Improving the Performance of Action Prediction through Identification of Abstract Tasks

Sira Panduranga Rao and Diane J. Cook
Department of Computer Science and Engineering
The University of Texas at Arlington
Arlington, Texas 76019-0015
{sprao, 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 action, the next location, and the next task that an inhabitant is likely to perform. In this paper we describe our approach to solving the problem of predicting inhabitant behavior in a smart home. We model the inhabitant actions as states in a simple Markov model, then improve the model by supplying it with data from discovered high-level inhabitant tasks. For simulated data we achieved good accuracy, whereas on real data we had marginal performance. We also investigate clustering of actions and subsequently predict the next action and the task with hidden Markov models created using the clusters.

Key words – Markov models, action prediction, task-based or conceptual clustering, machine learning, agent learning.

Introduction

Our aim is to solve the prediction problem with a view towards home automation. In automating the home, the inhabitant's future actions can be performed without the inhabitant or user intervening. Although some actions may warrant a need for an inhabitant's intervention, we foresee that most actions can be performed autonomously. Automation will provide a start towards the home adapting to the inhabitant's needs. We believe our efforts can be utilized in an intelligent home (e.g., Das et al. 2002).

A user interacts with a variety of devices in a home and we term each such interaction as an action. The string '10/25/2002 8:15:31 PM Kitchen Light D1 ON' is a representation of one such action, where D1 is the device identification, 'Light' and '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 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 in reality what is supplied are just the low-level actions. Our goal is to learn which sets of actions constitute a well-defined task.

Work in the area of prediction has been done in other environments, such as predicting Unix commands (Korvemaker and Greiner 2000), predicting user action sequences (Davison and Hirsh 1998) and predicting future user actions (Gorniak and Poole 2000). Our task involves identifying what actions are useful to be modeled as states in a Markov model and discovering sets of actions that form abstract tasks. 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 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 will help 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.

The remainder of the paper presents our work in detail. The next section explains the construction of the simple Markov model. We then delve into the heuristics that are employed to group sets of similar actions and the clustering of these groups. In the following section, we discuss the application of hidden Markov models to represent tasks. Following this, we discuss generation of the simulated data and validation of our approach. Finally, we discuss the results and conclude with directions for future studies.

Modeling and Clustering of Actions

Inhabitant actions need to be accessible in some tangible form. Common approaches to representing this information include generating a sequence of historic actions as a string of action symbols and representing individual actions as separate states with corresponding state transitions. We adopt the latter approach in this work.

Markov model of actions

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 remembering large amounts of history. 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_1: *10/25/2002 8:15:31 PM Kitchen Light D1 ON* which can be represented as a state in the model. We construct smaller models by discretizing the time of the action and neglecting the date. Consider another action A_2: *10/28/2002 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_1 and A_2 differ only in the time and date features. If the time difference between the actions is minimal, the corresponding states can be merged.

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 given that they both refer to the same device and location but differ in the time and this time difference is insignificant. 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 helps achieve a moderate number of states while increasing state transition probabilities. If each action were to be considered as a single state, there would be a great number of states giving us no scope to come up with a good prediction approach.

Heuristics to Partition Actions

Based on the initial Markov model, 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. 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.

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 $A_1, A_2, \ldots, A_N$ we can divide these into groups $G_1, G_2, \ldots, G_P$, where each group consists of a set of actions $A_I, \ldots, A_{I+J}$ and for simplicity every action is part of only one group.

We separate an action sequence $A_x, \ldots, A_y$ into groups $A_x, \ldots, A_z$ and $A_{z+1}, \ldots, A_y$ if at least one of the following conditions holds:

1) The time difference between A_z and A_{z+1} is > 90 minutes.
2) There is a difference in location between the A_z and A_{z+1}.
3) The number of devices in the group is > 15.

The result of the partitioning step is a set of groups, or individual tasks, that are ready to be clustered into sets of similar tasks.

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_I and thus we have partitions $P_1, P_2, \ldots, P_P$. 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 partitional clustering algorithms, we apply k-means clustering to partitions $P_1, P_2, \ldots, P_P$.

To perform clustering, we utilize LNKnet (Kukolich and Lippmann 1995), a public-domain software package made available from MIT Lincoln Labs. We use a shell program to invoke the software from our procedure. The results of the clustering are parsed to extract the cluster centroids and the cluster variance values. We average the cluster centroids and variances over values generated from ten separate runs to promote generality of the results. The resulting clusters represent the abstract inhabitant tasks. In the context of Hidden Markov models (HMMs) these clusters can be used as hidden states. We next discuss how the HMMs are used to perform action prediction.

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, and Rosenfeld 1999) and creation of user profiles for anomaly detection (Lane 1999). HMMs can be either hand built (Freitag and McCallum 1999) or can be 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.

After the clustering process, the resulting clusters are added to a HMM as hidden states. The hidden states thus have the same features as that of the clusters as described earlier. The terms tasks and hidden states thus imply the same in our discussion.

Terminology

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 $H = \{H_1, H_2, \ldots, H_N\}$ 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 $V = \{v_1, v_2, \ldots, v_M\}$.

3) A, the state transition probability distribution, $A = \{a_{ij}\}$, where

$a_{ij} = P\ [q_{t+1} = S_j \mid q_t = S_i],\ \ 1 \le i, j \le N$

4) B, the observation symbol probability distribution in state j, $B = \{b_j(k)\}$, where

$b_j(k) = P\ [v_k \text{ at } t \mid q_t = S_j],\ \ 1 \le j \le N$ and $1 \le k \le M$

5) π, the initial state distribution, $\pi = \{\ \pi_i\}$, where

$\pi_i = P\ [q_1 = S_i],\ \ 1 \le i \le N$

6) O, the observation symbol vector or the sequence of observations, $O{=}O_1O_2 \ldots O_T$, where each observation O_t is one of the symbols from V and T is the number of observations in the sequence.

HMM framework

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 symbol (a modified form of an action) upon arriving at one of the N states. The distribution π gives the likelihood of each state representing the initial state.

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 π together known as the triple _, written as _ = (A, B, π) in compact form. We use the Baum-Welch algorithm to train the model. The values of the triple are initialized randomly and the training is essentially an 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 observation sequences and the observation sequence that best fits this model is used for prediction of the next action. This is accomplished as follows: given a model _ and the observation sequence $O_{i+1} \ldots O_{i+k}$, we need to predict O_{i+k+1}. This symbol can be any one of the M alphabet symbols, thus 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 3 or 5 or some other configurable value. 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.

Data Synthesis and Validation

In this section we discuss how we obtained the data for training and testing and how we validate our approach to solving the prediction problem.

Data Generation

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. Activities are different on weekdays than on weekends. We generated about 1600 actions that corresponded to a period of 50 days of activity. We developed a synthetic data generator for the purpose of generating this data. The time interval for generating the data, the device types and their locations are specified. The generator executes the specified patterns to produce the simulated data.

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 real data so obtained had noise since we had no control over it. Whereas, the data we simulated had patterns embedded so as to test our different approaches.

Validation

We divided the data into training and testing data. The training data size varies from 100 to 900 instances in steps of 100 for the simulated data and for the real data from 100 to 500 in steps of 100. The remaining data is used for testing. 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 hidden Markov model by comparing each predicted action with the observed action and average the results over the entire test set.

Results and Discussion

In this section we describe our experimental results. The term time difference has been mentioned earlier and we keep this as 5 minutes in our experiments. For the clustering process we vary the number of clusters generated. The results shown here are for 47 generated clusters. We also discuss the results when these two parameters are changed.

In Figure 1 we plot the performance of the simple Markov model and the hidden Markov model. The accuracy of the top prediction as well as the top 5 (N=5) predictions for both models are shown. As the number of instances increases, the simple Markov model does very well but eventually plateaus. For the HMM, the top prediction does not do well but the top 5 predictions performs reasonably well.

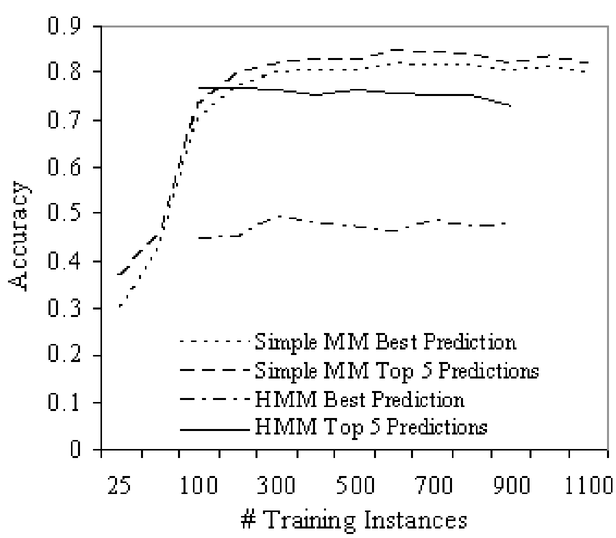

Figure 1: Performance of simple MM and HMM for simulated data.

In Figure 2, we compare the simple Markov model against the HMM when real data is used. Here, neither model performs well, but the HMM performs slightly better than the simple model. The best and top 5 predictions yield similar results for the simple Markov model. In the case of real data, the vast number of actions (devices along with the action time) as well as noise in the data hinders the simple Markov model from coming up with an efficient prediction algorithm. To clean up noise in the real data will require us to know what the actual tasks are and this defeats our purpose of finding the tasks to make better predictions. Also, in the figure we see that the best and top 5 predictions of the simple MM are almost equal and hence the overlap in the graphs.

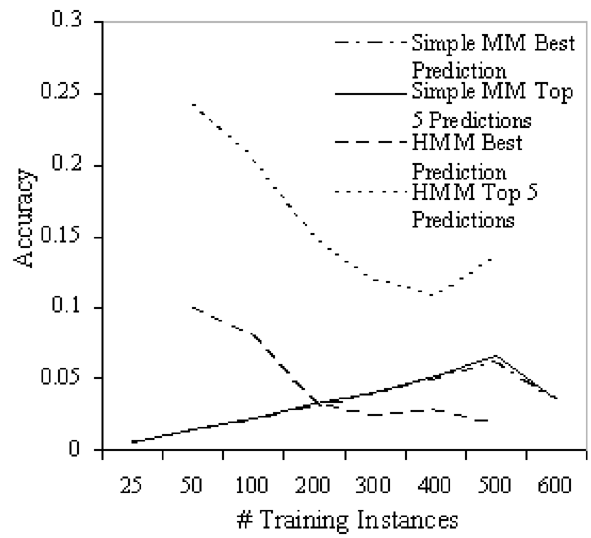

Figure 2: Performance of simple MM and HMM for real data.

The HMM does not perform as well either 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 inherent nature of the user pattern that has actions of different tasks interspersed. Employing these heuristics will not classify 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.

An improvement in the simple Markov models can be seen when we change the time difference feature, as shown in Table 1. However, the improvement in accuracy is slight. In Table 1, we vary the time difference keeping the number of training instances a constant. This value is 600 for the simulated data and 500 for the real data. We observe that as the time difference increases there is an improvement in the accuracy, but for higher time difference values, the performance suffers. This is due to the fact that actions that manipulate the same devices but at different times are now mapped to the same state. This generalization will lead to a smaller model. With this model accurate prediction of an action occurring at a certain time is not possible. In the case of real data, there is a constant increase in the predictive accuracy and beyond this the accuracy plateaus. The real data that we have needs more analysis and we expect that with more sets of real data and quantity in each set, a similar performance to that of the simulated data will be seen.

Time difference	Accuracy (Top5) Simulated data	Accuracy (Top5) Real data
100	0.52	0.02
300	0.84	0.046
600	0.92	0.137
1200	0.93	0.152
3600	0.872	0.273
10800	0.90	0.33
28800	0.88	0.32
57600	0.72	0.39

Table 1: Effect of change in time difference to predictive accuracy in simple Markov models.

We also observed the effect of the number of clusters on predictive accuracy for the HMM. We found that for few clusters, the accuracy is about 54-58%. As we increase the number of clusters, the accuracy increases to near 80% in some cases. Further increase does not greatly improve the accuracy.

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 modeling of an inhabitant's actions as states in a simple Markov model worked well on simulated data. The more we see the training instances, the lesser the number of states that are added because of the similarity between the action and the existing states. If we were to plot the number of states added for say, every 50 actions we will see a drop in this number as more training instances are seen.

Next, we refine this model by considering the abstract tasks that comprise the inhabitant's behavior. Hidden Markov models are used to make predictions based on the generated clusters. The HMM performs well on simulated data. The drop in precision for the real data for HMMs as the amount of training instances increases warrants a more detailed inspection. Tasks of users are difficult to identify given just the actions. What has been achieved is progress in the direction of task identification in an unsupervised mode.

Our immediate effort is to characterize the discrepancy in results between the real and simulated data sets. Looking at more data sets and characterizing them will be part of our future work. In addition, we are currently generating a larger database of smart home activity for testing. An alternative method of using cluster membership to seed the Markov model probability values is currently being investigated. We believe this will improve the performance of the task-based HMMs. An alternative effort that is being researched is the use of multiple single Markov models, where each model abstracts a task and is similar to a cluster. The use of abstract tasks for behavior prediction will also address scalability issues where large databases comes into play and using the simple Markov model will not suffice. Another element that needs to be considered once we achieve reasonable predictions is the cost associated with correct and incorrect predictions.

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, pages 1-8.

Das, S. K., Cook D. J., Bhattacharya, A., Heierman III, E. O., and 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., and Hirsh, H. 1998. Predicting Sequences of User Actions. *Predicting the Future: AI Approaches to Time Series Problems*, Technical Report WS-98-07, pages 5-12, AAAI Press.

Freitag, D., and 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, pages 31-36, AAAI press.

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

Korvemaker, B., and Greiner, R. 2000. Predicting Unix Command Lines: Adjusting to User Patterns. *National Conference on Artificial Intelligence*, AAAI 2000, pages 230-235, AAAI press.

Kukolich, L., and Lippmann, R. 1995. LNKNet User's Guide, http://www.ll.mit.edu/IST/lnknet/guide.pdf.

Lane, T. 1999. Hidden Markov Models for Human/ Computer Interface Modeling. *Proceedings of the IJCAI 99 Workshop on Learning about Users*, pages 35-44.

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

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