Forecasting Occurrences of Activities

Bryan Minor
Washington State University
Pullman, WA 99203 USA
bminor@eecs.wsu.edu

Diane J. Cook
Washington State University
Pullman, WA 99203 USA
cook@eecs.wsu.edu

Abstract

While activity recognition has been shown to be valuable for pervasive computing applications, less work has focused on techniques for forecasting the future occurrence of activities. We present an activity forecasting method to predict the time that will elapse until a target activity occurs. This method generates an activity forecast using a regression tree classifier and offers an advantage over sequence prediction methods in that it can predict expected time until an activity occurs. We evaluate this algorithm on real-world smart home datasets and provide evidence that our proposed approach is most effective at predicting activity timings.

Keywords

activity forecasting; activity recognition; regression trees; smart homes

1. Introduction

A significant component of any smart assistive technology is the ability to predict future occurrences of a user's activities based on current and past context. The ability of a system to forecast what activities the user will perform in the future allows the system to take anticipatory action to help the user complete those activities. Activity prediction can thus transform context-aware systems into *activity-predictive systems*.

One type of system that can benefit from the use of an activity prediction algorithm is the smart environment. A smart environment is a physical environment that is equipped to sense the state of its inhabitants and their surroundings in order to act to improve the wellbeing of the inhabitants and the environment [1,2]. Smart environments employ a variety of sensors to monitor their inhabitants and provide prompts and other actions to help complete those activities and provide for inhabitant safety.

In activity prediction, we desire to create a system which can forecast the number of time units (be it seconds, hours, or days) until future activities occur. The challenge lies in optimally using collected sensor data to inform activity forecasts without human intervention. Although the data can provide a wealth of information about user activities, it can be challenging to extract useful attributes to use in forecasting models. The relationship between sensor data and future activities is complex and nonlinear. Previous methods rely on user input and pre-defined rules, limiting their scope and flexibility for use in varied applications. An effective activity predictor needs to be able to discover and utilize deeper activity complexities and model them appropriately.

In this article, we introduce an activity forecasting algorithm that predicts future activity occurrences. Our algorithm, which we call AF, predicts the number of time units that will elapse until the next occurrence of a particular activity based on current and past sensor events. AF utilizes information about recent sensor events to generate both discrete-event and sampling-based features to provide improved prediction accuracy. These features can then be used with common machine learning models to learn the complex relationships between sensor data and

future activities in a straightforward way. Our approach provides new insight into activity forecasting by utilizing this combination of window-based feature extraction and machine learning. In the following sections, we discuss the implementation of this method and evaluate it using sensor and activity data gathered from smart homes.

2. Background and Related Work

Work on activity prediction follows two general approaches. The first approach is sequence prediction. In this approach, a predictor is trained using a sequence of observed symbols, $S=s_1,s_2,...,s_t$, and is then used to generate the symbol s_{t+1} that is likely to occur at time $t+1$. In the case of activity prediction, if activity information is available (e.g., from an expert observer or from an activity recognition algorithm) then the sequence predictor can output the most likely activity to occur next. Feder et al. adapted the LZ78 compression algorithms for use in sequential prediction [3,4]. The algorithms analyze the sequences of activities that occur and use Markov models to predict the next activity in the sequence. This method is further refined by the LeZi and Active LeZi algorithms [5,6]. A further development for application in smart homes is the SPEED algorithm, which utilizes the patterns of appliance usage in the home [7]. Other variable-order sequence prediction models have been examined by Begleiter et al. [8].

While these sequence prediction algorithms are able to predict the next activity that occurs in a sequence, they do not provide information about when that activity might occur. The algorithms simply predict the next activity in a sequence – there is no indication if this activity might occur in the next minute or in a few hours. Similarly, they do not provide information about target activities and their occurrence times when those activities are not in the short-term sequence horizon. For automation methods that are time-dependent, these can be important distinctions. In the smart home case, the next predicted event in a home automation system may be the resident's return home, but there is no indication of how much time will elapse until this activity occurs. This makes it more challenging for the system to determine when to start automation sequences in anticipation of their arrival.

In order to provide time-based forecasts of when each activity will occur, a second approach can be considered. This approach, called activity forecasting, involves predicting the time until an activity of interest will occur. Instead of determining a discrete classification for the next activity, activity forecasting methods determine a continuous-valued output prediction of the time until a particular activity occurs. This has been traditionally studied in the area of time series analysis. Until this point, it has not been explored in the context of activity prediction.

Popular time forecasting methods include autoregressive moving average (ARMA) and autoregressive integrated moving average (ARIMA) models [9]. While these models are relatively simple and have been widely discussed in the time series literature [10-12], they are based on the underlying time series being linear and stationary. These assumptions may not apply in the case of activity forecasting. The relationships between sensor data and activities are varied and cannot be readily decomposed into linear models. Furthermore, user behaviors tend to change over time as users experience life changes and adopt new routines, resulting in activity distributions which are not stationary.

Additionally, many time series models are designed for a univariate time series, where the previous values of the variable being forecasted are also used as inputs to the model. Since we do not precisely know the time an activity will occur until after it happens, this may be difficult

to apply for activity prediction. A multivariate model could be used instead [13,14]. These models incorporate time series values from a variety of variables, but are often limited to the same linearity and stationarity assumptions as univariate models. Our approach leverages feature generation and common machine learning models to build a system capable of forming activity predictions when these conditions do not hold. Nonlinear time series models, such as artificial neural networks [15] and autoregressive conditional heteroscedastic models [16], can sometimes be used. However, these methods can be difficult to train and use.

A variant on activity forecasting which is related to our work is the rule-based algorithm developed by Holder and Cook [2]. This method uses rules generated from the relationships between activities and their occurrence distributions. Activity predictions are made by using activity recognition to determine when a reference activity occurs so that the time to a desired activity can be predicted. They applied this algorithm to provide prompts in a smart home setting. While this model utilizes machine learning methods similar to ours, it relies upon the times between reference and forecasted activities to be relatively stable, which may not be the case as activity habits change. Furthermore, it uses only a limited scope of activity relationship information.

While the methods described above can be useful for activity prediction, they face some restrictions that leave room for improvement. Many of the time series methods are based on linear models that have difficulty modeling the more complex relationships between activities and current context. Rule-based approaches rely on the particular rule representation and discovery method that is used and will not be sensitive to more complex relationships between activities.

The proposed method is designed to overcome some of these challenges by including a forecasting model derived directly from training datasets. The combination of feature extraction and machine learning used in AF allows us to model the complex activity relationships without pre-defining activity rules. Further, by utilizing nonlinear models, we can overcome some limitations of linear methods.

Another key component in predicting sequences of activities is activity recognition. Not only does activity recognition inform the prediction algorithm of when the predicting activities actually do occur (thus providing both training data and feedback on predictive performance), but knowing when activities occurred in the past may be valuable for the activity forecasting algorithm.

Activity recognition algorithms label activities based on the sensor readings (or events) that are collected from the environment. The challenge of activity recognition is to map a sequence of sensor events, $x=e_1,e_2,...,e_n$, onto a value from a set of predefined activity labels, $a \in A$. Activity recognition can be viewed as a type of supervised machine learning problem. An activity recognition algorithm learns a function that maps a feature vector, X, describing a particular sensor event sequence onto an activity label, $h:X \rightarrow A$. The algorithm can then use the learned function to recognize and label occurrences of the learned activity. Our group and others have explored a large number of approaches to supervised activity recognition [17-31]. The learning methods can be broadly categorized into transductive, generative, discriminative, and ensemble approaches. Template matching techniques employ a kNN classifier to a fixed window size of

sensor data or with dynamic time warping to a varying window size [32]. Generative approaches such as naïve Bayes classifiers, Markov models and dynamic Bayes networks have yielded promising results for behavior modeling and offline activity recognition when a large amount of labeled data is available [33-40]. On the other hand, discriminative approaches that model the boundary between different activity classes offer an effective alternative. These techniques include decision trees, meta classifiers based on boosting and bagging, support vector machines, and discriminative probabilistic graphical models such as conditional random fields [33,41-43]. Other approaches combine these underlying learning algorithms, including boosting and other ensemble methods [35,44,45].

While many activity recognition algorithms have been proposed, they are typically designed for constrained situations with pre-segmented data, a single user, and no activity interruptions. Recent work has extended this to consider generalization of activity models over multiple users with real-time labeling. We have achieved >95% accuracy for 30 activities in (N=20) smart homes using our real-time AR activity recognition algorithm, which will be used as the activity recognition algorithm for our AF activity forecaster described in this paper. In order to avoid offline data segmentation that is used in other approaches [46-53], AR extracts features from a *sliding window* that moves over the data in real time as it is collected [54]. The window size dynamically adjusts to sensor readings based on likely current activities and their associated likely durations.

3. Methods

We introduce an activity forecasting algorithm, called AF. The AF algorithm can be applied to any type of sensor-based activity data. Previously, we described the basic forecasting approach and generated preliminary results on sample datasets [55]. In this paper, we describe an enhanced approach to AF that utilizes a richer set of feature descriptors and perform a comparative evaluation based on its performance as a component in the WSU CASAS smart home project. The CASAS smart home system is comprised of a wireless sensor network with a variety of sensors placed throughout the home. These sensors include motion, door, light, switch, and temperature sensors, among others. The sensor network monitors inhabitant activities within the home and sends sensor event information over a mesh network to be processed by the middleware and stored in a server database. The AF component uses these stored events to generate activity forecasts which can be used by other components in the system, such as prompt generation or energy management applications. AF provides a numeric prediction of the number of time units that will occur between any given sensor event and the next occurrence of an activity of interest.

The goal of AF is to map a feature vector, describing the current activity context and the state of the environment, onto a number representing the number of time units until a target activity will next occur. AF consists of two main components: the feature extraction component and the forecasting component. These are described in detail next.

Table 1: Example sensor event data from the CASAS smart home. The sensor message component can be either a discrete message (e.g. ON or OFF) or a numeric value (e.g. a light level value).

Date	Time	Sensor	Message
2012-07-20	11:36:25.77	M002	ON
2012-07-20	11:36:25.85	LS001	27
2012-07-20	11:36:26.89	M001	ON
2012-07-20	11:36:27.08	M002	OFF
2012-07-20	11:36:27.16	M003	ON
2012-07-20	11:36:29.71	LS001	36

3.1 Feature Extraction

The first step in the AF algorithm is to extract useful features from the raw CASAS sensor data. Sensor data stored in the database includes the date, time (at multiple resolutions), sensor ID, and sensor message for each event, as shown in Table 1. Some sensors, such as motion or door sensors, generate discrete-valued messages indicating their state (e.g. ON or OFF, OPEN or CLOSED). We call these *discrete* sensors. Other sensors, such as temperature and light sensors, generate messages which are continuous numeric values (e.g. the temperature in Celsius) and are referred to here as *sampling* sensors. Discrete sensors usually only generate events when there is a change in their state, typically as a result of direct interaction with the inhabitant, while sampling sensors will report values at a constant rate regardless of events that are (or are not) occurring in the home. By incorporating both types of sensors into AF's feature vector, a wider understanding of the inhabitant's activities can be gained and a more sensitive forecaster can be created. In addition, using both of these types of features allows us to apply activity forecasting to multiple types of sensor platforms, include smart phones, wearable sensors, and cameras in addition to smart home environment sensors by customizing them for the sensor types chosen. As a result, AF's features are divided into two types: discrete features and sampling features.

Table 2: The discrete features that are generated by the feature extraction component of AF. These features are based on a window of previous events and are combined with the sampling features as input to the regression tree classifier.

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	The most frequent sensor ID in the previous window
prevDominantSensor2	The most frequent sensor ID in the window before that
lastSensorID	The sensor ID of the current event
lastLocation	The sensor ID for the most recent *discrete sensor* event
sensorCount*	Number of times each sensor produced an event in the window
sensorElTime*	Time (seconds) since each sensor last produced an event
timeStamp	Time since beginning of the day (seconds) normalized by the total number of seconds in a day
lag*	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, the lag size is 12).**	

3.1.1 Discrete Features

The discrete features used in AF's feature vector describe the time and location where events occur in the home and are listed in Table 2. These features are generated using information found in the sliding window that contains the most recent sensor events, similar to the method described for the AR algorithm in the previous section. Temporal features (e.g., *lastSensorEventHours, timestamp, windowDuration*) are used to provide information on the time of day. This allows the classifier to take into account the fact that many activities occur during certain times of day. The *prevDominantSensor* features provide information about context leading up to the current window (in this case, the sensor generating the most events in the previous two sliding windows). Other features contain information about the location where an activity is taking place within the home based on the location of sensors that most recently generated events (e.g., *lastLocation*, *sensorCount*, and *sensorElTime*). The *lag* features describe the time of day at which previous sensor events occurred to provide a picture of the temporal patterns in the most recent sensor events.

Table 3: The sample features generated by the feature extraction component of AF. These features are based on a window of sample vectors, each containing the latest value of each sensor when the sample was taken. Each of these features is generated separately for each sensor using the sampled values for that sensor. Details and equations for these features are described in [56].

Sampling Feature Name	Description
sensorMaximumValue	The maximum value of the sensor
sensorMinimumValue	The minimum value of the sensor
sensorSum	The sum of the sensor's values
sensorMean	The arithmetic mean of the sensor's values
sensorMeanAbsDev	The mean absolute deviation of values from the arithmetic mean
sensorMedianAbsDev	The mean absolute deviation of values from the median
sensorStdDev	The standard deviation of the values
sensorCoeffVar	The coefficient of variation for the values
sensorNumZeroCross	The number of times the values cross the median in the window
sensor25thPercentile	The 25th percentile of the values
sensorSqSum25thPer	The sum of the squares of values less than the 25th percentile
sensor50thPercentile	The 50th percentile of the values
sensorSqSum50thPer	The sum of the squares of values less than the 50th percentile
sensor75thPercentile	The 75th percentile of the values
sensorSqSum75thPer	The sum of the squares of values less than the 75th percentile
sensorInterQuartRange	The difference between the 75th and 25th percentiles
sensorBinCount*	The number of values in each bin
sensorSkewness	The skewness of the values
sensorKurtosis	The kurtosis of the values
sensorSigEnergy	The signal energy (sum of squared values)
sensorLogSigEnergy	The logarithmic signal energy (sum of $\log_{10}$ of squared values)
sensorSigPower	The signal power (arithmetic mean of squared values)
sensorPeakToPeak	The difference of the maximum and minimum values
sensorAvgTimeBtwnPeaks	The average time between local maximum values
sensorNumPeaks	The number of local maximum values
***The values are binned into 10 bins divided equally across the range of values for each sensor**	

3.1.2 Sampling Features

Sampling features are generated from the values of each sensor sampled at given intervals. A sample interval and a sample lag are provided to the feature extraction component. Some sensors, such as accelerometers, are designed to provide readings at regular time intervals and thus can naturally be represented using sampling features. Other sensors, such as infrared motion detectors, do not provide readings at regular time intervals. As a result, researchers typically only represent these sensors using discrete features. There is a danger, however, on missing valuable information such as the delay between changes in sensor state, when we only employ discrete features. For these discrete sensors, we can generate sampling features by replicating the current state of the sensor at regular time intervals, thus simulating a sampling-type sensor. Because all of the sampling feature values are represented as numbers, discrete sensor values are mapped to binary numeric values (e.g., 1 for ON, 0 for OFF).

Similar to the case with discrete features, sampling features are created based on the sensor values contained within a recent sliding window of sensor sample vectors. AF's sampling features are listed in Table 3. All of the sampling features are generated for each sensor. These statistical features provide information about how each sensor has changed during the current window, providing more discriminative power for the classifier.

For example, consider the sequence of sensor events in Table 1. With a sample interval of one second, the feature extraction component generates the sensor state vectors shown in Table 4. Each sensor's state is changed in a sample vector if its value changed in the preceding sample interval. If the sample lag is four seconds, sample features generated for events occurring between 11:36:30 and 11:36:31 would use the last four sample vectors in the table (starting with 11:36:27). So, for example, the sensorMean and sensorSum for M002 would be 0.25 and 1, respectively.

The discrete and sampling features generated for each sensor event are combined into a single feature vector. This vector is then used by the second component of AF, the forecaster.

Table 4: Example sensor state vectors for the sensor events listed in Table 1 with a sampling interval of one second. Values that have changed since the previous vector are bolded.

Sample Time	M001	M002	M003	LS001
11:36:25	0	0	0	0
11:36:26	0	**1**	0	**27**
11:36:27	**1**	1	0	27
11:36:28	1	**0**	**1**	27
11:36:29	1	0	1	27
11:36:30	1	0	1	**36**

3.2 Activity Forecaster

The activity forecaster learns a model that maps extracted features to a forecast value. This forecast value represents the number of time units that will elapse between the time of the current sensor event and the time when the next occurrence of the target activity will start.

In order to generate a forecast value from the feature vector, AF requires a classifier that can output a numeric value based on the feature inputs. A simple linear regression could be used to

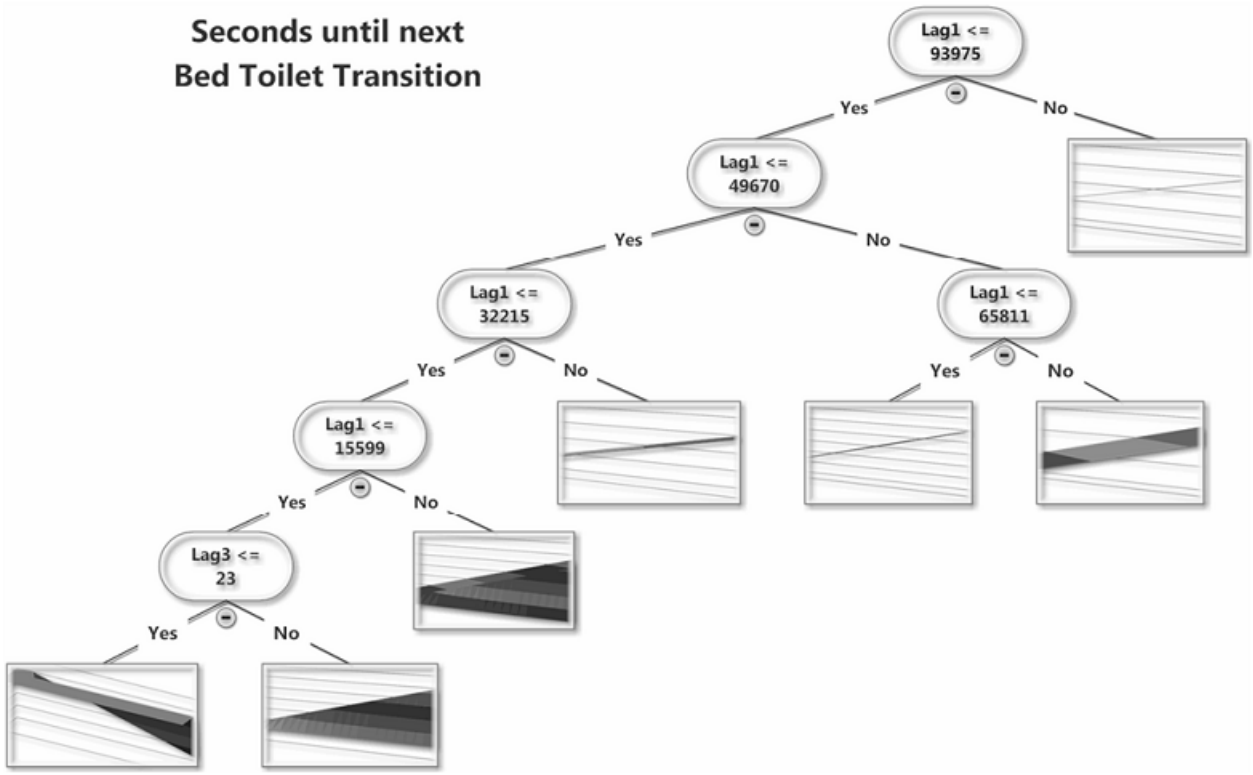

Figure 1: Example regression tree to predict the Bed Toilet Transition activity. Labels on the internal nodes indicate the split attribute and threshold checked at that node. Leaf nodes contain linear regression models used to compute a numeric activity prediction when the node is reached.

do this, but it would only be able to model the linear components of the activity relationships. Any complex nonlinear aspects of the activities would not be represented. Although a support vector machine (SVM) could be used, such a classifier has high computational cost and thus could not be quickly retrained. Such retraining is valuable when modeling daily activities for individuals whose routines may change over time. To address these needs, AF creates a regression tree to learn the forecasting model. Like decision trees, regression trees are traversed from the root node to the leaf nodes when generating a forecast value, following the path determined by split attributes at each node. However, while a decision tree has class labels that are applied at each leaf node, the regression tree's leaf nodes contain multivariate linear models that are used to determine the classification. Figure 1 shows an example regression tree generated to predict the Bed Toilet Transition activity.

The regression tree must initially be trained using a labeled set of training data. Once trained, however, the regression tree can be used to quickly generate a forecast value based on the current context. For example, after an initial training period, the regression tree classifier could be deployed to provide activity forecasts in a smart home environment based on recent sensor events.

When the regression tree is trained, the feature extractor generates feature vectors representing the training data. Each sensor event is labeled with the activity that was occurring when the event was generated. A human annotator or an activity recognition system may label these activity labels beforehand. A trained AR algorithm can be used to provide labels for AF to allow automated generation of new training instances. The classifier uses these activity labels to

compute the class labels. These class labels represent the actual time that elapses between the current sensor event and the next occurrence of the target activity (i.e., the first sensor event in the future that is labeled with the target activity).

The training feature vectors and associated class labels are used to generate the regression tree. Starting with the collection of training examples, T, we compute the standard deviation of its class labels, $\sigma(T)$. We then start building the tree from its root node. At each node, we choose an attribute to split the tree by finding the attribute that maximizes the reduction in error. For a particular split attribute, we let T_i denote the subset of examples produced by the i^{th} outcome of the split. We compute the standard deviation of this subset's class labels, $\sigma(T_i)$, as an estimate of the error in the examples in that subset. We then choose a split attribute, *attr*, which maximizes the gain, as defined in Equation 2.

$$Gain(T, attr) = \sigma(T) - \sum_{i \epsilon Values(attr)} \frac{|T_i|}{|T|} \times \sigma(T_i) \quad (2)$$

We continue this splitting process recursively with the node's children to form the tree structure. The splitting terminates when one of the following occurs: all attributes have been used along the current path, the number of examples remaining at a node is too small (four or fewer in our case), the standard deviation of the remaining example class values is less than a percentage of the original standard deviation (5% in this case), or the tree has reached a maximum depth limit (5000 nodes deep in this case).

In order to improve the regression tree's performance and reduce its complexity, we prune the tree in a bottom-up fashion, starting with the leaf nodes. For each node, we create a linear regression model using only the examples associated with that node and the attributes used as split attributes in the node's children. We then classify the subset of training examples associated with the node using this model and compute the root mean square (RMSE) of the computed class labels as shown in Equation 3. Here, we determine the error as the difference between the predicted and actual class labels (forecast values) for the data point (sensor event).

$$RMSE = \sqrt{\frac{\sum_{e=1}^{\#events} \left(predicted(e) - actual(e)\right)^2}{\#\ events}} \quad (3)$$

We also perform the classification using the node's subtree and compute the RMSE of the results. If the node's regression result has lower error than that of its subtree, we prune the subtree and the node becomes a leaf node instead.

Once the training of the regression tree is complete, it can be used to determine classification values for new events. We pass the generated features from the feature extraction component to the regression tree to compute a forecast. Starting from the root node, we compare the value of the feature specified as the split attribute at each node to decide which child node should be accessed. When we reach a leaf node, we use the regression model from that node to compute a class value.

4. Experimental Results

4.1 Setup

To demonstrate the effectiveness of our AF algorithm, we test it using sensor events collected from CASAS smart homes. In particular, we use data from N=25 smart home testbeds, each

housing one or two adults who were 73 years of age or older. Each of the datasets represents data collected over time periods ranging from a few weeks to a year while the residents performed their daily routines. Each of the datasets contains data from discrete sensors (motion, door, and light switch sensors) and sampling sensors (temperature, light level). The testbeds contain an average of 51 sensors. The floorplan and sensor layout for one of the testbeds is shown in Figure 2. Also included in the sensor events are battery status events used to periodically update the system on the wireless sensors' battery status. Information about the datasets is shown in Table 5.

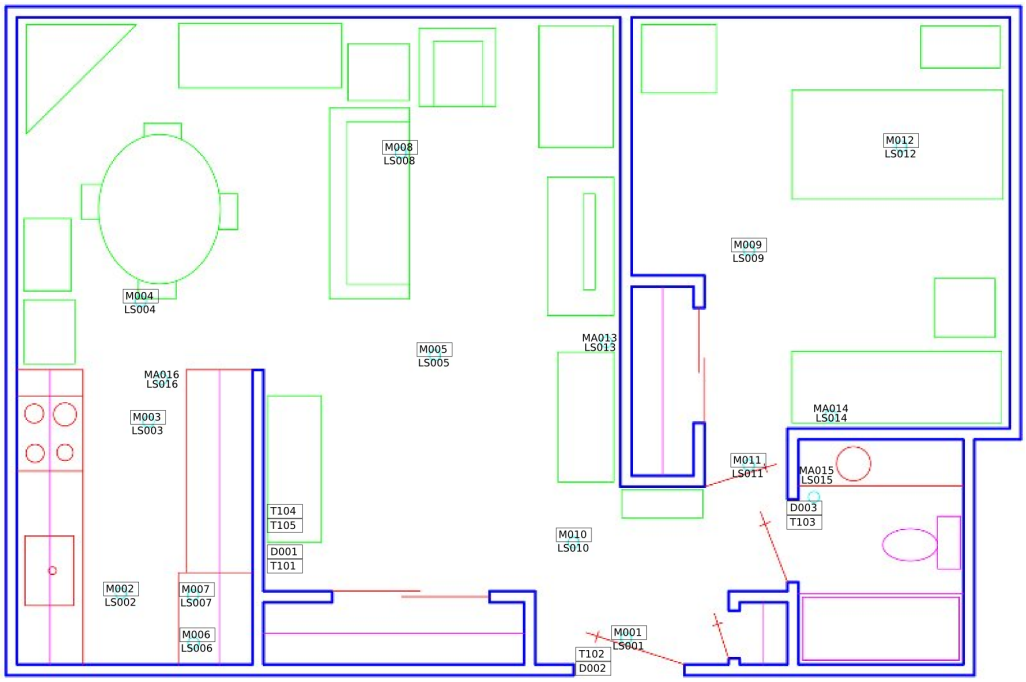

Figure 2: The floorplan and sensor layout for one of the CASAS smart home testbeds. Sensors labeled with "M" represent infrared motion sensors, "LS" are light level sensors, "D" are magnetic door sensors, and "T" are temperature sensors.

Human annotators labeled the sensor events with related activities by observing the home floorplan and sensor layout, interviewing residents to ascertain their daily routines, and visualizing the event sequences in software. Each dataset was annotated by multiple individuals to maximize label accuracy and consistency. In total, the datasets contain 118 activity classes, although not every activity appears in each dataset. However, there are 30 core activities that appear in at least 19 of the 25 datasets. These activities represent many aspects of daily living, including sleeping, bathing, eating, and personal hygiene. They are activities that are likely to be of interest in order to monitor the inhabitant's health and daily functioning [57]. Sensor events that do not fit in one of these core activity classes are labeled as "Other Activity" for these experiments.

To validate the performance of AF, we employ a *sliding window validation*. This is similar to a k-fold cross validation. However, the sequential nature of the sensor events is important for both model training and for activity prediction, so random data subsets cannot be selected for training and testing. Instead, this approach uses a sliding window of fixed length and moves it across the dataset to segment the sensor events. Events within the window are used to train AF, while the

next event after the window is used as a test instance. The window is then shifted by a specified number of events and the training and testing process is repeated. This approach maintains the temporal order of the data while also providing results for multiple training and test sets.

Table 5: The 25 CASAS smart home datasets used in our experiment. There are 118 separate labeled activities across all datasets, but some are only used with one or two datasets. A core group of 30 activities is used in almost all of the datasets.

Dataset	Start Date	End Date	Events	Labeled Activities
HH101	7/18/2012	9/17/2012	326066	30
HH102	6/15/2011	8/15/2011	413142	30
HH103	6/15/2011	8/11/2011	167183	29
HH104	6/15/2011	8/14/2011	484931	32
HH105	6/15/2011	8/14/2011	225820	30
HH106	6/15/2011	8/15/2011	263083	33
HH107	7/20/2012	8/20/2012	295165	27
HH108	9/1/2011	10/31/2011	362164	31
HH109	6/15/2011	8/14/2011	569331	32
HH110	6/15/2011	7/15/2011	138331	25
HH111	6/15/2011	8/14/2011	356496	33
HH112	6/15/2011	9/30/2011	669330	30
HH113	6/15/2011	11/5/2012	3250290	32
HH114	6/15/2011	7/15/2011	194345	28
HH115	6/15/2011	4/29/2012	2169339	64
HH116	6/15/2011	8/15/2011	507995	32
HH117	6/15/2011	5/25/2012	1118020	37
HH118	6/15/2011	7/15/2011	254112	30
HH119	1/28/2012	2/27/2012	140711	28
HH120	1/28/2012	3/31/2012	300037	32
HH121	3/1/2012	3/11/2012	170930	75
HH122	4/1/2013	4/30/2013	202112	32
HH123	3/2/2013	4/1/2013	154068	30
HH124	3/1/2013	4/30/2013	76024	21
HH125	3/1/2013	4/30/2013	216255	32

Although the sliding window approach only uses feature data from events within the window itself, the buffers used in feature generation are initially seeded with events from just prior to the window. This allows features to be generated starting with the first event in the window, rather than waiting for the buffers to fill. This is similar to what might occur in a real-world environment, where the classifier could observe sensor events for some time to build up the feature buffers before being trained. This approach also reduces the complexity of the experiment by allowing event features to be generated only once for each dataset.

Since AF outputs a numeric forecasted time value until the next activity, it is difficult to define the notion of classification accuracy or true positives. Instead, we determine the success of the forecaster by examining the difference between the predicted forecast value and the actual time until the next activity occurrence (the model error for a particular data point). However, since the time between activities can vary greatly depending on activity frequency and other factors, the forecast values and actual class labels also vary and differ for each activity. For example, an activity like toileting may occur multiple times per day, while housekeeping may only occur a few times per month.

Due to this variability between activities, a measurement of error such as the RMSE cannot be effectively used to compare the forecasting error for different activities. The mean absolute percentage error (MAPE), shown in Equation 4, could be used since it normalizes each event's forecast error. This measure, however, is subject to distortion due to outliers. For example, when the actual class label is near zero but the predicted value is only somewhat larger (perhaps off by only 10 time units), the normalization for that particular error could be very large. This in turn causes the MAPE to be distorted toward a larger value.

$$MAPE = \frac{\sum_{e=1}^{\#events} |(predicted(e) - actual(e))/actual(e)|}{\#\ events} * 100\% \quad (4)$$

As an alternative, we use a normalized measure of error based on the RMSE called range-normalized RMSE (RangeNRMSE), defined in Equation 5. The RangeNRMSE is the RMSE normalized by the range of the actual class labels. It also provides a normalized measure of the average error, but is not as subject to outliers as MAPE since the normalization occurs on the averaged error value rather than on a per-event basis. A perfectly accurate forecast would have a RangeNRMSE value of zero. Greater differences between the forecast and actual times will result in larger RangeNRMSE values indicating lower performance.

$$RangeNRMSE = \frac{RMSE}{\max(actual) - \min(actual)} \quad (5)$$

Since the actual class labels for each sensor event are derived based on the annotated activity labels, we can only provide class values, or elapsed time until the target activity occurs, up until the target activity is last performed in the dataset. Beyond this point, the next occurrence of the activity is unknown and the label cannot be determined. Thus, for each activity, sliding window validation is stopped at the last event labeled with that activity, instead of using the full dataset. However, the number of windows used for each activity is still sufficiently large and approximately the same across activities.

4.2 AF Parameter Comparison

We hypothesize that activity occurrences can be accurately learned from sensor data and AR-based activity labels. We also recognize that the choice of parameters values will affect the performance of AF. To validate this hypothesis and measure the effect of alternative parameter choices, we examine the performance of AF with varying window sizes. In addition, we analyze the effect of adding sampling features based on our discrete event sensors by comparing the performance with and without these features.

We run three categories of tests. The first category of tests is run by including only the events from discrete sensors and generating only discrete features for training and testing of AF. This

provides a baseline of comparison for the inclusion of other sensor types and sensor features. The second category of tests is run by including events from all sensors, but still with only discrete features. The final category of tests is run using events from all sensors and with both discrete and sampling features generated.

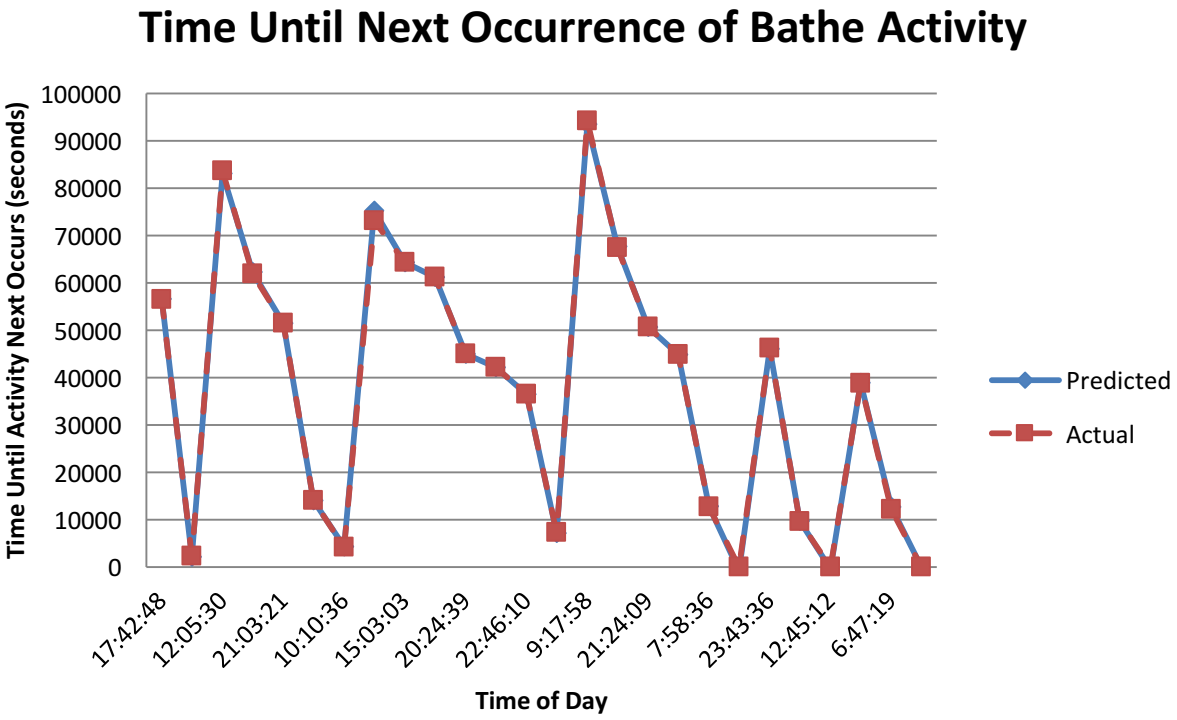

Figure 3: Example comparison of AF-predicted and actual times until the next occurrence of the Bathe activity. These values are from the test with no sampling sensors and no sampling features with a window size of 2000 for the HH101 dataset. The predicted values are almost indistinguishable from the actual values in most cases. Values near zero indicate that the activity is occurring at that time, while peaks indicate the maximum observed time that will elapse before the next activity occurrence.

Within each test category, we run the sliding window validation tests with window sizes of 2,000 and 5,000 events to determine the effect of training window size on performance. The window sizes must balance the need for a sufficient number of data points in each training window with the need for a sufficient sample size from which to analyze performance. Although larger window sizes could be used to capture more events, preliminary testing indicated that smaller window sizes produced better performance. For all tests we move the window forward by 1,000 events with each iteration. This allows us to obtain a sufficient number of test points while still maintaining reasonable test complexity.

Figure 3 shows an example of the predicted and actual times until the next occurrence of the Bathe activity from the test with no sampling sensors, no sampling features, and window size 2000 for the HH101 dataset. The time until the next occurrence of the activity varies from 0 to 95,000 seconds. In most cases, the forecast time to the activity is nearly identical to the actual value.

A summary of the test results is shown in Table 6, while Figures 4 and 5 show macro-averaged RangeNRMSE values by dataset and by activity, respectively. The table indicates that the highest-performing (lowest-average RangeNRMSE) models are the ones that utilize all sensors, utilize all features, and employ a sliding window of size 2000 events. The average error for this test category (0.0102) is lower than the average result without the sampling features (0.0112). As Table 6 shows, using all features leads to better performance for the greatest number of datasets. In contrast, using all sensors but only discrete features yields the best performance for only one dataset. This is supported by the results in Figure 4, where many of the datasets perform better with all features included. Thus, it appears that the inclusion of both sampling sensors and sampling features is helpful to the AF algorithm in forming predictions.

In fact, the best set of parameters can vary between different datasets and activities. For example, while most of the tests have lower performance with a window size of 5000 (compared to the equivalent test with a window size of 2000), this is not always the case. For both HH106 and HH109, the larger window sizes perform better than their smaller-size counterparts. Similarly, for some datasets (e.g. HH108 and HH 123), the tests with no sample sensors and no sample features performed the best. These differences may be due to factors such as the kinds of sensors used in the dataset and relationships between the activities and sensor events.

Table 6: Average and median RangeNRMSE results for the AF parameter comparison tests. The best-performing tests by each measure are bolded. The table also shows the number of datasets for which each test has the lowest error amongst all results for that datasets. The tests are listed by the combination of sensors and features used, along with the sliding window size. All tests are run with the window moving by 1000 events with each iteration.

Test	Average RangeNRMSE	Median RangeNRMSE	Number of Datasets Where Test Has Lowest Error
Disc Sensors/Disc Features 2000	0.0128	0.0052	5
Disc Sensors/Disc Features 5000	0.0184	0.0062	1
All Sensors/Disc Features 2000	0.0112	0.0043	1
All Sensors/Disc Features 5000	0.0106	0.0048	4
All Sensors/All Features 2000	**0.0102**	**0.0041**	**9**
All Sensors/All Features 5000	0.0109	0.0047	5

The activity error plot in Figure 5 provides some insight into how the type of activity and sensors affect AF performance. Activities such as cooking and eating, which are likely to occur in the same location and about the same time every day, generally have the lowest error rates. Activities such as entering and leaving the home, which are likely to occur at variable times of day and with different sets of preceding and following events, have much higher error with all parameter configurations.

Some activities, such as Toilet, Morning Meds, or Work At Table, show much better results by having all sensors included. This indicates that the use of some of the sampling sensors, such as light level sensors, may help to improve the accuracy of prediction by allowing AF to discern activity that is more evident with these sensors. For other activities, such as Watch TV or Bed-Toilet Transition, this difference is less. This may indicate that the information from the sampling sensors is less important for predicting these activities. This makes sense especially for the Bed-Toilet Transition activity, which is likely to occur at night and when there is generally less light throughout the house.

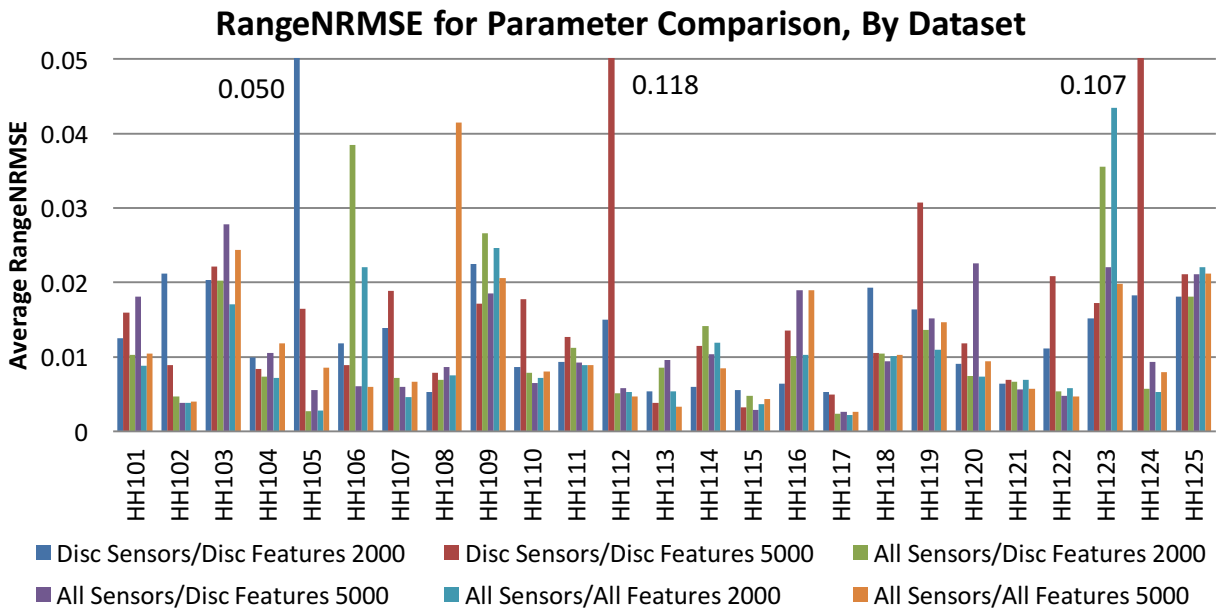

Figure 4: Average RangeNRMSE results for each dataset for the AF parameter comparison tests. Shown are macro-averages of the RangeNRMSE results for all activities in each dataset. The vertical axis has been limited to more clearly show results, and those values which were cut off are labeled with their respective values.

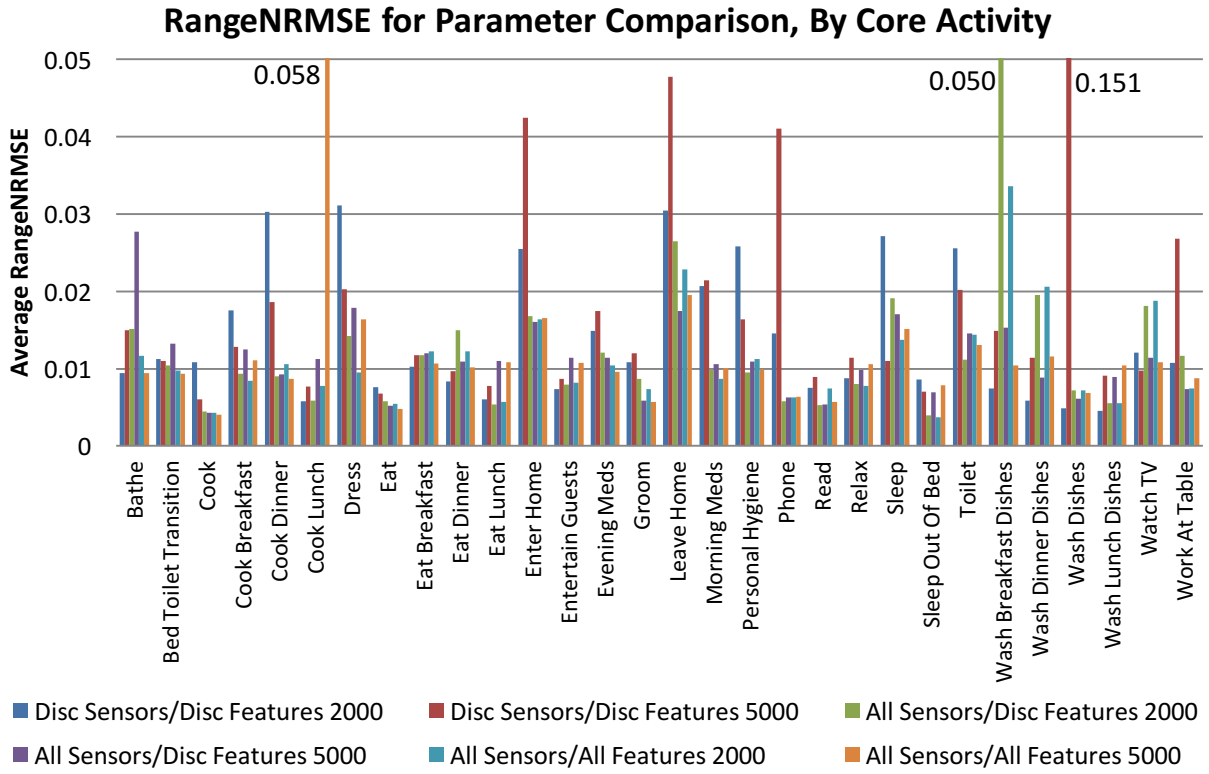

Figure 5: Average RangeNRMSE results for each of the core activities for the AF parameter comparison tests. Shown are macro-averages of the RangeNRMSE results for each activity across all datasets in which that activity occurred. The vertical axis has been limited to more clearly show results, and those values which were cut off are labeled with their respective values.

4.3 Performance Relationships to Dataset Characteristics

Next, we probe deeper to determine what the relationship is, if any, between characteristics of activities and the ability to forecast their occurrences. We perform this experiment because we would like to know if we can expect AF's performance to vary dramatically based on the type of data it is analyzing or the type of activity it is forecasting. To analyze this, we compare the correlation between characteristics of the activity and/or the dataset with AF performance (based on RangeNRMSE values). The results are summarized in Table 7. As the results indicate, none of the dataset characteristics shows a strong correlation with the error values, which may indicate that AF is not affected very much by the characteristics of the dataset. There is almost no correlation between the number of times per day an activity occurs and the error (0.0038), which suggests that AF does not need activities to occur frequently to provide a useful prediction. There is also little correlation with the variance in activities' start and end times, suggesting that AF is able to predict activities equally well if they happen at consistent times or have diverse occurrence times.

Table 7: Correlation coefficient of RangeNRMSE results to various dataset statistics for AF with all sensors, all features, and a window size of 2000. The first five correlation values represent the correlation between the statistic and the RangeNRMSE results, for all activities on all datasets. The last value is a correlation between the average number of events per day for each dataset and the median RangeNRMSE results for that dataset.

Statistic	Correlation
Activity Occurrences/Day	0.0038
Activity Circular Mean Start Time	-0.0220
Activity Circular Mean End Time	-0.0359
Activity Start Time Circular Variance	-0.0031
Activity End Time Circular Variance	-0.0059
Dataset Events/Day	-0.0634

There is some negative correlation with the start and end times of activities. This indicates that activities occurring in the evening may be slightly easier to predict (lower error) compared to those in the morning. This might be because residents are more active during the day and may follow more consistent schedules than at night or early morning, when they are primarily sleeping with little activity. Furthermore, there is a slight correlation between the number of events occurring per day in each dataset and the performance of the datasets. This supports the notion that having a higher density of events provides more information for AF to compute forecasts, leading to smaller error. Overall, however, there is not a strong correlation between the performance of AF and the statistical characteristics of the dataset.

4.4 Classifier Performance Comparison

In order to further evaluate the performance of the AF algorithm, we also perform a set of tests comparing the forecasting capabilities of AF's regression tree with two other popular regression classification methods: linear regression and support vector machine (SVM) regression. For these tests, the feature extraction component of AF is used to generate features for the sensor events as in the previous tests. For these tests, all sensors and all of the features are used. We train and test each of the three classifiers using a sliding window as before. All three classifiers are trained and tested with the same windows. We then compare the forecast predictions from the classifiers with the actual class labels for each test event.

Table 8: Average and median RangeNRMSE results for the classifier comparison tests. The best-performing tests by each measure are bolded. The table also shows the number of datasets for which each test had the lowest error amongst all results for that datasets. All tests were performed with a window of size 500 and moved by 500 events with each iteration.

Classifier Type	Average RangeNRMSE	Median RangeNRMSE	Number of Datasets Where Test Has Lowest Error
AF (Regression Tree)	**0.00996**	**0.00479**	**18**
Linear Regression	9167503	318159.8	0
SVM	0.01833	0.00496	7

We use the Weka data mining software [58] to implement the LR and SVM classifiers. We use the linear regression with no attribute elimination. The SVM is trained using the sequential minimal optimization algorithm [59] with a linear kernel and attribute normalization.

We originally used a window size of 2000 events, as with the previous tests. The regression tree and linear regression classifiers completed training and testing for all sliding windows for an activity in 12 hours or less. The SVM classifier had more difficulty with the data, however. After running for more than 10 hours, it had only completed training and testing for a fraction of the sliding windows (only completing two or three windows for some activities). We could not get any of the SVM tests to complete in less than 12 hours. We suspect that the SVM had difficulty learning a model due to the large number of features and data instances used for each training window. (There were approximately 2000 or more attributes generated for each data instance.) This indicates some of the difficulty of using an SVM classifier for the activity prediction problem due to its complexity and training time. In order to allow the SVM to finish, we reduce the sliding window size to 500 events, moving the window forward by 500 events each time.

The results of these tests are summarized in Table 8. The linear regression algorithm yields very large average and median RangeNRMSE values. These values are primarily due to a few predictions with very large errors that cause the overall RangeNRMSE to be large. While many of the linear regression errors are smaller, these large errors indicate that the linear regression may have trouble dealing with some nonlinear aspects of the activity patterns. They also suggest that it may be useful in activity forecasting to be able to detect and minimize predictions with large errors.

The SVM and regression tree classifiers have much lower error. The regression tree has a lower average and a lower median RangeNRMSE than the SVM. It also had the lowest error values of all three classifiers in 18 of the 25 datasets. This indicates that the regression tree is able to perform better than the SVM at forecasting activity occurrences.

Figure 6 shows the average RangeNRMSE results for the SVM and regression tree for each dataset. While many of the errors for both classifiers are relatively low, there are some large variations in the SVM results between datasets. In particular, the SVM error is much greater than the regression tree error for the HH114, HH115, and HH116 datasets. Figure 7 shows the errors plotted for each of the 30 core activities. Again, the regression tree outperforms the SVM in most cases. For many of the activities, the discrepancy is wide, with the SVM error much larger than the corresponding regression tree error. The SVM has particular trouble with some activities that mostly occur in the morning, such as cooking (especially breakfast), morning medications, and washing breakfast dishes. For most of these activities, however, the regression

tree has low error. This suggests that the regression tree is able to handle the possible variations in morning routines that the SVM struggles with. There are some activities for which the regression tree has lower performance, such as cooking lunch, washing lunch dishes, and sleeping. However, it shows a strong performance for most activities and consistently has relatively small error, while the SVM has more variability between activities. Anecdotally, the regression tree also had a faster training time compared to the SVM. Thus, these results suggest that the regression tree provides better classification capability for forecasting activities.

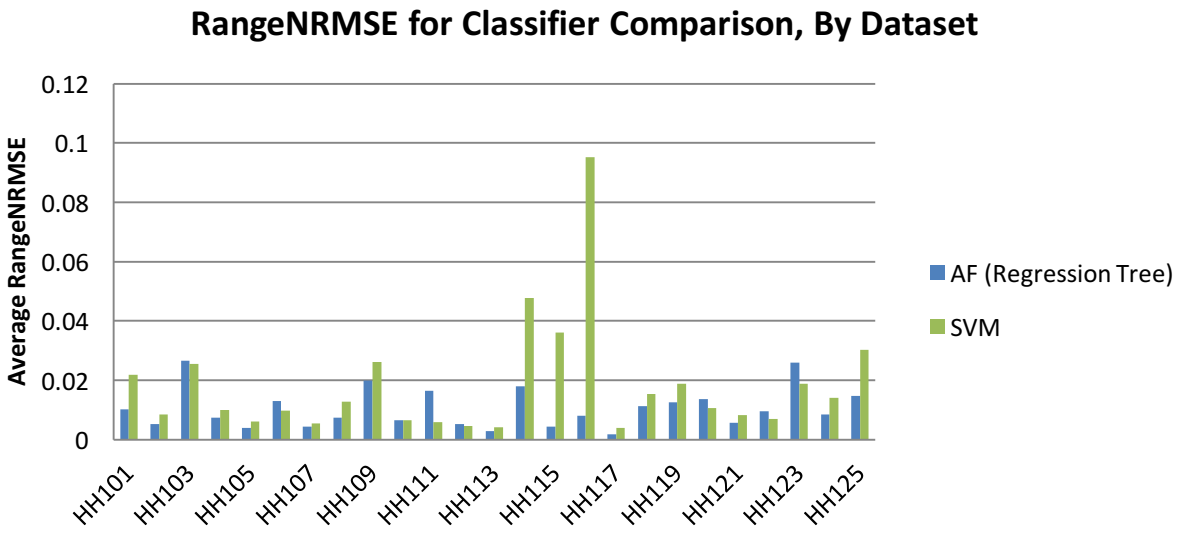

Figure 6: Average RangeNRMSE results for each dataset for the classifier comparison tests. Shown are averages of the RangeNRMSE results for all activities in each dataset. The linear regression results have been excluded due to their much larger values.

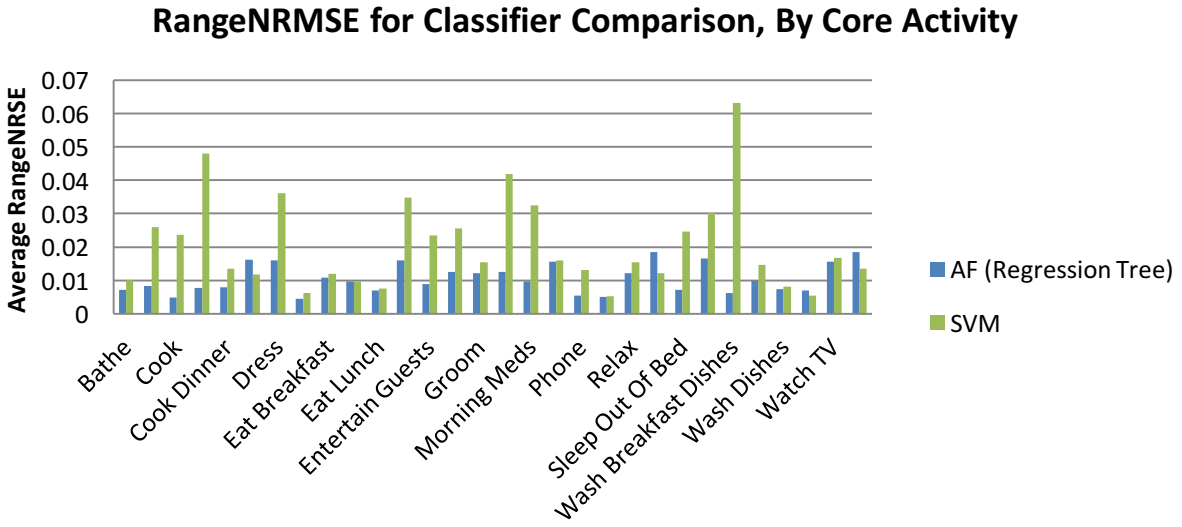

Figure 7: Average RangeNRMSE results for each of the core activities for the classifier comparison tests. Shown are averages of the RangeNRMSE results for each activity across all datasets in which that activity occurred. The linear regression results have been excluded due to their much larger values.

5. Discussion

These experiments demonstrate that AF provides activity forecasts with relatively high accuracy. Variations between datasets also do not impact the predictions very much, which allows AF to be applied in a variety of situations. The regression tree classifier of AF provides more accurate forecasts than either a linear regression or SVM regression.

In this paper, we also demonstrate the sensitivity of AF performance based on selected parameter values. The number of samples or their frequency can be increased to potentially gain further performance improvements depending on the desired complexity of the forecasting system. For example, using sample features can often improve the performance of AF, but they can be removed if the time to train the regression tree needs to be short. Although the best RangeNRMSE parameters tend to vary with the dataset and activities being predicted, using AF with all sensor types and all features with a window size of 2000 is a decent first choice. This configuration results in the lowest overall average error, performs better than the other configurations for most of the datasets in our tests, and is likely to perform well in other situations as well.

Smaller validation window sizes tend to perform better compared to larger ones, as was the case in our AF parameter comparison tests. This may be because smaller window sizes allow the training to be more focused on the sensor events occurring around the time of the test event. When the window size gets small enough, however, the performance starts to decrease. The higher RangeNRMSE results for a window size of 500 events in the classifier comparison tests demonstrate this. Consequentially, it may be worthwhile to perform further studies to determine how the window size impacts performance, especially in a live smart home environment.

We note that the activity learning and forecasting algorithms described in this paper do not rely on the availability of any particular sensor or suite of sensors. The methods can thus be utilized for mobile sensors as well, or for sensors that provide more fine-grained information.

6. Prompting Application

One use of activity forecasting is to support an activity prompting application. In such an application, we wish to prompt the user to perform activities based on times predicted by AF. A prompting application can be a useful tool for enabling a user facing cognitive decline to live more independently and improving their safety and wellbeing.

We designed a prompting application to run on mobile devices called CAFE (CASAS Activity Forecasting Environment). Figure 8 shows the application interface. AF runs on the server to generate activity predictions from observed CASAS sensor events. The application periodically queries the server to obtain the most recent predictions. When an activity is predicted to occur within the next few minutes and has not already been detected by the activity recognizer, a reminder prompt is displayed to the user. The user can then respond to indicate they are currently performing the activity. These responses, along with the predicted times, are then stored in a database for further analysis. We are currently testing CAFE in pilot studies and will analyze the usability of the app in future work.

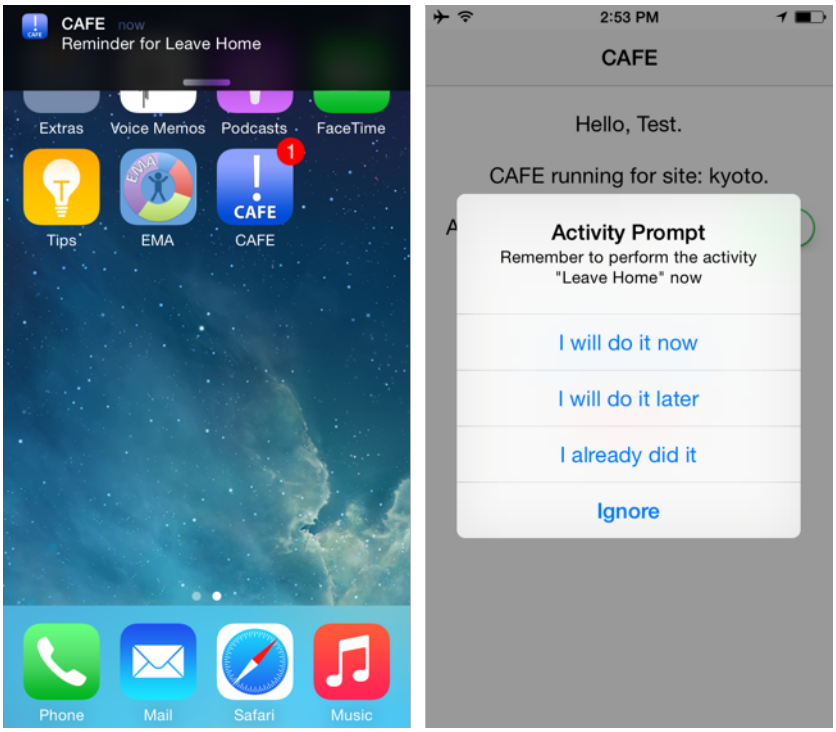

Figure 8: Layout of our CAFE activity prompting application for iOS mobile devices. AF is used to generate predictions of the user's activities. A prompt is shown at the predicted time, and the user responds regarding whether they are performing the activity.

7. Conclusion

In this article we have described AF, an algorithm for automated activity prediction in a smart home environment. AF can be used to provide forecasting services such as triggering activity prompts or managing home energy usage and resident comfort. It also can be applied in other forecasting situations where a numeric time prediction is valuable. It has an advantage over sequence prediction methods in that it produces a forecast conveying the time until an activity will next occur, allowing for scheduling of system actions that are time-dependent.

We have demonstrated that the inclusion of sampling-type sensors can boost the performance of the forecasting algorithm, as can the use of features based on samples of the sensor states. We also compared the performance of the regression tree classifier against two other popular classifiers. We found that the regression tree is able to produce forecasts of activity start times with lower error than the linear regression and SVM classifiers. It also has a faster training time and the ability to handle more complex datasets than the SVM classifier.

In our future work, we intend to explore further variations on the AF forecasting algorithm, including adapting other types of classifiers to produce improved forecasts with less complexity. We are also interested in the prospect of combining numeric forecasting techniques with sequence prediction to improve activity predictions.

We hypothesize that AF can provide a foundation for activity-predictive interventions, such as generate activity prompts in a smart home system. A prompting component could utilize the forecasts from AF for activities that require prompting, generating a prompt when a certain time criteria is met. For example, AF could be used to forecast the time until the medication activity

normally takes place. The prompting intervention could generate a prompt a few minutes before the forecasted time and repeat the prompt thereafter until the user takes the medication (as detected by an activity recognition system). We are currently gathering data for such a prompting-based intervention with a group of older adult participants.

8. Acknowledgments

This work is supported in part by NIH grant R01EB015853 and by NSF grant IIS-10644628.

9. References

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

[2] L. B. Holder and D. J. Cook, "Automated activity-aware prompting for activity initiation," *Gerontechnology*, vol. 11, no. 4, pp. 534-544, 2013.

[3] J. Ziv and A. Lempel, "A compression of individual sequences via variable rate coding," *IEEE Trans. Inf. Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[4] M. Feder, N. Merhav, and M. Gutman, "Universal prediction of individual sequences," *IEEE Trans. Inf. Theory*, vol. 38, pp. 1258–1270, 1992.

[5] A. Bhattacharya and S. K. Das, "LeZi-Update: An Information-theoretic approach for personal mobility tracking in PCS networks," *Wirel. Networks*, vol. 8, pp. 121–135, 2002.

[6] K. Gopalratnam and D. J. Cook, "Online sequential prediction via incremental parsing: The Active LeZi Algorithm," *IEEE Intell. Syst.*, vol. 22, 2007.

[7] M. R. Alam, M. B. I. Reaz, and M. A. M. Ali, "SPEED: An inhabitant activity prediction algorithm for smart homes," *IEEE Trans. on Systems, Man, and Cybernetics Part A: Systems and Humans*, vol. 42, no. 4, pp. 985-990, 2012.

[8] R. Begleiter, R. El-Yaniv, and G. Yona, "On prediction using variable order Markov models," *J. Artif. Intell. Res.*, vol. 22, pp. 385–421, 2004.

[9] G. E. P. Box and G. M. Jenkins, *Time series analysis: Forecasting and control*, Holden Day, San Francisco, 1970, revised. 1976.

[10] J. H. Kim, "Forecasting autoregressive time series with bias-corrected parameter estimators." *International Journal of Forecasting*, vol. 19, pp. 493 – 502, 2003.

[11] K. W. Hipel and A. I. McLeod, *Time Series Modeling of Water Resources and Environmental Systems*, Elsevier, Amsterdam, 1994.

[12] A. Zellner, *An introduction to Bayesian inference in econometrics*, Wiley, New York, 1971.

[13] M. H. Quenouille, *The analysis of multiple time-series*, (2nd ed. 1968), Griffin, London, 1957.

[14] T. Riise, and D. Tjøstheim, "Theory and practice of multivariate ARMA forecasting," *Journal of Forecasting*, vol. 3, pp. 309 – 317, 1984.

[15] G. A. Darbellay and M. Slama, "Forecasting the short-term demand for electricity: Do neural networks stand a better chance?," *International Journal of Forecasting*, vol. 16, pp. 71 – 83, 2000.

[16] R. F. Engle, "Autoregressive conditional heteroscedasticity with estimates of the variance of the United Kingdom inflation," *Econometrica*, vol. 50, pp. 987 – 1008, 1982.

[17] J. K. Aggarwal and M. S. Ryoo, "Human activity analysis: A review," *ACM Comput. Surv.*, vol. 43, no. 3, pp. 1–47, 2011.

[18] L. Chen, J. Hoey, C. D. Nugent, D. J. Cook, and Z. Yu, "Sensor-based activity recognition," *IEEE Trans. Syst. Man, Cybern. Part C Appl. Rev.*, vol. 42, no. 6, pp. 790–808, 2012.
[19] S.-R. Ke, H. L. U. Thuc, Y.-J. Lee, J.-N. Hwang, J.-H. Yoo, and K.-H. Choi, "A Review on Video-Based Human Activity Recognition," *Computers*, vol. 2, no. 2, pp. 88–131, 2013.
[20] A. Bulling, U. Blanke, and B. Schiele, "A tutorial on human activity recognition using body-worn inertial sensors," *ACM Comput. Surv.*, vol. 46, no. 3, pp. 107–140, 2014.
[21] A. Reiss, D. Stricker, and G. Hendeby, "Towards robust activity recognition for everyday life: Methods and evaluation," in *Pervasive Computing Technologies for Healthcare*, 2013, pp. 25–32.
[22] S. Vishwakarma and A. Agrawal, "A survey on activity recognition and behavior understanding in video surveillance," *Vis. Comput.*, vol. 29, no. 10, pp. 983–1009, 2013.
[23] O. Lara and M. A. Labrador, "A survey on human activity recognition using wearable sensors," *IEEE Commun. Surv. Tutorials*, vol. 15, no. 3, pp. 1192–1209, 2013.
[24] L. Chen and I. Khalil, "Activity recognition: Approaches, practices and trends," in *Activity Recognition in Pervasive Intelligent Environments*, L. Chen, C. D. Nugent, J. Biswas, and J. Hoey, Eds. Atlantis Ambient and Pervasive Intelligence, 2011, pp. 1–31.
[25] P. Tuaraga, R. Chellappa, V. S. Subrahmanian, O. Udrea, and P. Turaga, "Machine recognition of human activities: A survey," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 11, pp. 1473–1488, 2008.
[26] J. Alon, V. Athitsos, Q. Yuan, and S. Sclaroff, "A unified framework for gesture recognition and spatiotemporal gesture segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, pp. 1685–1699, 2008.
[27] J. A. Iglesias, P. Angelov, A. Ledezma, and A. Sanchis, "Human activity recognition based on evolving fuzzy systems," *Int. J. Neural Syst.*, vol. 20, no. 5, 2010.
[28] I. L. Liao, D. Fox, and H. Kautz, "Location-based activity recognition using relational Markov networks," in *International Joint Conference on Artificial Intelligence*, 2005, pp. 773–778.
[29] E. Guenterberg, H. Ghasemzadeh, and R. Jafari, "Automatic segmentation and recognition in body sensor networks using a hidden Markov model," *ACM Trans. Embed. Comput. Syst.*, vol. 11, pp. S2:1–S2:19, 2012.
[30] J. R. Doppa, A. Fern, and P. Tadepalli, "Structured prediction via output space search," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1317–1350, 2014.
[31] J. R. Doppa, A. Fern, and P. Tadepalli, "HC-Search: Learning heuristics and cost functions for structured prediction," *J. Artif. Intell. Res.*, vol. 50, pp. 369–407, 2014.
[32] K. Forster, S. Monteleone, A. Calatroni, D. Roggen, and G. Troster, "Incremental kNN classifier exploiting correct-error teacher for activity recognition," in *International Conference on Machine Learning and Applications*, 2010, pp. 445–450.
[33] D. Cook, "Learning setting-generalized activity models for smart spaces," *IEEE Intell. Syst.*, vol. 27, no. 1, pp. 32–38, 2012.
[34] L. Bao and S. Intille, "Activity recognition from user annotated acceleration data," in *Pervasive*, 2004, pp. 1–17.
[35] N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman, "Activity recognition from accelerometer data," in *Innovative Applications of Artificial Intelligence*, 2005, pp. 1541–1546.

[36] J. A. Ward, P. Lukowicz, G. Troster, and T. E. Starner, “Activity recognition of assembly tasks using body-worn microphones and accelerometers,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 10, pp. 1553–1567, 2006.
[37] G. Singla, D. J. Cook, and M. Schmitter-Edgecombe, “Recognizing independent and joint activities among multiple residents in smart environments,” *Ambient Intell. Humaniz. Comput. J.*, vol. 1, no. 1, pp. 57–63, 2010.
[38] J. Lester, T. Choudhury, N. Kern, G. Borriello, and B. Hannaford, “A hybrid discriminative/generative approach for modeling human activities,” in *International Joint Conference on Artificial Intelligence*, 2005, pp. 766–772.
[39] O. Amft and G. Troster, “On-body sensing solutions for automatic dietary monitoring,” *IEEE Pervasive Comput.*, vol. 8, pp. 62–70, 2009.
[40] M. Zhang and A. A. Sawchuk, “Motion Primitive-Based Human Activity Recognition Using a Bag-of-Features Approach,” in *ACM SIGHIT International Health Informatics Symposium*, 2012, pp. 631–640.
[41] U. Blanke, B. Schiele, M. Kreil, P. Lukowicz, B. Sick, and T. Gruber, “All for one or one for all? Combining heterogeneous features for activity spotting,” in *IEEE International Conference on Pervasive Computing and Communications Workshops*, 2010, pp. 18–24.
[42] T. van Kasteren, A. Noulas, G. Englebienne, and B. Krose, “Accurate activity recognition in a home setting,” in *ACM Conference on Ubiquitous Computing*, 2008, pp. 1–9.
[43] A. Bulling, J. A. Ward, and H. Gellersen, “Multimodal recognition of reading activity in transit using body-worn sensors,” *ACM Trans. Appl. Percept.*, vol. 9, no. 1, pp. 2:1–2:21, 2012.
[44] S. Wang, W. Pentney, A. M. Popescu, T. Choudhury, and M. Philipose, “Common sense based joint training of human activity recognizers,” in *International Joint Conference on Artificial Intelligence*, 2007, pp. 2237–2242.
[45] J. Lester, T. Choudhury, and G. Borriello, “A practical approach to recognizing physical activities,” in *International Conference on Pervasive Computing*, 2006, pp. 1–16.
[46] T. Gu, S. Chen, X. Tao, and J. Lu, “An unsupervised approach to activity recognition and segmentation based on object-use fingerprints,” *Data Knowl. Eng.*, vol. 69, no. 6, pp. 533–544, 2010.
[47] F. Niu and M. Abdel-Mottaleb, “HMM-based segmentation and recognition of human activities from video sequences,” in *IEEE International Conference on Multimedia and ExpoICME*, 2005, pp. 804–807.
[48] O. Duchenne, I. Laptev, J. Sivic, F. Bach, and J. Ponce, “Automatic annotation of human activities in video,” in *International Conference on Computer Vision*, 2009, pp. 1491–1498.
[49] Y. Zheng, W.-K. Wong, X. Guan, and S. Trost, “Physical activity recognition from accelerometer data using a multi-scale ensemble method,” in *Innovative Applications of Artificial Intelligence Conference*, 2013, pp. 1575–1581.
[50] X. Hong and C. D. Nugent, “Segmenting sensor data for activity monitoring in smart environments,” *Pers. Ubiquitous Comput.*, vol. 17, pp. 545–559, 2013.
[51] P. Palmes, H. K. Pung, T. Gu, W. Xue, and S. Chen, “Object relevance weight pattern mining for activity recognition and segmentation,” *Pervasive Mob. Comput.*, vol. 6, no. 1, pp. 43–57, 2010.

[52] T. Yamasaki and K. Aizawa, "Motion segmentation and retrieval for 3D video based on modified shape distribution," *EURASIP J. Appl. Signal Processing*, vol. 2007, no. 1, pp. 211–211, 2007.
[53] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series," in *IEEE International Conference on Data Mining*, 2001, pp. 289–296.
[54] N. Krishnan and D. J. Cook, "Activity recognition on streaming sensor data," *Pervasive Mob. Comput.*, vol. 10, pp. 138–154, 2014.
[55] B. Minor and D. J. Cook, "Regression tree classification for activity prediction in smart homes," *UbiComp AwareCast Workshop*, 2014.
[56] D. Cook and N. Krishnan, *Activity Learning from Sensor Data*, Wiley, 2014.
[57] V. G. Wadley, O. Okonkwo, M. Crowe, and L. A. Ross-Meadows, "Mild cognitive impairment and everyday function: Evidence of reduced speed in performing instrumental activities of daily living," *The American Journal of Geriatric Psychiatry*, vol. 15, no. 5, pp. 416-424, 2008.
[58] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explorations*, vol. 11, no. 1, 2009.
[59] 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*, vol. 11, no. 5, 2000.