

---

# Identifying Tasks and Predicting Action in Smart Homes using Unlabeled Data

---

Sira Panduranga Rao

Diane J. Cook

Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76019 USA

SPRAO@CSE.UTA.EDU

COOK@CSE.UTA.EDU

## Abstract

An intelligent home is likely in the near future. An important ingredient in an intelligent environment such as a home is prediction – of the next low-level action, the next location, and the next high-level task that an inhabitant is likely to perform. In this paper we model inhabitant actions as states in a simple Markov model. We introduce an enhancement, the Task-based Markov model (TMM) method. TMM discovers high-level inhabitant tasks using the unlabeled data supplied. We investigate clustering of actions to identify tasks, and integrate clusters into a hidden Markov model that predicts the next inhabitant action. We validate our approach and observe that for simulated data we achieve good accuracy using both the simple Markov model and the TMM, whereas on real data we see that simple Markov models outperform the TMM. We also perform an analysis of the performance of the HMM in the framework of the TMM when diverse patterns are introduced into the data.

## 1. Introduction

We envision that a home capable of being intelligent is a very likely possibility. A home that is capable of making decisions will provide a degree of autonomy not currently found in home environments. Individuals in such a home will find the home adapting to their needs and living styles while preserving comfort. In automating the home, many of the inhabitant's desired home interactions can be autonomously performed without the inhabitant or user intervening. Automation will provide a start towards the home adapting to the inhabitant's needs.

The Mavhome Smart Home research project (Cook et al., 2003; Das et al., 2002) focuses on the creation of an intelligent home that behaves as a rational agent. An important component is the ability to make decisions based on predicted activities. Values that can be predicted include the usage pattern of devices in the home, the movement patterns of the inhabitants and typical activities of the inhabitants. In this paper we model part of the prediction problem – predicting device usage in a home.

In a home, a user interacts with a variety of devices, and we term each such interaction as an *action*. For example, switching on the kitchen light at a given time is an action and can be represented as the string '10/25/2002 8:15:31 PM Kitchen Light D1 ON', where D1 is the device identifier, 'Light' is the device type, 'Kitchen' represents the location of the device, and 'ON' is the resulting status of the device. This interaction is time-stamped with the time and date. An inhabitant such as Sira can be characterized by his patterns which consist of a number of high-level tasks broken down into individual actions. For example, the task of 'Getting to Work' consists of turning on the bathroom light at 7am, turning on and later off the coffee maker in the kitchen, turning off the bathroom light at 7:30am, then turning on the kitchen light. After Sira has breakfast, the garage door is opened while the kitchen light is turned off and the door is closed after he leaves. In this case we know the task and the corresponding actions, but the observation data consists of just the low-level actions. Our goal is to learn which sets of actions constitute a well-defined task and use this information to make predictions.

Work in the area of prediction has been applied to other problems, such as predicting UNIX commands (Davison & Hirsh, 1998; Gorniak & Poole, 2000). The prediction problem is difficult for a home scenario where there are multiple inhabitants performing multiple tasks at the same or different times. As a first step, we investigate a portion of this problem: predicting a single inhabitant's actions and tasks.

Traditional machine learning techniques (e.g., decision trees, memory-based learners, etc.) that are employed for classification face difficulty in using historic information to make sequential predictions. Markov models offer an alternative representation that implicitly encodes a portion of the history. The distinguishing feature of our approach is that we identify the activity or task that encompasses a set of actions. We hypothesize that recognizing the task will help better understand the actions that are part of it, and we expect to better predict the inhabitant's next task and thus the next action. For example, if the home recognizes that Sira is making breakfast, it will better predict his next action as turning on the coffee maker given that the last action was turning off the bathroom light. This paper builds on our previous work in which we

introduce the application of TMM to smart home tasks (Cook et al., 2003; Rao & Cook, 2003).

The remainder of the paper presents our work. We first formalize the problem and explain application of a simple Markov model. We then explain a method of partitioning actions into sequences, clustering the sequences into tasks, and integrating task information into a hidden Markov model. Following this, we discuss generation of the simulated data, the collection of real data and the validation of our approach. Finally, we conclude with some directions for future work.

## 2. Problem Description and Modeling of Actions

Given actions  $A_1, A_2, \dots, A_N$  we want to predict the next action,  $A_{N+1}$ . An action  $A_i$  describes an event in a home. In the example of an action string provided earlier, “10/25/2002 8:15:31 PM Kitchen Light D1 ON”, the action string consists of many features or fields: date, time, location, device type, device identifier and device status. In this work we consider devices that have only binary states – ON and OFF. Also, the locations and the locations of devices in a home are assumed to be fixed and do not change over a period of time.

### 2.1 Markov Models

Our motivation to approach the prediction problem within the framework of Markov models is prompted by the need to learn the pattern of user actions without storing large amounts of history. As a first step we model the entire sequence of inhabitant actions as a simple Markov model, where each state corresponds to one or more actions. For example, consider the action  $A_{11}$ : *03/31/2003 8:17:31 PM Kitchen Light D1 ON* which can be represented as a state in the model. To create a unique state for each possible action, that is a device manipulation with the equivalent time stamp, would result in an unreasonably large model. We construct smaller models by discretizing the time of the action and ignoring the date, in effect merging similar states together. Consider another action  $A_{12}$ : *04/02/2003 8:11:46 PM Kitchen Light D1 ON*. The actions are composed of fields that we represent as features in our state representation: date, time, location, device description (e.g., Light), device id, and device status (e.g., ON). Observe that  $A_{11}$  and  $A_{12}$  differ only in the time and date features. Given two actions we can calculate the difference in the time of the two actions. If the difference in time between the actions is ‘minimal’, the corresponding states can be merged. We say the difference is ‘minimal’ or ‘negligible’ if this difference is within the allowable time.

When each new action  $A_i$  is processed, a check is made to see if the action is ‘close enough’ to an existing model state to merge with that state. An action is close enough to an existing model state if they both refer to the same device and location but differ in the time and this time difference is ‘negligible’. The term ‘negligible’ can be

fixed through use of different values and utilizing the performance results of these values. If not, a new state is created with action representative  $A_i$  and transitions are created from the previous action in the sequence to  $A_i$ . The values of each transition reflect the relative frequency of the transition. Thus, discretizing the time helps achieve a moderate number of states. If each action were to be considered as a single state, there would be a great number of states requiring a tremendous amount of training data to achieve good prediction performance.

Once the simple Markov model is constructed, it is used for prediction of actions. A prediction can be generated by matching the current state with a state in the model and predicting the action that corresponds to the outgoing transition from that state with the greatest probability.

## 3. Partitioning Actions and Clustering into Tasks

In the previous section we mention how a simple Markov model was built and used for the purpose of predicting user actions. Sometimes the simple Markov model may not yield useful predictions. This can be seen when there is an equal likelihood of a transition to many states from a particular state. This can be caused if there is too much noise in the data or there exists some randomness in the inhabitant’s actions. In this section and the next we describe enhancements to our model that we use to predict actions.

We hypothesize that the predictive accuracy can be improved by incorporating abstract task information into the model. If we can identify the actions that comprise a task, then we can identify the current task and generate more accurate transition probabilities for the corresponding task.

### 3.1 Partitioning Actions using Heuristics

Actions in smart home log data are not labeled with the corresponding high-level task, so we propose to discover these tasks using unsupervised learning. The first step is to partition the action sequence into subsequences that are likely to be part of the same task.

Given actions  $A_1, A_2, \dots, A_N$  we can divide these into disjoint groups  $G_1, G_2, \dots, G_p$ , where each group consists of a set of actions  $A_i, \dots, A_{i+j}$  and for simplicity every action is part of only one group. Note that each group maintains the sequence information and hence all actions within a group are ordered with respect to the time and date information.

The partitioning step extracts examples of activities. We attempt to separate the action sequences into groups that are likely to act as examples of activities. Given these examples we can cluster similar examples together and represent each of the similar examples by a single instance – a task. Since we have no prior information about the actions themselves, we need to use heuristics to group actions such that each group is an instance of a

task. Some of the heuristics that can be employed are ones that use the location of the device, the time interval between actions, the status of devices and the number of actions since the last grouping was formed.

We separate an action sequence  $A_x, \dots, A_y$  into groups  $A_x, \dots, A_z$  and  $A_{z+1}, \dots, A_y$  using any of the following rules:

- 1) The time difference between  $A_z$  and  $A_{z+1}$  is  $> 'P'$  minutes.
- 2) There is a difference in the device location (such as another room) between the  $A_z$  and  $A_{z+1}$ .
- 3) The number of actions in the group is  $> 'n'$ .

We allow for the use of these three rules either independently or in combination with the other rules. The result of the partitioning step is a set of groups or individual tasks. The groups so obtained are now ready to be clustered into sets of similar tasks.

### 3.2 Clustering Partitions into Abstract Tasks

Given the different groups from the partitioning process, we have to cluster similar groups together to form abstract task classes. As a first step, we extract meta-features describing the group as a whole. Information that we can collect from each group includes the number of actions in the group, the starting time and time duration of this group, the locations visited and devices acted upon in the group. This information is stored as a partition  $P_1$  and thus we have partitions  $P_1, P_2, \dots, P_p$ . The partitions are now supplied to the clustering algorithm.

Clustering is used to group instances that are similar to each other and the individual cluster represents an abstract task definition. A cluster consists of these similar instances, and we can consider the mean of the cluster distribution as the cluster representative, while the variance represents the disparity in the grouping of the cluster instances. Because we require clusters that group similar instances and because of the inherent simplicity of the partitioning algorithms, we apply k-means clustering to partitions  $P_1, P_2, \dots, P_p$ .

To perform clustering, we utilize LNKnet (Kukulich & Lippmann, 1995), from MIT Lincoln Labs. The results are parsed to extract the cluster centroids and variance values. We average the cluster centroids and variances over values generated from ten separate runs to promote generality of the results. For certain data sets we find that the clustering results show the tasks clearly. As the data is more disparate we find the clusters less reflect the true tasks. The resulting clusters represent the abstract inhabitant tasks. In the context of hidden Markov models (HMMs) these clusters can be used as hidden states, and the HMMs can be used in the TMM framework to perform action prediction.

## 4. Hidden Markov Models

Hidden Markov models have been used extensively in a variety of environments that include speech recognition

(Rabiner, 1989), information extraction (Seymore, McCallum, & Rosenfeld, 1999) and recognizing human activities (Luhr et al., 2003). HMMs can be either hand built (Freitag & McCallum, 1999) or constructed assuming a fixed-model structure that is subsequently trained using the Baum-Welch algorithm (Baum, 1972). To our knowledge HMMs have not been used for prediction of tasks, and our use of HMMs to predict actions and tasks is a new direction of research.

In this section we discuss how clusters obtained from the partitioning and clustering process are used in a hidden Markov model, how HMM parameters are learned, and the use of HMM within the framework of our task-based Markov model for predicting actions.

### 4.1 HMM Terminology

The elements of a HMM can be defined formally. A HMM is characterized by the following (Rabiner, 1989):

- 1)  $N$ , the number of hidden states in the model: We denote these individual states as  $S = \{S_1, S_2, \dots, S_N\}$  and we denote the state at time  $t$  as  $q_t$ .
- 2)  $M$ , the number of distinct observation symbols per state or the discrete alphabet size: The individual symbols are denoted as  $V = \{v_1, v_2, \dots, v_M\}$ .
- 3)  $A$ , the state transition probability distribution,  $A = \{a_{ij}\}$ , where  $a_{ij} = P[q_{t+1} = S_j | q_t = S_i]$ ,  $1 \leq i, j \leq N$
- 4)  $B$ , the observation symbol probability distribution in state  $j$ ,  $B = \{b_j(k)\}$ , where  $b_j(k) = P[v_k \text{ at } t | q_t = S_j]$ ,  $1 \leq j \leq N$  and  $1 \leq k \leq M$
- 5)  $\delta$ , the initial state probability distribution,  $\delta = \{\delta_i\}$ , where  $\delta_i = P[q_1 = S_i]$ ,  $1 \leq i \leq N$
- 6)  $O$ , the observation symbol vector or the sequence of observations,  $O = O_1 O_2 \dots O_T$ , where each observation  $O_t$  is one of the symbols from  $V$ , and  $T$  is the number of observations in the sequence.

In our problem,  $N$  represents the number of clusters, and  $v_i$  is a symbol representing a single discretized action. We want a moderate set of symbols that represent the entire training action instances. The number of such symbols is the value  $M$ .  $T$  is the size of our training data, and  $O$  is a vector containing the modified form of the actions represented as symbols.  $A$ , also known as the *transition matrix*, represents the transition probability distribution between hidden states.  $B$ , also known as the *confusion matrix*, represents the probability distribution of observing a particular symbol (a modified form of an action) upon arriving at one of the  $N$  states. The distribution  $\delta$  gives the likelihood of each state representing the initial state.

### 4.2 TMM framework

We use the clusters obtained from the clustering step as hidden states in a HMM. The hidden states thus have the same features as that of the clusters; namely, the number of actions in the cluster, the start time of actions in the cluster, the time duration of the set of actions in the

cluster, the location of the devices and the device identifiers. Construction of a HMM needs more than just the information about the hidden states. We need to have the matrices and the vectors defined before we can proceed with using the HMM for prediction.

Given that the actions repeat over time, we require a model that is ergodic. Since our model has only  $N$  and  $M$  defined we need to learn the three probability measures  $A$ ,  $B$  and  $\delta$  together known as the triple  $\tilde{e}$ , written as  $\tilde{e} = (A, B, \delta)$  in compact form. We use the Baum-Welch algorithm to train the model. The values of the triple are initialized randomly, uniformly or through a seeding procedure (these are discussed in the next section), and the training is essentially an expectation-modification (EM) procedure. Following the training of the model, the forward algorithm (Rabiner 1989) is used to calculate the probability of an observation sequence given the triple.

Typically, the forward algorithm is used to evaluate an observation sequence with respect to different HMMs and the model yielding the greatest probability is the model that best explains the sequence. However, we consider a different evaluation mechanism where we have a single model and many possible observation sequences. The observation sequence that best fits this model is used for prediction of the next action. This is accomplished as follows: given a model  $\tilde{e}$  and the observation sequence  $O_{i+1}O_{i+2} \dots O_{i+k}$ , we need to predict  $O_{i+k+1}$ . Symbols  $O_{i+1}O_{i+2} \dots O_{i+k}$  represent the observed recent history of actions. Since the predicted symbol can be any one of the  $M$  alphabet symbols, we have  $M$  possible sequences with the first  $k$  symbols in common but different last symbols.

The sequence that yields the greatest probability value for the model offers symbol  $k+1$  as the predicted next action. Since the probabilities associated with the model differ only slightly, we consider the top  $N$  predictions as valid predictions, where  $N$  is a configurable value. The probability distributions of the top  $N$  predictions will be fed to a decision maker, which will learn over time how the top  $N$  predictions are to be considered. Alternatively a user can be queried regarding the predictions suggested. Since this method requires remembering  $k$  symbols to make the next prediction, we need to remember at least  $k$  actions. But remembering the entire observation sequence can be prohibitive while making prediction for the next action, hence we have a sliding window that keeps only the last  $k$  symbols or actions, where  $k$  is a configurable value.

### 4.3 Initialization of Parameters

The last section mentions that the parameters of the HMM need to be initialized before using the Baum-Welch method to optimize the parameters. The initialization of the parameters can be done in one of three ways. The three approaches are: random initialization, uniform initialization and initialization through the process of seeding. Previously (Rao & Cook, 2003) we have shown the results when the *confusion matrix* is initialized using

the random method. In this paper we show results when the initialization of the *confusion matrix* is done through a process of seeding while the *transition matrix* and the *initial state vector* are initialized using the different techniques. Random initialization is done with a different seed each time so that we obtain different probability measure values. This does lead to evaluation problems since multiple runs are needed to average the results of the randomness we have to deal with. Another approach is to uniformly initialize the values of the two probability measures and this method produces more consistency in the results. This method of initialization is detrimental as the model size increases and our observations will show this to be true.

To seed the triple requires deliberately fixing the values of one or more of the probability measures that comprise the triple. Seeding is done prior to the Baum-Welch training so that the model is trained differently in the anticipation that the model thus obtained will present us with better predictions.

#### 4.3.1 SEEDING THE CONFUSION MATRIX

Seeding of the confusion matrix is done using the cluster centroids and variances simultaneously or the hidden states along with the observation vector or the inhabitant actions. The method of seeding is described now.

We know that matrix  $B$  represents the observation symbol probability distribution in state  $j$ , i.e.,  $B = \{b_j(k)\}$  and each  $b_j(k)$  represents the probability of observing symbol  $v_k$  at time  $t$  given that the state at time  $t$  is  $S_j$ , i.e.,  $b_j(k) = P[v_k \text{ at } t \mid q_t = S_j]$ . The observation vector is a sequence of observations where each observation is one of the  $M$  symbols in the set  $V$ . If a symbol has a higher likelihood of being part of a particular cluster we assign a high probability for the entry in the matrix that corresponds to observing this symbol upon reaching the corresponding state (or cluster, since they imply the same). If the likelihood is small or moderate, the appropriate probability is assigned. The value assigned is not a fixed value but a small range around the probability value computed. This is repeated for every symbol and for each of the hidden states.

#### 4.3.2 INITIALIZING THE OTHER PROBABILITY MEASURES

Once the matrix  $B$  is seeded, the matrix  $A$  and the vector  $\delta$  are initialized. We use random and uniform initializations for these two probability measures. In addition the matrix  $A$  can also be initialized through a seeding process. In this case, we seed the *transition matrix* using information about the clusters as well as the cluster transition information obtained from a classification mechanism. The classification mechanism uses cluster centers to classify a data point to a particular cluster. We classify the data points that were used for clustering, i.e., the partitions that were clustered are now classified. We determine the cluster to which each partition classifies to, and we obtain a cluster classification for each partition. The sequence of partitions is equivalent to a sequence of

cluster classifications that indicates which cluster or activity follows another.

Once the probability measures are initialized, the Baum-Welch method is run to optimize the parameters. The seeding of parameters influences the optimization and hence is an important step towards constructing the HMM. We show the performance for the different types of initializations in the section on experimental results.

## 5. Data Synthesis and Validation

We first created a synthetic data generator to validate our approach. We developed a model of a user's pattern which consists of different activities comprising different locations (e.g., kitchen, bathroom) and containing devices (e.g., TV, lamp). Devices are turned on or off according to the predefined pattern. Randomness is incorporated into the time at which the devices are used and in the inhabitant's activities using a Gaussian distribution. We generate different data sets corresponding to the usage of eight devices. Data set 1 has 10000 actions, corresponding to a period of 1250 days of activity. Data set 2 has 10000 actions corresponding to 1250 days of activity. Data set 0 has about 10000 actions, which corresponds to a period of 325 days of activity. In data set 1 the pattern was fixed except for the small differences in time of occurrence of the actions. Data set 2 has the same fixed patterns but there is a substantial difference in time between activities on weekdays and on weekends. This forces our approach to learn the same set of actions as two different tasks since they differ substantially in time. Data set 0 has more patterns and randomness than the other data sets.

Real data was obtained from a student test group who used X10 lamp and appliance modules to control devices in their homes. One such sample contained around 700 actions or data instances over a period of one month. The difference between the real and simulated data was that the real data contains noise over which we had no control. In contrast, the data we simulated had patterns cleanly embedded so as to test and evaluate our different approaches. The labeling of actions to tasks for the real data defeats the purpose of action prediction through recognition of patterns in the data and in most cases it is difficult to come up with labels unless the data is inspected. So the real data is useful for a test of how well the approaches do in practice.

We divide the data into training and testing data. For data set 1, data set 2 and data set 0 we divide the data into 10 sets of 1000 actions and perform the training and testing on each set of 1000 actions. For these data sets we vary the training data from 100 to 900 in steps of 200 and test on 100 subsequent actions. We also experiment with different values of the number of clusters, the sequence length and the allowable time difference. For data sets 1 and 2 we set the sequence length to 20 and the time difference (defined in section 3.1 as 'P' minutes) to 600 and the number of clusters is set to 3 and 6 respectively.

For lack of space we show in this paper only the significant results and refer the reader to Rao (2003) for an extensive set of experiments.

For the real data we perform similar experiments. The training data size we use varies from 100 to 500 in steps of 200. The values for the time difference and the length of action sequence to be retained are the same as for the simulated data. The value of the number of clusters used is set to 11, 23 and 47. For the real data we have to test with different value choices and these values were used as a representative of the number of different values that can be used.

For both the simulated and the real data, we employ the three different techniques for initialization of the probability measures as mentioned in section 4.3. We also use a single heuristic that employs both time and location information. Since the data is of a sequential nature, we do not perform a cross-validation for this data but average the results from multiple runs. We test the predictive accuracy of the simple Markov model and the Task-based Markov model by comparing each predicted action with the observed action and average the results over the entire test set for both the simulated and real data.

## 6. Experimental Results

In the set of experiments we describe, we seed the *confusion matrix* while the other probability measures are initialized using either random, uniform or seeding methods. The best and top 5 predictions for the simple Markov model yield similar results and hence we show only the top 5 predictions and compare these with the top 5 predictions suggested by the HMM.

In figure 1 we show the performance of the top 5 predictions of the simple Markov model against the predictions suggested by the HMM. The performance of the HMM and the simple Markov model for the top 5 predictions are similar and near accurate for this data set.

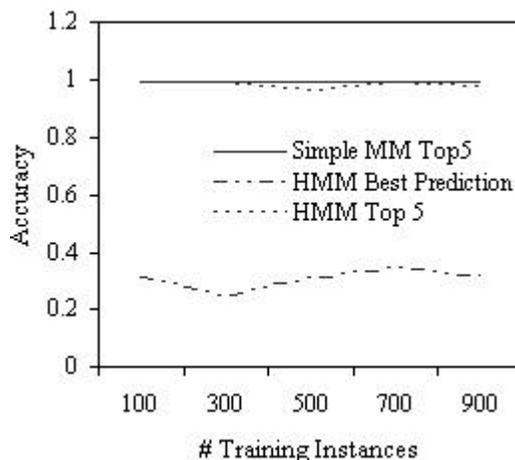


Figure 1. Data set 1 – Performance of simple Markov model and HMM (uniform initialization).

Observe that the best prediction of the HMM is low because of the added complexity introduced into the model. The action that is to be performed next is one of the actions among the different activities. The accuracy value increases if we were to consider the top 'N' choices for the next action. Using the knowledge of the top 'N' values we can consider eliminating wrong choices over a period of time through feedback. The method of feedback and improving the top 'N' values is one of the future work directions.

We compare the performance of the HMM using the different initializations – random, uniform and through seeding. We observe that the uniform initialization significantly outperforms the other two ( $p < 0.025$ ) and random outperforms the seeded method ( $p < 0.03$ ).

The data set 2 has patterns that are temporally shifted and has more randomness than data set 1. In figure 2 we see the performance of the simple Markov model and the HMM when the data set 2 is used. We see that the performance of the HMM for the top 5 actions is lower than for the simple Markov model. The uniform initialization and random initialization did comparably (significant to  $t=0.45$ ). These methods outperformed the seeding method ( $p < 0.02$  and  $p < 0.048$  respectively).

We observe in this experiment that the clustering of the partitions of actions is imperfect. Just as data set 1 can be hand labeled into patterns and clusters, so can this data set be hand labeled into clusters. We observe the imperfection in the clustering process by comparing the results of the clustering to the results obtained through hand labeling. This can be attributed to the difference in the time of actions between weekends and weekdays and hence an increase in model size.

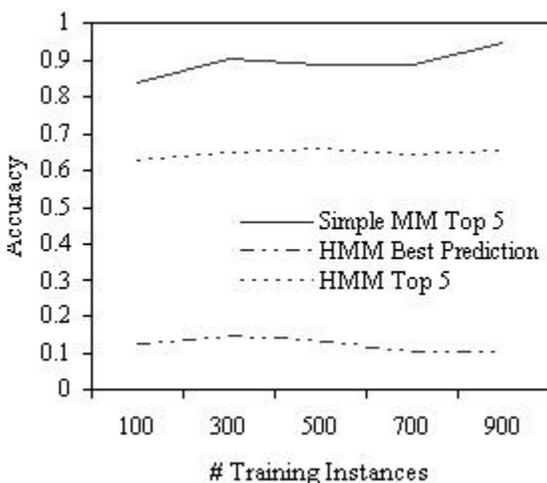


Figure 2. Data set 2 – Performance of simple Markov model and HMM (uniform initialization).

The effect of seeding improves the performance of the HMM and this is seen in the experiments that follow on the data set 0 and the real data set. The next experiment uses the data set 0 that has significantly more number of

patterns and randomness. We observe here that the seeding method performs better than the other initialization methods ( $p < 0.005$  for uniform and  $p < 0.023$  for random). We also observe that for certain values of the training instance the random initialization does better than the uniform method ( $p < 0.005$ ) but for some values (training size = 900) the two methods are comparable. In figure 3 we show the performance of the simple Markov model and the HMM. We observe that with increase in patterns and randomness there is a loss in the cluster quality. This affects the HMM from coming up with sequence of hidden states that best explains a sequence of actions. Thus, we see more state transitions and the symbols that are output reflect the likelihood of seeing the best symbol for a particular state. Thus the task identification for more disparate data sets becomes more difficult.

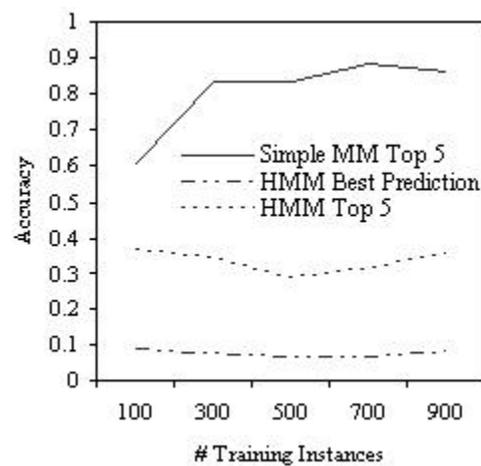


Figure 3. Data set 0 – Performance of simple Markov model and HMM (initialization through seeding).

However, as more clusters are introduced the seeding method performs better than the uniform initialization method. Similarly, we can conjecture that the greater the disparity in the data, the better the random initialization will perform since uniform initialization can be harmful leading to local minima when using the Baum-Welch method to train the HMM. Our purpose in testing these three techniques is to show that we can apply these different techniques for different types of data and observe the circumstances under which a particular technique performs well.

The HMM does not perform as well as the simple Markov model in the experiments with disparate data and we discuss why this is so.

- 1) The heuristics that are employed to partition the actions are not able to exactly divide these actions into tasks. This is because of the nature of the user pattern that interleaves actions from different tasks. Employing these heuristics will not determine whether the interspersing is deliberate and is likely to

be a task by itself or the mixing was a random occurrence.

- 2) The clustering of these partitions employs a Euclidean distance measure. The simple use of a distance function may not be sufficient towards clustering the tasks together. The similarity between the task instances may need to be considered apart from the dissimilarity feature.
- 3) When using HMMs, we are dealing with probabilities that are multiplied so that even a small change can cause significant changes in the best prediction.

In figure 4 we show the performance of the real data for the top 5 predictions of the simple Markov model and the HMM. We employ the seeding method to initialize the HMM and due to data insufficiency, we are not able to test for significance.

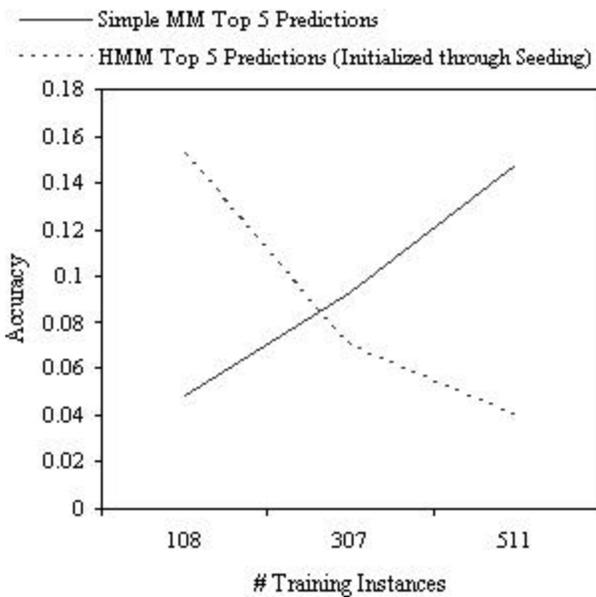


Figure 4. Real data – Performance of simple Markov model and HMM.

In the case of real data, the vast number of actions (devices along with the action time) and noise in the data hinders the simple Markov model from generating accurate predictions. To clean up noise in the real data will require us to know the actual task labels, which are not always available. We see that the simple Markov model improves its performance as the size of the training data increases. This is in contrast to what happens with the HMM.

The HMM does not perform well as the number of training instances increases. This is due in part to the noisy nature of the data sample we have. The decrease in performance is due to the increase in the number of symbols in the alphabet of the HMM that adds to the complexity in terms of increasing the model size as well as evaluating the forward probabilities when determining the next action. The effect is a decrease in the probability

of choosing the correct observation as the next action. Hence there is a decrease in the accuracy with an increase in the amount of training data. To verify the decrease is due to the complexity we vary the training instances and test on a fixed size of the test data size (200 instances). This is what is shown in figure 4. A random predictor that chooses among the many actions does not compare well. Using a random predictor the choice for a simple Markov model is one of the ‘N’ different states whereas for a HMM the choice is the alphabet size, that is one in ‘M’. For 300 training instances we observe 225 states added to the simple Markov model and 215 symbols added to the alphabet. The random predictor’s accuracy for simple Markov is 0.0044 and for the HMM is 0.0047.

In figure 5 we illustrate the performance of the HMM (initialized through seeding) when the number of clusters is varied. We observe here that the increase in the number of clusters decreases the accuracy, while accuracy drops when the amount of training instances is increased for the same number of clusters. Due to the limited data available we do not have significance tests for these experiments.

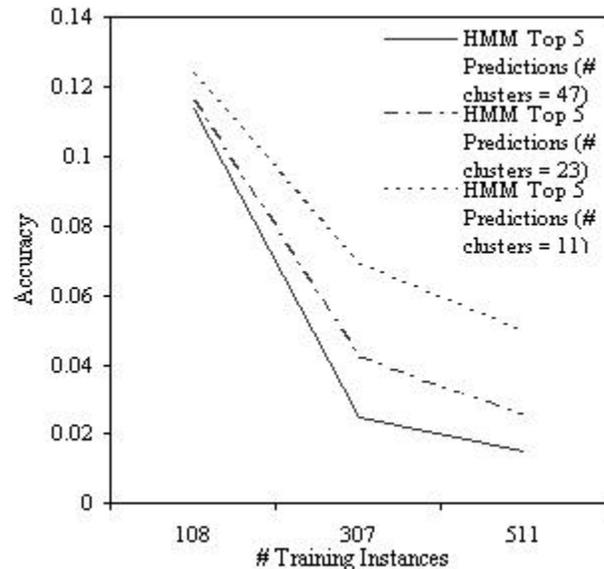


Figure 5. Real data – Performance of HMM for a range of the number of clusters.

We also ran tests varying the sequence length and the allowable time difference. We find that as the sequence length and the time difference increases, the accuracy increases. Further increase in sequence length or time difference offers no appreciable increase in accuracy and can be detrimental for large values.

## 7. Conclusions and Future Work

In this paper we have described our approach to predicting an inhabitant’s behavior in an intelligent environment such as a smart home. The role of prediction here is to provide assistance for home automation and adaptation to an inhabitant’s needs. To predict the next

action we use a simple Markov model that models each action as a state. Our enhancement to this model is to categorize the actions into abstract tasks and use this information to make subsequent predictions. This is part of our Task-based Markov model approach. TMM employs heuristics to group actions, clusters the groups and uses a HMM to perform prediction.

Our experiments show the performance of the simple Markov model and HMM for real and simulated data sets. The results reveal that HMM performs better than the simple Markov models for certain data sets. The results of the various experiments also illustrate that the real data and simulated data differ, and the solutions that work for simulated data do not work for the real data. Generalizing we can say that the choices of the parameters – allowable time difference, number of clusters and sequence length are important in deciding the overall accuracy. In spite of this, we cannot have a ‘one solution fits all’ policy. For real data this is especially true and only based on experience we can find a choice of parameters that performs reasonably well for a given data set.

We can say that although identifying abstract tasks of users is difficult given just a history of executed actions, what has been achieved is progress in the direction of task identification in an unsupervised mode. We obtain good clustering results for data such as the data sets 1 that has identifiable patterns. Thus the TMM finds a right model in certain instances that validates the identification of tasks. Identifying the task and its associated actions can be used to predict future actions.

One of the pressing issues for future work is to test the model using different sources of real data and verify the resulting performance. As part of this effort, we are currently generating a larger database of smart home activity for testing in our Artificial Intelligence lab.

Another effort that can be pursued is the use of multiple simple Markov models, where each model abstracts a task and is similar to a cluster. Once the clusters are obtained, a Markov model is constructed to encompass the actions of each cluster while also forming the transitions from one Markov model to another. We can also look into alternate methods of cluster generation and compare the performance of HMMs with the multiple simple Markov models. Other elements that need to be considered once we achieve reasonable predictions is the cost associated with correct and incorrect predictions and the use of feedback to alter the model.

### Acknowledgements

This work is supported by the National Science Foundation grant IIS-0121297.

### References

Baum, L. (1972). An inequality and associated maximization technique in statistical estimation of

probabilistic functions of Markov processes. *Inequalities*, Vol. 3, (pp. 1-8).

Cook, D. J., Youngblood, M., Heierman, III, E. O., Gopalratnam, K., Rao, S., Litvin, A., & Khawaja, F. (2003). MavHome: An Agent-Based Smart Home. *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications – PerCom 2003*, (pp. 521-524), March 2003.

Das, S. K., Cook D. J., Bhattacharya, A., Heierman III, E. O., & Lin, T-Y. (2002). The Role of Prediction Algorithms in the MavHome Smart Home Architecture. *IEEE Wireless Communications Special Issue on Smart Homes*, Vol. 9, No. 6, 2002, 77-84.

Davison, B. D., & Hirsh, H. (1998). Predicting Sequences of User Actions. *Predicting the Future: AI Approaches to Time Series Problems*, (Technical Report WS-98-07), (pp. 5-12), AAAI Press.

Freitag, D., & McCallum, A. (1999). Information extraction using HMMs and shrinkage. *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction*, (Technical Report WS-99-11), (pp. 31-36), AAAI press.

Gorniak, P., & Poole, D. (2000). Predicting Future User Actions by Observing Unmodified Applications. *National Conference on Artificial Intelligence*, AAAI 2000, (pp. 217-222), AAAI press.

Kukolich, L., & Lippmann, R. (1995). LNKNet User's Guide, <http://www.ll.mit.edu/IST/lnknet/guide.pdf>.

Luhr, S., Bui, H. H., Venkatesh, S., & West, G. A. W. (2003). Recognition of Human Activity through Hierarchical Stochastic Learning. *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications – PerCom 2003*, (pp. 416-422), March 2003.

Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257-285.

Rao, S. (2003). *Inhabitant Action Prediction in Smart Homes using Action and Task Models*. Master's thesis, Department of Computer Science and Engineering, University of Texas at Arlington.

Rao, S., & Cook, D. J. (2003). Improving the Performance of Action Prediction through Identification of Abstract Tasks. In *Proceedings of the 16th International FLAIRS-2003 Conference*, (pp. 43-47), AAAI press.

Seymore, K., McCallum, A., & Rosenfeld, R. (1999). Learning hidden Markov model structure for information extraction. *Proceedings of AAAI-99 Workshop on Machine Learning for Information Extraction*, (Technical Report WS-99-11), (pp. 37-42), AAAI press.