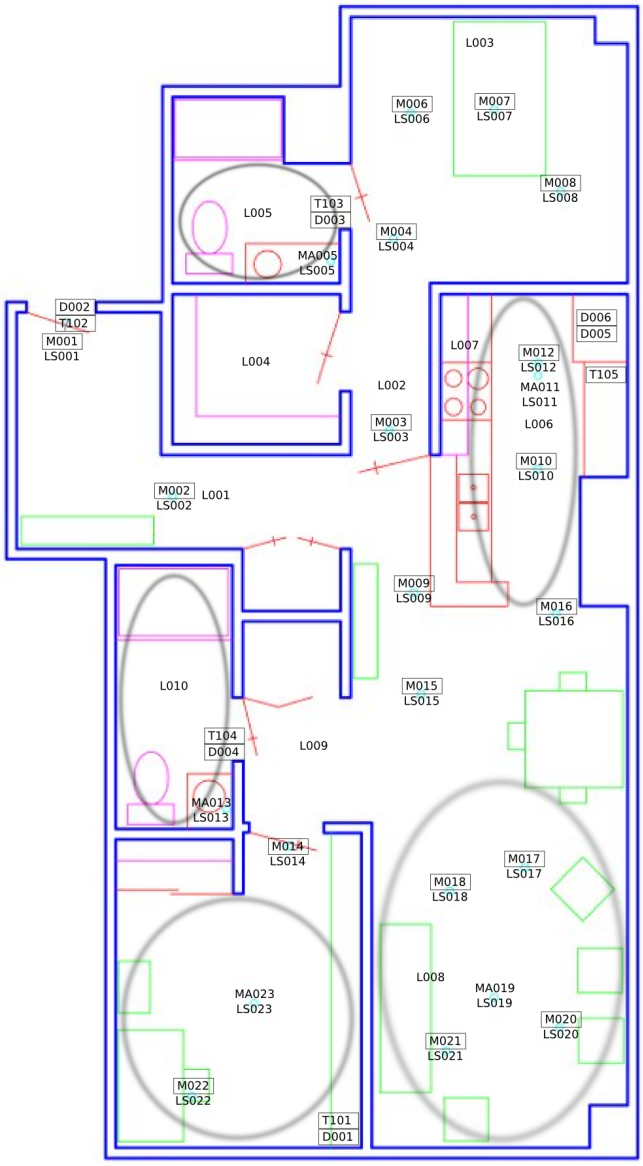
HH110


M016
LS016
M017
LS017
MA015
LS015
M022
LS022
M020
LS020
M018
LS018
M013
LS013
M014
LS014
D004
T104
MA021
LS021
M012
LS012
L008
MA019
LS019
M008
LS008
M007
LS007
M009
LS009
M010
LS010
D005
D006
T105
D003
T103
L007
MA011
LS011
D001
T101
M003
LS003
M004
LS004
L002
M002
LS002
L006
L004
MA006
LS006
L003
L001
M005
LS005
L005
D002
T102
M001
LS001

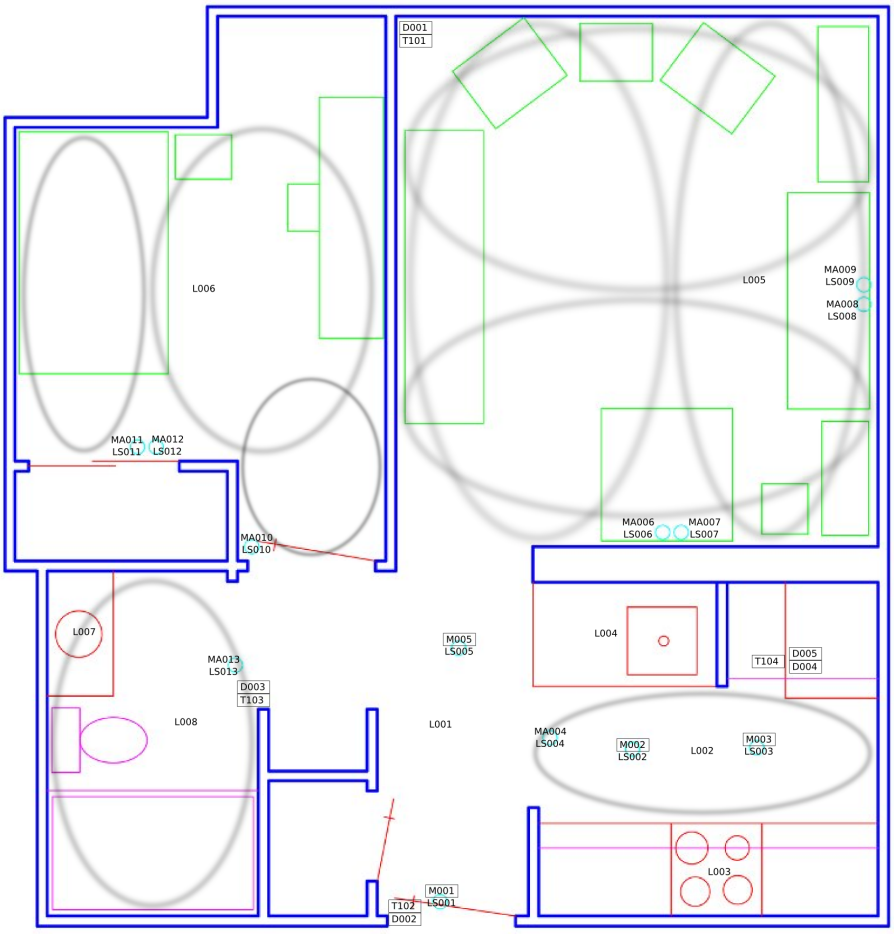
HH111


M008
LS008
M006
LS006
L003
MA005
LS005
D003
T103
M002
LS002
M004
LS004
M003
LS003
M007
LS007
L001
M001
LS001
T102
D002
M013
LS013
M009
LS009
M010
LS010
D001
T101
MA012
LS012
M014
LS014
M011
LS011
M015
LS015
D004
T104
M016
LS016
L002

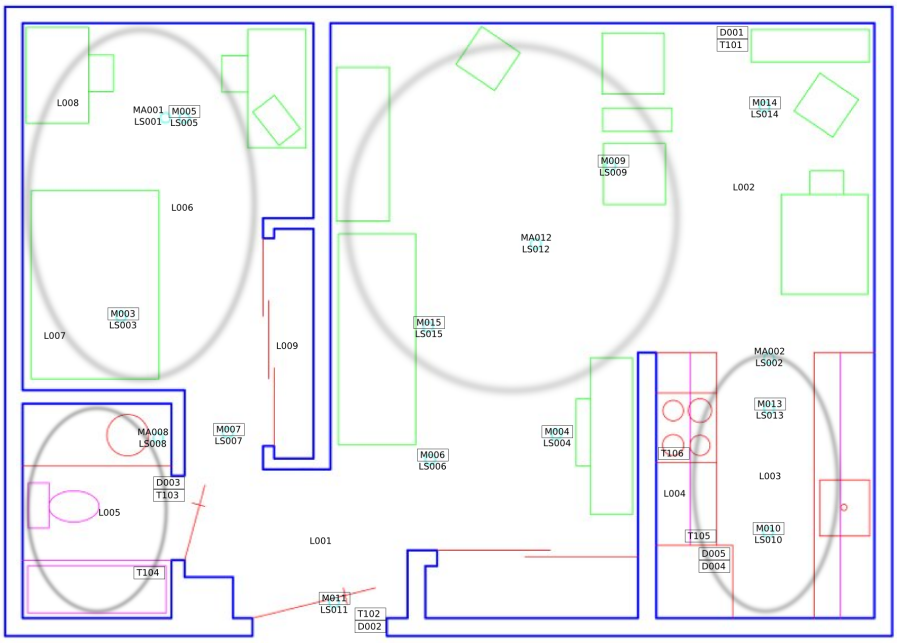
HH112


L003
M006
LS006
M007
LS007
M008
LS008
L005
T103
D003
M004
LS004
MA005
LS005
D002
T102
M001
LS001
L004
L007
M012
LS012
D006
D005
T105
MA011
LS011
L002
L006
M003
LS003
M010
LS010
M002
LS002
L001
M009
LS009
M016
LS016
L010
M015
LS015
T104
D004
L009
MA013
LS013
M014
LS014
M017
LS017
M018
LS018
L008
MA019
LS019
MA023
LS023
M020
LS020
M021
LS021
M022
LS022
T101
D001

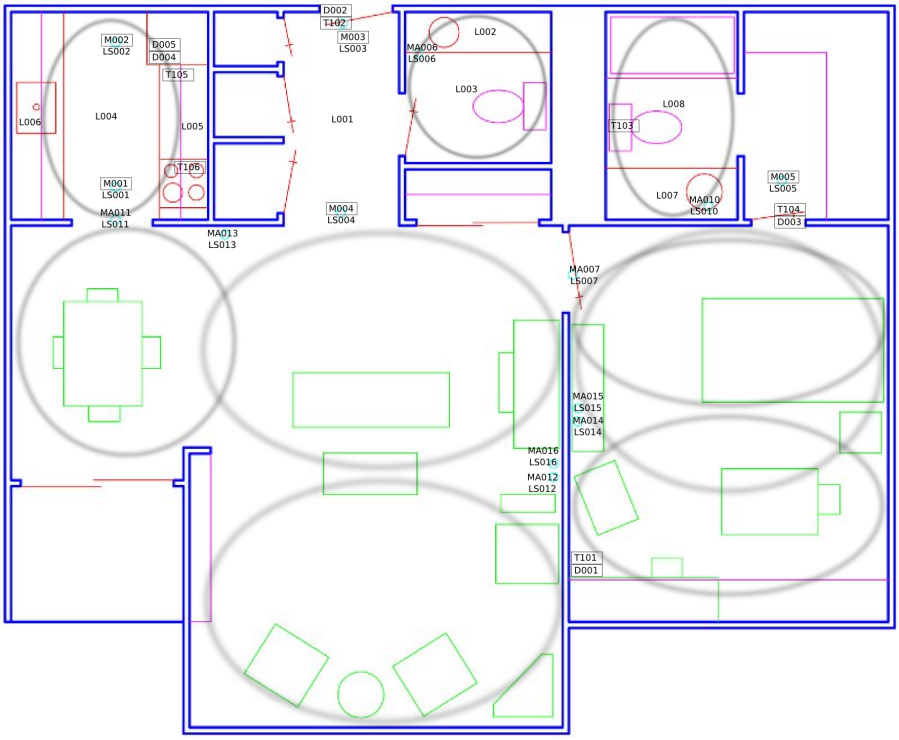
HH113


D001
T101
L006
L005
MA009
LS009
MA008
LS008
MA011
LS011
MA012
LS012
MA010
LS010
MA006
LS006
MA007
LS007
L007
M005
LS005
L004
T104
D005
D004
MA013
LS013
D003
T103
L008
L001
MA004
LS004
M002
LS002
L002
M003
LS003
L003
M001
LS001
T102
D002

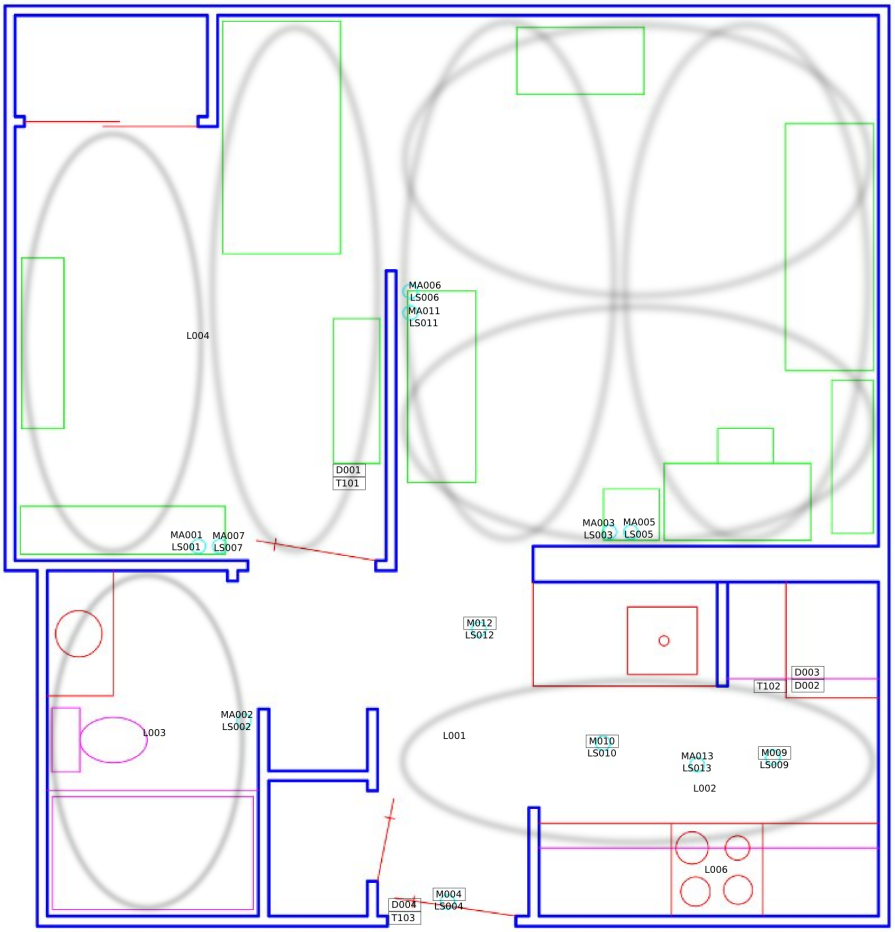
HH114


D001
T101
L008
MA001
LS001
M005
LS005
M014
LS014
M009
LS009
L002
L006
MA012
LS012
M003
LS003
L007
M015
LS015
L009
MA002
LS002
M013
LS013
MA008
LS008
M007
LS007
M004
LS004
T106
M006
LS006
L003
D003
T103
L004
L005
M010
LS010
T105
L001
D005
D004
T104
M011
LS011
T102
D002

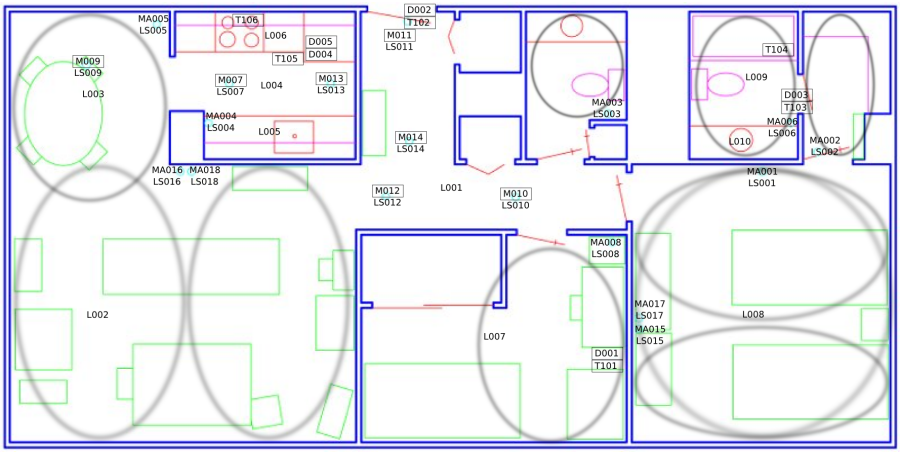
HH115


D002
T102
M002
LS002
D005
D004
M003
LS003
MA008
LS006
L002
T105
L003
L008
L006
L004
L005
L001
T103
T106
M005
LS005
M001
LS001
L007
MA011
LS011
M004
LS004
MA010
LS010
T104
D003
MA013
LS013
MA007
LS007
MA015
LS015
MA014
LS014
MA016
LS016
MA012
LS012
T101
D001

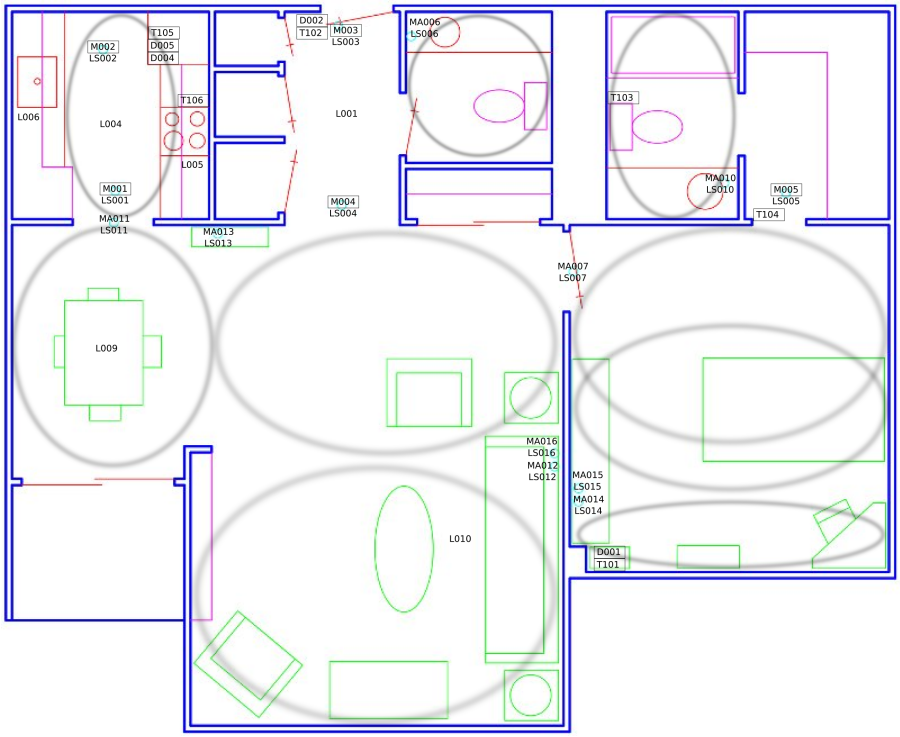
HH116


MA006
LS006
MA011
LS011
L004
D001
T101
MA001
LS001
MA007
LS007
MA003
LS003
MA005
LS005
M012
LS012
D003
D002
T102
MA002
LS002
L003
L001
M010
LS010
MA013
LS013
M009
LS009
L002
L006
M004
LS004
D004
T103

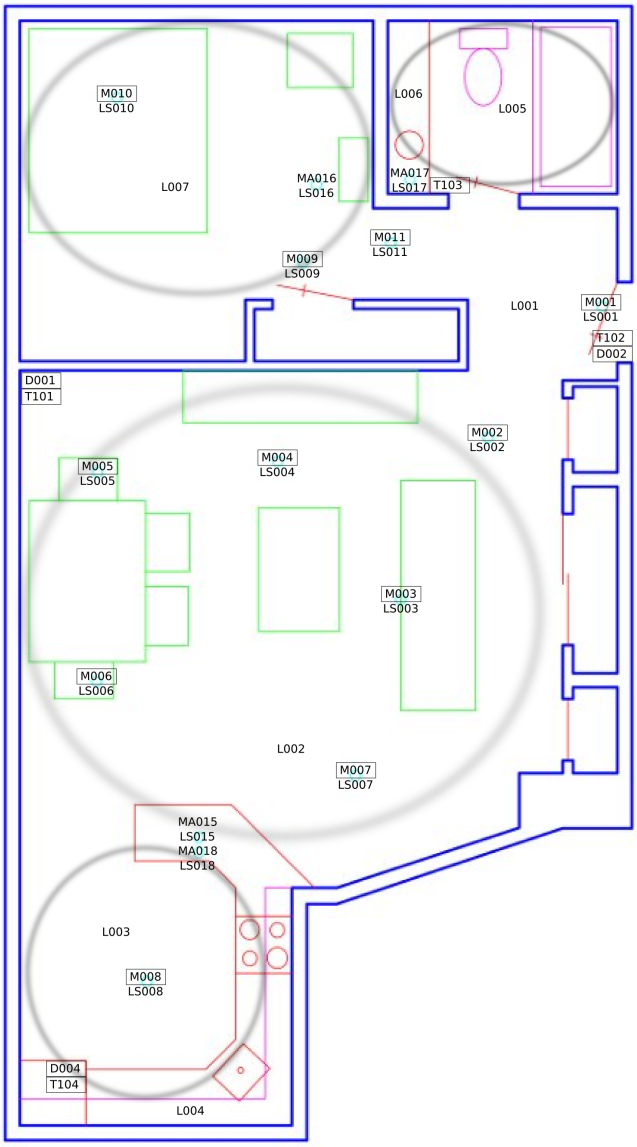
HH117


MA005
LS005
T106
L006
D005
D004
T105
M009
LS009
L003
M007
LS007
L004
M013
LS013
MA004
LS004
L005
MA016
LS016
MA018
LS018
D002
T102
M011
LS011
M014
LS014
M012
LS012
L001
M010
LS010
MA003
LS003
T104
L009
D003
T103
MA006
LS006
L010
MA002
LS002
MA001
LS001
MA008
LS008
L002
L007
MA017
LS017
MA015
LS015
L008
D001
T101

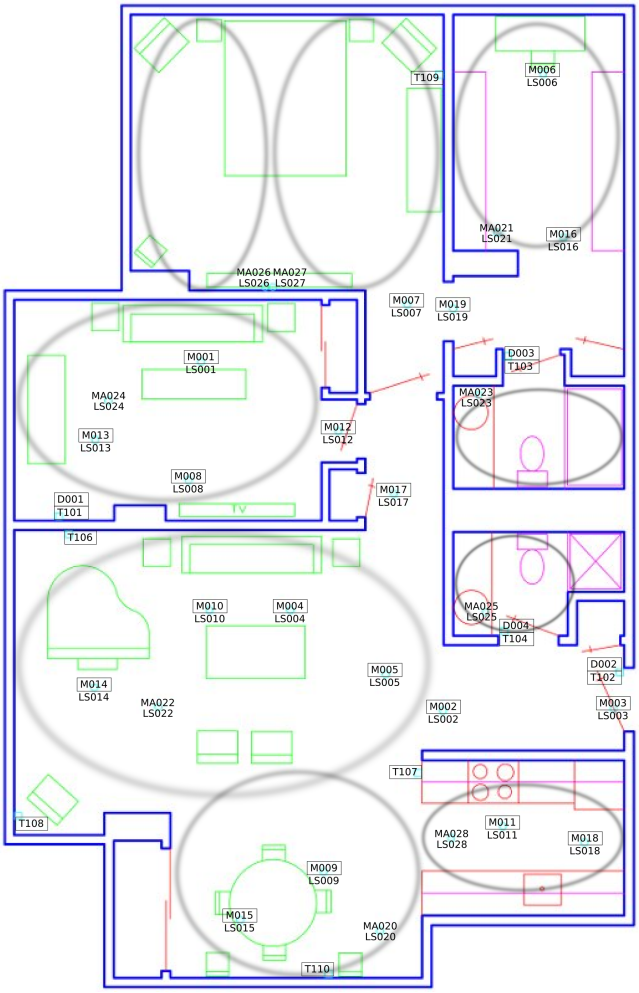
HH118


D002
T102
M003
LS003
MA006
LS006
T105
D005
D004
M002
LS002
T106
L006
L004
L001
L005
T103
MA010
LS010
M005
LS005
T104
M001
LS001
MA011
LS011
MA013
LS013
M004
LS004
MA007
LS007
L009
MA016
LS016
MA012
LS012
MA015
LS015
MA014
LS014
L010
D001
T101

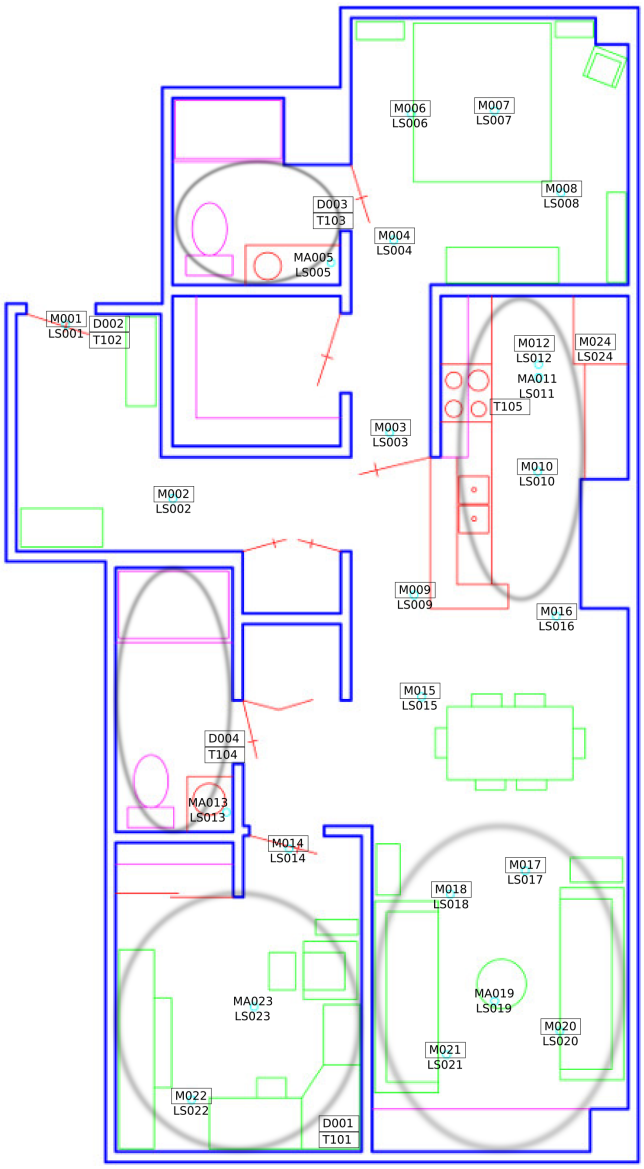
HH119


M010
LS010
L007
MA016
LS016
L006
L005
MA017
LS017
T103
M011
LS011
M009
LS009
L001
M001
LS001
T102
D002
D001
T101
M002
LS002
M004
LS004
M005
LS005
M003
LS003
M006
LS006
L002
M007
LS007
MA015
LS015
MA018
LS018
L003
M008
LS008
D004
T104
L004

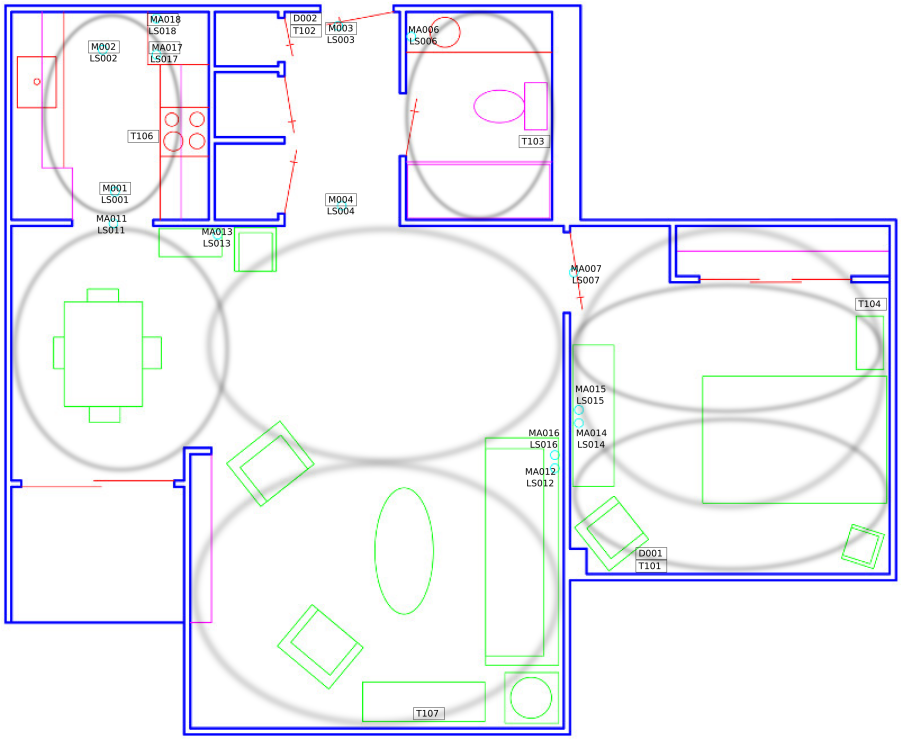
HH120


T109
M006
LS006
MA021
LS021
M016
LS016
MA026 MA027
LS026 LS027
M007
LS007
M019
LS019
M001
LS001
D003
T103
MA024
LS024
MA023
LS023
M013
LS013
M012
LS012
M008
LS008
M017
LS017
D001
T101
T106
M010
LS010
M004
LS004
MA025
LS025
D004
T104
D002
T102
M014
LS014
M005
LS005
MA022
LS022
M002
LS002
M003
LS003
T107
T108
M011
LS011
MA028
LS028
M018
LS018
M009
LS009
M015
LS015
MA020
LS020
T110

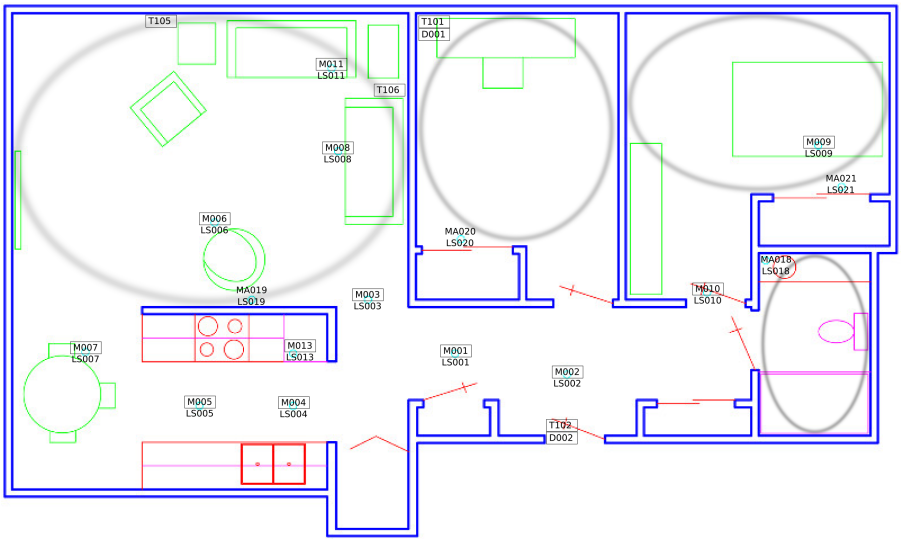
HH121


M006
LS006
M007
LS007
M008
LS008
D003
T103
M004
LS004
MA005
LS005
M001
LS001
D002
T102
M012
LS012
M024
LS024
MA011
LS011
T105
M003
LS003
M010
LS010
M002
LS002
M009
LS009
M016
LS016
M015
LS015
D004
T104
MA013
LS013
M014
LS014
M017
LS017
M018
LS018
MA023
LS023
MA019
LS019
M020
LS020
M021
LS021
M022
LS022
D001
T101

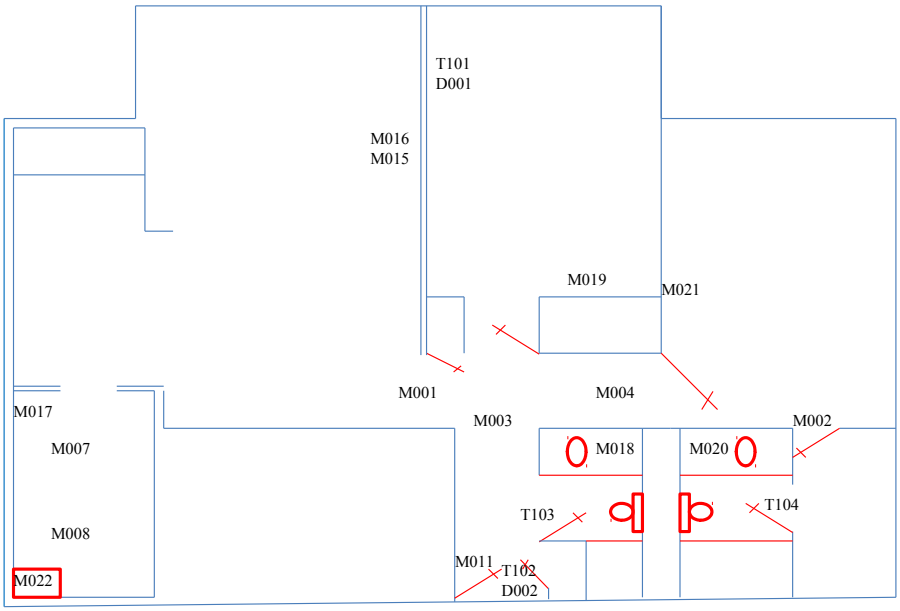
HH122


MA018
LS018
D002
T102
M003
LS003
MA006
LS006
M002
LS002
MA017
LS017
T106
T103
M001
LS001
M004
LS004
MA011
LS011
MA013
LS013
MA007
LS007
T104
MA015
LS015
MA016
LS016
MA014
LS014
MA012
LS012
D001
T101
T107

HH123


T105
T101
D001
M011
LS011
T106
M009
LS009
M008
LS008
MA021
LS021
M006
LS006
MA020
LS020
MA018
LS018
MA019
LS019
M003
LS003
M010
LS010
M007
LS007
M013
LS013
M001
LS001
M002
LS002
M005
LS005
M004
LS004
T102
D002

HH124


T101
D001
M016
M015
M019
M021
M001
M004
M017
M003
M002
M007
M018
M020
T103
T104
M008
M011
T102
D002
M022

HH125

REFERENCES

[1] Diane J. Cook and S.K. Das. *Smart environments: technology, protocols, and applications*. John Wiley & Sons, Hoboken, New Jersey, 2005.

[2] Grayson K. Vincent and Victoria A. Velkoff. The Next Four Decades - The Older Population in the United States : 2010 to 2050. Technical Report May, 2010.

[3] Jane Bates, Jonathan Boote, and Catherine Beverley. Psychosocial interventions for people with a milder dementing illness: a systematic review. *Journal of advanced nursing*, 45(6):644–58, March 2004.

[4] Virginia G. Wadley, Ozioma Okonkwo, Michael Crowe, and Lesley A. Ross-Meadows. Mild Cognitive Impairment and Everyday Function : Evidence of Reduced Speed in Daily Living. *American Journal of Geriatric Psychiatry*, 15(5):416–424, 2008.

[5] J Ziv and A Lempel. Compression of individual sequences by variable rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

[6] Meir Feder, Neri Merhav, and Michael Gutman. Universal Prediction of Individual Sequences. *IEEE Transactions on Information Theory*, 38(4):1258–1270, 1992.

[7] Amiya Bhattacharya and Sajal K. Das. LeZi-update: An information-theoretic framework for personal mobility tracking in PCS networks. *Wireless Networks*, 8(2-3):121–135, 2002.

[8] Diane J Cook, Manfred Huber, Karthik Gopalratnam, and Michael Youngblood. Learning to Control a Smart Home Environment. In *Innovative applications of artificial intelligence*, 2003.

[9] S.K. Das and D.J. Cook. Health monitoring in an agent-based smart home by activity prediction. In *Toward a human-friendly assistive environment: ICOST'2004, 2nd International Conference on Smart Homes and Health Telematics*, pages 3–14, 2004.

[10] Karthik Gopalratnam and Diane Cook. Online Sequential Prediction via Incremental Parsing: The Active LeZi Algorithm. *IEEE Intelligent Systems*, 22(1):52–58, January 2007.

[11] MR Alam, MBI Reaz, and M Ali. SPEED: An inhabitant activity prediction algorithm for smart homes. *Systems, Man and Cybernetics, ...*, 42(4):985–990, 2012.

[12] Vikramaditya R. Jakkula and Diane J Cook. Using Temporal Relations in Smart Environment Data for Activity Prediction. In *International Conference on Machine Learning*, pages 20–24, 2007.

[13] Ron Begleiter, Ran El-Yaniv, and Golan Yona. On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004.

[14] Ehsan Nazerfard and Diane J. Cook. Bayesian Networks Structure Learning for Activity Prediction in Smart Homes. In *2012 Eighth International Conference on Intelligent Environments*, pages 50–56. Ieee, June 2012.

[15] Kp Hawkins and N Vo. Probabilistic Human Action Prediction and Wait-sensitive Planning for Responsive Human-robot Collaboration. In *013 13th IEEE-RAS International Conference on Humanoid Robots*, pages 499–506, 2013.

[16] Kris M Kitani, Brian D Ziebart, James Andrew Bagnell, and Martial Hebert. Activity Forecasting. In *Proceedings of the European Conference on Computer Vision*, pages 201–214, 2012.

[17] Hema Swetha Koppula and Ashutosh Saxena. Anticipating human activities for reactive robotic response. In *IEEE International Conference on Intelligent Robots and Systems*, 2013.

[18] Moshe Lichman and Padhraic Smyth. Modeling human location data with mixtures of kernel densities. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 35–44, 2014.

[19] a. Gunawardana, C. Meek, and P. Xu. A Model for Temporal Dependencies in Event Streams. *Advances in Neural Information Processing Systems*, pages 1962–1970, 2011.

[20] Jan G. De Gooijer and Rob J. Hyndman. 25 Years of Time Series Forecasting. *International Journal of Forecasting*, 22(3):443–473, January 2006.

[21] Reza Reyhani and Amir masud eftekhari Moghadam. A heuristic method for forecasting chaotic time series based on economic variables. In *Sixth International Conference on Digital Information Management*, pages 300–304, 2011.

[22] Aranildo R Lima Junior, David A Silva, Paulo S Mattos Neto, and Tiago A. E. Ferreira. An Experimental Study of Fitness Function and Time. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pages 2015–2018, 2010.

[23] Ricardo de A. Araujo, Aranildo R. L. Junior, and Tiago A. E. Ferreira. Morphological-Rank-Linear Time-lag Added Evolutionary Forecasting Method for Financial Time Series Forecasting. In *2008 IEEE Congress on Evolutionary Computation*, 2008.

[24] G. E. P. Box and G. M. Jenkins. *Time series analysis: Forecasting and control*. 1976.

[25] Keith W. Hipel. *Time Series Modeling of Water Resources and Environmental Systems*. 1994.

[26] Jae H. Kim. Forecasting autoregressive time series with bias-corrected parameter estimators. *International Journal of Forecasting*, 19(3):493–502, 2003.

[27] Arnold Zellner. *An introduction to Bayesian inference in econometrics*. 1971.

[28] Robert F. Engle. Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation. *Econometrica*, 50(4):987–1007, 1982.

[29] Syed Rahat Abbas and Muhammad Arif. Competitive Neural Network Based Algorithm for Long Range Time Series Forecasting Case Study : Electric Load Forecasting. In *International Multitopic Conference, IEEE*, 2005.

[30] Georges a. Darbellay and Marek Slama. Forecasting the short-term demand for electricity: Do neural networks stand a better chance? *International Journal of Forecasting*, 16(1):71–83, 2000.

[31] M. H. Quenouille. *The analysis of multiple time-series. 2nd Edition.* 2nd editio edition, 1968.

[32] Trond Riise and Dag Tjozstheim. Theory and practice of multivariate ARMA forecasting. *Journal of Forecasting*, 3(3):309–317, 1984.

[33] Yong Ding, Martin Alexander, Per Goncalves Da Silva, and Michael Beigl. A Framework for Short-Term Activity-Aware Load Forecasting Categories and Subject Descriptors. In *Joint Proceedings of the Workshop on AI Problems and Approaches for Intelligent Environments and Workshop on Semantic Cities*, pages 23–28, 2013.

[34] N I U Dongxiao, S H I Hui, L I Jianqing, and W E I Yanan. Research on Short-term Power Load Time Series Forecasting model Based on BP Neural Network. pages 509–512, 2010.

[35] Chien-Yu Kuo, Ming-Feng Lee, Chia-Lin Fu, Yao-Hua Ho, and Ling-Jyh Chen. An In-depth Study of Forecasting Household Electricity Demand using Realistic Datasets. In *Proceedings of the 5th international conference on Future energy systems*, pages 145–155, 2014.

[36] Nagender Kumar Suryadevara and Subhas Chandra Mukhopadhyay. Wireless Sensor Network Based Home Monitoring System for Wellness Determination of Elderly. *IEEE SENSORS JOURNAL*, 12(6):1965–1972, 2012.

[37] N. K. Suryadevara, S. C. Mukhopadhyay, R. Wang, and R. K. Rayudu. Forecasting the behavior of an elderly using wireless sensors data in a smart home. *Engineering Applications of Artificial Intelligence*, 26(10):2641–2652, 2013.

[38] N. K. Suryadevara, S. C. Mukhopadhyay, and R. K. Rayudu. Applying SARIMA time series to forecast sleeping activity for wellness model of elderly monitoring in smart home. In *International Conference on Sensing Technology*, pages 157–162, 2012.

[39] N.K. Suryadevara, S.C. Mukhopadhyay, R. Wang, R.K. Rayudu, and Y.M. Huang. Reliable measurement of wireless sensor network data for forecasting wellness of elderly at smart home. In *Instrumentation and Measurement Technology Conference*, pages 16–21, 2013.

[40] N. K. Suryadevara, a. Gaddam, R. K. Rayudu, and S. C. Mukhopadhyay. Wireless sensors network based safe home to care elderly people: Behaviour detection. *Sensors and Actuators, A: Physical*, 186:277–283, 2012.

[41] Michael C Mozer. Lessons from an Adaptive Home. In Diane J. Cook and Sajal K. Das, editors, *Smart Environments*, pages 271–294. John Wiley & Sons, 2005.

[42] Mohamed Tarik Moutacalli, Kevin Bouchard, Abdenour Bouzouane, and Bruno Bouchard. Activity Prediction Based on Time Series Forcasting. In *AAAI Workshop on Artificial Intelligence Applied to Assistive Technologies and Smart Environments*, pages 32–37, 2014.

[43] Nicholas Navaroli, Christopher Dubois, and Padhraic Smyth. Modeling individual email patterns over time with latent variable models. *Machine Learning*, 92(2-3):431–455, 2013.

[44] Dq Vu and Au Asuncion. Continuous-time regression models for longitudinal networks. *Advances in Neural Information Processing Systems*, 2011.

[45] Ngoc Cuong Truong, James McInerney, Long Tran-Thanh, Enrico Costanza, and Sarvapali D. Ramchurn. Forecasting multi-appliance usage for smart home energy management. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 2908–2914, 2013.

[46] Javier Gil-Quijano and Nicolas Sabouret. Prediction of humans' activity for learning the behaviors of electrical appliances in an intelligent ambient environment. In *Proceedings - 2010 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2010*, volume 2, pages 283–286, 2010.

[47] a. Barbato, A. Capone, M. Rodolfi, and D. Tagliaferri. Forecasting the usage of household appliances through power meter sensors for demand management in the smart grid. In *2011 IEEE International Conference on Smart Grid Communications, SmartGridComm 2011*, pages 404–409, 2011.

[48] Myungeun Lim, Jaehun Choi, Daehee Kim, and Soojun Park. A Smart Medication Prompting System and Context Reasoning in Home Environments. In *2008 Fourth International Conference on Networked Computing and Advanced Information Management*, volume 1, pages 115–118. IEEE, September 2008.

[49] Martha E. Pollack, Laura Brown, Dirk Colbry, Colleen McCarthy, Cheryl Orosz, Bart Peintner, Sailesh Ramkrishnan, and Ioannis Tsamardinos. Autominder: An intelligent cognitive orthotic system for people with memory impairment. *Robotics and Autonomous Systems*, 44(3-4):1–10, 2003.

[50] Matthew Rudary, S Singh, and ME Pollack. Adaptive cognitive orthotics: combining reinforcement learning and constraint-based temporal reasoning. In *Procedings of the International conference on Machine learning*, pages 91–98, 2004.

[51] Joelle Pineau, Michael Montemerlo, Martha Pollack, Nicholas Roy, and Sebastian Thrun. Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, 42(3-4):271–281, March 2003.

[52] Jennifer Boger, Pascal Poupart, Jesse Hoey, Craig Boutilier, Geoff Fernie, and Alex Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *Proceedings of the 19th international joint conference on Artificial intelligence*, pages 1293–1299, 2005.

[53] Barnan Das, Diane J. Cook, Maureen Schmitter-Edgecombe, and Adriana M. Seelye. PUCK: an automated prompting system for smart environments: toward achieving automated promptingchallenges involved. *Personal and Ubiquitous Computing*, 16(7):859–873, September 2011.

[54] Lawrence B Holder and Diane J Cook. Automated activity-aware prompting for activity initiation. *Gerontechnology*, 11(4):534–544, January 2013.

[55] John Lafferty, Andrew Mccallum, and Fernando C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of International Conference on Machine Learning*, pages 282–289, 2001.

[56] Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine Learning*, 75(3):297–325, 2009.

[57] Alan Fern, Sungwook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *Journal of Artificial Intelligence Research*, 25:75–118, 2006.

[58] R Khardon. Learning to Take Actions. *Machine Learning Journal*, 35(1):57–90, 1999.

[59] Stéphane Ross and J Andrew Bagnell. Efficient Reductions for Imitation Learning. *Journal of Machine Learning Research - Proceedings Track*, 9:661–668, 2010.

[60] Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *Journal of Machine Learning Research - Proceedings Track*, 15:627–635, 2011.

[61] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Hc-search: A learning framework for search-based structured prediction. *Journal of Artificial Intelligence Research (JAIR)*, 50:369–407, 2014.

[62] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Structured Prediction via Output Space Search. *Journal of Machine Learning Research*, 14:1317–1350, 2014.

[63] Jun Xie, Chao Ma, Janardhan Rao Doppa, Prashanth Mannem, Xiaoli Fern, Tom Dietterich, and Prasad Tadepalli. Learning Greedy Policies for the Easy-First Framework. In *2AAAI Conference on Artificial Intelligence*, pages 2339–2345, 2015.

[64] Chao Ma, Janardhan Rao Doppa, J Walker Orr, Prashanth Mannem, Xiaoli Fern, Tom Dietterich, and Prasad Tadepalli. Prune-and-Score : Learning for Greedy Coreference Resolution. In *EMNLP2*, 2014.

[65] Michael Lam, Janardhan Rao Doppa, Sinisa Todorovic, and Thomas G Dietterich. HC -Search for Structured Prediction in Computer Vision. In *CVPR*, 2015.

[66] Michael Lam, Janardhan Rao Doppa, Xu Hu, Sinisa Todorovic, Thomas Dietterich, Abigail Reft, and Marymegan Daly. Learning to detect basal tubules of nematocysts in SEM images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 190–196, 2013.

[67] Matti Kääriäinen. Lower Bounds for Reductions. *Atomic Learning Workshop*, 2006.

[68] Diane Cook and Narayanan C. Krishnan. *Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data*. John Wiley & Sons, Hoboken, New Jersey, 2015.

[69] J.C.Van HOuwelingen S. Le Cessie. Ridge Estimators in Logical Regression. *Applied Statistics*, 41(1):191–201, 1992.

[70] S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R K Murthy. Improvements to the SMO algorithm for SVM regression. *IEEE Transactions on Neural Networks*, 11(5):1188–1193, 2000.

[71] Alex Smola and Bernhard Schölkopf. A tutorial on support vector regression, 1998.

[72] J R Quinlan. Learning with Continuous Classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.

[73] Y Wang and Ian H. Witten. Induction of model trees for predicting continuous classes. In *Proceedings of the 9th European Conference on Machine Learning Poster Papers*, pages 128–137, 1997.

[74] Janardhan Rao Doppa, Jun Yu, Chao Ma, Alan Fern, and Prasad Tadepalli. HC -Search for Multi-Label Prediction : An Empirical Study. *AAAI*, 2014.

[75] Mark Hall, Hazeltine National, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA Data Mining Software : An Update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[76] Yoav Freund and Re Robert E Schapire. Experiments with a New Boosting Algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[77] C. W. Dawson, R. J. Abrahart, and L. M. See. HydroTest: A web-based toolbox of evaluation metrics for the standardised assessment of hydrological forecasts. *Environmental Modelling and Software*, 22(7):1034–1052, 2007.

[78] Asela Gunawardana and Guy Shani. A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. *The Journal of Machine Learning Research*, 10:2935–2962, 2009.

[79] Luis Torgo and Rita Ribeiro. Precision and recall for regression. *Lecture Notes in Computer Science*, 5808:332–346, 2009.

[80] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit. A simulation study of the model evaluation criterion MMRE. *IEEE Transactions on Software Engineering*, 29(11):985–995, 2003.

[81] K L Gwet. *Handbook of Inter-Rater Reliability*. Advanced Analytics, LLC, 2014.

[82] Diane J. Cook, Aaron S. Crandall, Brian L. Thomas, and Narayanan C. Krishnan. CASAS: A smart home in a box. *Computer*, 46(7):62–69, 2013.

[83] Brian L. Thomas and Aaron S. Crandall. A demonstration of PyViz, a flexible smart home visualization tool. In *2011 IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 304–306, 2011.

[84] Diane J. Cook, Narayanan C. Krishnan, and Parisa Rashidi. Activity discovery and activity recognition: A new partnership. *IEEE Transactions on Cybernetics*, 43(3):820–828, 2013.

[85] Eric Guenterberg, Hassan Ghasemzadeh, and Roozbeh Jafari. Automatic Segmentation and Recognition in Body Sensor Networks Using a Hidden Markov Model, 2012.

[86] Lin Liao, Dieter Fox, and Henry Kautz. Location-Based Activity Recognition using Relational Markov Networks. In *Proceedings of the Nineteenth International Conference on Artificial Intelligence, IJCAI'05*, pages 773–778, 2005.

[87] Jose Antonio Iglesias, Plamen Angelov, Agapito Ledezma, and Araceli Sanchis. Human activity recognition based on Evolving Fuzzy Systems. *International journal of neural systems*, 20(5):355–364, 2010.

[88] Jonathan Alon, Vassilis Athitsos, Quan Yuan, and Stan Sclaroff. A unified framework for gesture recognition and spatiotemporal gesture segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(9):1685–1699, 2009.

[89] Pavan Turaga, Rama Chellappa, V. S. Subrahmanian, and Octavian Udrea. Machine recognition of human activities: A survey. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(11):1473–1488, 2008.

[90] Oscar D. Lara and Miguel A. Labrador. A Survey on Human Activity Recognition using Wearable Sensors. *IEEE Communications Surveys & Tutorials*, 15(3):1192–1209, 2013.

[91] Sarvesh Vishwakarma and Anupam Agrawal. A survey on activity recognition and behavior understanding in video surveillance. In *Visual Computer*, volume 29, pages 983–1009, 2013.

[92] Andreas Bulling, Ulf Blanke, and Bernt Schiele. A tutorial on human activity recognition using body-worn inertial sensors. *ACM Computing Surveys (CSUR)*, 46(3):1–33, 2014.

[93] Shian Ru Ke, Hoang Thuc, Yong Jin Lee, Jenq Neng Hwang, Jang Hee Yoo, and Kyoung Ho Choi. A Review on Video-Based Human Activity Recognition. *Computers*, 2:88–131, 2013.

[94] Liming Chen and Ismail Khalil. Activity Recognition: Approaches, Practices and Trends. In *Activity Recognition in Pervasive Intelligent Environments*, volume 4, pages 1–31. 2011.

[95] Liming Chen, Jesse Hoey, Chris D. Nugent, Diane J. Cook, and Zhiwen Yu. Sensor-based activity recognition, 2012.

[96] JK Aggarwal and MS Ryoo. Human activity analysis: A review. *ACM Computing Surveys (CSUR)*, 43(3):16:1–16:43, 2011.

[97] Attila Reiss, Gustaf Hendeby, and Didier Stricker. Towards Robust Activity Recognition for Everyday Life : Methods and Evaluation. *Proceedings of 7th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, (May):25–32, 2013.

[98] S. Seth Long and Lawrence B. Holder. Using graphs to improve activity prediction in smart environments based on motion sensor data. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6719 LNCS:57–64, 2011.

[99] Diane J. Cook. Learning setting-generalized activity models for smart spaces. *IEEE Intelligent Systems*, 27(1):32–38, 2012.

[100] H Hagras, F Doctor, V Callaghan, and A Lopez. An incremental adaptive life long learning approach for type-2 fuzzy embedded agents in ambient intelligent environments. *IEEE Transactions on Fuzzy Systems*, 15(1):41–55, 2007.

[101] Emmanuel Munguia Tapia, Stephen S Intille, and Kent Larson. Activity Recognition in the Home Using Simple and Ubiquitous Sensors. *Pervasive Computing*, 3001:158–175, 2004.

[102] Holger Junker, Oliver Amft, Paul Lukowicz, and Gerhard Tröster. Gesture spotting with body-worn inertial sensors to detect user activities. *Pattern Recognition*, 41(6):2010–2024, 2008.

[103] U Maurer, A Smailagic, D P Siewiorek, and M Deisher. Activity recognition and monitoring using multiple sensors on different body positions. In *Proceedings - BSN 2006: International Workshop on Wearable and Implantable Body Sensor Networks*, volume 2006, pages 113–116, 2006.

[104] Ay Yang, Roozbeh Jafari, Ss Sastry, and Ruzena Bajcsy. Distributed recognition of human actions using wearable motion sensor networks. *Journal of Ambient Intelligence and Smart Environments*, 1:1–5, 2009.

[105] Liang Wang, Tao Gu, Xianping Tao, Hanhua Chen, and Jian Lu. Recognizing multi-user activities using wearable sensors in a smart home. *Pervasive and Mobile Computing*, 7(3):287–298, 2011.

[106] Tao Gu, Shaxun Chen, Xianping Tao, and Jian Lu. An unsupervised approach to activity recognition and segmentation based on object-use fingerprints. *Data and Knowledge Engineering*, 69(6):533–544, 2010.

[107] Matthai Philipose, Kenneth P. Fishkin, Mike Perkowitz, Donald J. Patterson, Dieter Fox, Henry Kautz, and Dirk Hähnel. Inferring activities from interactions with objects, 2004.

[108] Joshua Candamo, Matthew Shreve, Dmitry B. Goldgof, Deborah B. Sapper, and Rangachar Kasturi. Understanding transit scenes: A survey on human behavior-recognition algorithms. *IEEE Transactions on Intelligent Transportation Systems*, 11(1):206–224, 2010.

[109] Oliver Brdiczka, James L. Crowley, and Patrick Reignier. Learning situation models in a smart home. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(1):56–63, 2009.

[110] M. S. Ryoo. Human activity prediction: Early recognition of ongoing activities from streaming videos. *Proceedings of the IEEE International Conference on Computer Vision*, (Iccv):1036–1043, 2011.

[111] Gang Yu, Junsong Yuan, and Zicheng Liu. Predicting human activities using spatio-temporal structure of interest points. *Proceedings of the 20th ACM international conference on Multimedia - MM '12*, page 1049, 2012.

[112] J R Kwapisz, G M Weiss, and S A Moore. Activity Recognition using Cell Phone Accelerometers. In *Proceedings of the Fourth International Workshop on Knowledge Discovery from Sensor Data*, pages 10–18, 2010.

[113] Norbert Gyrbíró, Ákos Fábián, and Gergely Hományi. An Activity Recognition System For Mobile Phones. *Mobile Networks and Applications*, 14(1):82–91, 2008.

[114] Qinran Hu and Fangxing Li. Hardware design of smart home energy management system with dynamic price response. *IEEE Transactions on Smart Grid*, 4(4):1878–1887, 2013.

[115] Carlos Fernandez, Juan-Pablo Lázaro, and José Miguel Benedí. Workflow Mining Application to Ambient Intelligence Behaviour Modelling. pages 160–167, 2009.

[116] Kilian Förster, Samuel Monteleone, Alberto Calatroni, Daniel Roggen, and Gerhard Tröster. Incremental kNN classifier exploiting correct-error teacher for activity recognition. In *Proceedings - 9th International Conference on Machine Learning and Applications, ICMLA 2010*, pages 445–450, 2010.

[117] Oliver Amft and Gerhard Tröster. On-body sensing solutions for automatic dietary monitoring. *IEEE Pervasive Computing*, 8(2):62–70, 2009.

[118] Mi Zhang and AA Sawchuk. Motion primitive-based human activity recognition using a bag-of-features approach. In *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium*, number 1, page 631, 2012.

[119] Ling Bao and Stephen S. Intille. *Activity Recognition from User-Annotated Acceleration Data*. 2004.

[120] Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Ml Littman. Activity recognition from accelerometer data. In *Proceedings of the 17th conference on Innovative applications of artificial intelligence*, pages 1541–1546, 2005.

[121] Jamie A. Ward, Paul Lukowicz, Gerhard Tröster, and Thad E. Starner. Activity recognition of assembly tasks using body-worn microphones and accelerometers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1553–1566, 2006.

[122] Geetika Singla, Diane J. Cook, and Maureen Schmitter-Edgecombe. Recognizing independent and joint activities among multiple residents in smart environments. *Journal of Ambient Intelligence and Humanized Computing*, 1(1):57–63, 2010.

[123] Jonathan Lester, Tanzeem Choudhury, Nicky Kern, Gaetano Borriello, and Blake Hannaford. A hybrid discriminative/generative approach for modeling human activities. In *Proc of the International Joint Conference on Artificial Intelligence IJCAI*, pages 766–772, 2005.

[124] Ulf Blanke, Bernt Schiele, Matthias Kreil, Paul Lukowicz, Thiemo Gruber, and Bernhard Sick. All for one or one for all? Combining heterogeneous features for activity spotting. In *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops, PERCOM Workshops 2010*, pages 18–24, 2010.

[125] Tim Van Kasteren, Athanasios Noulas, Gwenn Englebienne, and Ben Kr. Accurate Activity Recognition in a Home Setting. In *UbiComp '08 Proceedings of the 10th international conference on Ubiquitous computing*, pages 1–9, 2008.

[126] Andreas Bulling, Jamie A. Ward, and Hans Gellerson. Multimodal recognition of reading activity in transit using body-worn sensors. *ACM Transactions on Applied Perception*, 9(1):2:1–2:21, 2012.

[127] Jonathan Lester, Tanzeem Choudhury, and Gaetano Borriello. A Practical Approach to Recognizing Physical Activities. *Pervasive Computing*, 3968:1–16, 2006.

[128] Shiaokai Wang, William Pentney, Ana-Maria Popescu, Tanzeem Choudhury, and Matthai Philipose. Common sense based joint training of human activity recognizers. In *Proceedings of the 20th international joint conference on Artificial intelligence*, pages 2237–2242, 2007.

[129] Aaron S. Crandall and Diane J. Cook. *Behaviometrics for Identifying Smart Home Residents*. 2013.

[130] Feng Niu and M Abdel-Mottaleb. HMM-Based Segmentation and Recognition of Human Activities from Video Sequences. In *2005 IEEE International Conference on Multimedia and Expo*, pages 804–807, 2005.

[131] Olivier Duchenne, Ivan Laptev, Josef Sivic, Francis Bach, and Jean Ponce. Automatic annotation of human activities in video. In *International Conference on Computer Vision*, pages 1491–1498, 2009.

[132] Yonglei Zheng, Weng-keen Wong, Xinze Guan, and Stewart Trost. Physical Activity Recognition from Accelerometer Data Using a Multi-Scale Ensemble Method. In *Proceedings of the Twenty-Fifth Innovative Applications of Artificial Intelligence Conference*, pages 1575–1581, 2010.

[133] Xin Hong and Chris D. Nugent. Segmenting sensor data for activity monitoring in smart environments. *Personal and Ubiquitous Computing*, 17(3):545–559, 2013.

[134] Paulito Palmes, Hung Keng Pung, Tao Gu, Wenwei Xue, and Shaxun Chen. Object relevance weight pattern mining for activity recognition and segmentation. *Pervasive and Mobile Computing*, 6(1):43–57, 2010.

[135] Toshihiko Yamasaki and Kiyoharu Aizawa. Motion segmentation and retrieval for 3D video based on modified shape distribution. *Eurasip Journal on Advances in Signal Processing*, 2007(1):211–211, 2007.

[136] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 289–296, 2001.

[137] Narayanan C. Krishnan and Diane J. Cook. Activity recognition on streaming sensor data. *Pervasive and Mobile Computing*, 10(PART B):138–154, 2014.

[138] Nuria M. Oliver, Barbara Rosario, and Alex P. Pentland. A bayesian computer vision system for modeling human interactions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):831–843, 2000.

[139] Tomoya Hirano and Takuya Maekawa. A Hybrid Unsupervised/Supervised Model for Group Activity Recognition. In *International symposium on wearable computers*, pages 21–24, 2013.

[140] Derek Hao Hu and Qiang Yang. CIGAR : Concurrent and Interleaving Goal and Activity Recognition. In *AAAI Conference on Artificial Intelligence*, pages 1363–1368, 2008.

[141] Hayley Hung, Gwenn Englebienne, and Jeroen Kools. Classifying social actions with a single accelerometer. In *ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 207–210, 2013.

[142] M.B. Kjaergaard, Martin Wirz, Daniel Roggen, and Gerhard Troster. Detecting pedestrian flocks by fusion of multi-modal sensors in mobile phones. In *ACM Conference on Ubiquitous Computing*, pages 240–249, 2012.

[143] Yi Ting Chiang, Kuo Chung Hsu, Ching Hu Lu, Li Chen Fu, and Jane Yung Jen Hsu. Interaction models for multiple-resident activity recognition in a smart home. In *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pages 3753–3758, 2010.

[144] Ching-hu Lu and Yi-ting Chiang. Interaction-Feature Enhanced Multiuser Model Learning for a Home Environment Using Ambient Sensors. *International Journal of Intelligent Systems*, 29(11):1015–1046, 2014.

[145] Tsu-yu Wu, Chia-chun Lian, and Jane Yung-jen Hsu. Joint recognition of multiple concurrent activities using factorial conditional random fields. In *AAAI Workshop on Plan, Activity, and Intent Recognition*, pages 82–87, 2007.

[146] Andrei Tolstikov, Clifton Phua, Jit Biswas, and Weimin Huang. Multiple people activity recognition using MHT over DBN. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6719 LNCS, pages 313–318, 2011.

[147] Liang Wang, Tao Gu, Xianping Tao, Hanhua Chen, and Jian Lu. Multi-User Activity Recognition in a Smart Home. In *Activity Recognition in Pervasive Intelligent Environments*, volume 4, pages 59–81. 2011.

[148] Tao Gu, Liang Wang, Hanhua Chen, Xianping Tao, and Jian Lu. Recognizing multiuser activities using wireless body sensor networks. *IEEE Transactions on Mobile Computing*, 10(11):1618–1631, 2011.

[149] Mikkel Baun Kjaergaard. Studying sensing-based systems: Scaling to human crowds in the real world. *IEEE Internet Computing*, 17(5):80–84, 2013.

[150] Johanna Petersen, Nicole Larimer, Jeffrey A. Kaye, Misha Pavel, and Tamara L. Hayes. SVM to detect the presence of visitors in a smart home environment. In *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, pages 5850–5853, 2012.

[151] Dawud Gordon, Jan-Hendrik Hanne, Martin Berchtold, Ali Asghar Nazari Shirehjini, and Michael Beigl. Towards collaborative group activity recognition using mobile devices. *Mobile Networks and Applications*, 18(3):326–340, 2013.

[152] J R Quinlan. *C4.5: Programs for Machine Learning*. 1993.

[153] N. Epstein, M. G. Willis, C. K. Conners, and D. E. Johnson. Use of technological prompting device to aid a student with attention deficit hyperactivity disorder to initiate and complete daily activities: An exploratory study. *Journal of Special Education Technology*, 16(1):19–28, 2001.

[154] M. Schmitter-Edgecombe, J. T. Howard, S. Pavawalla, L. Howeel, and A. Rueda. Multidyad memory notebook intervention for very mild dementia: A pilot study. *American Journal of Alzheimers Disease and Other Dementias*, 23(5):477–487, 2008.

[155] Pallavi Kaushik, S. S. Intille, and K. Larson. User-adaptive Reminders for Home-based Medical Tasks: A Case Study. *Methods of Information in Medicine*, 47(3):203–207, 2008.

[156] Kristin E Heron and Joshua M Smyth. Ecological momentary interventions: incorporating mobile technology into psychosocial and health behaviour treatments. *British journal of health psychology*, 15(Pt 1):1–39, 2010.