Automated Activity Interventions to Assist with Activities of Daily Living

Barnan Das[1], Narayanan C. Krishnan and Diane J. Cook
School of Electrical Engineering and Computer Science
Washington State University, Pullman WA, USA

Abstract. Over the last decade there has been a significant growth of research endeavors in the area of ambient intelligence or smart environments. An anticipated increase in the older adult population around the globe and an increase in health care expenditures as a result, has increased the demand of smart health assistance systems. Along with the classical problems of remote health monitoring and activity tracking, delivering in-home interventions to residents for timely reminders or brief instructions to ensure successful completion of daily activities, is receiving a significant amount of attention in the community. In this chapter, the problem of delivering in-home interventions has been described in detail and some of the prospective approaches have been compared and contrasted. The approaches, details and challenges mentioned in this chapter revolve around a prototypic model of an automated prompting system, namely PUCK, which is an on-going project at the Center for Advanced Studies in Adaptive Systems at Washington State University. The previous study done on this project investigated the application of machine learning techniques to identify appropriate timing of prompts based on data provided by off-the-shelf sensors. The fundamental machine learning problem faced while learning the timing of prompts is that the class of training instances that represent prompt situations is under represented as compared to no-prompt situations. While a method was originally proposed to deal with this problem, popularly known as learning from imbalanced class distributions in this chapter a novel Cluster-Based Under-sampling (CBU) approach is proposed that shows promising results.

Keywords. Prompting system, smart environments, machine learning, imbalanced class distribution, activities of daily living

Introduction

Research in the area of smart environments has gained popularity over the last decade. Most attention has been directed towards health monitoring and activity recognition [1-4]. 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 [5], 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

[1] Corresponding Author: EME 130 Spokane Street, PO Box 642752, School of Electrical Engineering and Computer Science, Washington State University, Pullman WA, 99164-2752, USA; Email: barnandas@wsu.edu.

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 [6]. These older adults face problems completing both simple (e.g. eating, dressing) and complex (e.g. cooking, taking medicine) Activities of Daily Living (ADLs) [7].

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 within an activity and at that time provides a prompt that helps 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. 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 alleviating the burden of many caregivers that are helping a large section of the population.

In this chapter, we describe in detail the problem of delivering in-home interventions and compare some of the prospective approaches. The approaches, details and challenges mentioned in this chapter are based on a prototypic model of an automated prompting system, namely PUCK, which is an on-going project at the Center for Advanced Studies in Adaptive Systems (CASAS) at Washington State University. In an earlier study [8, 9] PUCK learned the timing of prompts for eight different activities of daily living, based on real sensor data collected in a smart home with volunteer participants. The purpose was to achieve the goal of automating prompt timing without any direct user feedback. In this study, a challenge faced while learning the timing of prompts because of the nature of the data collected from the sensors. The class of training instances that represent a prompt situation is under represented as compared to no-prompt situations. In order to address this problem, which is popularly known as learning from an imbalanced class distribution [10], a fundamental modification was introduced to the Synthetic Minority Over-sampling Technique or SMOTE [11] proposed by Nitesh Chawla et al. Due to certain limitations of this approach, in this chapter a novel Cluster-Based Under-sampling (CBU) technique is proposed that can handle more realistic situations and that performs better than the aforementioned technique.

The chapter starts by describing the problem of in-home interventions for activities, in detail. The discussion proceeds by detailing the system architecture for conducting the study at Center for Advanced Studies in Adaptive Systems at Washington State University, followed by a description of the data collection methodology and data representation. Previous approaches taken to address the problem are described and the limitations are highlighted. Next, the Cluster-Based Under-sampling (CBU) technique is proposed and new experimental results are shown. The chapter ends with a related work section that covers both applied and theoretical aspects of the problem.

1. Problem Definition

A "prompt" in the context of a smart home environment can be defined [8] 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. Although the literature is flooded with similar terms such as reminders, alerts, and notifications, "prompt" is generically used to represent interventions that ensure accomplishment of certain activity goals. Prompts can provide a 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 are the prompts 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 multi-modal)?
- User Environment: What is the physical layout of the home and how does this affect the timing and mode of prompts?

The goal of the Prompting Users and Control Kiosk or PUCK project is to develop an automated prompting system that guides a smart home inhabitant through every step of an activity. The PUCK project operates on the hypothesis that the timing of the prompt within an activity can be learned by identifying when an activity step has been missed or performed erroneously. As a result, PUCK's goal is to deliver an appropriate prompt when, and only when, one is required. The prompt granularity for this system is individual activity steps, unlike other projects which consider activities as a whole. A unique combination of pervasive computing and machine learning technologies is required to meet this goal. More explanation on other projects that deal with the prompting problem can be found in Section 8 of this chapter.

In order to address the applied problem of automated prompting, a more theoretical machine learning problem needs to be addressed, namely the ability to learn from an imbalanced class distribution. The fundamental challenge of this domain is to learn the appropriate timing of prompts when the vast majority of training situations do not require prompts. Thus, the data gathered from the sensor grid in the testbed, consists of very few prompt situations as compared to no-prompt situations.

In the following, the bigger problem of automated activity interventions has been broken down into layers of applied problems and theoretical problems. A top down approach is taken to transition from the applied problem to in-depth machine learning challenges.

2. 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 machine learning algorithms.

In the previous work, it was possible 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. Past research has also included the deployment of time-based and context-aware prompts with one or two older adult residents. In these deployments, touch screen monitors with embedded speakers were used 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. The same interface is being used for the automated prompting system.

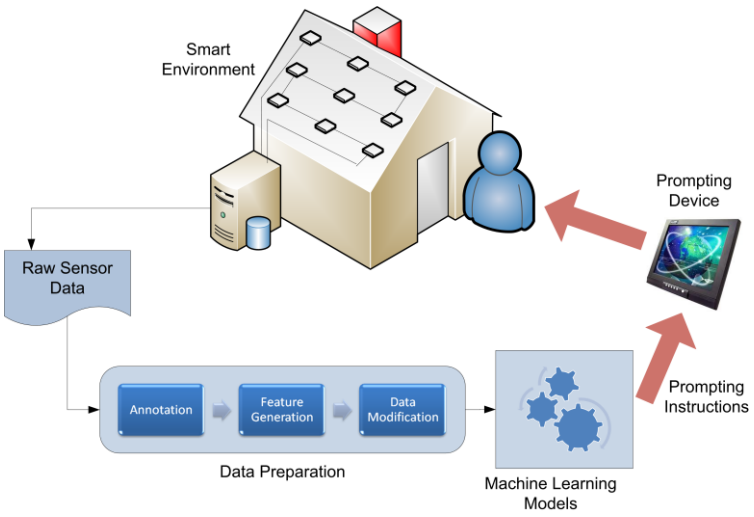

Figure 1: System architecture.

The PUCK system architecture can be broadly divided into four major modules:

- **Smart Environment**: The smart home environment testbed is a two-story apartment located on the Washington State University campus. The apartment 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 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 study, the data gathered by motion, door and item sensors have been considered. 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 clinically-trained experimenters monitor the activities performed by the participants (via web cameras) and deliver prompts through an audio delivery system whenever they deem a prompt is necessary. The goal of PUCK is to automate the role of the experimenter in this setting. The sensor data is stored in a SQL database in real time.

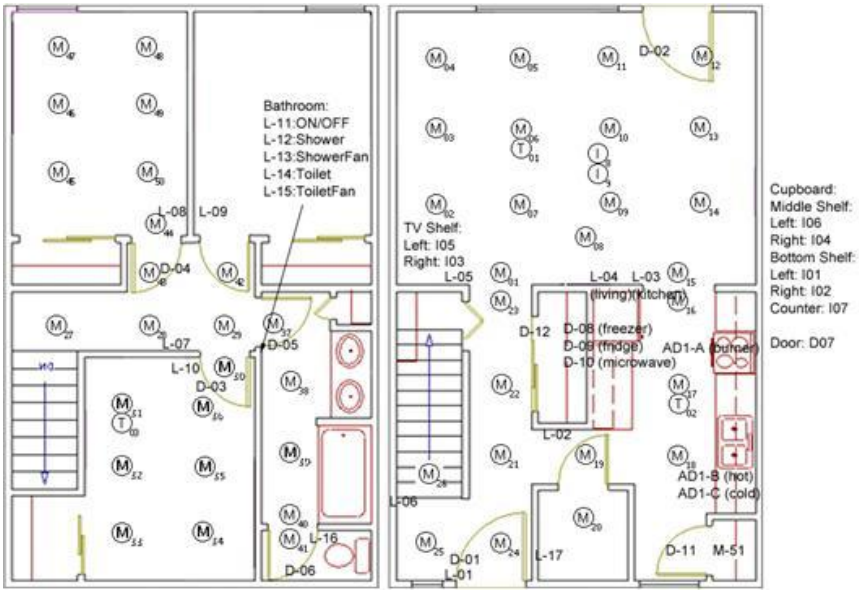

Figure 2: Three-bedroom smart apartment used for data collection (Sensors: motion (M), temperature (T), water (W), burner (B), telephone (P) and item (I)).

- **Data Preparation**: The portion of raw sensor data that would be used by the learning models is collected from the database. This data is manually annotated for activities and activity steps 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 of the imbalanced nature of the training data, the data is modified before applying the machine learning models.
- **Machine Learning Model**: Once the data is prepared by the Data Preparation module, machine learning techniques are employed to determine 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.

The focus of this chapter is on the unique approach taken in the Machine Learning Model to handle the imbalanced data that is inherent in an activity-prompting application.

3. Data Collection

The PUCK data collection is done in collaboration with Washington State University'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 (TBI).

3.1. Experimentation Methodology

The experiments are conducted by the psychologists by inviting participants to the 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 is considered for our experiments:

1. Sweeping	2. Taking Medication	3. Writing Birthday Card
4. Watching DVD	5. Watering Plants	6. Receiving Phone Call
7. Cooking	8. Selecting Outfit	

These activities are subdivided into relevant steps by the psychologists in order to track their proper completion. The detailed description of all the activities and their steps is beyond the scope of this chapter. Therefore, the steps of a sample "Cooking" activity (Table 1) are provided 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.

Table 1. Steps of "Cooking" activity.

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

The participants are asked to perform activities of daily living in the testbed. A prompt is issued by the experimenter if the participant misses any critical step, performs the step erroneously or takes longer than usual. The goal of the experimenter is to issue as few prompts as possible but at the same time ensure a successful completion of the activity. The prompt issuance time is logged in the database and is later used to determine the activity step to which it corresponds. This information is incorporated into the vector of features describing the raw data.

3.2. 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 the SQL database is expressed by several features. A sample of sensor data collected in the smart apartment and described by the features is given in Table 2.

Table 2. 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
2009-02-07	11:18:37	P001	1.929kWh
2009-02-09	21:15:28	P001	2.536kWh

After collecting data, the sensor events are annotated with the corresponding activities (as shown in Table 3) that were performed while the sensor events were generated. Activities are labeled with their corresponding activity IDs (as listed in Section 3.1) and step IDs. This is done in the following format: **<*Activity ID*>.<*Step Number*>**. For example, **7.4** would indicate the fourth step of the seventh activity "Cooking", i.e., "Participant pours water into cup of noodles".

Table 3. Annotated steps for activity 7 (Cooking).

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	M017	ON	7.9
2009-05-11	15:00:45.008148	M014	ON	7.9

As the annotated data is used to train the learning models, the quality of annotation is very important for the appropriate 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 [13] without the use of visualization tools. Therefore to enhance the quality of the annotated data, an open source Python Visualizer, called PyViz [14] and developed by CASAS research team members, is used to visualize the sensor events.

4. Dataset and Performance Measures

4.1. Feature Generation

Relevant features are generated from the annotated data that is 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 pertinent features are defined to describe the step based on sensor data. 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 4 provides a summary of all generated features. It should be noted that the machine learning models learn and predict class labels from this refined dataset. This way PUCK predicts if an instance (steps of activities in this context) constitutes a *prompt* instance. Thus, the problem of *when* a prompt should be delivered is addressed.

Table 4. Generated features.

Feature	Description
stepLength	Length of step in time (seconds)
numSensors	Number of unique sensors involves with the step
numEvents	Number of sensor events associated with the step
prevStep	Previous step ID
nextStep	Next step ID
timeActBegin	Time (seconds) elapsed since the beginning of the activity
timePrevAct	Time (seconds) difference between the last event of the previous step and first event of the current step
stepsActBegin	Number of steps visited since the beginning of the activity
activityID	Activity ID
stepID	Current step ID
M01…M51	All of M01 to M51 are individual features denoting the frequency of firing of these sensors associated with the step
Class	Binary class representing *prompt* and *no-prompt*

Sensor data was collected for 128 participants and was used to train the machine 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 steps are associated with specific object interactions which could not be tracked by the current sensor infrastructure. 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 (*prompt* steps) and the rest are negative (*no-prompt* steps). 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.

4.2. Performance Measures

Conventional performance measures such as accuracy and error rate consider different types of classification errors as equally important. For example, the purpose of this work is not to predict whether a prompt should not be delivered in a step, but to predict when to issue the prompt. 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 of the entire system.

Therefore, performance measures that directly measure the classification performance for positive and negative classes independently are considered. The True Positive (TP) Rate (the positive and in this case 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 steps that are accurately labeled 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 classifier's false positive rate [15] on the x-axis and the true positive rate on the y-axis. A ROC curve is generated by plotting the accuracy obtained by

varying different parameters of the classifiers. The primary advantage of using these is that they illustrate the classifier's performance without taking into account class distribution or error cost. We report AUC, or the area under ROC curve [16], in order to average the performance over all costs and distributions. Also, the geometric mean of TP and TN rates denoted by G_{acc} is reported, which is commonly used as a performance metric in imbalanced class learning. G_{acc} is calculated as $\sqrt{TP\ Rate \times TN\ Rate}$. To evaluate the overall effect of classification, the conventional accuracy of the classifiers is also considered.

5. Background on Machine Learning Methods

5.1. Decision Tree

A decision tree classifier [17] 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:

$$Entropy(S) \equiv -p_+ \log_2 p_+ - p_- \log_2 p_- \tag{1}$$

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:

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

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.

5.2. Nearest Neighbor

k-Nearest Neighbor [18] is the most basic technique amongst instance based learning methods in which all instances are assumed to correspond to points in the n-dimensional space. The distance measure that is used to find the neighbors is Euclidean distance given by the following:

$$\mathrm{d}(\mathrm{x_i}, \mathrm{x_j}) \equiv \sqrt{\sum_{\mathrm{r}=1}^{\mathrm{n}} (\mathrm{a_r}(\mathrm{x_i}) - \mathrm{a_r}(\mathrm{x_j}))^2} \tag{3}$$

When a query instance x_q is to be classified, the training examples denoted by $x_1 \ldots x_k$ (such that there are k training examples), which are nearest to x_q, are found by the equation:

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\arg\max} \sum_{i=1}^{k} \delta(v, f(x_i)) \quad (4)$$

where $\delta(a,b) = 1$ if $a = b$ and where $\delta(a,b) = 0$ otherwise. For any value of *k*, the algorithm assigns the most common classification label that appears among the *k* nearest training examples.

5.3. Support Vector Machines

Support Vector Machines (SVMs) were first introduced in 1992 [19]. This is an 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 a 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.

$$minimize \frac{1}{2}||w|| \qquad \text{subject to: } y_i(w^t x_i + b) \geq 1 \quad (5)$$

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

$$L_d \equiv \sum_i a_i - \frac{1}{2}\sum_{i,j} a_i a_j y_i y_j K(x_i, x_j) \quad (6)$$

$$0 \leq a_i \leq C \quad and \quad \sum a_i y_i = 0$$

The target function can be computed by:

$$f(x) = \text{sign}\left(\sum_{i=1}^{N} a_i y_i k(x_i, x) + b\right) \quad (7)$$

For a traditional SVM, the quadratic programming problem introduces a matrix, whose dimensions 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) [20] 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.

6. Sampling to Handle Imbalanced Prompt Cases

6.1. Experimental Results With No Sampling

Experiments were initially 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. As can be seen from Figure 3(a), the usage of classical machine learning algorithms on original dataset obtains a high accuracy. However, Figure 3(b) depicts that the TP Rates are extremely low as compared to the TN Rates.

From this experiment it can be inferred that traditional classifiers are not 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.

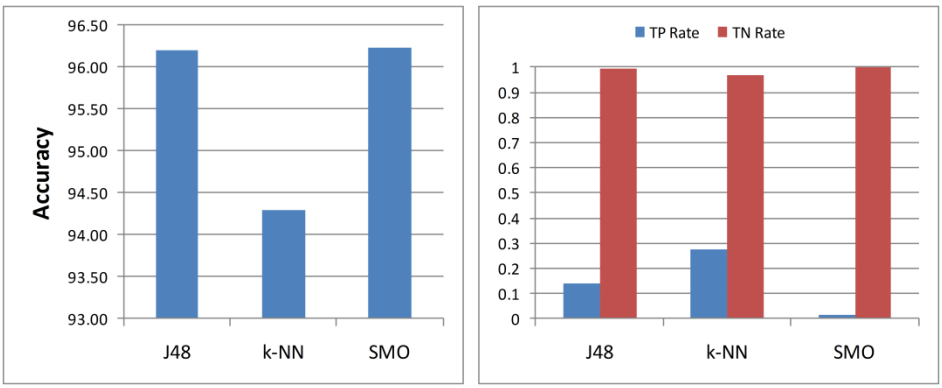

Figure 3: (a)Accuracy, (b)TP and TN Rates obtained without any preprocessing.

There can be number of reasons for the dataset to be skewed. In this 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 high accuracies, direct application these algorithms is not suitable for the goal of the project as they either fail to predict the steps in which the prompt should be issues or do that job with poor performance.

6.2. 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 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 it estimates the function 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 an incorrect class. But, k-Nearest Neighbor is capable of taking care of this issue and the result is reflected in Figure 3(b).

As the number of attributes is quite high (61 in this 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). However, due to a lesser number of positive class instances in this case, the functional margin is quite small and thus results in a lower TP rate.

6.3. SMOTE-Variant

Sampling is a technique of rebalancing the dataset synthetically and can be accomplished by under-sampling or over-sampling. While under-sampling can throw away potentially useful data, oversampling can overfit the classifier if it is done by data replication. As a solution to these challenges, SMOTE [11] uses a combination of both under and over sampling, but without data replication. Over-sampling is performed by taking each minority class sample and synthesizing a new sample by randomly choosing any or all (depending upon the desired size of the class) of its *k* minority class nearest neighbors. Generation of the synthetic sample is accomplished by first computing the difference between the feature vector (sample) under consideration and its nearest neighbor. Next, this difference is multiplied by a random number between *0* and *1*. Finally, the product is added to the feature vector under consideration. In the dataset under consideration, the minority class instances are not only small in terms of percentage of the entire dataset, but also in absolute number. Therefore, if the nearest neighbors are conventionally calculated (as in original SMOTE) and the value of k is small, there would be null neighbors. Unlike SMOTE, in SMOTE-Variant the k-nearest neighbors are calculated on the basis of just two features: *activityID* and *stepID*. Under-sampling is done by randomly choosing a sample of size *k* (as per the desired size of the majority class) from the entire population without repetition.

6.4. Experimental Results

The purpose of sampling is to rebalance a dataset by increasing the number of minority class instances, enabling the classifiers to learn more relevant rules on positive instances. However, there is no ideal class distribution. A study done by Weiss et al. [21] shows that, given plenty of data when only *n* instances are considered, the optimal distribution generally contains 50% to 90% of the minority class instances. Therefore, in order to empirically determine the class distribution, the J48 decision tree is considered as the baseline classifier and the experiments are repeated by varying percentages of minority class instances from 5% up to 95%, by increments of 5%. A sample size of 50% of the instance space is chosen.

Figure 4: TP Rate, TN Rate and AUC for different class distributions

While any lower sample size will cause loss of potential information; any higher size will make the sample susceptible to overfitting. Figure 4 shows that the *TP* rate increases while the *TN* rate decreases as the percentage of the minority class is increased. These two points intersect each other at some point that corresponds to somewhere between 50-55% of minority class. Also, the AUC value is between *0.923* and *0.934,* a relatively high value, near this point. Therefore, 55% of the minority class is chosen to be the appropriate sample distribution for further experimentation.

Three different algorithms, namely J48, IBk (a k-nearest neighbor algorithm), and SMO are run on the sampled dataset. From Figure 5 (a) it is seen that the *TP* rate has increased tremendously for all the algorithms without compromising too much the *TN* rate (shown in Figure 5(b)). Also, the area under ROC curve and G_{acc} have increased (Figure 5(c) and 5(d), respectively) indicating that the overall performances of all the learning methods have increased. Clearly, sampling encouraged the learning methods to learn more rules for the positive class. However, it should also be noted that the average accuracy decreases by a few percentage points. This is acceptable until the *TP* rate is high.

7. Improvements on Basic Sampling

7.1. Analysis of Previous Approach

The results of the previous approach are fairly optimistic but do not reflect reality. A deeper analysis of the previous methodology and results indicates that there were a number of implicit assumptions that do not hold in realistic settings. In the following, an analysis of the previous approach is done and simultaneous a new improved method is been proposed.

Evaluation of the methods was performed using cross validation. In case of SMOTE-Variant, as the training and testing were done on the same examples that were

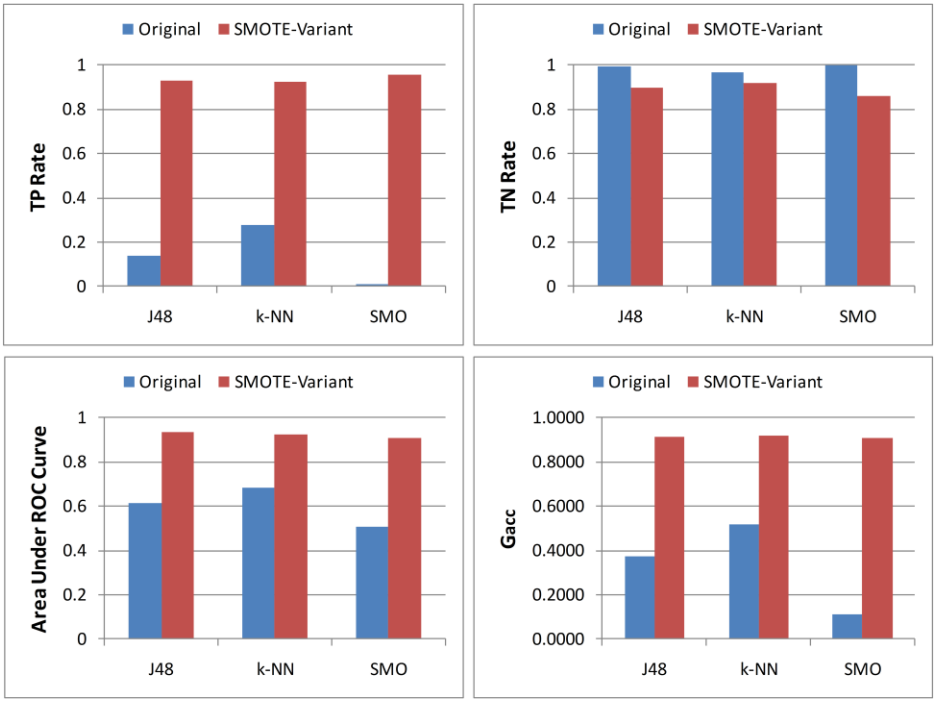

Figure 5: Comparison of (a)TP Rate, (b)TN Rate, (c) AUC, and (d) Gacc.

synthetically generated, the overfitting of the classifiers caused by an overwhelmed synthesis of artificial minority class examples, was never detected. In order to avoid this inappropriate evaluation technique, the current approach trains the classifiers on 80% data and considers the rest for testing. Also, the degree of imbalance in the original dataset is maintained in training and testing examples.

While studying the nature of the data and trying to find the heuristic that essentially makes a *prompt* instance different from a *no-prompt* instance, it was found that there are minor differences in the values of the attributes of *prompt* and *no-prompt* instances. This means that there is no crisp boundary between positive and negative class examples. Positive data points are embedded into negative data points causing a high degree of overlap between the two classes.

7.2. The Overlap Problem and Its Existence in Prompting Data

The overlap problem [22] occurs when there are ambiguous regions in the data space where there are approximately the same number of training examples from both classes. Conceptually, ambiguous regions can be visualized as regions where the prior probability for both classes is approximately equal and thus makes it difficult or impossible to distinguish between the two classes. This is because it is difficult to make a principled choice of where to place the class boundary in this region since it is expected that the accuracy will be equal to the proportion of the volume assigned to each class. Figure 6, illustrates the difference between normal data and data with class overlap.

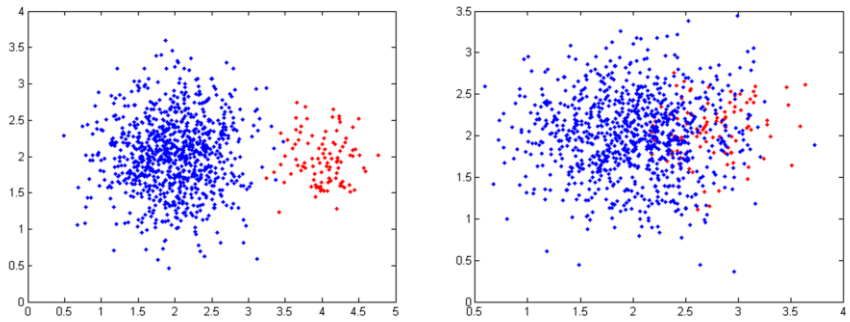

Figure 6: (a) Data without overlap, (b) Data with overlap

The prompting data has similar overlapping nature in between the two classes and that is confirmed by performing a dimensionality reduction on the attributes. A Principal Component Analysis (PCA) [23] is considered for this purpose. The dimension is reduced to three and then plotted. Figure 7 shows a reduced three dimension plot of the prompting data. It can be easily seen from the figure that the positive (*prompt*) class instances are highly embedded in negative (*no-prompt*) class instances.

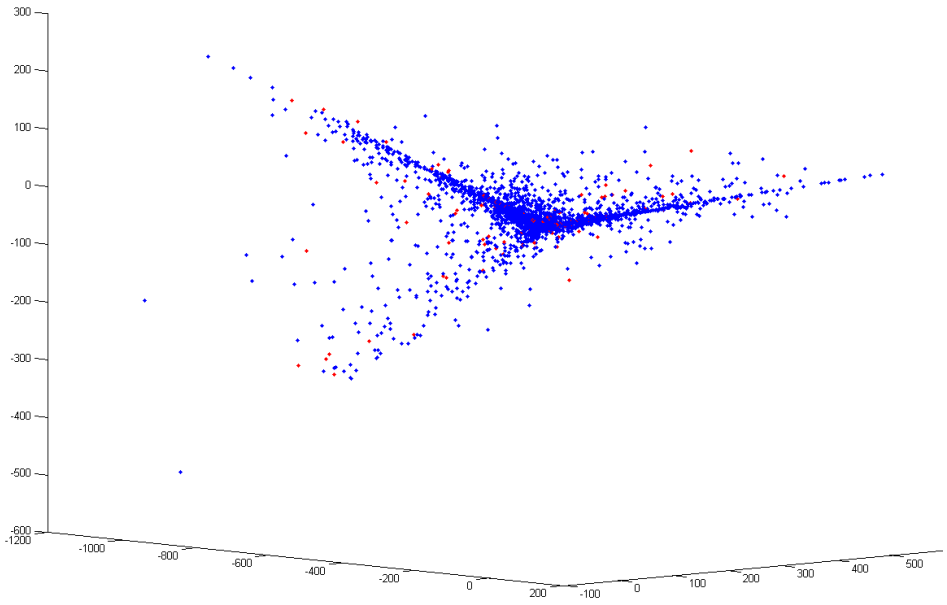

Figure 7: 3D PCA plot of prompting data.

7.3. Cluster-Based Under-sampling

By performing a hypothesis testing, Denil et al. proved [24] that overlap and imbalance are not independent factors. They showed that if overlap and imbalance levels are too high, good performance cannot be achieved regardless of amount of available training data. Therefore, in Cluster-Based Under-sampling (CBU) method, the purpose is to get rid of the overlapping problem and the hypothesis is that achieving success with the overlap problem would also be helpful in getting rid of the imbalance problem to some extent as the majority class is under-sampled. It should also be kept in mind that the prompting data has an absolute rarity imbalance problem, that is, the minority class instances are not only relatively less as compared to majority class, but also rare in absolute number. Therefore, no sampling method that involves throwing away minority class instances can be employed.

The idea of devising this technique is derived from the use of Tomek links [25] combined with other sampling methods like Condensed Nearest Neighbor [26] and SMOTE [27]. Tomek links are defined as: given two examples E_i and E_j belonging to different classes, and $d(E_i,E_j)$ being the distance between E_i and E_j, a (E_i,E_j) pair is called a Tomek link if there is not an example E_k such that $d(E_i,E_k) < d(E_i,E_j)$. If two examples form a Tomek link, then either one of these examples is noise or both examples are on or near the class boundary. Tomek links are used both as a data cleaning method and an under-sampling method. As a data cleaning method, examples of both classes are removed, and as an under-sampling method, only examples belonging to the majority class are eliminated.

One-sided selection [28] is an under-sampling method that applies Tomek links followed by the applying Condensed Nearest Neighbor (CNN). In this method, Tomek links are used to remove noisy and borderline majority class examples. As a small amount of noise can make the borderline examples fall on the wrong side of the decision boundary, borderline examples are considered as unsafe. CNN is used to remove examples from the majority class that are far away from the decision boundary. The rest of the majority and minority class examples are used for learning.

As opposed to the use of Tomek links in OSS to find closest minority and majority class example pairs and then remove majority class examples, in the Cluster-Based Under-sampling (CBU) method, clusters of minority and majority class examples are considered and majority class examples from those clusters are removed.

Table 5. Algorithm of Cluster-Based Under-sampling.

1.	Let S be the original training set.
2.	Use K-means clustering to form clusters on S denoted by C_i where $1<i<\|C\|$.
3.	Find the degree of imbalance for all the clusters denoted by: $r_i = \frac{\text{Number of minority class examples in } C_i}{\|C_i\|}$
4.	For clusters which satisfy: $0<r_i<1$ and $r>=\tau$ (where, τ is an empirically determined threshold value for r and is uniform over all the clusters), remove all the majority class examples and retain all the minority class example.

Table 5 summarizes the CBU algorithm. First, the entire training data is clustered ignoring the class attribute and using simple K-means clustering which uses Euclidean distance as the distance measure. The degree of *minority dominance*, denoted by *r*, for

each of these clusters is calculated as the ratio of number of minority class examples to the size of the cluster. Therefore, *r=0* indicates that all the examples of the cluster belong to the majority class, and *r=1* indicate that all the examples belong to the minority class. The clusters whose *r* lies between 0 and 1 are of interest in this method as indicates that the cluster has both minority and majority class examples. For this kind of cluster, the majority class examples are removed if *r* is equal to or greater that an empirically determined threshold value τ. Clearly, if the threshold τ is low more majority class examples would be removed as compared to when τ is high. This method creates a "vacuum" around the minority class examples and thus helps the machine learning classifiers learn the decision boundary more efficiently.

7.4. Experimental Results

The set of data attributes considered for CBU is a bit different from the data considered in the previous study. The motion sensor frequencies for each activity step, which was previously considered as 51 different attributes, have been merged to form locations in the apartment that would represent regions of the smart home such as *kitchen*, *living room*, *dining room*, *bedroom*, *kitchen doors*, etc. As mentioned in Section 7.1, 80% of the original data is used for training and the rest for testing. However, the degree of imbalance that is originally present in the data is maintained in both the training and the test data.

With the training data is of size 3184, the number of clusters that is considered to be formed using K-means clusters is 100. Although the current study does not involve any empirical study on the effects of number of clusters to be formed on the training data, it would be considered in the future. As mentioned in the CBU algorithm, the threshold value on the degree of imbalance for the clusters is calculated empirical, Figure 8 shows the outcome. The TP Rate for three different classifiers: J48 decision tree, k-nearest neighbor and SMO, is varied with a decreasing value of the threshold τ. TP Rate has a clear increasing trend irrespective of the classifier and becomes constant after a certain point. From the figure, it can be seen that there is no further increase in TP Rate beyond τ=0.04, that is, 4% of the cluster size is minority class examples. Therefore, 0.04 is considered as τ.

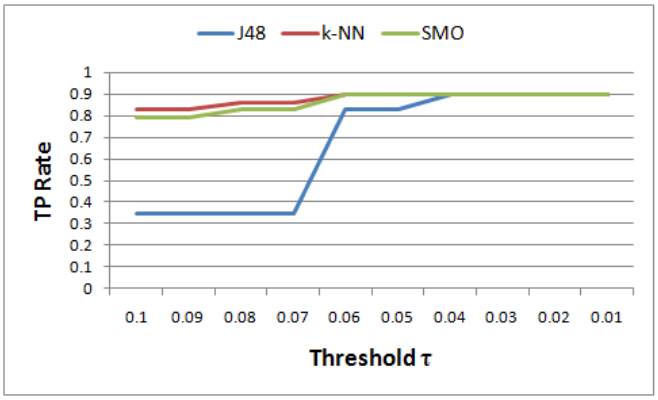

Figure 8: Change in TP Rate with decreasing τ.

Further experimentation to find out the performance of the classifiers for other performance measures is done with τ=0.04. The CBU method is compared with the performance of the classifiers on the original un-sampled dataset and after the application of SMOTE.

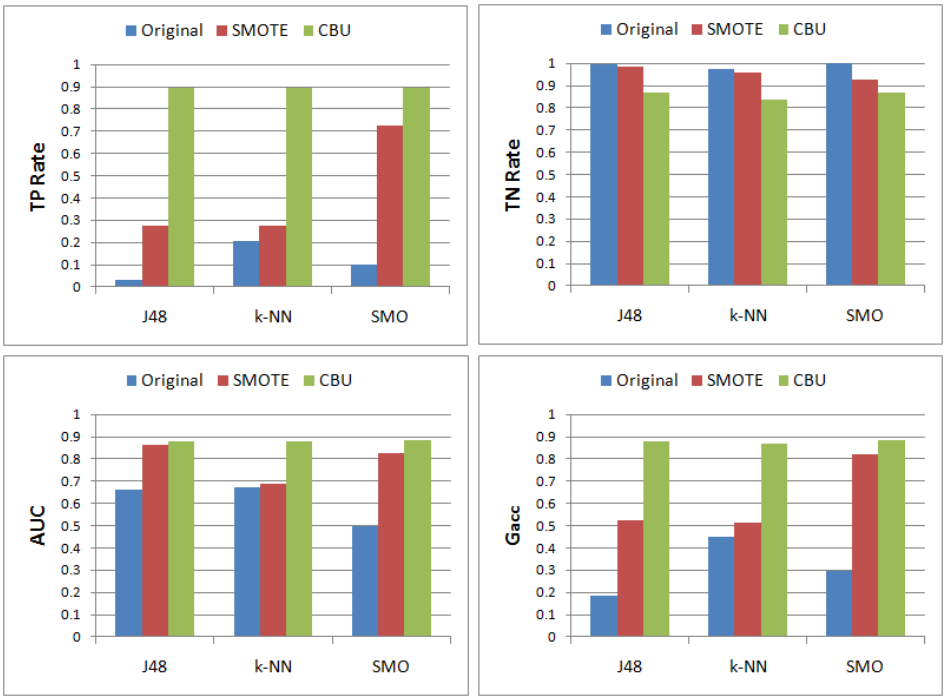

Figure 9: Comparison of (a)TP Rate(top left), (B)TN Rate(top right), (c)AUC(bottom left), and (d)G_{acc}(bottom right).

It should be noted that with the SMOTE method for this dataset, the minority class examples were boosted up to 50% of the entire dataset. Any further increase would incur more cost to synthesize new training examples in a real-life situation.

Figure 9 shows the comparison of TP Rate, TN Rate, Area Under ROC Curve and G_{acc} for all three approaches. Figure 9(a) shows a significantly better performance of CBU as compared to SMOTE in terms of TP Rate. The most interested thing to keep in mind is that the significant increase in TP Rate for CBU was achieved with under-sampling and without incurring extra cost of synthesizing new training examples. AUC and G_{acc} values have also increased. However, the decrease in TN Rate is more as compared to SMOTE, but it is in an acceptable range. k-NN has the lowest TN Rate for CBU and it is approximately 83% which is quite acceptable.

8. Related Work

8.1. 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 prompting systems is flourishing, research groups are taking their own unique way of solving the problem. From a computational perspective the approaches can be broadly classified into five major types: time-based, location-based, context-aware, AI planning and machine learning.

The simplest example of a time based prompt (or reminder, to be more appropriate in this case) is an alarm clock. It produces a general audible or vibration alert to grab the user's attention when the appointed time arrives. Most of the electronic organizers have this feature. Lim et al. [29] 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 .[30] 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.

Location based prompts are more complex than time-based prompts. They are usually used in association with time-based or context-aware prompts (as discussed next) to provide more precise location related reminders to the users. Marmasse et al. [31] did a pioneering work on delivering proactive location-based reminders and messages with the help of their system comMotion that uses GPS to determine location. The Assistive Cognition Project at University of Washington [32] sensed aspects of individual's location and environment (both outdoors and at home) by relying on a wide range of sensors such as GPS, active badges, motion detectors and other ubiquitous computing infrastructure.

Context-aware prompting is the oldest and first of its kind technology. A milestone was set by Dey et al. with the development of CyberMinder [33], a context-aware reminder application based on Context Toolkit [34] that focused on using complex contextual information to determine a prompt situation. Mihailidis et al. [35] proposed the usage of context-aware computing to assist people with dementia for ADLs that needs more privacy such as toileting. HYCARE [36] or hybrid context-aware reminding framework uses a novel scheduling mechanism to handle synchronous time-based and asynchronous event-based reminding services. By conducting an extensive user study, the authors classify the reminding services into four categories and formulate the context-aware reminding rules accordingly.

There has also been a significant amount of work in the application of AI planning approaches for prompting. The Autominder System developed by Pollack et al. [37] uses dynamic Bayesian networks as an underlying domain model to coordinate pre-planned events in an attempt to ensure that scheduled tasks are executed without interfering with each other or with other activities, such as watching television. Pineau et. al. [38] uses variant of partially observable Markov decision processes (POMDPs) to design the high level control system for “Nursebot”, an artificially intelligent robot designed to assist elderly people with daily activities. Boger et al. [39] proposed a planning system that uses Markov decision processes (MDPs) to determine when and

how to provide prompts to a user with dementia for guidance through the activity of hand washing.

There has not been any significant work on applying machine learning methods in this domain. Rudary et al. [40] integrated temporal constraint reasoning with reinforcement learning to build an adaptive reminder system. This algorithm can personalize to a user and adapt to both short and long term changes. Although this approach is useful when there is no direct or indirect user feedback, it relies on a complete schedule of user activities. The Independent LifeStyle Assistant project [41] used machine learning techniques to capture interactions among devices, environment and humans. Patterned behavior profiles were created to build models of what sensor firings correspond to what activities in what order and at what time. Alerts were raised when activities that were probabilistically unlikely, occurred. Also, schedule information for regular activities was learned using machine learning techniques.

8.2. Imbalanced Class Distribution Problem

Several solutions have been proposed over the years to deal with the imbalanced class distribution problem. He et al. [10] have done an extensive literature review on the approaches that have been developed over the past decade to deal with imbalanced class problem. The methods can be broadly classified into data-level methods and algorithm-level methods. Data level methods mainly include under-sampling the majority class [42] to match the size of the other class; over-sampling the minority class [28] to match the size of the other class; and combination of both under and over sampling as proposed by Chawla et al. [11]. Algorithm level methods include: threshold method [43] in which the classifiers yield a score that represents the degree to which an example is a member of a class; one-class learning [44]; and cost-sensitive learning [45] in which unequal misclassification cost is considered between classes.

8.3. Class Overlap Problem

Class overlap is a much more difficult problem than imbalanced class distribution. Therefore, comparatively very less work has been done in this area. One of the reasons for this is that there is no widely accepted way to identify overlap in dataset. Yaohua et al. [46] apply different classifiers to ambiguous and non-ambiguous region and report improved overall performance. However, others [47, 48] proposed that instead of making a decision on the ambiguous region, these points should be categorized as a third “I don’t know” class. Prati et al. [49] explore how overlapping classes affect classifier accuracy in the presence of imbalance in synthetic data.

9. Conclusion

In this chapter, the problem of automated activity intervention in the domain of ambient intelligence has been addressed. In order to ensure a clear understanding of the problem definition, the problem has been discussed in detail. The approaches developed to address this problem are based on the system architecture developed at the Center for Advanced Studies in Adaptive Systems at Washington State University. An initial sampling-based approach that was attempted for this domain is reported and its limitations have been highlighted. A novel method (CBU) to address the issues of

the previous approach is proposed and the experimental results show that CBU performs significantly better than the previous approach.

References

[1] G. Singla, D. J. Cook, and M. Schmitter-Edgecombe, Recognizing independent and joint activities among multiple residents in smart environments, *Journal of ambient intelligence and humanized computing*, vol. 1, pp. 57-63, 2010.

[2] G. Singla, D. J. Cook, and M. Schmitter-Edgecombe, Tracking activities in complex settings using smart environment technologies, *International journal of biosciences, psychiatry, and technology (IJBSPT)*, vol. 1, p. 25, 2009.

[3] E. M. Tapia, S. S. Intille, and K. Larson, Activity recognition in the home using simple and ubiquitous sensors, *Pervasive Computing*, pp. 158-175, 2004.

[4] U. Maurer, A. Smailagic, D. P. Siewiorek, and M. Deisher, Activity recognition and monitoring using multiple sensors on different body positions, 2006, pp. 4 pp.-116.

[5] U. C. Bureau. (2011). *US Population Projections*. Available: http://www.census.gov/population/www/projections/natdet-D1A.html

[6] J. Bates, J. Boote, and C. Beverley, Psychosocial interventions for people with a milder dementing illness: a systematic review, *Journal of Advanced Nursing*, vol. 45, pp. 644-658, 2004.

[7] V. G. Wadley, O. Okonkwo, M. Crowe, and L. A. Ross-Meadows, Mild Cognitive Impairment and everyday function: Evidence of reduced speed in performing instrumental activities of daily living, *American Journal of Geriatric Psych*, vol. 16, p. 416, 2008.

[8] B. Das, C. Chen, A. M. Seelye, and D. J. Cook, An Automated Prompting System for Smart Environments, presented at the 9th International Conference on Smart Homes and Health Telematics, 2011.

[9] B. Das, C. Chen, N. Dasgupta, D. J. Cook, and A. M. Seelye, Automated prompting in a smart home environment, presented at the 2010 IEEE International Conference on Data Mining Workshops, 2010.

[10] H. He and E. A. Garcia, Learning from imbalanced data, *IEEE Transactions on Knowledge and Data Engineering*, pp. 1263-1284, 2008.

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, SMOTE: synthetic minority over-sampling technique, *Journal of Artificial Intelligence Research*, vol. 16, pp. 321-357, 2002.

[12] B. Das and D. J. Cook, Data Mining Challenges in Automated Prompting Systems, presented at the Proceedings of 2011 Internatuional Conference on Intelligent User Interfaces Workshop on Interaction with Smart Objects, 2011.

[13] S. Szewcyzk, K. Dwan, B. Minor, B. Swedlove, and D. Cook, Annotating smart environment sensor data for activity learning, *Technology and Health Care*, vol. 17, pp. 161-169, 2009.

[14] B. L. Thomas and A. S. Crandall, A Demonstration of PyViz, a Flexible Smart Home Visualization Tool, in *IEEE International Conference on Pervasive Computing and Communications*, Seattle, WA, 2011.

[15] F. Provost, T. Fawcett, and R. Kohavi, The case against accuracy estimation for comparing induction algorithms, 1998.

[16] D. J. Hand, *Construction and assessment of classification rules* vol. 15: Wiley, 1997.

[17] J. R. Quinlan, Induction of decision trees, *Machine learning*, vol. 1, pp. 81-106, 1986.

[18] T. M. Mitchell, Machine learning. WCB, *Mac Graw Hill*, p. 368, 1997.

[19] B. E. Boser, I. M. Guyon, and V. N. Vapnik, A training algorithm for optimal margin classifiers, 1992, pp. 144-152.

[20] J. Platt, Sequential minimal optimization: A fast algorithm for training support vector machines, *Advances in Kernel Methods-Support Vector Learning*, vol. 208, pp. 98–112, 1999.

[21] G. M. Weiss and F. Provost, The effect of class distribution on classifier learning: an empirical study, *Rutgers Univ*, 2001.

[22] M. Denil, The Effects of Overlap and Imbalance on SVM Classification, Master's, Dalhousie University, 2010.

[23] I. Jolliffe, Principal component analysis, 2002.

[24] M. Denil and T. Trappenberg, Overlap versus Imbalance, *Advances in Artificial Intelligence*, pp. 220-231, 2010.

[25] I. Tomek, Two modifications of CNN, *IEEE Trans. Syst. Man Cybern.*, vol. 6, pp. 769-772, 1976.

[26] P. Hart, The condensed nearest neighbor rule (corresp.), *Information Theory, IEEE Transactions on*, vol. 14, pp. 515-516, 1968.

[27] G. E. Batista, R. C. Prati, and M. C. Monard, A study of the behavior of several methods for balancing machine learning training data, *ACM SIGKDD Explorations Newsletter*, vol. 6, pp. 20-29, 2004.
[28] M. Kubat and S. Matwin, Addressing the curse of imbalanced training sets: one-sided selection, 1997, pp. 179-186.
[29] M. Lim, J. Choi, D. Kim, and S. Park, A smart medication prompting system and context reasoning in home environments, 2008, pp. 115-118.
[30] M. Oriani, E. Moniz-Cook, G. Binetti, G. Zanieri, G. Frisoni, 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 & Mental health*, vol. 7, pp. 22-27, 2003.
[31] N. Marmasse and C. Schmandt, Location-aware information delivery with commotion, 2000, pp. 361-370.
[32] H. Kautz, O. Etzioni, D. Fox, D. Weld, and L. Shastri, Foundations of assisted cognition systems, *University of Washington, Computer Science Department, Technical Report, Tech. Rep*, 2003.
[33] A. Dey and G. Abowd, Cybreminder: A context-aware system for supporting reminders, 2000, pp. 201-207.
[34] A. K. Dey, G. D. Abowd, and D. Salber, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, *Human–Computer Interaction*, vol. 16, pp. 97-166, 2001.
[35] A. Mihailidis and G. Fernie, Context-aware assistive devices for older adults with dementia, *Gerontechnology*, vol. 2, pp. 173-189, 2002.
[36] K. Du, D. Zhang, X. Zhou, M. Mokhtari, M. Hariz, and W. Qin, HYCARE: A hybrid context-aware reminding framework for elders with mild dementia, *Smart Homes and Health Telematics*, pp. 9-17, 2008.
[37] M. E. Pollack, L. Brown, D. Colbry, C. E. McCarthy, C. Orosz, B. Peintner, S. Ramakrishnan, and I. Tsamardinos, Autominder: An intelligent cognitive orthotic system for people with memory impairment, *Robotics and Autonomous Systems*, vol. 44, pp. 273-282, 2003.
[38] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun, Towards robotic assistants in nursing homes: Challenges and results, *Robotics and Autonomous Systems*, vol. 42, pp. 271-281, 2003.
[39] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis, A decision-theoretic approach to task assistance for persons with dementia, 2005, p. 1293.
[40] M. Rudary, S. Singh, and M. E. Pollack, Adaptive cognitive orthotics: combining reinforcement learning and constraint-based temporal reasoning, 2004, p. 91.
[41] K. Z. Haigh, L. M. Kiff, and G. Ho, The independent lifestyle assistant: Lessons learned, *Assistive Technology*, vol. 18, pp. 87-106, 2006.
[42] S. Kotsiantis and P. Pintelas, Mixture of expert agents for handling imbalanced data sets, *Annals of Mathematics, Computing & TeleInformatics*, vol. 1, pp. 46-55, 2003.
[43] G. Weiss, Mining with rarity: A unified framework, *SIGKDD explorations*, vol. 6, pp. 7-14, 2004.
[44] B. Raskutti and A. Kowalczyk, Extreme re-balancing for SVMs: a case study, *ACM SIGKDD Explorations Newsletter*, vol. 6, pp. 60-69, 2004.
[45] C. Elkan, The foundations of cost-sensitive learning, 2001, pp. 973-978.
46] T. Yaohua and G. Jinghuai, Improved classification for problem involving overlapping patterns, *IEICE TRANSACTIONS on Information and Systems*, vol. 90, pp. 1787–1795, 2007.
[47] T. P. Trappenberg and A. D. Back, A classification scheme for applications with ambiguous data, 2000, p. 6296.
[48] S. Hashemi and T. Trappenberg, Using SVM for Classification in Datasets with Ambiguous data, *SCI 2002*, 2002.
[49] R. C. Prati, G. E. Batista, and M. C. Monard, Class imbalances versus class overlapping: an analysis of a learning system behavior, *MICAI 2004: Advances in Artificial Intelligence*, pp. 312-321, 2004.