An Automated Prompting System for Smart Environments

Barnan Das[1,a] , Chao Chen[1,b], Adriana M. Seelye[2,a], Diane J. Cook[1,b]

[1] School of Electrical Engineering and Computer Science
[2] Department of Psychology
Washington State University, Pullman WA 99164, USA

[a]{barnandas, aseelye}@wsu.edu, [b]{cchen, cook}@eecs.wsu.edu

Abstract. The growth in popularity of smart environments has been quite steep in the last decade and so has the demand for smart health assistance systems. A smart home-based prompting system can enhance these technologies to deliver in-home interventions to a user for timely reminders or a brief instruction describing the way a task should be done for successful completion. This technology is in high demand with the desire for people who have physical or cognitive limitations to live independently in their homes. In this paper, we take the approach to fully automating a prompting system without any predefined rule set or user feedback. Unlike other approaches, we use simple off-the-shelf sensors and learn the timing for prompts based on real data that is collected with volunteer participants in our smart home testbed.

Keywords: prompting system; automated prompting; smart environments; sensor network; machine learning.

1 Introduction

With an increasing population of older adults in US [1] and rising trends of human care-giving facilities, assistive healthcare systems have started gaining popularity. A smart-home based prompting system is one such technology that delivers in-home interventions to a user for timely reminders or brief instructions describing the way a task should be done for successful completion. Intuitively, prompts in the context of a smart home environment can be defined as any form of verbal or non-verbal intervention delivered to a user on the basis of time, context or acquired intelligence that helps in successful (in terms of time and purpose) completion of an activity. Prompts can provide critical service in a smart home setting especially to older adults and inhabitants with some form of cognitive impairment.

In this paper, we describe the system architecture and functionality of our automated prompting system for smart home inhabitants named Prompting Users and Control Kiosk (PUCK). Developing automated prompting systems like PUCK requires a unique combination of pervasive computing and machine learning techniques. Using real data collected from our volunteer participants in our smart

home testbed, we learn the timing of the prompts. Moreover, we achieve this goal without direct user feedback unlike other existing prompting systems. In addition, a major challenge of dealing with a minority of prompting situations while training as compared to a vast majority of situations that do not require a prompt, is highlighted and a prospective solution is proposed.

2 Related Work

Reminder systems have been in existence for quite some time now with different research groups taking their own unique approach to solving the problem. From a machine learning perspective these approaches can be broadly classified into four types: rule based (time and context), reinforcement learning, planning and supervised learning. Most of the early and modern reminder systems are rule based. In this approach, a set of rules is defined based on time, the context of an activity and user preferences. Lim et al. [2] designed a medication reminder system that recognizes the reminders suitable for medication situation. Rudary et al. [3] integrated temporal constraint reasoning with reinforcement learning to build an adaptive reminder system. Although this approach is useful when there is no direct or indirect user feedback, it relies on a complete schedule of user activities. The Autominder System [4] developed by Pollack et al. provides adaptive personalized activity reminders using a dynamic Bayesian network as an underlying domain model to coordinate preplanned events. Boger et al. [5] designed a Markov decision process-based planning system that used video inputs to determine when and how to provide prompts to dementia patients for guidance through the activity of hand washing.

In our current study, we use environmental sensors that do not intervene in people's day to day lives. For the sake of privacy concerns we avoid the use of video or audio inputs. Instead, the sensors used by us are inexpensive and can be deployed in a few hours. Moreover, our system and learning models are not reliant on user feedbacks. In term of learning models, we take a supervised learning approach in order to make the prompts more accurate and the system more robust.

3 System Architecture

PUCK is not just a single device but a framework (Fig. 1) that helps in providing automatic interventions to inhabitants of a smart home environment. In our current work we have been able to reach the phase where the system is able to predict a relative time in the activity when a prompt is required after learning intensively from training data collected over a period of time. We are in the process of using touch screen interface for delivering audio cues along with images that are relevant to the activity for which the prompt is being given. The system architecture of PUCK can be broadly divided into four major modules:

- **Smart Environment:** This is the smart home infrastructure that acts as a testbed where experiments are performed. It has a sensor network that keeps track of the activities performed and stores the data in a SQL database in real time.

- **Data Preparation:** The portion of raw sensor data that would be used by the learning models are collected from the database. Features or attributes that would be helpful in differentiating a "Prompt" step from a "No-Prompt" step are generated. As there are very few training examples that have "Prompt" steps, we use a sub-module, namely Sampling to generate new and unique "Prompt" examples.
- **Machine Learning Model:** Once the data is prepared by the Data Preparation Module, we employ machine learning strategies to identify whether a prompt should be issued. This module is the "brain" of the entire system where the decision making process takes place.
- **Prompting Device:** This device acts as a bridge between the user or inhabitant and the digital world of sensor network, data and learning models. Prompting devices can range from simple speakers to basic computers, PDAs, or even smart phone (a project that we are currently pursuing).

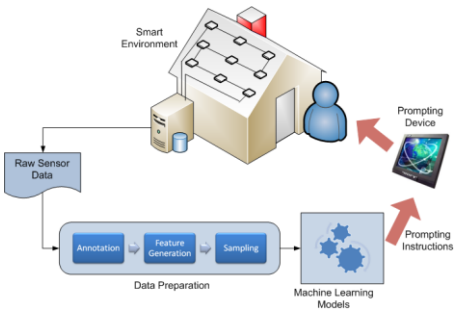

Fig. 1. System Architecture of the PUCK.

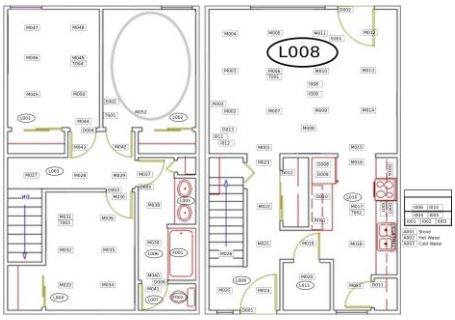

Fig. 2. Three-bedroom smart apartment used for data collection (Sensors: motion (M), temperature (T), water (W), burner (B), telephone (P) and item (I)).

4 Experimental Setup

Testbed. The data collection is done in collaboration with the Department of Psychology of the university. The smart home testbed is a 2 story apartment located on the university campus. It contains a living room, dining area and kitchen on the first floor and three bedrooms and bathroom on the second. All of these rooms are equipped with a grid of motion sensors on the ceiling, door sensors on the apartment entrances and on cabinets, the refrigerator and the microwave oven, temperature sensors in each room, a power meter, and analog sensors for burner and water usage. Fig. 2 depicts the structural and sensor layout of the apartment. Data was collected while volunteer older adult participants performed activities in the smart apartment. One of the bedrooms on the second floor is used as a control room where the experimenters monitor the activities performed by the participants (via web cameras) and deliver pre-defined prompts through an audio delivery system whenever necessary. The goal of PUCK is

to learn from this collected data how to time the delivery of prompts and ultimately to automate the role of the experimenter in this setting.

The following activities are used in our experiments: Sweep, Refill Medication, Birthday Card, Watch DVD, Water Plants, Make Phone Call, Cook and Select Outfit. These activities are subdivided into relevant steps by the psychologists in order to track their proper completion. Fig. 3 shows an example of the Cooking task. Participants are asked to perform these specific activities in the smart apartment. While going through the steps of an activity, a prompt is given if he/she performs steps for other activities rather than the current one, if a step is skipped, if extra/erroneous steps are performed, or if too much time has elapsed since the beginning of the activity. Note that there is no ideal order of steps by which the activity can be completed. Therefore, a prompt is given only when one of the conditions mentioned above occurs. Moreover, the goal is to deliver as few prompts as possible. The experimenters keep track of all the errors done by the participants and the steps at which a prompt was delivered, which are later extracted and used to train PUCK.

Cooking

1. Participant retrieves materials from cupboard.
2. Participant fills measuring cup with water.
3. Participant boils water in microwave.
4. Participant pours water into cup of noodles.
5. Participant retrieves pitcher of water from refrigerator.
6. Participant pours glass of water.
7. Participant returns pitcher of water.
8. Participant waits for water to simmer in cup of water.
9. Participant brings all items to dining rooms table.

Fig 3. Steps of Cooking Activity

```
2009-05-11 14:59:54.934979D010   CLOSE  7.3
2009-05-11 14:59:55.213769M017   ON     7.4
2009-05-11 15:00:02.062455M017   OFF
2009-05-11 15:00:17.348279M017   ON     7.8
2009-05-11 15:00:34.006763M018   ON     7.8
2009-05-11 15:00:35.487639M051   ON     7.8
2009-05-11 15:00:43.028589M016   ON     7.8
2009-05-11 15:00:43.091891M015   ON     7.9
2009-05-11 15:00:45.008148M014   ON     7.9
```

Fig 4. Annotated data snippet

Annotation. An in-house sensor network captures all sensor events and stores them in a SQL database in real time. The sensor data gathered for our SQL database is expressed by several features summarized in Table 1. These four fields (Date, Time, Sensor, ID and Message) are generated by the data collection system.

Table 1. Sample of sensor events used for our study

Date	Time	Sensor ID	Message
2009-02-06	17:17:36	M45	ON
2009-02-06	17:17:40	M45	OFF
2009-02-06	11:13:26	T004	21.5
2009-02-05	11:18:37	P001	747W
2009-02-09	21:15:28	P001	1.929kWh

After collecting data, sensor events are labeled with the specific activity and step within the activity, {activity#.step#}, that was being performed while the sensor events were generated, as shown in Fig 4.

Feature Generation. From the annotated data we generate relevant features that would be helpful in predicting whether a step is a “Prompt” step or a “No Prompt”

step. After the features are generated, the modified form of the data set contains steps performed by participants as instances. This data set is then re-annotated for the prompt steps. Table 2 provides a summary of all generated features.

Table 2. List of Features and Description

Feature #	Feature Name	Description
1	stepLength	Length of the step in time (seconds)
2	numSensors	Number of unique sensors involved with the step
3	numEvents	Number of sensor events associated with the step
4	prevStep	Previous step
5	nextStep	Next step
6	timeActBegin	Time (seconds) since beginning of the activity
7	timePrevStep	Time (seconds) between the end of the previous
8	stepsActBegin	Number of steps since beginning of the activity
9	activityID	Activity ID
10	stepID	Step ID
11	M01 … M51	Frequency of firing each sensor during the step
12	Class	Binary Class. 1=”Prompt”, 0=”No Prompt”

5 Theoretical Background

From machine learning perspective, PUCK works on classification algorithms. The purpose of the classifiers is to predict if a particular step of an activity needs a prompt or not. That is, it can be viewed as a binary class learning problem. However, PUCK delivers prompts only when it is critically important rather than at every possible step, which would make it a cause of annoyance rather than an aid. Intuitively, there are far more "no-prompt" instances in the dataset than "prompt" instances because there are very few situations that would require a prompt. Thus, the data in this domain is inherently skewed or in other words the data has a class imbalance problem.

We use data collected from 128 older adult participants, with different levels of mild cognitive impairment (MCI), to train the learning models. Thus the errors committed by the participants are real and not simulated. There are 53 steps in total for all the activities, out of which 38 are recognizable by the annotators. The participants were delivered prompts in 149. Therefore, approximately 3.74% of the total instances are positive (“Prompt” steps) and the rest are negative (“No-Prompt” steps). This creates a bias in the classifiers to learn more on positive class and thus perform better on them than the negative class. Therefore, performance metrics that measure the classification performance on positive and negative classes independently are considered. True Positive (TP) Rate: represents the percentage of activity steps that are correctly classified as requiring a prompt; True Negative (TN) Rate: represents the percentage of correct steps that are accurately labeled as not requiring a prompt; Area Under ROC Curve (AUC): evaluate overall classifier performance without taking into account class distribution or error cost; and Accuracy (Acc): conventional accuracy of classifiers. Learning models are described in the following:

Decision Tree: A decision tree [6] classifier uses information gain to create a classification model, a statistical property that measures how well a given attribute separates the training examples according to their target classification. In our experiments, we use the J48 decision tree.

k - Nearest Neighbor: The k-Nearest Neighbor is an instance based learning method [7] in which all instances correspond to points in an n-dimensional space. Instance neighbors are calculated using Euclidean distance. For any value of k, the algorithm assigns a class label to a data point that represents the most common value among the k training examples which are nearest to the data point.

Support Vector Machines: Support Vector Machines (SVMs) is a non-probabilistic binary linear classifier [8] that learns a hyperplane or set of hyperplanes which separates a series of positive data instances and a series of negative data instances with maximum margin. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data points of any class, since in general the larger the margin the lower the generalization error of the classifier.

As mentioned earlier, the purpose of PUCK is not to prompt an inhabitant in every step of an activity but to deliver the prompt only for steps where individuals need help to complete the task. Therefore, although the classical machine learning algorithms can have high accuracies, they are not suitable for our purpose as they either fail to predict the steps in which the prompt should be fired or do so with poor performance. In order to resolve this issue we use a technique known as Sampling as described below.

Sampling is a technique of rebalancing the dataset synthetically and can be accomplished by under-sampling or over-sampling. However, under-sampling can throw away potentially useful data, oversampling can overfit the classifier if it is done by data replication. As a solution SMOTE [9] uses a combination of both under and over sampling, but without data replication. Over-sampling is performed by taking each minority class sample and synthesizing a new sample by randomly choosing any or all (depending upon the desired size of the class) of its *k* minority class nearest neighbors. Generation of the synthetic sample is accomplished by first computing the difference between the feature vector (sample) under consideration and its nearest neighbor. Next, this difference is multiplied by a random number between *0* and *1*. Finally, the product is added to the feature vector under consideration. In our dataset the minority class instances are not only small in terms of percentage of the entire dataset, but also in absolute number. Therefore, if the nearest neighbors are conventionally calculated (as in original SMOTE) and the value of k is small, we would have null neighbors. Unlike SMOTE, in our algorithm the k-nearest neighbors are calculated on the basis of just two features: *activityID* and *stepID*. Under-sampling is done by randomly choosing a sample of size *k* (as per the desired size of the majority class) from the entire population without repetition.

6 Experiments and Discussion

All the experiments are evaluated with 10 fold cross validation to see how well the learning models perform at the task of predicting timing (in terms of activity steps) of prompts.

As discussed earlier due to the inherent imbalanced class distribution in the dataset, the classical machine learning algorithms fail drastically on the original dataset.

Therefore, we adopt the method of sampling the original dataset. The purpose of sampling is to rebalance a dataset by increasing the number of minority class instances, enabling the classifiers to learn more relevant rules on positive instances. However, there is no ideal class distribution. A study done by Weiss et al [10] shows that, given plenty of data when only *n* instances are considered, the optimal distribution generally contains 50% to 90% of the minority class instances. Therefore, in order to empirically determine the class distribution we consider J48 as the baseline classifier and repeat the experiments, varying percentages of minority class instances from 5% up to 95%, by increments of 5%. A sample size of 50% of the instance space is chosen.

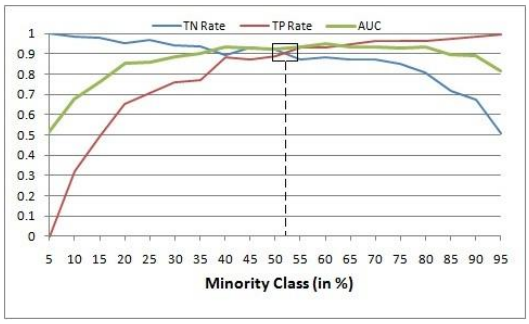

Fig 5: TP Rate, TN Rate and AUC for different class distributions

While, any lower size will cause loss of potential information; any higher size will make the sample susceptible to overfitting. From Figure 5 we see that the *TP* rate increases while the *TN* rate decreases as the percentage of the minority class is increased. These two points intersect each other at some point that corresponds to somewhere between 50-55% of minority class. Also, AUC is between *0.923* and *0.934* near this point. Therefore, we decide to choose 55% of minority class to be the appropriate sample distribution for further experimentation.

Three different algorithms, namely, J48, IBk (a k-nearest neighbor algorithm) and SMO are run on the sampled dataset. From Figure 6 (a) it is seen that the *TP* rate has increased tremendously for all the algorithms without compromising too much with *TN* rate (shown in Figure 6(b)). Also the area under ROC curve has also increased (Figure 6 (c)) indicating that the overall performances of all the learning methods have increased. Clearly, sampling encouraged the learning methods to learn more rules for the positive class. But, it should also be noted that average accuracy decreases by a few percentage. This is acceptable till the *TP* rate is high.

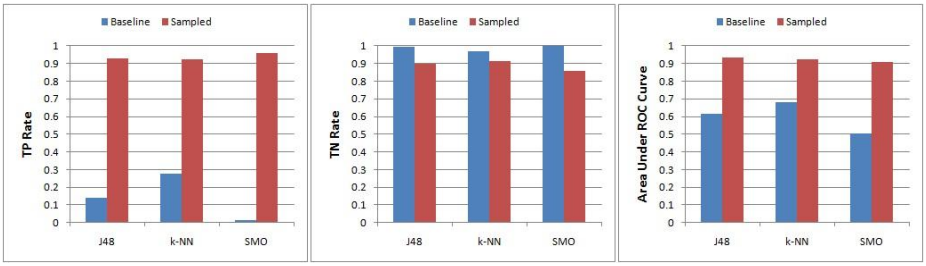

Fig 6: Comparison of (a) TP Rates, (b) TN Rates and (c) Area Under ROC Curve

5 Conclusion and Future Work

In this paper we described PUCK, a system that automates activity prompts in a smart home environment by identifying the steps at which prompts are required. We discussed the framework in which the prompting system was developed and some challenges faced. We also mentioned some noteworthy properties that are characteristic to the domain of automated prompting.

In our future work, we want to predict the prompts in real time and automating its content. For making PUCK more acceptable to a larger population of older adults and cognitively impaired people, it is essential to determine most appropriate prompt in real time. In a current project we are developing a smart phone prompting interface, as we understand, physical proximity of the recipient to the prompting interface is a crucial factor, and problems like recipient failing to hear the prompts, might be avoided by making a phone the prompting interface that people can always carry with them.

References

1. U.S. Census Bureau. International database. Table 094. Midyear population, by age and sex, http://www.census.gov/population/www/projections/natdet-D1A.html
2. Lim, M., Choi, J., Kim, D., Park, S.: A Smart Medication Prompting System and Context Reasoning in Home Environments. In: Proceedings of the 2008 Fourth International Conference on Networked Computing and Advanced Information Management, vol 0, pp 115-118 (2008)
3. Rudary, M., Singh, S., Pollack, M. E.: Adaptive Cognitive Orthotics: Combining Reinforcement Learning and Constraint-Based Temporal Reasoning. In: Proceedings of the 21st International Conference on Machine Learning, pp 719-726 (2004)
4. Pollack, M., Brown, L., Colbry, D., McCarthy, C., Orosz, C., Peintner, B., Ramakrishnan, S., Tsamardinos, I.: Autominder: An intelligent cognitive orthotic system for people with memory impairment. In: Robot.Auton. Syst., vol. 44, pp. 273-282 (2003)
5. Boger, J., Hoey, J., Poupart, P., Boutilier, C., Fernie, G., Mihailidis, A.: A decision-theoretic approach to task assistance for persons with dementia. In: Proceedings for International Joint Conferences on Artificial Intelligence, pp 1293 -1299 (2005)
6. Quinlan, J.R.: Induction of Decision Trees, Machine Learning, Vol. 1, No. 1, pp81-106 (2005)
7. Mitchell, T.: Machine Learning. McGraw Hill (1997)
8. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory 5, pp144-152, Pittsburgh (1992)
9. Chawla, N. V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W. P.: SMOTE: Synthetic Minority Over-sampling Technique. In: Journal of Aritificial Intelligence Research, 16:321-357, (2002)
10. Weiss, G.M., Provost, F.: The effect of class distribution on classifier learning: An empirical study. In: Technical Report ML-TR44, Rutgers University, Department of Computer Science (2001)