Automated Prompting in a Smart Home Environment

Barnan Das[1], Chao Chen[1], Nairanjana Dasgupta[2], Diane J. Cook[1], Adriyana M. Seelye[3]
[1]School of Electrical Engineering and Computer Science, [2]Department of Statistics, [3]Department of Psychology
Washington State University
Pullman, USA
barnandas@wsu.edu, cchen@eecs.wsu.edu, cook@eecs.wsu.edu, dasgupta@wsu.edu, aseelye@wsu.edu

Abstract— **With more older adults and people with cognitive disorders preferring to stay independently at home, prompting systems that assist with Activities of Daily Living (ADLs) are in demand. In this paper, with the introduction of "The PUCK", we take the very first approach to automate a prompting system without any predefined rule set or user feedback. We statistically analyze realistic prompting data and devise a classifier from statistical outlier detection methods. Further, we devise a sampling technique to help with skewed and scanty data sets. We empirically find a class distribution that would be suitable for our work and validate our claims with the help of three classical machine learning algorithms.**

Keywords-smart environments; automated prompting; prompting systems; machine learning

I. INTRODUCTION

This paper introduces an automated prompting system that guides a smart home inhabitant through each step of an activity. The goal of our prompting system, called "The 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. A unique combination of data mining and pervasive computing technologies is required to meet this goal. In this paper, we explore the areas of fundamental statistical analysis and supervised machine learning to solve the problem. Earlier works that addressed this problem were associated with either direct or indirect feedback from the user. In contrast, we solve it without any user feedback as we believe that in a real smart environment housing cognitively impaired people, it would not be possible to rely upon or even receive user feedback. We discuss our experiences in dealing with noisy and skewed data which is common for this domain, and provide a sampling technique, SMOTE-Variant, to deal with them. Our approach will thus provide insights for analyzing and applying data mining techniques in other application domains with similar characteristics. The following paragraph illustrates the problem definition is more detail.

Research in the area of smart environments has gained popularity in the last decade. However, most attention has been directed towards health monitoring and activity recognition [1],[2]. 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 US Census Bureau [3], the number of older adults in US aged 65+ is expected to increase from approximately 35 million in 2000 to an estimated 71 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 [4]. These older adults face problems completing both simple (e.g. eating, dressing) and complex (e.g. cooking, taking medicine) Activities of Daily Living (ADLs) [5]. They often leave an activity incomplete as they forget the steps or get distracted in the middle of the activity. On observing a care recipient facing a problem with an activity, the caregiver gives a prompt that helps her/him to perform the activity completely. The number of prompts that a caregiver typically provides depends upon the level of cognitive impairment. Increased impairment demands increased caregiver duties and thus places heavier burden on the caregiver. 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 would help in addressing a large section of the population.

II. RELATED WORK

Reminder systems have been in existence for quite some time now, the simplest form being a traditional alarm clock. As the technology for building innovative reminder or prompting systems is flourishing, research groups are exploring alternative methods to solve the problem.

Most reminder systems are rule based. In this approach, a set of rules is defined based on time, an activity context and user preferences. The work by Lim et al. [6] is concerned with a medication reminder system that recognizes the service suitable for medication situation. Oriani et al [7] 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 a predefined time. Rudary et al. [8] 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 [9] developed by Pollack et al. provides adaptive personalized reminders of activities of daily living using a dynamic Bayesian network to coordinate preplanned events. Pineau et al. [10] use partially ordered Markov decision processes control robots that can assist elderly with daily activities. Boger et al. [11] designed a Markov decision process based planning system guide dementia patients through the activity of hand washing. There has been limited work based on a supervised learning approach. The work by Weber et al. [12] focuses on online active learning for interactive calendar management using an entropy-driven technique. In this paper, we take a supervised learning approach without a user feedback or intervention and deal with the possible problems of data in this domain.

III. Data Collection

To validate our data mining-based approach to automatic prompt generation, we test The PUCK using real data collected in our smart environment testbed. The data collection is done in collaboration with the WSU's Department of Psychology. Participants in these experiments are volunteers who are either healthy older adults, people with mild cognitive disorders (MCI) or people with dementia, Alzheimer's or traumatic brain injury.

A. *Testbed*

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 entry and on doors for cabinets, refrigerator and microwave oven, temperature sensors in each room, a power meter, and analog sensors for burner and water usage. Figure 1 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 The PUCK is to automate the role of the experimenter in this setting.

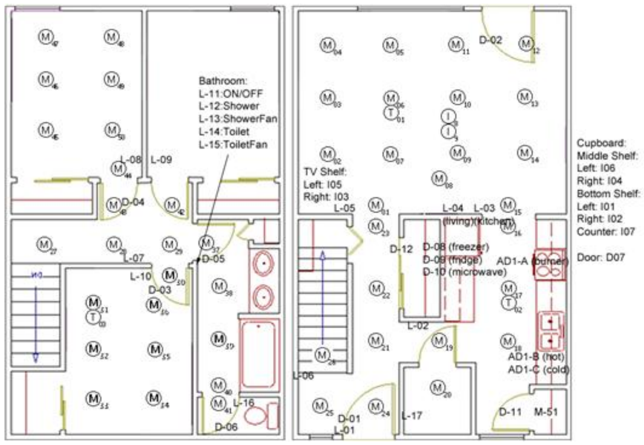


Figure 1. Three-bedroom smart apartment used for data collection (Sensors: motion (M), temperature (T), water (W), burner (B), telephone (P) and item (I)).

B. *Experimentation Methodology*

The following activities are used in our experiments:

1. Sweeping
2. Medication
3. Birthday Card
4. DVD
5. Water Plants
6. Phone Call
7. Cooking
8. Select Outfit

These activities are subdivided into relevant steps by the psychologists in order to track their proper completion. For example, individual Cooking steps consist of: 1) retrieving materials from cupboard, 2) filling measuring cup with water, 3) boiling water in microwave, 4) pouring water into cup of noodles, 5) retrieving pitcher of water from refrigerator, 6) pouring glass of water, 7) returning pitcher of water, 8) waiting for water to simmer, and 9) bringing all items to dining room table.

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 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 we then mine and use to train The PUCK.

C. *Annotation*

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

TABLE I. 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 {activity#.step#} that was being performed while the sensor events were generated, as shown here:

```
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
```

D. 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 prompts. Table II provides a summary of all generated features.

TABLE II 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	timePrevAct	Time (seconds) between the end of the previous step and the beginning of the current step
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"

IV. DATASET AND PERFORMANCE MEASURES

Before devising a data mining method that is appropriate for this task, it is essential to understand the nature of the dataset and specify suitable performance measures. We use data collected from 20 participants to train the learning models. There are 53 steps in total for all the activities, out of which 38 are recognizable and labeled. The participants were delivered prompts in 39 cases. Therefore, approximately 8% of the total instances are positive and the rest are negative. Essentially, this means that, predicting all the instances as negative, would give at least 92% accuracy even though all the predictions for positive instances were incorrect.

Conventional accuracy performance measures consider different classification errors as equally important. However, the purpose of our work is not to predict whether a prompt should be delivered in a step, but to predict when to fire the prompt. An important thing to keep in mind about automated prompting is that it is more desirable to deliver a prompt when it is not needed than to miss a prompt in a step when it is needed, especially while dealing with dementia patients. There should be a trade-off between the correctness of predicting a "prompt" step and the overall accuracy of the system. This is discussed in Section VII in further detail.

Therefore, it would be more interesting to consider performance measures that directly measure the classification performance on positive and negative classes independently. The True Positive (TP) Rate 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 The PUCK accurately labels as not requiring a prompt. We consider these two metrics as we want to see the performance of all the classifiers for both positive and negative classes.

A method to evaluate overall classifier performance is using a ROC curve analysis. A ROC curve for a classifier plots the false positive rate 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 The PUCK. The primary advantage of using these is that they illustrate The PUCK's performance without taking into account class distribution or error cost. We also report the AUC, or the area under ROC curve, in order to average the performance over all costs and distributions.

V. STATISTICAL ANALYSIS

We hypothesize that activity steps requiring a prompt will exhibit properties that are very different from the same activity step performed successfully by a participant. To validate this hypothesis, we apply a basic statistical approach to detect outliers in the data. In most instances of our dataset, the prompt does not exist. Thus, it is reasonable to consider the prompts as outliers. We match these outliers with the "prompt" steps (from the experimenters' data) to see if they have any relation with the theoretical definition of outliers. Thus, we essentially convert the outlier detection method in statistical analysis to a classifier for our data.

A. Selecting the Feature Subset for Analysis

Not all of the features mentioned in Table II are statistically feasible for analysis. Therefore, the five features summarized in Table III are selected for initial consideration. Table IV describes the Pearson correlation coefficients between different features. From Table IV, we note that there are two feature pairs which have a rather high correlation. StepLength and numEvents are correlated because when the time length of a step is longer, the number of sensor events has a high probability of increasing. NumSensors and numEvents are correlated because when the number of sensors is high, generally the number of events is also high.

TABLE III SIMPLE STATISTICAL INFORMATION

Feature	Mean	Std Dev	Sum	Min	Max
stepLength	71.57	137.31	36930	1	815.00
numSensors	1.98	1.46	1021	1	9.00
numEvents	4.80	5.70	2479	1	45.00
timeActBegin	82.29	122.63	42464	0	864.00
timePrevAct	12.92	32.64	5003	0	323.00

TABLE IV PEARSON CORRELATION COEFFICIENTS

	lengthStep	numSensors	numEvents	timeActBegin	timePrevAct
stepLength	1	0.21	0.74	-0.047	-0.083
numSensors	0.21	1	0.59	0.1	-0.05
numEvents	0.74	0.59	1	0.043	-0.07
timeActBegin	-0.047	0.1	0.043	1	0.27
timePrevAct	-0.083	-0.05	-0.08	0.27	1

B. *minimal-Redundancy-Maximal-Relevance (mRMR) Feature Selection*

Analysis of the remainder of the features is beyond the scope of this paper. However, before we start finding the outliers for all the features individually we consider a feature selection technique mRMR that would further identify the best relevant features for use by our supervised learner.

One of the most popular feature selection approaches is the Max-Relevance method, which selects the features with the highest relevance to the target class . Using Max-Relevance, the selected features x_i are selected individually for the largest mutual information *I(*x_i*;c)* with the target class *c*, reflecting the largest dependency on the target class. The definition of mutual information *I(*x_i*;c)* is as following:

$$I(x_i;y)=\iint P(x_i;y)\log\frac{p(x_i;y)}{p(x_i)p(y)}dxdy$$

However, Cover [13] shows that combinations of individually good features do not necessarily lead to good classification performance. To address this problem, Peng et al. [14] proposed a heuristic minimum-Redundancy-Maximum-Relevance (mRMR) selection framework, which selects features mutually far away from each other, while still maintaining high relevance to the classification target. This method has proven to be more efficient than Max-Relevance selection. In mRMR, Max-Relevance is to search features with the mean value of all mutual information values between an individual feature x_i and class *c*:

$$\max D(S,c), D=\frac{1}{|S|}\sum_{x_i\in S} I(x_i;c)$$

Because the Max-Relevance features may have a very high possibility of redundancy, a minimal redundancy condition can be added to select mutually exclusive features:

$$\min R(S), R=\frac{1}{|S|^2}\sum_{x_i,x_j\in S} I(x_i;x_j)$$

Thus, the operator *Φ(D,R)* is the mRMR criterion combining the above two constraints:

$$\max \Phi(D,R), \Phi=D-R$$

Running the mRMR to select a subset of features yields the following features in the order of their ranking:

1. stepLength
2. numSensors
3. timeActBegin
4. timePrevAct

C. *BoxPlot*

A number of outlier detection techniques are available in statistics, but we found Box Plot to be the appropriate one for our study. As the data is not normally distributed, mean and standard deviation are not good measures for detecting outliers. The box plot [15] is a graphic approach for examining sets of data. A box plot displays five important data summaries: 1) lowest value; 2) lower quartile; 3) median 4) upper quartile 5) highest value. As shown in Figure 2, the box plot is constructed by drawing a rectangle between the upper and lower quartiles with a solid line drawn across the box to locate the median. The lowest and highest values exist at the boundary of the solid line. The advantages of the box plot are that it can display differences between populations without making assumptions about the underlying statistical distribution and the distance between the parts of the box indicates the degree of spread and skewness in the data set.

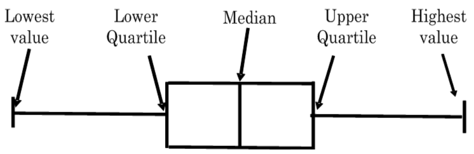
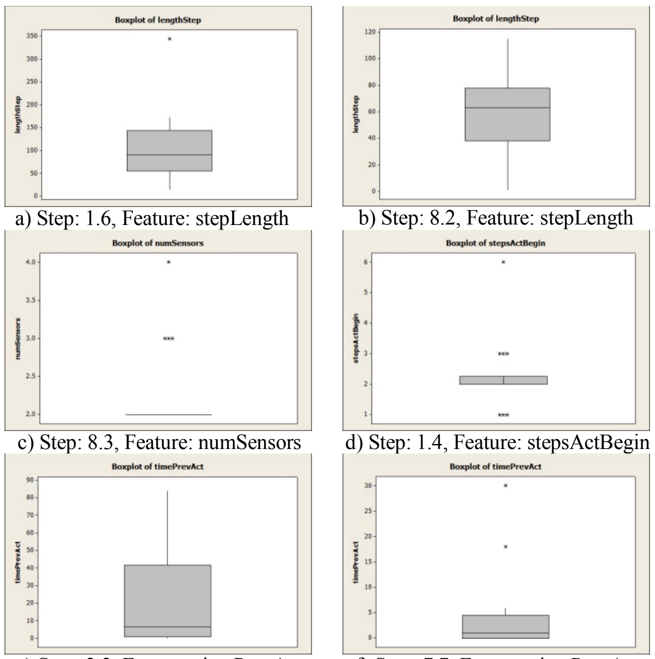


Figure 2. Configuration of a Box Plot

In this paper, we make use of box plot to identify the outliers of the steps, which lie unusually far from the main body of the data. Because even a single outlier can drastically affect the values of the mean and the standard deviation, a box plot is based on measures that could be resistant to the presence of the outliers. A measure of spread that is resistant to the outliers is the inter-quartile range, defined as *IQ = Upper Quartile – Lower Quartile.* Any data point farther than 1.5*IQ from the closest quartile is an outlier. An outlier is extreme if it is more than 3*IQ from the nearest fourth, and it is mild otherwise.

D. *Experiment Result*



a) Step: 1.6, Feature: stepLength b) Step: 8.2, Feature: stepLength
c) Step: 8.3, Feature: numSensors d) Step: 1.4, Feature: stepsActBegin
e) Step: 2.2, Feature: timePrevAct f) Step: 7.7, Feature: timePrevAct

Figure 3. Box Plots for some steps and features

We detect outliers for all the features associated with each step. Figure 3 shows box plots for some of the features. Note that some steps have features with high IQ (Figure 3b and 3e) and thus fail in detecting outliers even when there exists one. On the other hand, some steps have features with low IQ (Figure 3c and 3d) but are capable of detecting outliers better than others.

Although all the features are capable of detecting the outliers in the data individually, it becomes difficult to select a "silver bullet" feature that defines all error-based outliers in activity steps. In order to solve this problem, we devise a data mining method that considers a combination of these features. However, all the features are not equally important in detecting outliers and therefore are assigned a ranked weight. We assign weights to all the features according to the rank obtained from mRMR and obtain a score denoted by *p* for every instance in the dataset, in the following way:

$$p = w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$$

where *n* is the total number of features (*n=4* in our case), f_i for $1 \leq i \leq n$, is 1 if the value f_i is an outlier for the corresponding box plot and 0 otherwise; w_i for $1 \leq i \leq n$, are the weights assigned to each feature as per the rank. For some *x*, such that $0 \leq x \leq 1$, an instance is marked as an outlier if $p \geq x$.

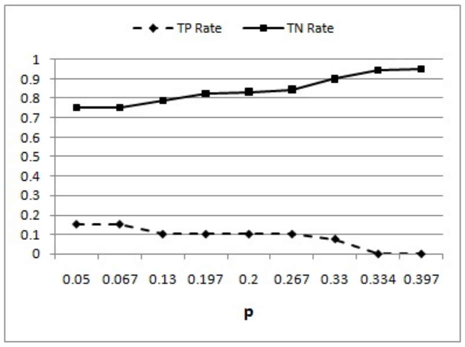


Figure 4. Performance of Outlier Detection

In order to evaluate the performance of outlier detection as a classifier we compare the predictions for every instance with the original "prompt" steps to obtain plots for the TP rate and TN rate against the score as shown in Figure 4. As it can be seen, the TP rate starts from a value of 0.154 and keeps decreasing until it reaches close to 0.05. On the other hand, the TN rate ranges from 0.75 to 0.95. Clearly, the TP rate is extremely poor as compared to the TN rate. As the TP rate is crucial for the evaluation of outlier detection in our work, we do not delve further into any other performance metric and proceed with the next learning method.

VI. LEARNING MODELS

A. Background

Decision Tree: A decision tree 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 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.

Boosting: Boosting, a type of ensemble method [16], 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 criteria for a binary class problem.

We use LogitBoost [17] that uses adaptive Newton steps for fitting an additive symmetric logistic model by maximum likelihood. It minimizes the expectation of the loss function $E(e^{-yF(x)})$ 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 output error and is less sensitive to noise.

B. Experiments with Original Dataset

As can be seen from Figure 5, the usage of classical machine learning algorithms on our dataset obtains a high accuracy. When we look at the TP and TN rates in Table V, we see that they are low for the positive class and high for negative class. This means that the classifiers are not able to effectively handle the positive instances. The reason is that the dataset has a highly imbalanced class distribution, it is more skewed towards negative instances than positive.

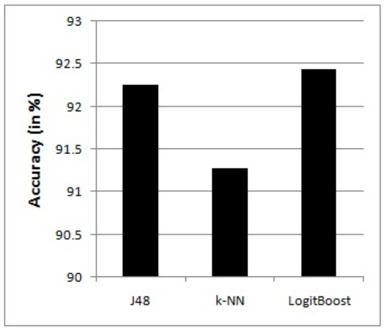


Figure 5. Accuracy (in %) for J48, k-NN and LogitBoost

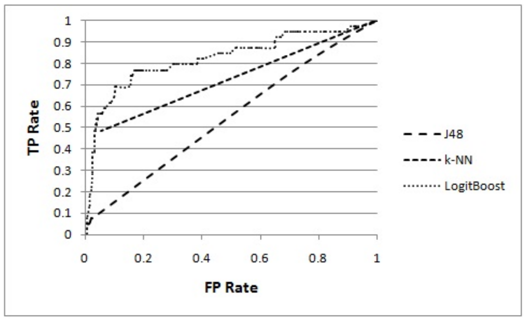


Figure 6. ROC curves

TABLE V TRUE POSITIVE AND TRUE NEGATIVE RATES

	TN Rate	TP Rate
J48	0.994	0.051
k-NN	0.948	0.487
LogitBoost	0.977	0.282

C. *Failure and Partial Success of Learning Algorithms on Original Dataset*

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 negative class. As mentioned before, the purpose of The PUCK is not to prompt an inhabitant in every step of an activity but to deliver prompts only for steps where individuals need the help to complete the task. Therefore, in spite of having such good 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. In the following, we analyze the reasons for the failure.

Decision trees do not take all attributes into consideration to form a hypothesis. The inductive bias is to prefer a shorter 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 once and thus compromises with unique properties of the instances that might lie with an attribute that has not been considered.

Unlike decision tree, k-Nearest Neighbor does not estimate the target function once for the entire instance space, rather does it locally and differently for each new instance to be classified. Also, this method calculates the distance between instances based on all attributes of the instance i.e. on all axes in the Euclidean space containing the instances. This is in contrast to methods such as rule and decision tree that selects a subset of the learning attributes while forming the hypothesis.

As the data is highly skewed, considering a subset of all the instances might not even consider the attributes activityID and stepID, which are unique identifiers of an instance belonging to a particular step, thus predicting a wrong class. But, k-Nearest Neighbor takes cares of this and the result is reflected in the Table V. Also, we make sure never to consider the value of *k* to be more than 2. As the percentage of minority class in the original dataset is quite less, a higher value of *k* will include more instances from the majority class and would predict wrongly for the instances in the minority class. In our Boosting technique, a learning algorithm is run several times each time with a different subset of training examples.

VII. SAMPLING

A common solution to deal with imbalanced datasets is to re-balance them synthetically, which is a technique known as Sampling. It is done in two ways, either by under-sampling (the majority class is shortened by ignoring some of its instances) or by oversampling (the number of instances in the minority class is increased). Note 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 difficulties [18]. 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 let the classifier formulate rules on insufficient or replicated data. There has been research to overcome both of these drawbacks at the same time. The most common of them is the SMOTE method [19]. 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 generating new instances which are similar to others. We describe a new mining approach that represents a variation of SMOTE applicable to this type of situation.

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 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 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 small, we would have null neighbors. Due to these limitations we had to make variations that would suit our domain. Unlike 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. One of these neighbors is randomly chosen and the new instance is synthesized as done in SMOTE. Figure 7 summarizes the approach.

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
    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
```

Figure 7. SMOTE-Variant Algorithm

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.

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. A study done by Weiss et al [20] 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.

In the previous section we demonstrated that the natural class distribution is not the best for training a classifier. As we noted, decision trees 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 varying percentages of minority class instances, from 5% up to 95%, by increments of 5%.

By repeating the experiments it was possible to determine empirically that a good class distribution for our domain occurs when the original dataset has 8%-10% of the minority (positive) class. We choose the sample size to be about the 40% of the instance space. Choosing any size lower than this causes loss of potential information. Any higher size requires generation of 500%-600% of the positive instances that were originally present, making the sample susceptible to overfitting. The reason is that our sampling technique SMOTE-Variant randomly picks some instances from the minority class and considers only the activityID and stepID features to get the nearest neighbors.

As mentioned before, we not only want a better TP rate but also a fairly good TN rate. In order to get a balance between the two we plot the TP and TN rate together in Figure 8(*left*). The TP rate increases while TN rate decreases as the percentage of minority class is increased. These two points intersect each other somewhere between 70-75% of the minority class. The area under ROC curve is 0.854 (Figure 8(*middle*)) and accuracy 83.5% (Figure 8(*right*)) for 70% of the minority class. Therefore, we decide to choose 70% of minority class to be the sample distribution for further experimentation. It should also be noted that, keeping in mind the type of dataset we obtain (i.e. specific properties related to size and degree of skewness) for our domain of prompting, a class distribution with 60% - 80% minority class should be good for experimentation. This can also be used as a baseline performance for future work.

VIII. Improvement of Learning Algorithms

From Figure 9(*left*) it is seen that the TP rate has increased tremendously for all the algorithms without compromising too much with TN rate (shown in Figure 9(*middle*)). Also the area under ROC curve has also increased (Figure 9(*right*)) indicating that the overall performances of all the learning methods have increased.

Clearly, sampling the data encouraged the methods to learn more rules for the positive class. However, the average accuracy decreases (see Table VI). This is acceptable until the TP rate is high. Also, an interesting thing to note is that the k-nearest neighbor consistently performed best. Although, it has a lower TN rate and AUC after sampling, as compared to others, but at the same time it managed to keep average accuracy at 89%, which is fairly high. This indicates that the size of the optimal subset of features to make the best classification is not small. This violates our initial assumption of feature subset in statistical analysis, when we neglected many features considering they are statistically infeasible for analysis.

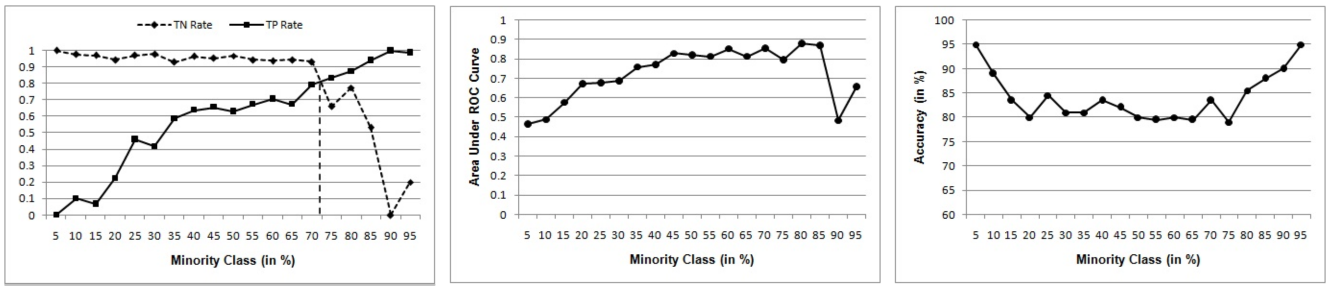


Figure 8. (*left*)TN and TP Rates; (*middle*) AUC; (*right*) Accuracies; for different class distributions

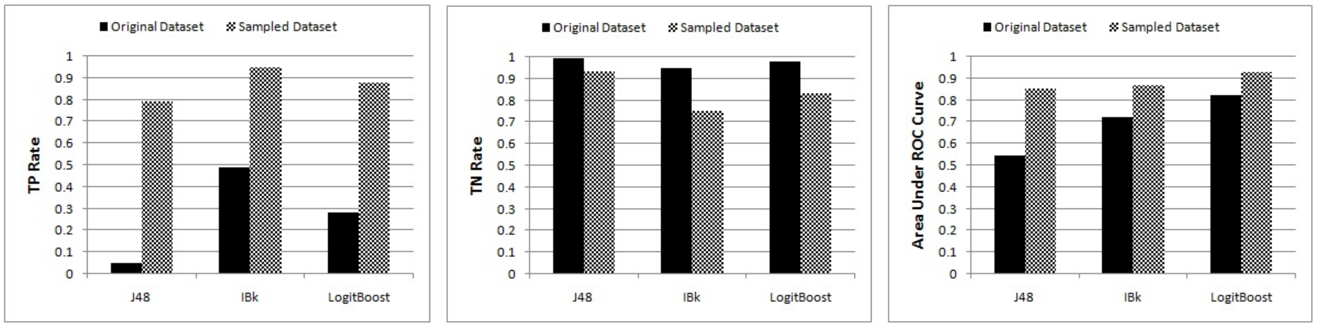


Figure 9. Comparison of (*left*)TP; (*middle*) TN; (*right*)AUC

TABLE VI COMPARISON OF ACCURACIES

	Original Dataset	Sampled Dataset
J48	92.25	83.5
k-NN	91.28	89.0
LogitBoost	92.44	86.5

IX. CONCLUSIONS AND FUTURE WORK

In this paper we introduced The PUCK, a system that automates activity prompts in a smart home environment by identifying the steps at which prompts are required. We studied the nature of the data by performing fundamental statistical analysis and highlighting some noteworthy properties that are characteristic to the domain of automated prompting. We proposed a variant of SMOTE sampling technique that would help in dealing with skewed data sets. With the help of this we empirically found the near best class distribution for our data that helps in boosting the performance of classical learning model.

In our future work, we would deal with the problem of scanty data by using semi-supervised learning techniques. Furthermore, we want to predict the prompts for real time with precision of every second. The model would redefine itself every second and predict whether a prompt should be fired, rather than predicting the "Prompt State" of every state. We will next test the accuracy of The PUCK by providing fully-automated prompts for activities in the smart home testbed environment with older adult participants.

REFERENCES

[1] U. Maurer, A. Smailagic, D. Siewiorek, M. Deisher. "Activity recognition and monitoring using multiple sensors on different body positions," Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks, 2006.

[2] G. Singla, D. Cook, and "Tracking activities in complex settings using smart environment technologies," International Journal of BioSciences, Psychiatry and Technology, Vol. 1, pp. 25-36, 2009.

[3] U.S. Census Bureau. Table 094. Midyear population, by age and sex. www.census.gov/population/www/projections/natdet-D1A.html.

[4] J. Bates, J. Boote, and C. Beverly, "Psychological interventions for people with a dementing illness: A systematic review," Journal of Advanced Nursing, vol. 4, no. 6, pp 644-658, 2004.

[5] V. Wadley, O. Okonkwo, M. Crowe, and L.A. Ross-Meadows. "Mild Cognitive Impairment and everyday functions: Evidence of reduced speed in performing instrumental activities of daily living." American Journal of Geriatric Psychiatry, vol. 16, no. 5, pp 416-424, 2007.

[6] M. Lim, J. Choi, D. Kim, and S. Park. "A Smart Medication Prompting System and Context Reasoning in Home Environments," Proceedings of the Fourth International Conference on Networked Computing and Advanced Information Management, vol 0, pp 115-118, 2008.

[7] M. Oriani, E. Moniz-Cook, G. Binetti, G. Zanieri, G. Frisono, C. Geroldi, L. De Vreese, and O. Zanetti, "An electronic memory aid to support prospective memory in patients in the early stages of Alzheimer's disease: A pilot study," Aging Ment. Health., vol. 7, no. 1, pp. 22-27, 2003.

[8] M. Rudary, S. Singh, and M. E. Pollack. "Adaptive Cognitive Orthotics: Combining Reinforcement Learning and Constraint-Based Temporal Reasoning," Proceedings of the 21st International Conference on Machine Learning, pp 719-726, 2004.

[9] M. Pollack, L. Brown, D. Colbry, C. McCarthy, C. Orosz, B. Peintner, S. Ramakrishnan, and I. Tsamardinos, "Autominder: An intelligent cognitive orthotic system for people with memory impairment," Robot.Auton. Syst., vol. 44, pp. 273-282, 2003.

[10] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun, "Towards robotic assistants in nursing homes: challenges and results," Rob. Auton. Syst., vol. 42, pp. 271-281, 2003.

[11] J. Boger, J. Hoey, P. Poupart, C. Boutilier, G. Fernie, and A. Mihailidis. "A decision-theoretic approach to task assistance for persons with dementia," Proceedings of the International Joint Conferences on Artificial Intelligence, pp 1293 -1299, 2005.

[12] J. S. Weber and M. E. Pollack, "Entropy-Driven online active learning for interactive calendar management," In Proceedings of the 12th international Conference on intelligent User interfaces, 2007.

[13] T.M. Cover, "The Best two Independent Measurements Are Not the Two Best", IEEE Trans. Systems, Man, and Cybernetics, vol. 4, pp. 116-117, 1974.

[14] H. C. Peng, F. Long, and C. Ding, "Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 27, No. 8, pp. 1226-1238, 2005.

[15] J. W. Tukey, Exploratory data analysis. Reading, MA: Addison-Wesley, 1977.

[16] T. G. Dietterich. "Ensemble methods in machine learning". International Workshop on Multiple Classifier Systems. Springer-Verlag pp. 1-15, 2000.

[17] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. Annals of Statistics 28(2), pp 337-407, 2000.

[18] M. Monard and G. Batista, "Learning with skewed class distribution," Frontiers in Artificial Int. and its Applications, IOS Press, 2002.

[19] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. "SMOTE: Synthetic Minority Over-sampling Technique," Journal of Aritificial Intelligence Research, 16:321-357, 2002.

[20] G. M. Weiss and F. Provost, The Effect of Class Distribution on Classifier Learning: An Empirical Study. Technical Report ML-TR-44, Rutgers University, Department of Computer Science, 2001.