Published in final edited form as:
Pers Ubiquitous Comput. 2012 October 1; 16(7): 859–873. doi:10.1007/s00779-011-0445-6.

PUCK: An Automated Prompting System for Smart Environments:

Towards achieving automated prompting; Challenges involved

Barnan Das,
School of Electrical Engineering and Computer Science, EME 130 Spokane Street, Box 642752, Washington State University, Pullman, WA 99164-2752, Tel.: +1-208-596-1169

Diane J. Cook,
School of Electrical Engineering and Computer Science, EME 121 Spokane Street, Box 642752, Washington State University, Pullman, WA 99164-2752, Tel.: +1-509-335-4985, Fax: +1-509-335-3818

Maureen Schmitter-Edgecombe, and
Department of Psychology, Johnson Tower 312, P.O. Box 644820 Washington State University, Pullman, WA 99164-4820, Tel.: +1-509-335-0170, Fax: +1-509-335-5043

Adriana M. Seelye
Department of Psychology, Johnson Tower 321, Washington State University, Pullman, WA 99164-4820

Barnan Das: arnandas@wsu.edu; Diane J. Cook: cook@eecs.wsu.edu; Maureen Schmitter-Edgecombe: schmitter-e@wsu.edu; Adriana M. Seelye: aseelye@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 users for timely reminders or brief instructions describing the way a task should be done for successful completion. This technology is in high demand given the desire of people who have physical or cognitive limitations to live independently in their homes. In this paper, with the introduction of the “PUCK” prompting system, we take an approach to automate prompting-based interventions without any predefined rule sets 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 test bed. The data mining approaches taken to solve this problem come with the challenge of an imbalanced class distribution that occurs naturally in the data. We propose a variant of an existing sampling technique, SMOTE, to deal with the class imbalance problem. To validate the approach, a comparative analysis with Cost Sensitive Learning is performed.

Keywords

Automated prompting; Prompting systems; Smart environments; Machine learning; Imbalanced class distribution

1 Introduction

1.1 Problem Definition

A "prompt" 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 a task. Prompts can provide critical service in a smart home setting especially for older adults and inhabitants with cognitive impairment. Prompts can remind individuals to initiate an activity or to complete incorrect or missing steps of an activity. However, a number of challenges rise when creating prompting systems:

- **Problem Identification:** When and for which tasks are the prompts necessary?
- **Justification:** When and for which tasks would the prompts be effective?
- **Prompt Granularity:** Which tasks require what level of prompting granularity (in terms of activity step detail)?
- **Media:** What type of prompt is most effective (audio, video or multimodal)?
- **Users Environment:** What is the physical layout of the home and how does this affect the timing and mode of prompts?

Our current work deals with developing an automated prompting system that can guide a smart home inhabitant through various steps of activities. The goal of our prompting system, called "PUCK" (Prompting Users and Control Kiosk) is to identify when an activity step has been missed or performed erroneously and deliver an appropriate prompt when one is required. In other words, the goal of PUCK is to learn the timings of a prompt within an activity. A unique combination of pervasive computing and machine learning technologies is required to meet this goal.

In this paper, we describe a prompting system architecture that accomplishes our goal. In addition, we highlight a fundamental challenge of this domain: learning the appropriate timing of prompts when the vast majority of training situations do not require prompts. By proposing a solution to the problem in the form of a sampling technique SMOTE-Variant, we introduce an approach to overcome this challenge. We validate the effectiveness of our approach by performing a comparative analysis with another commonly used technique to deal with imbalanced class distribution problem, known as Cost Sensitive Learning. Moreover, earlier prompting systems have relied upon either direct or indirect feedback from the user. In contrast, we solve it without any user feedback as we believe that in a real world smart environment housing residents with cognitive limitations, it would not be possible to rely upon or even consistently receive user feedback.

1.2 Motivation

Research in the area of smart environments has gained popularity in the last decade. Most attention has been directed towards health monitoring and activity recognition [Maurer et al (2006) Maurer, Smailagic, Siewiorek, and Deisher, Singla et al (2009) Singla, Cook, and Schmitter-Edgecombe]. Recently, assistive health care systems have started making an

impact in society, especially in countries where human care-giving facilities are expensive and a large population of adults prefers an independent lifestyle. According to the studies conducted by the US Census Bureau [Bureau (2010)], the number of older adults in the US aged 65+ is expected to increase from approximately 35 million in 2000 to an estimated 71 million in 2030, and adults aged 80+ from 9.3 million in 2000 to 19.5 million in 2030. Moreover, there are currently 18 million people worldwide who are diagnosed with dementia and this number is predicted to reach 35 million by 2050 [Bates et al (2004) Bates, Boote, and Beverley]. These older adults face problems completing both simple (e.g. eating, dressing) and complex (e.g. cooking, taking medicine) Activities of Daily Living (ADLs) [Wadley et al (2008) Wadley, Okonkwo, Crowe, and Ross-Meadows].

We note that real-world caregivers do not perform all activities for the care recipient, nor do they prompt each step of a task. Instead, the caregiver recognizes when the care recipient is experiencing difficulty with an activity and provides a prompt at that time to help in performing the activity completely. The number of prompts that a caregiver typically provides depends upon the level of cognitive impairment. Worsening of the level of impairment demands an increased number of caregiver duties and thus places a heavier burden on the caregiver. Caregivers for individuals with dementia are also at increased risk for health problems, including higher levels of stress hormones, reduced immune function, slower wound healing, new hypertension, higher serum insulin levels and related markers of diabetes, cardiovascular disease, increased morbidity and premature death [Dementia (2010)]. Therefore, an automated computerized system that would be able to provide some of the facilities of a human caregiver is the call of the hour and may reduce health risks and alleviate the burden of many caregivers that are helping a large section of the population.

1.3 Contribution

There are a number of contributions of this paper. We proposed a system architecture useful for developing an automated prompting system for smart home inhabitants. Using real data collected from volunteer participants in our smart home testbed, we learn the timing of the prompts. Moreover, we achieve this goal without direct user feedback. In addition, we discuss the class imbalance challenge that arises in this problem domain and describe a prospective solution. Finally, our approach is evaluated by comparing it with an existing technique.

2 Related Work

2.1 Automated Prompting Systems

Reminder systems have been in existence for quite some time now, the simplest form being an alarm clock which is used to provide an alarm tone at a time that is determined by the user. As the technology for building innovative reminder or prompting systems is flourishing, research groups are taking their own unique way of solving the problem. From a machine learning perspective the 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.

[Lim et al(2008) Lim, Choi, Kim, and Park] designed a medication reminder system that recognizes the service (composed of a digital health frame, medicine chest and medication prompting application) suitable for a medication situation. Oriani et al [Oriani et al(2003) Oriani, Moniz-Cook, Binetti, Zanieri, Frisoni, Geroldi, De Vreese, and Zanetti] developed an electronic memory aid that allows a user or caregiver to prerecord messages (e.g. reminders to complete a task) that can be played back to the user at predefined times. Rudary et al. [Rudary et al(2004) Rudary, Singh, and Pollack] 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. Autominder [Pollack et al(2003) Pollack, Brown, Colbry, McCarthy, Orosz, Peintner, Ramakrishnan, and Tsamardinos], developed by Pollack et al., provides adaptive personalized activity reminders using a dynamic Bayesian network to coordinate preplanned events. The system is deployed on the Pearl nursebot. Pineau et al. used a variant of partially ordered Markov decision process [Pineau et al(2003) Pineau, Montemerlo, Pollack, Roy, and Thrun] to design a high level control system for robots that can assist older adults with daily activities. Boger et al. designed a Markov decision process-based planning system [Boger et al(2005) Boger, Poupart, Hoey, Boutilier, Fernie, and Mihailidis] that used video inputs to determine when and how to provide prompts to dementia patients for guidance through the activity of hand washing. The work by Weber et al. [Weber and Pollack(2007)] focuses on online active learning for interactive calendar management using an entropy-driven technique.

In the context of automated prompting systems, it should be noted that any computational approach chosen to solve the problem requires human intervention to some extent. But, inspite of this limitation, the system can be considered as automatic because it does not require the presence of a human caregiver after it is deployed. From the computational perspective, a machine learning technique is *more* automatic than a rule-based system. In context of the PUCK system, it can be argued that if defining rules determining prompt situations can automate the system post deployment, what is the use of manually annotating large datasets of sensor events. A counter argument would be that the level of prompt granularity that has been considered in the current project, is nearly impossible to satisfy with human defined rules. Rule sets could suffice prompting situations that look at activities as a whole, such as time to initiate an activity or specific contextual situations, but they would not be sufficient for prompting situations that consider activity steps. Therefore, the outcome of PUCK is worth the effort of manual annotation of large datasets of sensor event.

2.2 Handling the Class Imbalance Problem

A dataset is said to have a skewed imbalanced class distribution when one class is largely under-represented in comparison to the others. In other words, the dataset has a much lower number of instances of one class than the other classes. Several solutions have been proposed to deal with this issue. These methods are broadly classified into data-level methods and algorithm-level methods. Data level methods mainly include under-sampling the majority class [Kotsiantis and Pintelas(2003)] to match the size of the other class, over-sampling the minority class [Kubat and Matwin(1997)] to match the size of the other class and combination of both under and over sampling as proposed by Chawla et al [Chawla et

al(2002) Chawla, Bowyer, Hall, and Kegelmeyer]. Algorithm level methods include: threshold method [Weiss(2004)] in which the classifiers yield a score that represents the degree to which an example is a member of a class; one-class learning [Raskutti and Kowalczyk(2004)]; and cost-sensitive learning [Elkan(2001)] in which unequal misclassification cost is considered between classes.

In our current study, we use environmental sensors that do not interface with inhabitants' 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 feedback. In term of learning models, we take a supervised learning approach. Instead of prompting the participant each time a potential error is detected, we prompt in situations where a human caregiver would normally prompt. This allows the prompt timing to be more accurate and better emulate human intervention. We develop SMOTE-Variant to make the original SMOTE better able to process the data collected in our domain.

In this context, is would be good to mention that detecting prompting situations is similar to the detecting confident events from data sources. Event Detection is the task of monitoring a data source and detecting the occurrence of an event that is captured within that source. The source of complexity for the current application is an array of heterogeneous mix of various sensors thats are used in our smart environment. In the Watchdog project [Keally et al(2010) Keally, Zhou, and Xing] a modality-agnostic based event detection framework has been proposed that clusters the right sensors to meet user specified detection accuracy during runtime while significantly reducing energy consumption. Neill et al. [Neill and Cooper(2010)] take a multivariate Bayesian scan statistics (MBSS) approach that integrates prior domain knowledge for a highly powerful detector of emerging patterns. Singliar et al. [Singliar and Hauskrecht(2010)] defined detection requirements, developed, [3] and analyzed over different parameter ranges, a series of detectors for traffic incidents handling streaming data affected by noise and data that is not aligned.

3 System Architecture

PUCK is not just a single device but a framework that helps in providing automatic interventions to inhabitants of a smart home environment. Therefore, this framework (Figure 1) includes every component necessary for a working prompting system, including data collection, data preparation and learning algorithms. 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. In the past we have deployed rule based and context aware prompts in the homes of two participant older adults. In these deployments we used touch screen monitors with embedded speakers to deliver the prompts. The prompts included audio cues along with images that are relevant to the activity for which the prompt is being given. We are in the process of using the same interface for the automated prompting system.

The system architecture of PUCK can be broadly comprises of four major modules:

- **Smart Environment:** 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. This data is manually annotated and made suitable for feature generation. Features or attributes that would be helpful in differentiating a “Prompt” step from a “No Prompt” step are generated. Because there are very few training examples that are labeled as “Prompt” steps, we use a Sampling sub-module 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 is the primary decision making module of the entire system.
- **Prompting Device:** The prompting device acts as a bridge between the resident and the digital world that contains the sensor network as well as the data and learning models. Prompting devices can range from simple speakers to basic computers, PDAs, or even smart phones.

In the following sections, we provide descriptions of the functionality of each of these modules and how every single one them contributes towards the goal of achieving automated prompting.

4 Data Collection

The data collection is done in collaboration with Washington State University’s Department of Psychology. Participants in these experiments are volunteers, aged 50 or older, who are either healthy older adults, individuals with mild cognitive disorders (MCI) or individuals diagnosed with a progressive dementia, most commonly Alzheimer’s disease.

4.1 Test bed

The smart home testbed is a two-story apartment located on the Washington State University campus. It has 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 entrance and on doors for cabinets, refrigerator and microwave oven, item sensors on containers in the cabinet, temperature sensors in each room, a power meter, analog sensors for burner and water usage, and a sensor that keeps track of telephone use. For this particular study we have only considered the data gathered by motion, door and item sensors. Figure 2 depicts the structural and sensor layout of the 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 prompts through an audio delivery system whenever necessary. The goal of PUCK is to automate the role of the experimenter in this setting.

4.2 Experimentation Methodology

The experiments are conducted by psychologists in an on-campus smart apartment testbed. The participants are requested to perform different activities of daily living which are monitored by the experimenters via web cam from the control room. The following set of eight ADLs are considered for our experiments:

1. Sweep and Dust	4. DVD Selection and Operation	7. Prepare Meal
2. Fill Medication Dispenser	5. Water Plants	8. Outfit Selection
3. Write Birthday Card	6. Converse on Phone	

These activities are subdivided into relevant steps to track their proper completion. Although the detailed description of all the activities and their steps is beyond the scope of this paper, we present the steps of the activity “Prepare Meal” as an example to illustrate how every activity is subdivided into individual steps that need to be completed to meet the goal of the activity. This activity requires participants to use the supplies located in a kitchen cupboard to prepare a cup of noodle soup according to the direction on the container. The participant also had to fill a glass with water using the pitcher of water located on the top shelf of the refrigerator.

Prepare Meal	
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.

An argument against the use of predefined steps is that, different individuals might have different ways of completing an activity. But, the division of the steps has been done by the psychologist after studying the common nature or pattern of individuals to proceed in an activity. Although, there is no ideal order of the steps for activity completion, a subsets of these steps are critically important for successful completion of activities.

The following stepwise procedure gives a vivid description of the methodology used by the psychologists to conduct the experiments and helps us understand the relevance from the perspective of data mining and the view point of computer scientists:

1. Participants are asked to perform activities in the smart apartment.

2. When an activity is about to begin, an instruction (with basic details of the activity) is delivered to the participant.
3. The participant, while going through the steps of the activity, is given a prompt in the following conditions:
    - The participant performs steps for other activities and not the current one.
    - Steps irrelevant to the current activities are being performed.
    - A specified amount of time has elapsed since the initiation of the current step.
4. After the activity is completed, the participant is asked a set of questions that help in determining the difficulties they might have faced while doing the activity or following the prompts.
5. The instruction of the next activity is given and the same process continues.

Note that there is no single or required order of steps by which the activity can be completed. For many activities there are many sequences of activity steps that can lead to accurate task completion. Therefore, a prompt is given only when one of the conditions mentioned in step 3 occurs. Moreover, the goal is to deliver as few prompts as possible. The experimenters keep track of all the errors committed by the participants and the steps at which a prompt was delivered, which is a later retrieved and used to train PUCK.

4.3 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.

After collecting data, the sensor events are annotated with the corresponding activities (as shown below) that were being performed while the sensor events were generated.

Activities are labeled with their corresponding activity IDs (as listed in Section 4.2) and step IDs. The steps are in the format of ⟨*ActivityID*⟩.⟨*StepNumber*⟩.

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

For example, 7.4 would indicate the fourth step of the seventh activity “Cooking”, i.e., “Participant pours water into cup of noodles”.

Because the annotated data is used to train the learning models, the quality of annotation is very important for the performance of the system. Generation of a large number of sensor data events in a smart home environment makes it difficult for researchers and users to interpret raw data into residents’ activities [Szewcyzk et al(2009) Szewcyzk, Dwan, Minor, Swedlove, and Cook] without the use of visualization tools. Therefore to enhance the quality of the annotated data, we built an open source Python Visualizer, called PyViz [Thomas and Crandall(2011)], to visualize the sensor events. Figure 3 shows the user interface of PyViz in our project. It is a graphical visualizer application that represent smart home inhabitant motion in real-time or from an archived raw sensor data file. It uses a Scalable Vector Graphics (SVG) file to represent the floor plan and sensor layout of the smart environment, in which, motion and temperature sensors are marked in circles, door sensors marked as rectangles and so on. PyViz visually highlights sensors that are triggered so provide an annotator or an analyst with a clearer understanding of the participants movement and object interaction in the apartment. PyViz has an option to label sensor events as they are displayed, which provides a more interactive mechanism for the analyst to label events with the corresponding activity steps

As mentioned earlier, the participants in our study are older adults. Some are healthy, some have MCI (Mild Cognitive Impairment), and some have dementia. Compared to healthy older adults, MCI and dementia participants required more prompts across the eight activities. In general, MCI participants were able to get through the activities with prompting assistance. Because of their more severe cognitive impairment, Dementia participants had more difficulty completing activities with the prompting assistance. When dementia participants were unable to successfully complete an activity, it was due to reasons such as forgetting the activity steps, getting distracted in the middle of the activity or performing steps in errorneous ways. Many of the MCI and dementia participants may struggle with an activity because they forget the steps, get distracted in the middle of the activity or perform steps in erroneous ways. As some of these steps are not critical for activity completion, it is acceptable if the non-critical steps are missed. However, in most of the cases the activity is left incomplete because of distraction. Many of the participants, particularly those in the MCI category, were benefited from the prompts and were able to complete activities with this intervention.

4.4 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. Each step of an activity is treated as a separate training instance, and we define pertinent features to describe the step based on sensor data. This is done by considering each raw data sequence, that represents an activity step, at a time. Features are generated on this data sequence and converted into a feature vector that acts as a single instance in the dataset. Each data instance is tagged with the class value. Specifically, a step at which a participant received a prompt, is marked as “1” indicating “Prompt”, others are hence assumed to be “No Prompt” steps and marked as “0”.

Table 2 provides a summary of all generated features. It should be noted that the machine learning models learn a mapping from a data pointed described by these features to a Prompt / No prompt class label. In this way, PUCK addresses the problem of deciding when a prompt needs to be delivered by predicting of an activity step is a “Prompt” instance.

It should be noted that in order to deploy this system as a real-time prompt situation predictor, there needs to be a separate real-time activity recognition module. The activity recognition module will keep on tagging every sensor event with an activity and a relevant step. Contiguous sensor events tagged as a particular activity step would be considered as a sensor data sequence for that activity step and features would be generated on it.

The temporal features in the table, such as feature *stepLength*, *timeActBegin* and *timePrevAct* calculate the time in seconds. A common problem with MCI patients is that they get confused in the middle of an activity and just do nothing until the caregiver issues to prompt to proceed. These situations will get represented in the temporal features as unusually higher values as compared to others. This makes the job of the classifiers easier in determining that something went wrong in such a situation and thus issue a prompt. Therefore, the timeout in determining a prompt situation is implicitly taken care of by the classifier. There are some features which could be NULL, such as *prevStep* and *nextStep*. *prevStep* could be NULL for the very first step of activities. Similarly, the *nextStep* could be NULL for the last step of activities. To note: it might appear that *activityID* and *stepID* could also be NULL, but it is not the case. The sensor events that do not belong to any activity or any step within an activity are not considered for feature generation. Features are generated only on the data sequence that belongs to a specific activity step. The features *M01…M51* are the frequencies of sensor triggering associated with an activity step. It is determined from the sensor event sequence that belongs to each activity step. While each step has sensor associations with specific locations of the apartment, there could also be multiple step association for some sensors. For example, “Prepare Meal” and “Fill Medication Dispenser” activities take place in the kitchen for our smart environment setting and thus multiple motion and door sensors are associated with these two activities.

5 Dataset and Performance Metrics

Before designing a system that would be smart enough to automate prompts, it is essential to understand the nature of the dataset and specify performance metrics that are suitable for analyzing the effectiveness of the machine learning models. We use data collected from 128 participants to train the learning models. There are 53 steps in total for all the activities, out of which 38 are recognizable by the annotators. The rest of the activities could not be a recognized as they involved object interaction and the PUCK infrastructure did not have object interaction sensors back then. The participants were delivered prompts in 149 cases which involved any of the 38 recognizable steps. Therefore, approximately 3.74% of the total instances are positive, indicating “Prompt” steps, and the rest are negative, indicating “No-Prompt” steps. Positive and negative classes have also been synonymously used as minority and majority classes respectively. Essentially, this means that predicting all the instances as negative, would give more than 96% accuracy even though all the predictions for positive instances were incorrect.

Conventional performance measures such as accuracy and error rate consider different types of classification errors as equally important. For example, the purpose of our work is not to predict whether a prompt should not be delivered in a step, but to predict when to fire the prompt and with what efficiency. An important thing to keep in mind about this domain of automated prompting is that false positives are more acceptable than false negatives. While a prompt that is delivered when it is not needed is a nuisance, that type of mistake is less costly than not delivering a prompt when one is needed, particularly for a resident with dementia. In addition, considering that the purpose of the research is to assist people by delivering a lesser number of prompts, there should be a trade-off between the correctness of predicting a "prompt" step and the total accuracy on the entire system. This has been discussed in Section 7.1 in further detail.

As a result, we consider performance measures that directly measure the classification performance on positive and negative classes independently. The True Positive (TP) Rate (for positive or the minority class) here represents the percentage of activity steps that are correctly classified as requiring a prompt; the True Negative (TN) Rate here represents the percentage of correct steps that PUCK accurately labels as not requiring a prompt. TP and TN Rates are thus capable of measuring the performance of the classifiers separately for the positive and negative classes. ROC curve analysis is used to evaluate overall classifier performance. An ROC curve plots the classifiers false positive rate [Provost et al(1998) Provost, Fawcett, and Kohavi] on the x-axis and the true positive rate on the y-axis. We generate an ROC curve by plotting the ROC obtained by varying different parameters of PUCK. The primary advantage of using these is that they illustrate PUCKs performance without taking into account class distribution or error cost. We report AUC, or the area under ROC curve [Hand(1997)], in order to average the performance over all costs and distributions. Also, the geometric mean of TP Rate and TN Rate is denoted by G_{acc} which is commonly used as a performance metric in imbalanced class learning, has been used in our evaluation. G_{acc} is given by: $\sqrt{TP\ Rate \times TN\ Rate}$. To evaluate overall effects of classification the conventional accuracy of classifiers is considered.

6 Learning Models

6.1 Background

Decision Tree—A decision tree classifier [Quinlan(1986)] 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. Information gain is a measure based on entropy, a parameter used in information theory to characterize the purity of an arbitrary collection of examples. It is measured as:

$$Entropy(S) \equiv -p_+ log_2 p_+ - p_- log_2 p_- \quad (1)$$

where, S is the set of data points, p_+ is the number of data points that belong to the positive class and p_- is the number of data points that belong to the negative class. The information gain for each attribute is as follows:

$$Gain(S,A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{S} Entropy(S_v) \quad (2)$$

where *Values*(*A*) is the set of all possible values for feature *A. Gain*(*S*, *A*) measures how well a given feature separates the training examples according to their target classification. In our experiments, we use the J48 decision tree provided with the Weka distribution.

Support Vector Machines—Support Vector Machines (SVMs) were first introduced in 1992 [Boser et al(1992) Boser, Guyon, and Vapnik]. This is a training algorithm for data classification which maximizes the margin between the training examples and the class boundary. The SVM learns a hyperplane which separates a series of positive data instances and a series of negative data instances with maximum margin. Each training data instance should contain one class label and several features. The target of an SVM is to generate a hyperplane which provides a class label for each data point described by a set of feature values.

The class boundary of SVM can be solved by the following constrained optimization problem.

$$minimize \frac{1}{2}\|w\| \quad subject\ to: y_i(w^t x_i + b) \geq 1 \quad (3)$$

To introduce a non-linear kernel function, the optimal problem can be converted into a dual form which is a quadratic programming problem:

$$\begin{gathered} L_d \equiv \sum_i a_i - \frac{1}{2}\sum_{i,j} a_i a_j y_i y_j K(x_i, x_j) \\ 0 \leq a_i \leq C \ \ and \ \ \sum a_i y_i = 0 \end{gathered} \quad (4)$$

The target function can be computed by:

$$f(x) = sign\left(\sum_{i=1}^{N} a_i y_i k(x_i, x) + b\right) \quad (5)$$

For a traditional SVM, the quadratic programming problem involves a matrix, whose elements are equal to the number of training examples. If the training set is large, the SVM algorithm will use a lot of memory. To solve such a problem, Sequential Minimal Optimization (SMO) [Platt(1998)] decomposes the overall quadratic programming problem into a series of smaller quadratic programming problems. During the training process, SMO picks a pair of Lagrange multipliers (a_i; a_j) in each iteration and solves the quadratic programming problem, then repeats the same process until it converges on a solution. SMO significantly improves the ability to scale and the computation time for SVMs.

Boosting—Boosting, a type of ensemble method [Dietterich(2000)], manipulates the training examples to generate multiple hypotheses. It is an approximation to additive

modeling on the logistic scale using maximum Bernoulli likelihood as a criterion for a binary class problem. AdaBoost, an abbreviation for Adaptive Boosting [Freund and Schapire(1996)], is the most commonly used boosting algorithm. By giving higher weights to the training examples that are misclassified, AdaBoost builds a powerful classifier iteratively according to the performance of the previous weak classifiers. AdaBoost fits an additive $F(x)=\sum_{m=1}^{M} c_m f_m(x)$ to the training data. It minimizes the expectation of an exponential loss function $E(e^{-yF(x)})$. However, the performance of AdaBoost is harmed by noisy data because the expectation of an exponential loss function changes exponentially.

To solve such a problem, LogitBoost [Hastie and Tibshirani(2000)] was formulated by Friedman et al. This approach uses adaptive Newton steps for fitting an additive symmetric logistic model by maximum likelihood. The algorithm minimizes the expectation of the loss function to fit an additive logistic regression model to directly optimize a binomial log-likelihood represented by $-\log(1 + e^{-2yF(x)})$. The property of LogitBoost changes linearly with the output error and is therefore less sensitive to noisy data.

6.2 Experiments with Original Dataset

All of our experiments are run with 10 fold cross validation to see how well the learning models perform at the task of predicting the timing (in terms of activity steps) of prompts. In 10 fold cross validation the data is split into 10 segments, out of which 9 segments are used for training and one for testing. To reduce variability, multiple rounds of cross-validation are performed, and the validation or test results are averaged over all the rounds. As can be seen from Figure 4 (a), the usage of classical machine learning algorithms on our original dataset obtains a high accuracy. However, Figure 4 (b) depicts that the TP Rates are extremely low as compared to the TN Rates.

From this experiment we can conclude that traditional classifiers are able to effectively learn to recognize the positive instances of the dataset. The reason for this is that the dataset has a highly imbalanced class distribution; it is more skewed towards negative instances than positive.

There can be number of reasons for the dataset to be skewed. In our case, there is a domain-specific reason for the data to be skewed towards the negative class. As mentioned before, 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, in spite of having such high accuracies these algorithms are not suitable for our work as they either fail to predict the steps in which the prompt should be fired or do that job with poor performance.

6.3 Reasons for Failure of Learning Algorithms

Decision trees do not take all attributes into consideration to form a hypothesis. The inductive bias is to prefer a smaller tree over larger trees. Moreover, like many other learning methods (e.g. rule based), a decision tree searches for a hypotheses from a hypotheses space that would be able to classify all new incoming instances. While doing so,

it prefers shorter hypothesis trees over longer ones and thus ignores unique properties of the data points that might be represented by attributes not included in the tree.

As the number of attributes is quite high (61 in our case), the SVM algorithm, SMO, constructs a set of hyperplanes for the purpose of classification. Usually a good separation is achieved by a hyperplane that has the largest distance to the nearest training data points of any class (the functional margin). In the case of PUCK, the positive class instances are not too difierent from negative class instances, because out of 61 attributes, only a few attributes like *stepLength* and *timeActBegin* are difierent for positive instances as compared to negative instances. Therefore, the functional margin is quite short causing a lower TP rate.

In our Boosting technique, a learning algorithm is run several times, each time with a difierent subset of training examples. Dieterrich demonstrated [Dietterich(2000)] that this technique works well for algorithms whose output classifier undergoes major changes in response to small changes in the training data, also known as unstable learning algorithms. As a decision tree is an unstable algorithm, we use Decision Stump [Iba and Langley(1992)], a decision tree with root immediately connected to the terminal nodes, as the base learner.

7 Dealing with Imbalanced Class Distribution

In the following sections, we discuss two difierent techniques for dealing with this problem of imbalanced class distributions.

7.1 Sampling

A common solution to deal with imbalanced datasets is to re-balance them synthetically. This is a widely known technique in the area of data mining referred to as Sampling. Sampling is accomplished in two ways: under-sampling, in which the majority class is shortened by ignoring some of its instances, and oversampling, in which the number of instances in the minority class is increased by some means. It should be noted that both under-sampling and oversampling are done taking into account the balancing factor (percentage of minority class in the sample) on a fixed sample size.

However, both under-sampling and oversampling have their own dificulties [Monard and Batista(2002)]. While under-sampling can throw away potentially useful data, oversampling can cause the classifier to overfit as a common technique is to replicate data and thus letting the classifier to formulate rules on insuficient or replicated data. There has been some research to overcome both of these drawbacks at the same time. The most common of them is the SMOTE method [Chawla et al(2002) Chawla, Bowyer, Hall, and Kegelmeyer] proposed by Chawla, et al. It is a combination of both under-sampling and over-sampling, where over-sampling is not done by just replicating the positive instances but by the generation of new instances which are similar to others. We describe a new mining approach in the following discussion that represents a variation of SMOTE applicable to this type of situation.

Over-sampling—In the original SMOTE method, over-sampling is done by taking each minority class sample and synthesizing a new sample by randomly choosing any/all (depending upon the desired size of the class) of its *k* minority class nearest neighbors. Generation of the synthetic sample is done in the following way:

1. Compute the difierence between the feature vector (sample) under consideration and its nearest neighbor.
2. Multiply this difierence by a random number between 0 and 1.
3. Add the product to the feature vector under consideration.

In our dataset the minority class instances were not only small in terms of percentage of the entire dataset, but also in absolute number. Therefore, if the nearest neighbors are conventionally calculated and the value of *k* is taken to be small, we would have null neighbors. Due to these limitations on the behavior of our dataset we could not use the SMOTE directly and had to make variations that would suit our domain. Unlike original SMOTE, we do not find the *k* nearest neighbors for all instances in the minority class. We randomly pick an instance from the class and consider its nearest neighbors as all minority class instances that have the same *activityID* and *stepID*. In most of the cases the number of such instances is as low as 2 or 3. One of these neighbors is randomly chosen and the new instance is synthesized the same way as done in SMOTE. Note that all the randomized functions are based on the pseudorandom number generator, Mersenne Twister [Matsumoto and Nishimura(1998)], developed by Matsumoto and Nishimura. The algorithm on the next page summarizes the approach.

Under-sampling—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. It is again based on the Mersenns Twister pseudo-random number generator.

Effect of Class Distribution—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, we note that there is no ideal class distribution. In addition to domain dependency, the optimal class distribution also depends upon the classifier for which we want to observe the improvement. A thorough study done by Weiss et al [Weiss and Provost(2001)] shows that, given plenty of data when only *n* instances are considered due to computational restrictions, the optimal distribution generally contains 50% to 90% of the minority class instances in the entire dataset.

Algorithm

SMOTE-Variant (T, N, M)

Input: Instances of minority class T, Number of minority class samples N, Desired number of minority class samples M

Output: M minority class samples stored in list S

if (M >N)

then S = T //*S stores all the instances in T*

for i ← 1 to M - N

```
        t = randomize(T) //Randomly chosen instance from T
        neighbor[ ] = null //List storing all the nearest neighbors of instance t
      for j ← 1 to length(T)
          if (t.activityID==T[j].activityID) AND (t.stepID==T[j].stepID)
              neighbor[ ].append(T[j]) //Appending the list with the neighbor
      endfor
      s = randomize(neighbor) //Randomly choose neighbor instance
      diff = t - s //Both t and s are vectors, so this is a vector subtraction
      rand = random number between 0 and 1
      newInstance = t + diff * rand
      S.append(newInstance)
    endfor
  endif
```

From the results shown in the previous section, it has already been demonstrated that the natural class distribution is not the best for training a classifier. Moreover, decision trees like J48 are not inherently capable of handling imbalanced class distribution. Therefore, in order to empirically determine the class distribution of our domain we consider J48 as the baseline classifier. If our new approach performs better on J48, it can be assumed that it will do so on the classifiers that are inherently capable of handling imbalanced datasets. We repeat the experiments by varying percentages of minority class instances in the dataset, from 5% up to 95%, by increments of 5% and by considering the sample size of about 50% of the original instance space. As mentioned before, we not only want a better TP rate but also a fairly good TN rate. We plot the TP and TN rates and the Area Under the ROC Curve in Figure 5.

In the domain of automated prompting and while dealing mainly with older adults it is necessary to predict actual prompt steps as prompt steps, that is high True Positive (TP) rate, but it is not suficient. It is also critical to ensure that not too many false positives are generated (i.e., that not too many no-prompt steps are classified as prompt steps), because this situation can be a source of annoyance for older adults. To satisfy this criterion, we consider a percentage of minority class which balances both the TP and TN rate. In Figure 5, the plots of TP an TN Rates intersect somewhere near 55% of minority class instances in the dataset. This is also validated by a substantial increase in AUC. The intesection of TP and TN rate plots represents a proper trade off between boosting TP rate without compromising too much with TN rate. Therefore, we use the 55% minority class dataset for the remaining experiments in this paper.

7.2 Cost Sensitive Learning

The goal of most classical machine learning techniques is to achieve a high classification accuracy; in other words, minimize the error rate. In order to accomplish this, the difference between difierent misclassification errors is ignored, assuming that the costs of all misclassification errors are equal. This assumption drastically fails in many real world applications where the experimenters are more interested in the minority class. For example,

in our system, the cost of not issuing a prompt when it is critically required is far greater than the cost of issuing a prompt when it is not required.

In this paper, we take only misclassification costs into consideration, although there are some other costs like data acquisition costs, computation costs, etc., that can be considered [Turney(2010)]. Note that the costs considered in these discussions are not necessarily monetary. It can be wastage of time, severity of illness, etc. In generic terms, “any undesired outcome” can be considered a “cost”.

We consider a confusion matrix shown in Table 3 which also represents a cost matrix [Elkan(2001), Weiss et al(2007) Weiss, McCarthy, and Zabar] for difierent categories of classified instances and is denoted by *C* with the classification category in subscript. These misclassification costs can be assigned from the domain expert knowledge or can be learned via different techniques.

Mathematically, let (*i; j*) in C_{ij} be the cost of predicting class *i* when the actual class is *j*. When *i* = *j*, the prediction is correct and incorrect otherwise. Given the cost matrix, an example is classified as class *i* with the minimum expected cost by using the Bayes risk criterion:

$$L(x)=arg\,min_i \left(\sum_{j\in\{0,1\}} P(j|x)C(i,j) \right) \quad (6)$$

where *L*(*x*) is a sum over the alternative possibilities for the true class of *x* and *P*(*j*|*x*) is the posterior probability of classifying an example *x* as class *j*.

In this work we use a meta-learning based cost sensitive learner CostSensitiveClassifier, proposed by Witten and Frank [Witten and Frank(2005)], which predicts the class with the smallest expected misclassification cost. Let us illustrate this with an example. Consider the classifier assigns classes 0 or 1 to a test instance with probability p_0 and p_1 and the cost matrix:

$$\begin{bmatrix} 0 & 5 \\ 1 & 0 \end{bmatrix} \quad (7)$$

If it predicts 1, the expected cost of the prediction is obtained by multiplying the second column of the matrix [5,0] by the probability vector [p_0; p_1], yielding a normalized value of $^5p_0$. Similarly, the cost of predicting class 0 is p_1.

As mentioned earlier, the cost matrix can be hard coded by the domain expert or it can be learned. In this paper, we take an empirical approach to determine the cost matrix. The following points should be noted about the cost matrix:

- **i.** For correct predictions there are no costs, i.e. $C_{00}=C_{TN}$ and $C_{11}=C_{TP}$ are 0.
- **ii.** As the number of false positives (FP) is low, we work with the default misclassification cost of 1, i.e. C_{FP} = 1.

iii. The false negatives (FN) are critically important in out domain. Therefore, we repeated the experiments with different values of C_{FN} to determine a near ideal cost matrix.

We repeat the experiments using J48 decision tree as the baseline classifier and varied the value of C_{FN} from 2 to 100 to determine the cost matrix that works best for our dataset. In Figure 6, we see that as C_{FN} increases the *TN* rate decreases, but not drastically. On the other hand, *TP* rate takes a steep increase until $C_{FN} = 70$ when it becomes fairly constant. At $C_{FN} = 90$ it takes a rises a bit and stays constant henceforth. Therefore, at $C_{FN} = 70$, when the *TP* rate is 0:705, the *TN* rate is 0:899 and there is a fairly good AUC of 0:817, the cost matrix is:

$$\begin{bmatrix} 0 & 70 \\ 1 & 0 \end{bmatrix} \quad (8)$$

We consider this cost matrix for further experimentation on rest of the algorithms.

8 Experiments and Discussion

Experiments with sampled dataset (with 55% minority class) are performed on the classifiers. On the other hand, the classical classifiers are made cost sensitive with the help of meta-learner *Cost Sensitive Classifier* and tested on the original dataset. In Figure 7 we compare TP Rates, AUC and G_{acc} for difierent approaches across different classifier. Tables 4, 5 and 6 gives a summary of all the performance metrics. From Figure 7, it can be said that both the techniques of handling class imbalance help in achieving better performance than using the classical machine learning methods directly. On the other hand, our proposed technique SMOTE-Variant performs better than cost sensitive learning (CSL). One possible reason could be that, with an overwhelming increase in minority class instances helps the classifiers learn most of the rules for classification. On the other hand, in order to achieve this level of performance with CSL we need to compromise with accuracy performance on negative class which is not advisable.

The TN rates have dropped by a fair amount in CSL (Table 6) which has caused a drastic decrease in accuracy. Decreases in accuracy by 5–6% in the original dataset are acceptable, as in case of sampling. However, in CSL the accuracy decreases more than 26% for LogitBoost. This decrease in accuracy in CSL, caused by a major decrease in the TN Rate, essentially means that PUCK will deliver prompts even when they are not required. Determination of a threshold of tolerance for unnecessary prompts is a matter that needs to be considered further in the context of user tolerance and human factors.

An interesting thing to note is that the TN Rate of J48 in CSL is comparable to the TN Rate for sampling. This might imply that, even though empirically determining a cost matrix is a good idea to get the near best performance, the cost matrix cannot be generalized over all algorithms. Every algorithm has its own way of handling the hypothesis and building the objective function.

Therefore, it would be a good idea to empirically determine the cost matrix separately for all the algorithms. We tried to find a better cost matrix for SMO and LogitBoost and found that, on the basic cost matrix used in our experiments a lower C_{FN} as compared to the J48 achieves better performance. For SMO and LogitBoost, a cost value for C_{FN} which is close to 30 can achieve a greater than 0:8 TN Rate. In future, we would like to deal with this issue and try to find cost matrices for the classifiers separately.

9 Conclusions and Future Work

In this paper we introduced 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 is developed and some challenges faced. We also mentioned some noteworthy properties that are characteristic to the domain of automated prompting. We proposed SMOTE-Variant sampling technique that helps in dealing with skewed datasets. Moreover, we validated our approach by making a comparative study with Cost Sensitive Learning.

One limitation of this work is that the experiments were conducted in a highly structured and controlled smart apartment, which might limit the generalizability of the finding to other more real-world smart environment settings. In addition, the majority of participants were well educated and Caucasian, which might not be representative of larger population of older adults with MCI and dementia. Finally, this work currently requires manual annotation of a large dataset on individual sensor events, which limits its flexibility and generalizability to more real-world environments.

In our future work, we want to learn the timing of prompts at a finer time granularity. This means that we will identify prompt situations for each sensor event, not just for each activity step. In addition, we will consider automatically generating prompt content. To ensure that PUCK will be accepted by the target population of older adults and individuals with cognitive impairment, the content and delivery of prompts needs to be carefully selected and adapted to the individual's needs in real time. To extend the applicability of the prompting system we are currently developing a smart phone prompting interface. Problems such as the individual not hearing a prompt in a large home might be avoided by using a phone that individuals typically keep on or near them.

Acknowledgments

This work was supported by the United States National Institutes of Health Grant R01EB009675 and National Science Foundation Grant CRI-0852172.

References

Bates J, Boote J, Beverley C. Psychosocial interventions for people with a milder dementing illness: a systematic review. Journal of Advanced Nursing. 2004; 45(6):644–658. [PubMed: 15012642]

Boger J, Poupart P, Hoey J, Boutilier C, Fernie G, Mihailidis A. A decision-theoretic approach to task assistance for persons with dementia. In: International Joint Conference on Artificial Intelligence, Citeseer, vol. 2005; 19:1293.

Boser, B.; Guyon, I.; Vapnik, V. A training algorithm for optimal margin classifiers. Proceedings of the fifth annual workshop on Computational learning theory; ACM; 1992. p. 144-152.

Bureau UC. International database. table 094. 2010. URL http://www.census.gov/population/www/projections/natdet-D1A.html

Chawla N, Bowyer K, Hall L, Kegelmeyer W. Smote: synthetic minority over-sampling technique. Journal of Artificial Intelligence Research. 2002; 16(1):321–357.

Dementia A. Alzheimer Association Report: Alzheimer's disease facts and figures. 2010; 6

Dietterich T. Ensemble methods in machine learning. Multiple classifier systems. 2000:1–15.

Elkan, C. The foundations of cost-sensitive learning. International Joint Conference on Artificial Intelligence; Citeseer. 2001. p. 973-978.

Freund, Y.; Schapire, R. Experiments with a new boosting algorithm. Machine Learning-International Workshop then Conference; Citeseer. 1996. p. 148-156.

Hand D. Construction and assessment of classification rules. 1997

Hastie J, Tibshirani R. Additive logistic regression: A statistical view of boosting. The Annals of Statistics. 2000; 28(2):337–374.

Iba, W.; Langley, P. Induction of one-level decision trees. Proceedings of the Ninth International Conference on Machine Learning; 1992. p. 233-240.

Keally, M.; Zhou, G.; Xing, G. Watchdog: Confident event detection in heterogeneous sensor networks. 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium, IEEE; 2010. p. 279-288.

Kotsiantis S, Pintelas P. Mixture of expert agents for handling imbalanced data sets. Annals of Mathematics, Computing and Teleinformatics. 2003; 1(1):46–55.

Kubat, M.; Matwin, S. Addressing the curse of imbalanced training sets: one-sided selection. Machine Learning-International Workshop then Conference; Citeseer. 1997. p. 179-186.

Lim, M.; Choi, J.; Kim, D.; Park, S. A smart medication prompting system and context reasoning in home environments. Networked Computing and Advanced Information Management, 2008. NCM'08. Fourth International Conference on, IEEE; 2008. p. 115-118.

Matsumoto M, Nishimura T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation (TOMACS). 1998; 8(1):3–30.

Maurer, U.; Smailagic, A.; Siewiorek, D.; Deisher, M. Activity recognition and monitoring using multiple sensors on different body positions. Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on, IEEE; 2006. p. 4-116.

Monard M, Batista G. Learning with skewed class distributions. Advances in Logic, Artificial Intelligence and Robotics. 2002:173–180.

Neill D, Cooper G. A multivariate bayesian scan statistic for early event detection and characterization. Machine learning. 2010; 79(3):261–282.

Oriani M, Moniz-Cook E, Binetti G, Zanieri G, Frisoni G, Geroldi C, De Vreese L, Zanetti O. An electronic memory aid to support prospective memory in patients in the early stages of Alzheimer's disease: a pilot study. Aging & Mental health. 2003; 7(1):22–27. [PubMed: 12554311]

Pineau J, Montemerlo M, Pollack M, Roy N, Thrun S. Towards robotic assistants in nursing homes: Challenges and results. Robotics and Autonomous Systems. 2003; 42(3–4):271–281.

Platt J. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998

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. Robotics and Autonomous Systems. 2003; 44(3–4):273–282.

Provost, F.; Fawcett, T.; Kohavi, R. The case against accuracy estimation for comparing induction algorithms. Proceedings of the Fifteenth International Conference on Machine Learning; Citeseer. 1998.

Quinlan J. Induction of decision trees. Machine learning. 1986; 1(1):81–106.

Raskutti B, Kowalczyk A. Extreme re-balancing for svms: a case study. ACM SIGKDD Explorations Newsletter. 2004; 6(1):60–69.

Rudary, M.; Singh, S.; Pollack, M. Adaptive cognitive orthotics: combining reinforcement learning and constraint-based temporal reasoning. Proceedings of the twenty-first international conference on Machine learning; ACM; 2004. p. 91

Singla G, Cook D, Schmitter-Edgecombe M. Tracking activities in complex settings using smart environment technologies. International journal of biosciences, psychiatry, and technology (IJBSPT). 2009; 1(1):25.

Šingliar T, Hauskrecht M. Learning to detect incidents from noisily labeled data. Machine learning. 2010; 79(3):335–354.

Szewcyzk S, Dwan K, Minor B, Swedlove B, Cook D. Annotating smart environment sensor data for activity learning. Technology and Health Care. 2009; 17(3):161–169. [PubMed: 19641255]

Thomas, BL.; Crandall, AS. A demonstration of PyViz, a flexible smart home visualization tool. IEEE International Conference on Pervasive Computing and Communications, PerCom '11; 2011. to appear

Turney, P. Types of cost in inductive concept learning. Proceedings of the Cost-Sensitive Learning Workshop at the 17th ICML-2000 Conference; 2010. p. 15-21.

Wadley V, Okonkwo O, Crowe M, Ross-Meadows L. Mild cognitive impairment and everyday function: evidence of reduced speed in performing instrumental activities of daily living. American Journal of Geriatric Psych. 2008; 16(5):416.

Weber, J.; Pollack, M. Entropy-driven online active learning for interactive calendar management. Proceedings of the 12th international conference on Intelligent user interfaces; ACM; 2007. p. 141-150.

Weiss G. Mining with rarity: A unified framework. SIGKDD explorations. 2004; 6(1):7–14.

Weiss, G.; Provost, F. The effect of class distribution on classifier learning: an empirical study. Rutgers Univ; 2001.

Weiss, G.; McCarthy, K.; Zabar, B. Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs. Int. Conf. on Data Mining; Citeseer. 2007. p. 35-41.

Witten, I.; Frank, E. Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann Pub; 2005.

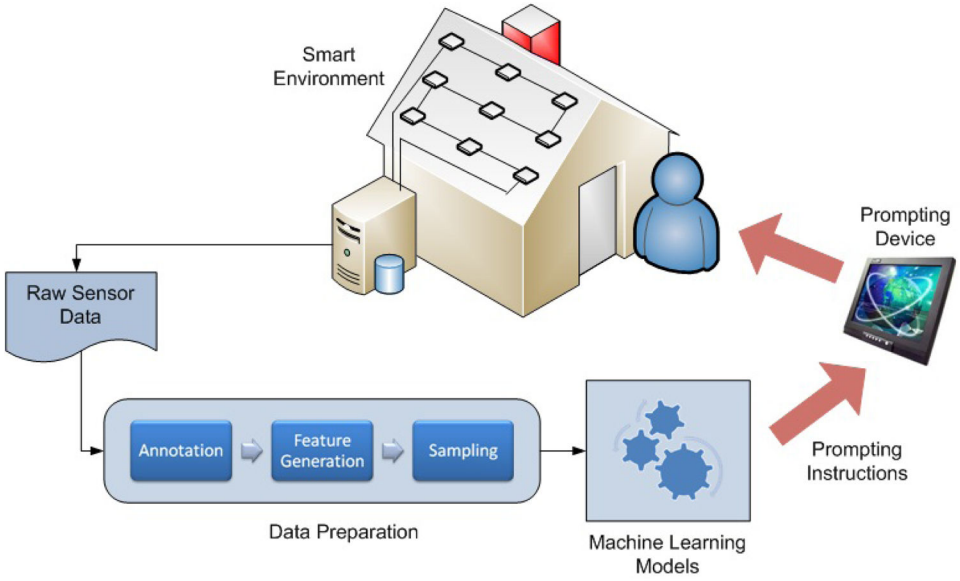

Fig. 1.
System Architecture of PUCK

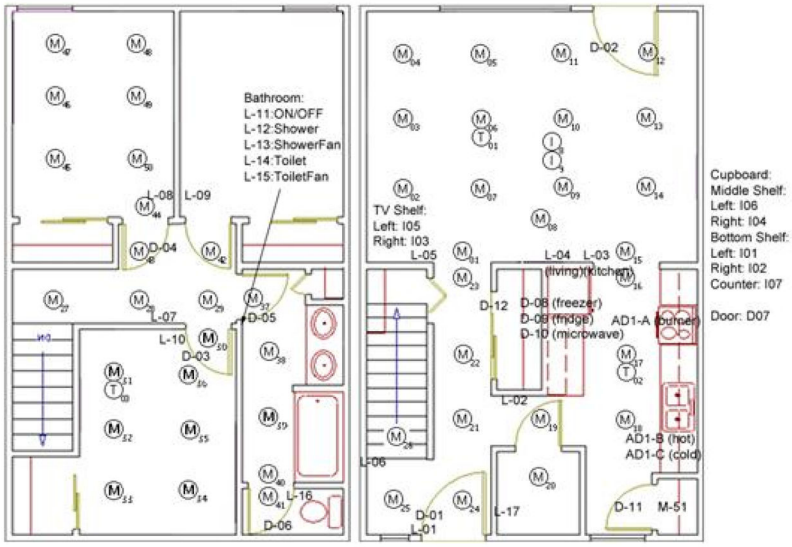

Fig. 2.

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

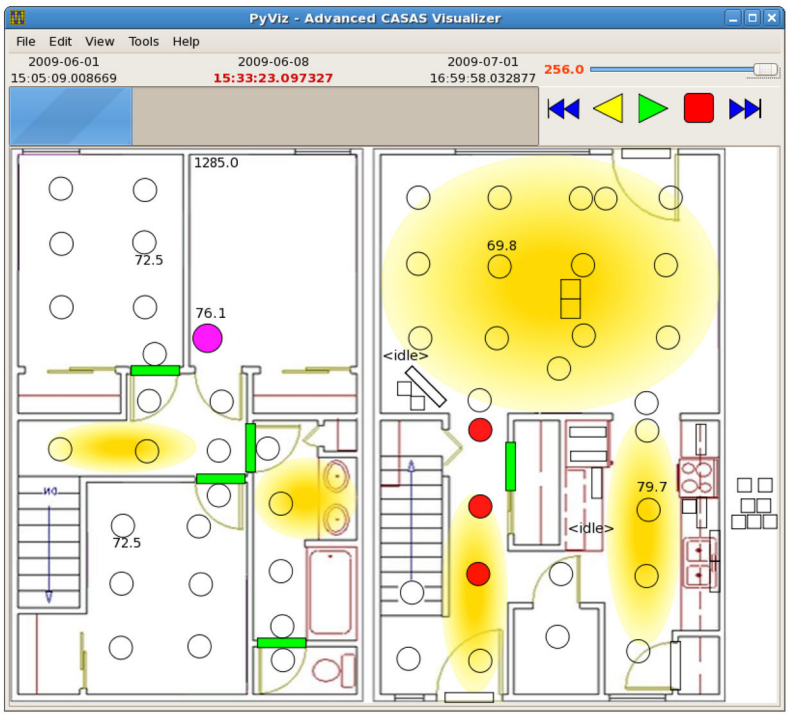

Fig. 3.
Screenshot of PyViz

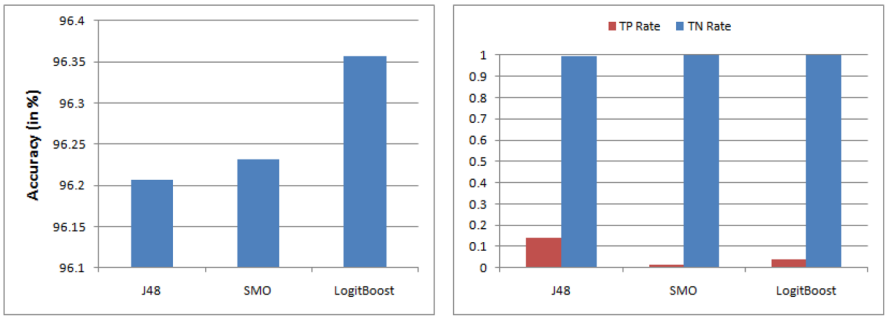

Fig. 4.

(a) Accuracy, (b)TP and TN Rates of classical machine learning algorithms

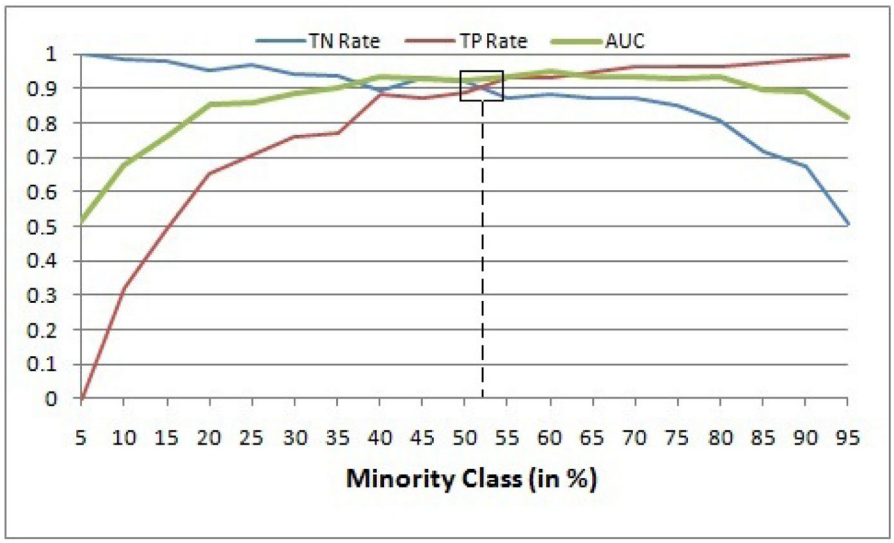

Fig. 5.
TP Rate, TN Rate and AUC for difierent class distributions

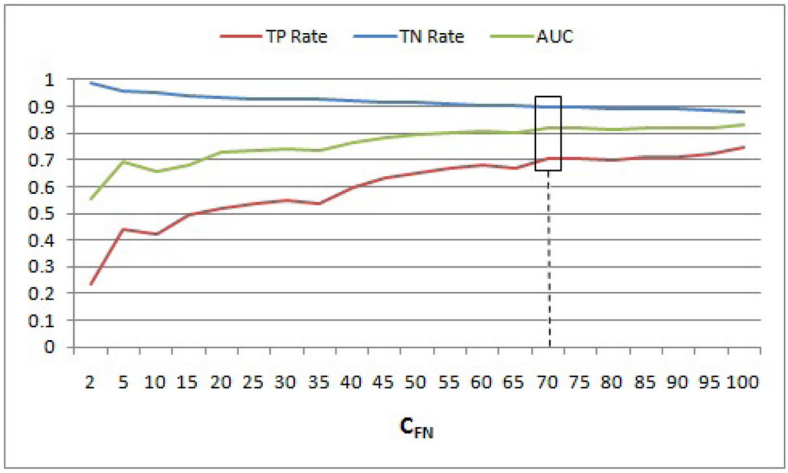

Fig. 6.
TP Rate, TN Rate and AUC for difierent C_{FN}

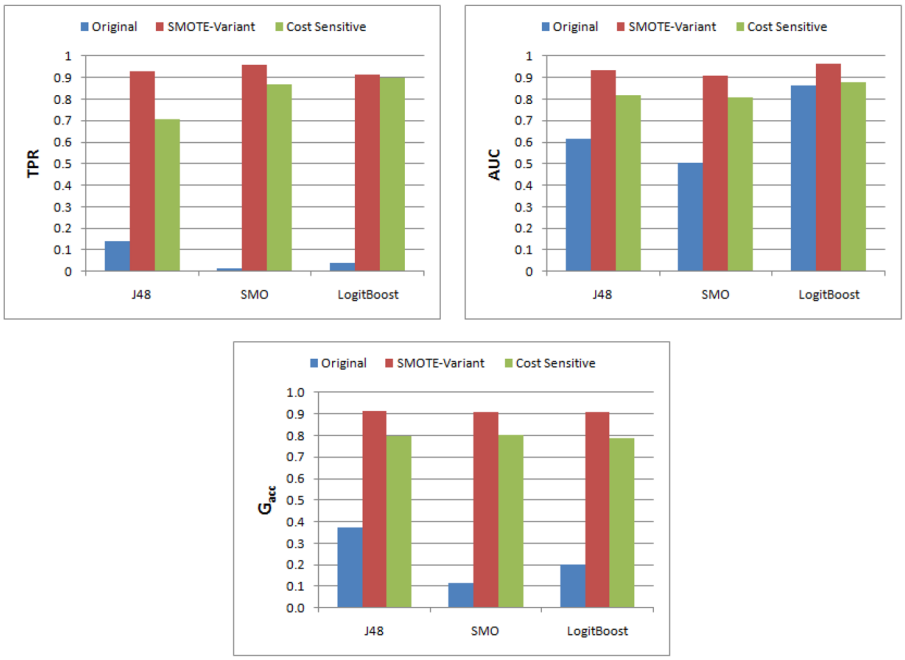

Fig. 7.
(a)TP Rate (*top-left*), (b)AUC(*top-right*), and (c)G_{acc}(*bottom*)

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

Table 2

Sample of sensor events used for our study

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 ID
5	nextStep	Next step ID
6	timeActBegin	Time (seconds) elapsed since the beginning of the activity
7	timePrevAct	Time (seconds) difference between the last event of the previous step and first event of the current step
8	stepsActBegin	Number of steps visited since the beginning of the activity
9	activityID	Activity ID
10	stepID	Current Step ID
11	M01 …M51	All of M01 to M51 are individual features denoting the frequency of firing of these sensors associated with the step
12	Class	Binary Class. 1-Prompt, 0-No Prompt

Table 3

Confusion Matrix

		Actual	
		Negative	**Positive**
Predicted	Negative	True Negative(TN)/C_{00} or C_{TN}	False Negative(FN)/C_{01} or C_{FN}
	Positive	False Positive(FP)/C_{10} or C_{FP}	True Positive(TP)/C_{11} or C_{TP}

Table 4

Performance on Original Dataset

Algorithms	TP Rate	TN Rate	AUC	Accuracy	G_{acc}
J48	0.141	0.994	0.615	96.21	0.3744
SMO	0.013	0.999	0.506	96.23	0.1140
LogitBoost	0.04	0.999	0.865	96.36	0.1999

Table 5

Performance on Sampled Dataset

Algorithms	TP Rate	TN Rate	AUC	Accuracy	G_{acc}
J48	0.931	0.897	0.935	91.55	0.9138
SMO	0.957	0.86	0.909	91.35	0.9072
LogitBoost	0.912	0.902	0.966	90.75	0.9070

Table 6

Performance Cost Sensitive Classifiers

Algorithms	TP Rate	TN Rate	AUC	Accuracy	G_{acc}
J48	0.705	0.899	0.817	89.19	0.7961
SMO	0.866	0.746	0.806	75.03	0.8038
LogitBoost	0.899	0.687	0.88	69.45	0.7859