MACHINE LEARNING CHALLENGES FOR
AUTOMATED PROMPTING IN
SMART HOMES

By
BARNAN DAS

A dissertation submitted in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

AUGUST 2014

© Copyright by BARNAN DAS, 2014
All Rights Reserved
To the Faculty of Washington State University:

The members of the committee appointed to examine the dissertation of BARNAN DAS find it satisfactory and recommend that it be accepted.

____________________
Diane J. Cook, Ph.D., Chair

____________________
Narayanan C. Krishnan, Ph.D.

____________________
Lawrence B. Holder, Ph.D.

____________________
Maureen Schmitter-Edgecombe, Ph.D.

____________________
Behrooz A. Shirazi, Ph.D.

____________________
Alex Mihailidis, Ph.D.
ACKNOWLEDGMENTS

Although this dissertation is an individual work, I could never have reached the heights or explored the depths without the help, support, guidance and efforts of a lot of people.

First and foremost, I would like to thank my advisor, Dr. Diane Cook, whose guidance and encouragement motivated me to do focused research throughout my time at the CASAS Lab. The joy and enthusiasm she has for her research is contagious and truly inspirational. From very short response times of emails to understanding the big picture of research projects, the list of lessons I have learned from Dr. Cook is beyond the scope of a dissertation's acknowledgement section. I will always be indebted to her for instilling in me the qualities of a high-impact researcher.

Alongside Dr. Cook, I would like to thank CK (Dr. Narayanan C. Krishnan) for being a mentor and friend since 2010. CK’s strong theoretical understanding has always amazed me, and was also the primary reason why I bothered him whenever I was stuck with a problem. His patience in understanding my doubts/concerns and innumerable number of brainstorming sessions helped me throughout the technically challenging times in my Ph.D. pursuit.
My committee members played a pivotal role in shaping the problem I have addressed in this dissertation. I am particularly grateful to Dr. Maureen Schmitter-Edgecombe and her team for their help and support in conducting the smart home studies and providing me with the data I needed for the computational experiments. Dr. Schmitter-Edgecombe’s insightful comments on my work helped me attain clarity of my research goals. Dr. Larry Holder’s machine learning course project, in my first semester of Ph.D., ignited my passion for machine learning and the wonders data science can do. I would also like to thank Dr. Behrooz Shirazi and Dr. Alex Mihailidis for their advise and guidance since my preliminary examination.

A very special thanks to the person who has been my idol since as far back in my childhood as I can remember, my elder brother, Mistida (Dr. Samik Ghosh). Whenever I am in a philosophical dilemma, there is only one person on planet earth to whom I can go to, and its you. You have given me courage, inspiration and clarity of life. Thank you!

I am, and will always be thankful to my parents and my brother Barshan for their unconditional love and support. They believed in me during those rough patches of my life when I stopped believing in myself.

They say one best book is equal to hundred good friends, but one good friend is equal to a library. I can proudly say that I have three such libraries. Thank you Tanmoy, Oishik and Kanchan for keeping me grounded and making me realize that
there is life beyond work.

In the end, I would like to thank the person who stood by my side through all the perilous fights, Sabita Nahata, my best friend, my greatest critic, my toughest competitor, and my strongest support system. You were with me when I had nothing, and I promise to be with you forever and beyond!
MACHINE LEARNING CHALLENGES FOR
AUTOMATED PROMPTING IN
SMART HOMES

Abstract

by Barnan Das, Ph.D.
Washington State University
August 2014

Chair: Diane J. Cook

As the world’s population ages, there is an increased prevalence of diseases related to aging, such as dementia. Caring for individuals with dementia is frequently associated with extreme physical and emotional stress, which often leads to depression. Smart home technology and advances in machine learning techniques can provide innovative solutions to reduce caregiver burden. One key service that caregivers provide is prompting individuals with memory limitations to initiate and complete daily activities. We hypothesize that sensor technologies combined with machine learning techniques can automate the process of providing reminder-based interventions or prompts. This dissertation focuses on addressing machine learning challenges that arise while devising an effective automated prompting system.
Our first goal is to emulate natural interventions provided by a caregiver to individuals with memory impairments, by using a supervised machine learning approach to classify pre-segmented activity steps into prompt or no-prompt classes. However, the lack of training examples representing prompt situations causes imbalanced class distribution. We proposed two probabilistic oversampling techniques, RACOG and wRACOG, that help in better learning of the “prompt” class. Moreover, there are certain prompt situations where the sensor triggering signature is quite similar to the situations when the participant would probably need no prompt. The absence of sufficient data attributes to differentiate between prompt and no-prompt classes causes class overlap. We propose ClusBUS, a clustering-based under-sampling technique that identifies ambiguous data regions. ClusBUS preprocesses the data in order to give more importance to the minority class during classification.

Our second goal is to automatically detect activity errors in real time, while an individual performs an activity. We propose a collection of one-class classification-based algorithms, known as DERT, that learns only from the normal activity patterns and without using any training examples for the activity errors. When evaluated on unseen activity data, DERT is able to identify abnormalities or errors, which can be potential prompt situations.
We validate the effectiveness of the proposed algorithms in predicting potential prompt situations on the sensor data of ten activities of daily living, collected from 580 participants, who were part of two smart home studies.
# TABLE OF CONTENTS

ACKNOWLEDGMENTS ................................................................. iii

ABSTRACT ................................................................. vi

LIST OF TABLES ............................................................... xi

LIST OF FIGURES ............................................................... xiii

1. Introduction ............................................................... 1
   1.1 Motivation ............................................................... 1
   1.2 Research Hypothesis .................................................... 6
   1.3 Related Work in Prompting Technologies ............................... 7
   1.4 Summary of Proposed Approaches ..................................... 19
   1.5 Contribution ........................................................... 22

2. The CASAS Smart Home Infrastructure ..................................... 24
   2.1 Smart Home Test Bed .................................................. 25
   2.2 Smart Home Sensors ................................................... 28
   2.3 CASAS Lightweight Middleware (CLM) .................................. 33
   2.4 Smart Home in a Box (SHiB) .......................................... 39
   2.5 Studies ................................................................. 42

3. Imbalanced Class Distribution ............................................... 46
   3.1 Introduction ........................................................... 46
   3.2 Related Work .......................................................... 51
   3.3 The Prompting Dataset ................................................ 56
   3.4 Probabilistic Approaches for Oversampling ............................. 57
   3.5 Experimental Setup ................................................... 72
   3.6 Results and Discussion ............................................... 82
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Raw-sensor data stored in the database.</td>
</tr>
<tr>
<td>2.2</td>
<td>Activities of daily living for two different studies.</td>
</tr>
<tr>
<td>2.3</td>
<td>Human annotated sensor events.</td>
</tr>
<tr>
<td>3.1</td>
<td>Description of generated features.</td>
</tr>
<tr>
<td>3.2</td>
<td>Description of selected datasets.</td>
</tr>
<tr>
<td>3.3</td>
<td>Classifiers and parameter values.</td>
</tr>
<tr>
<td>3.4</td>
<td>AUC-ROC for C4.5 decision tree.</td>
</tr>
<tr>
<td>3.5</td>
<td>Results for SVM.</td>
</tr>
<tr>
<td>3.6</td>
<td>Results for k-nearest neighbor.</td>
</tr>
<tr>
<td>3.7</td>
<td>Results for logistic regression.</td>
</tr>
<tr>
<td>5.1</td>
<td>One-class classification vs supervised learning.</td>
</tr>
<tr>
<td>5.2</td>
<td>Distribution of data samples from Studies 1 and 2.</td>
</tr>
<tr>
<td>5.3</td>
<td>Distribution of error data samples on various activity error types.</td>
</tr>
<tr>
<td>5.4</td>
<td>Confusion matrix for one-class classification.</td>
</tr>
<tr>
<td>5.5</td>
<td>$\nu$ and $\gamma$ values for all activities.</td>
</tr>
<tr>
<td>5.6</td>
<td>Error classification results for Study 1 dataset.</td>
</tr>
<tr>
<td>5.7</td>
<td>Error classification results for Study 2 dataset.</td>
</tr>
<tr>
<td>5.8</td>
<td>$F_1$-score and MCC for Study 1 dataset.</td>
</tr>
<tr>
<td>5.9</td>
<td>$F_1$-score and MCC for Study 2 dataset.</td>
</tr>
</tbody>
</table>
6.1 Attribute information for prompting dataset. ........................ 189
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Popularity trend of the phrase “connected home”</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>Summary of proposed solutions</td>
<td>21</td>
</tr>
<tr>
<td>2.1</td>
<td>On-campus smart home</td>
<td>25</td>
</tr>
<tr>
<td>2.2</td>
<td>Sensor layout of on-campus smart apartment</td>
<td>27</td>
</tr>
<tr>
<td>2.3</td>
<td>Wireless motion sensor</td>
<td>30</td>
</tr>
<tr>
<td>2.4</td>
<td>Magnetic door sensor</td>
<td>31</td>
</tr>
<tr>
<td>2.5</td>
<td>Item pressure sensor</td>
<td>32</td>
</tr>
<tr>
<td>2.6</td>
<td>Vibration sensor</td>
<td>33</td>
</tr>
<tr>
<td>2.7</td>
<td>CLM middleware architecture</td>
<td>35</td>
</tr>
<tr>
<td>2.8</td>
<td>Smart Home in a Box kit</td>
<td>41</td>
</tr>
<tr>
<td>3.1</td>
<td>Chow-Liu dependence tree construction</td>
<td>61</td>
</tr>
<tr>
<td>3.2</td>
<td>Dependence trees for the <em>prompting</em> dataset</td>
<td>62</td>
</tr>
<tr>
<td>3.3</td>
<td>A Markov chain</td>
<td>63</td>
</tr>
<tr>
<td>3.4</td>
<td>Gibbs sampling</td>
<td>65</td>
</tr>
<tr>
<td>3.5</td>
<td>RACOG algorithm</td>
<td>68</td>
</tr>
<tr>
<td>3.6</td>
<td>wRACOG algorithm</td>
<td>71</td>
</tr>
<tr>
<td>3.7</td>
<td>Sensitivity for C4.5 decision tree</td>
<td>84</td>
</tr>
<tr>
<td>3.8</td>
<td>G-mean for C4.5 decision tree</td>
<td>85</td>
</tr>
<tr>
<td>3.9</td>
<td>ROC curves produced by C4.5 decision tree</td>
<td>86</td>
</tr>
</tbody>
</table>
3.10 Comparison of number of instances added by different methods. 91
3.11 Comparison of PromptingBaseline with RACOG and wRACOG 93
4.1 Data without overlap (left) and data with overlap (right) 100
4.2 3D PCA plot for prompting data 101
4.3 Steps taken to address class overlap 102
4.4 ClusBUS algorithm 109
4.5 Schematic representation of ClusBUS algorithm 110
4.6 DBSCAN algorithm 112
4.7 Ensemble created from the classifiers of all activities 114
4.8 TPR, FPR, and AUC obtained from C4.5 for different 1st quantile
boundary values of q-quantile r values 117
4.9 Comparison based on True Positive Rate 119
4.10 Comparison based on False Positive Rate 119
4.11 Comparison based on AUC 121
4.12 Comparison based on G-mean 121
5.1 Distribution of error annotation across 8 activities of Study 1 129
5.2 Modeling activity error prediction 144
5.3 Distribution of normal and erroneous activity data in Studies 1 and 2 145
5.4 Baseline algorithm for outlier detection 152
5.5 Ensemble of OCSVM and error classification models 159
5.6 Temporal accuracy 162
5.7 Surface plot of \( \nu \) and \( \gamma \) values for OCSVM 164
5.8 Recall (left) and precision (right) values for all activities. ............ 166
5.9 RMSE values for all activities in seconds (left) and number of events (right). ................................................................. 169
5.10 Recall (left) and precision (precision) values for all methods. ....... 175
5.11 RMSE values of all methods in seconds (left) and number of events (right). ................................................................. 178
Dedication

To

my life mentor, Mistida,

and

my best friend, Sabita.
CHAPTER 1. INTRODUCTION

1.1 Motivation

The world’s population is aging [WHO, 2011]. The estimated number of individuals over the age of 85 is expected to triple by 2050 [Vincent and Velkoff, 2010]. The increase in the number of individuals crossing higher life expectancy thresholds has in turn made a large section of the older population susceptible to cognitive impairments such as Alzheimer’s disease and dementia. According to the Alzheimer’s Association’s report [ALZ, 2014], an estimated 5.5 million Americans have Alzheimer’s disease in 2014. By 2050, the global number of people who are 65 and older with some form of cognitive impairment may nearly triple, barring medical breakthroughs to prevent, slow or stop the disease. These older adults face difficulties in completing both simple (for example, eating and dressing) and complex (for example, cooking and taking medication) activities of daily living (ADLs) [Wadley et al., 2008].

There are currently 15.5 million family and friends providing unpaid care to those with Alzheimer’s and other dementias. One key service that caregivers provide is prompting individuals with memory limitations to initiate and complete daily ac-
activities. Generally, the caregivers do not prompt each step of an activity. Instead, the
caregiver recognizes when the care recipient is experiencing difficulty with an activity
and provides a prompt at that time to help in performing the activity completely.
The number of prompts that a caregiver typically provides depends upon the level
of cognitive impairment. Worsening of the level of impairment demands an increased
number of caregiver duties and thus places a heavier burden on the caregiver. Care-
givers for individuals with dementia are also at increased risk of health problems,
including higher levels of stress hormones, reduced immune function, slower wound
healing, new hypertension, higher serum insulin levels and related markers of dia-
betes, cardiovascular disease, increased morbidity and premature death [ALZ, 2014].
This issue motivates the need for an automated intervention system which can help
older adults with ADLs by prompting them with effective instructions to complete
their activities successfully. This is a vital step in reducing health risks and alleviat-
ing the burden of many caregivers that are helping a large section of the population.
Technology-oriented long-term care facilities in individual homes can also provide a
low-cost alternative to spending substantial amount of time and money in hospitals,
nursing homes or health clinics. Moreover, nearly 90% of people over age 65 want to
stay in their home for as long as possible [NCSL, 2011].

With recent advances in sensor technologies, pervasive computing and machine
learning, it is possible to transform a regular home into a smart home with bare
minimum infrastructural modifications and at a reasonable cost. A smart home is typically equipped with ambient sensors that can monitor the resident’s movements and interactions with home appliances and objects of daily use. Some of the efforts for realizing such smart homes have been demonstrated in actual physical testbeds such as CASAS [Rashidi and Cook, 2009], Gator Tech Smart House [Helal et al., 2005], iDorn [Doctor et al., 2005], Aware Home [Abowd and Mynatt, 2004], PlaceLab [LaMarca et al., 2005], and Tiger Place [Rantz et al., 2008].

We hypothesize that smart home technology-driven solutions can reduce caregiver burden and provide long-term low cost care facilities in individuals’ homes. For realization of these goals, smart homes need to play two different roles. First, a smart home should be able to understand the resident’s daily behavior which involves her/his daily activity routine and interaction with household objects and electronic appliances. Second, the smart home should be able to provide proactive intervention to the residents in order to help them maintain a healthy lifestyle. Over the past decade a significant amount of work has been done in the former area, which includes activity recognition [Singla et al., 2009, Krishnan and Cook, 2014, Chen et al., 2012], activity discovery [Cook et al., 2013], activity prediction [Nazerfard, 2014] and energy usage monitoring [Jahn et al., 2010], among other applications [Demiris and Hensel, 2008]. Moreover, home monitoring technologies are likely to witness further advancement with the recent boom of connected homes in the consumer electronic industry. A
Google Trends™ image (shown in Figure 1.1) for the search query “connected home” from December 2008 to January 2014 clearly reflects the increase in popularity of home monitoring technologies in the US and the world. However, apart from few academic research groups [Kautz et al., 2003, Pollack et al., 2003, Hoey et al., 2010], much less work has been done in the area of proactive automated interventions. The cycle of holistic care based on smart homes is not complete until this is done.

The goal of this dissertation is to lay the foundation of an automated prompting
system that can assist older adults with successful completion of ADLs. A prerequisite to providing effective assistance is the accurate detection of the “difficulties” individuals face while performing the ADLs. These activity-related difficulties can be considered as activity “errors” and can vary widely in terms of their criticality with respect to a specific activity. In this dissertation, we propose approaches that can detect as many errors as possible in the ADLs. However, it should be noted that in a real-world setting prompts are generally not issued for all error types. In our smart home studies we found that prompts are mainly issued for critical omission or substitution types of errors. For example, in the watering plants activity, failing to fill up the watering can with water is a crucial omission and needs a prompt. On the other hand, failing to empty extra water from the can after watering the plants is not a crucial error and does not affect the successful completion of the activity.

Choosing the most critical errors out of all that are detected in order to issue prompts is an important problem, particularly with older adults. This is because a fraction of the detected errors can possibly be false positives. Issuing prompts for all detected errors can lead to annoyance and sometimes prove to be unsafe for specific activities. For example, if an individual is prompted very frequently while taking medication, it can be a major cause of distraction which can lead to the intake of incorrect medicines. On the other hand, not catching the errors that can result in incomplete activities or unsafe states, such as leaving the water faucet or burner on
after cooking, can lead to potential hazards. Therefore, the fraction of detected errors that should be used to issue prompts is activity specific and needs to be carefully chosen by gerontologists and clinical psychologists.

An accurate and precise solution to detecting activity errors necessitates us to address machine learning problems that arise in smart home sensor data. In the following sections, we state our research hypothesis, discuss related work in proactive prompting technologies and give a brief overview of how we address the machine learning challenges.

1.2 Research Hypothesis

The primary goal of this dissertation is to propose technology-based solutions that can aid older adults with independent living in their own homes. We hypothesize that machine learning algorithms trained on smart home sensor data can predict when an individual faces difficulty while performing everyday activities. The prediction of activity-related “difficulties” can be used to prompt older adults with effective instructions to complete their activities successfully. Our hypothesis is dependent on another hypothesis that smart home sensors are capable of recognizing and tracking everyday activities, which has been validated by Krishnan and Cook [Krishnan and Cook, 2014].
We validate our hypothesis by considering two different methodologies. First, we emulate natural interventions provided by a caregiver by using a supervised machine learning approach to classify activity steps, pre-segmented by human annotators, into prompt or no-prompt classes. An efficient solution to this classification problem faces two fundamental machine learning challenges: imbalanced class distribution and overlapping classes. Our second methodology is to automatically detect activity errors in real time, while an individual performs an activity, using no training data for the activity errors. We hypothesize that one-class classification-based algorithms trained on the normal activity patterns can accurately detect activity errors.

1.3 Related Work in Prompting Technologies

In a smart home, a prompt can be any form of verbal or non-verbal intervention delivered to a resident on the basis of time, context or acquired intelligence that helps in successful (in terms of time and purpose) completion of an activity. Research groups have also used terms such as reminders, notifications and alerts, depending on the granularity of interventions, methods of delivery, and applications [Kaushik et al., 2008]. Intelligent prompting technologies have been in existence for quite some time with different research groups taking their own unique approach to arrive at a solution. In the following, we discuss the various approaches of automated prompting
1.3.1 Time-Based

The simplest example of a time-based prompting or reminder system is an alarm clock, which produces a general audible or vibration alert to acquire the user’s attention when the appointed time arrives. Most electronic devices such as phones and organizers include this feature. In addition, Google Calendar [Sjogreen, 2006] is currently being widely used on the Internet to set reminders for certain events.

Despite the simplicity, reliability, availability, and precision of time-based prompts, they have inherent limitations. Most notably, they have to be delivered at a specific time and do not take into account important aspects of an individual’s environment. These characteristics can make time-based prompts ineffective in everyday situations, such as when an individual is sleeping, or in the middle of a complex and time-sensitive activity. Consistent with this, Lundell et al. [Lundell et al., 2007] found that time-based prompts resulted in significantly lower medication adherence than context-based prompts in a study of medication prompting with healthy older adults. Similarly, Kaushik et al. [Kaushik et al., 2008] found that older adults perceived context-aware prompts as more helpful than time-based prompts. In addition, while time-based prompts can assist with activity initiation at predetermined times, they
cannot detect errors once an activity has begun and subsequently assist with activity completion. Furthermore, even if time-based prompts could be adapted according to the user’s demands, the burden to set the prompt remains on the user, which poses a challenge for older adults with cognitive impairment.

1.3.2 Location-Based

Location-based prompts are more complex than time-based prompts. They are often used in association with time-based or context-aware prompts to provide more precise location-related reminders to a user. By taking into account an individual’s environment and delivering prompts targeted to a specific location, location-based prompts are usually more effective than simple time-based prompts. For example, Marmasse et al. [Marmasse and Schmandt, 2000] used GPS to determine location and pioneered work on delivering proactive location-based reminders and messages, such as sending a grocery list through a mobile device when in close physical proximity to a grocery store. Similarly, Place-Its application [Sohn et al., 2005], designed around the post-it usage metaphor, used mobile phones as the ubiquitous platform to detect an individual’s location and deliver reminders at remote locations.

Smart home sensor technology has made it possible to determine indoor location of the user by using simple, low-cost, passive infrared motion detectors in different
rooms of the house. While indoor location information of a user can be used to provide simple location-based reminders, such as taking medication in the morning when the user is in the kitchen area, it can also be combined with the knowledge of context-awareness to recognize complex everyday activities [Cook and Das, 2005, 2007, Rashidi and Cook, 2009]. Although location-based reminder systems have the advantage of being relatively simple and effective to remind a person to perform an activity at the appropriate location, limitations of these systems are that they do not have the capability of detecting activity errors and delivering prompts during a complex activity.

1.3.3 Context-Aware

Although context-aware prompting systems have been used for a number of years, several groups have made important developments in this area recently. Context-aware prompting systems have a significant advantage of delivering assistance to individuals when needed, as compared to predefined rules of prompt delivery used in time-based and location-based prompting systems.

The Forget-me-not project [Lamming and Flynn, 1994], one of the pioneering works in this area, used a small electronic device similar to a PDA to gather contextual information from a user’s environment. It associated items of interest with icons to
help users to remember various tasks that needed to be completed, such as email and telephone calls. Another milestone was set by Dey et al. when they developed CyberMinder [Dey and Abowd, 2000], a context-aware reminder application based on the Context Toolkit [Dey et al., 2001] that focused on using complex contextual information beyond time and location to determine a prompt situation. Hristova et al. [Hristova et al., 2008] later used the Context Toolkit to build a prompting application for ambient assisted living to support heart rate monitoring, medication prompting, generation of agenda reminders, weather alerts, and emergency notifications. The system was, however, limited by a lack of data privacy, security, and robust conflict reasoning mechanisms.

Context-aware computing to assist people with dementia to complete ADLs that require privacy, such as toileting, was proposed by Mihailidis and Fernie [Mihailidis and Fernie, 2002]. The authors highlighted the importance of gaining and using information about the older adult population’s unique needs to inform the design of prompting technologies that act like human caregivers and have the capability to continually learn and grow with the user. Later, Kautz et al. developed an artificial intelligence planning based approach, called the Assistive Cognition Project [Kautz et al., 2002, 2003], which included novel representation and reasoning techniques for context-aware cognitive assistance that can help individuals with Alzheimer’s disease carry out multi-step everyday tasks like cooking.
HYCARE [Du et al., 2008], or hybrid context-aware reminding framework, used a novel scheduling mechanism to handle synchronous time-based and asynchronous event-based reminding services. The scheduling mechanism was designed to organize, coordinate, and resolve conflicts between the different types of reminders, categorized by time and event, and further into fixed/urgent or relevant. In order to deal with possible conflict between the prompts or asynchronous events happening (e.g., phone call) while a prompt is being delivered, the processing system prioritizes the types of prompts into a hierarchy and the more urgent prompts take precedence over the less urgent prompts. Overall, these unique aspects of this prompting system allow it to be more flexible and generalizable to what happens in older adults’ daily lives, allowing for and handling things such as interruptions and deviations from strict time and event based schedules when appropriate.

As smart phones gain popularity over personal computers, smart phone-based cognitive assistance [Helal et al., 2003, Das et al., 2012] is also gaining increased attention. For example, Helal et al. [Helal et al., 2003] developed smart phone applications that interact with a smart house sensor network to assist individuals with activities of daily living by means of reminders, orientation, context-sensitive teaching and monitoring. The applications also play a proactive role in enhancing the level of awareness of the users by notifying them when certain events occur.

Due to its broader application opportunities, context-aware prompting is also
used in workplaces to help users remain engaged and recall task routines. For example, Chang et al. [Chang et al., 2009] proposed an approach to provide distributed cognition support of work engagement for persons with traumatic brain injury, cerebral palsy and intellectual disability among others. This system is capable of providing highly customized prompts to a user that are mainly triggered by context.

1.3.4 AI Planning

Assistance in a smart home to increase the independence of individuals with cognitive impairment involves not only reminders of events at certain times or places, but also provide guidance when errors are committed during multi-step activities, such as cooking, taking medication, or carrying out household chores. Approaches are likely to be benefitted from utilization of artificial intelligence that enables detection of critical errors as they occur in real time and provision of prompting assistance after errors occur. However, the applications of AI-based planning approaches in smart homes have been very limited.

Marquardt et al. used a planning approach for service composition [Marquardt and Uhrmacher, 2008] to facilitate the cooperation of multiple devices in smart environments to enable real-time service support. Other groups have used plan recognition and temporal reasoning techniques to describe goals and plans of users at various lev-
els of abstraction in order to allow for greater flexibility and spontaneity of prompt delivery in various changing situations [Kautz et al., 2003]. Levinson [Levinson, 1997] used a planning based handheld prompting system to help individuals with traumatic brain injury maintain their independence in carrying out daily activities. This prompting system used classical deterministic planning algorithms to compute the best plan for completion of an activity and provided step-by-step guidance through tasks in the form of visual and audio cues. Pollack et al. developed a dynamic Bayesian network-based prompting system [Pollack et al., 2003], named Autominder, that used a Bayesian network as the 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.

A planning system that uses Markov decision processes (MDPs) to determine when and how to provide prompts to users with dementia to guide them through the activity of handwashing was proposed by Boger et al. [Boger et al., 2006]. The hand washing task was divided into five different sequential steps required for proper completion. With the help of a video camera, the user’s progress through the activity was monitored and a prompt was issued when there was a user regression/departure from the appropriate sequence of steps. This framework was named as the COACH system and extended to have three significant changes [Mihailidis et al., 2008]: use of marker-less flocking to track the activity, usage of partially observable Markov
decision process (POMDP) to model the hand washing guidance problem, and the refinement of audio prompts and video demonstrations. Although this framework was proposed to be generalizable to other ADLs, no further work has been published. Pineau et. al. [Pineau et al., 2003] used variant of POMDPs to design the high level control system for an artificially intelligent robot, Nursebot, to assist older adults with daily activities. The robot primarily provided intelligent reminders regarding specific activities but also engaged in a certain degree of social interaction.

1.3.5 Machine Learning

Older adults require assistance for initiation or completion of highly complex activities, which necessitates the development of effective machine learning-based prompting approaches. Due to the variety of ways in which activity errors can be committed, AI planning-based approaches might be inefficient. In these situations, machine learning techniques could be used effectively to predict a prompt situation based on activity data collected from the smart home sensors.

In the Assistive Cognition Project [Kautz et al., 2003], dynamic Bayesian networks were used to create predictive models of user behavior from sensor data. Rudary et al. [Rudary et al., 2004] integrated temporal constraint reasoning with reinforcement learning to build an adaptive reminder system, which 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 preprogrammed schedule of user activities. The Independent LifeStyle Assistant project [Haigh et al., 2006] 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 were learned using machine learning techniques. Machine learning based prompting techniques can harness the power to sophisticated statistical models to spontaneously adapt to a user’s changing environment.

Most of the existing prompting technologies usually work well for activity level prompting situations and typically for activity initiation. Automated prompting at the granularity of sub-activities (for example, retrieve broom from supply closet is a sub-activity step for the Sweeping and Dusting task), however, is a harder problem than activity-level prompts. Designing a technique to make the prompting system work in real-time on streaming sensor data is even harder. There are only a few research groups who have worked on sub-activity level prompting. Most of these solutions are either based on video analysis [Boger et al., 2006, Hoey et al., 2010] or monitoring interactions with a variety of objects of daily use [Zhang et al., 2008, Feki
et al., 2009] in the smart home using sensors, such as RFID tags.

Hoey et al. [Hoey et al., 2010] proposed a real-time computer vision system to assist dementia patients with hand washing task. The proposed approach combines a Bayesian sequential estimation framework for tracking hands and towel. A decision theoretic framework is used for computing policies of all possible actions. The decision policies which dictate system actions are computed using Partially Observable Markov Decision Process (POMDP) using a point-based approximate solution technique. The tracking and decision making systems are coupled using a heuristic method for temporally segmenting the input video stream based on the continuity of the belief state. This methodology performs a very good job with prompting at appropriate activity steps. However, it can cause major privacy concerns to the participants due to the use of video input. Moreover, due to reliance of this technique on video analysis, it is not a very generalizable framework because significant modifications need to be made to use similar strategies with other activities of daily living.

On the other hand, other approaches [Zhang et al., 2008, Feki et al., 2009, Chu et al., 2011] use sensor platforms that can provide deep and precise insight into sub-activity steps. For example, usage of RFID tags with objects of daily use in the smart home is a very common strategy to gather 1:1 mapping between sensors and activity steps. Some of these approaches use complex plan recognition models for predicting the probability of a certain action for a state when activity is modeled as a stochastic
process such as a Markov chain. A survey on a variety of prompting technologies can be found in the paper by Seelye et al. [Seelye et al., 2012].

To evaluate the effectiveness of technology-based prompts, Seelye et al. [Seelye et al., 2013] developed and experimented with a graded hierarchy of prompts (based on audio or video and direct or indirect) to investigate both the amount of prompting and type of prompts required to assist individuals with mild cognitive impairment (MCI) in completing routine activities. It was found in this study that the technology-based prompts enabled participants to correct critical errors in activities and get back on track. Moreover, user feedback from participants indicated that in general they perceived the prompting technology to be very helpful and appropriately timed.

In our smart home infrastructure, we avoid using any sensor technology that represent a potential threat to privacy. This, however, causes major hurdles in sensing fine grained activity information. On the other hand, we are motivated to propose an efficient, non-intrusive and generic solution to the automated sub-activity level prompting problem. To the best of our knowledge, this problem with the given infrastructural limitations has not been addressed in the past.
1.4 Summary of Proposed Approaches

We hypothesize that sensor technologies combined with machine learning techniques can automate the process of providing reminder-based interventions or prompts. This dissertation focuses on addressing machine learning challenges that arise while devising an accurate and precise automated prompting system.

Our first goal is to automate prompt timing using supervised learning methods by emulating the timing of natural interventions provided by a caregiver. We postulate that prompt timing can be automated by incorporating contextual information of activities gathered from sensors located in a smart home. The sensor data collected from the smart home are used to generate various contextual features of an individual’s daily activities. Machine learning algorithms are trained on the data to classify a “prompt” situation from a “no-prompt” situation. However, lack of training samples representing prompt situations raises a fundamental machine learning problem known as imbalanced class distribution.

We propose two probabilistic oversampling approaches, namely RACOG and wRACOG, to synthetically generating and strategically selecting new minority class samples. The proposed approaches use the joint probability distribution of data attributes and Gibbs sampling to generate new minority class samples. While RACOG selects samples produced by the Gibbs sampler based on a predefined lag, wRACOG
selects those samples that have the highest probability of being misclassified by the existing learning model. We validate our approach on the prompting dataset collected from 400 older adults and nine publicly available datasets that are carefully modified to exhibit class imbalance. In addition, we compare the classification performance of the proposed methods with three other existing resampling techniques. Imbalanced class distribution in the context of automated prompting is discussed in Chapter 3.

In addition, studies conducted with older adult participants in our smart home testbed show that there are certain prompt situations where the sensor triggering signature is quite similar to the situations when the participant would probably need no prompt. This kind of situation is prevalent in daily activities that involve object interactions. As our sensor infrastructure when the study was conducted did not have dedicated sensors to track object usage, it is difficult to gauge from the raw sensor data if the participant actually committed an error in such activities. Thus, the absence of sufficient data attributes to differentiate between prompt and no-prompt classes causes class overlap.

Our solution, ClusBUS, is a clustering-based under-sampling technique that identifies data regions where minority class samples are embedded deep inside majority class. By removing majority class samples from these regions, ClusBUS preprocesses the data in order to give more importance to the minority class during classification. We discuss the class overlap problem in prompting data and ClusBUS
in Chapter 4.

Our second goal is to automatically detect activity errors in real time, while an individual performs an activity. We propose a collection of one-class classification-based algorithms, known as DERT, that learns only from the normal activity patterns and without using any training examples for the activity errors. When evaluated on unseen activity data, DERT is able to identify abnormalities or errors, which can be potential prompt situations. We validate our approaches on smart home sensor data obtained from 580 older adult participants, some of whom faced difficulties performing routine activities and thus committed errors. One-class classification-based real-time activity error detection is discussed in Chapter 5.

The proposed solutions are summarized visually in Figure 1.2.

![Figure 1.2: Summary of proposed solutions.](image)
1.5 Contribution

The contributions of this dissertation are listed below:

(i) We propose the foundation of an automated prompting system that relies on smart home sensors. Residents’ privacy is taken into account by avoiding video or audio input.

(ii) While most of the existing prompting systems work at the level of activities, our approach works on fine-grained sub-activity level information.

(iii) We propose two novel probabilistic oversampling techniques, RACOG and wRACOG, to handle imbalanced class distribution.

(iv) A novel clustering-based under-sampling technique, ClusBUS, is proposed to address overlapping classes problem in the presence of class imbalance.

(v) The proposed one-class classification-based activity error detection algorithm, DERT, automatically detects activity errors in real time, while an individual performs an activity, using no training data for the activity errors.
Before discussing the details of our proposed algorithms, in the next chapter, we describe our smart home sensor infrastructure and the studies conducted with older adults in our on-campus smart home. The sensor data for activities of daily living, gathered from these studies, are used in our experiments.
CHAPTER 2. THE CASAS SMART HOME

INFRASTRUCTURE

The CASAS smart home infrastructure is a combination of software and hardware components/tool that work together to provide pervasive computing solutions for health monitoring and intervention needs. The components work on three different layers: physical, middleware, and application. The physical layer contains hardware components including sensors and actuators that use ZigBee wireless mesh network to communicate with the middleware or with each other. The middleware layer is the backbone software architecture that controls the physical layer components using a publish-subscribe paradigm. It passes sensor information to the application layer for various application needs and controls the actuators in the physical layer as per the commands of the applications. The application layer contains the AI and machine learning-based software agents that help in health monitoring and in providing necessary interventions. Activity recognition and activity discovery are some examples of application layer software agents.
2.1 Smart Home Test Bed

The CASAS smart home test bed is used to replicate day to day lives of individuals in their homes. The facility is used for sensor data collection of activities of daily living conducted in a realistic setting. The test bed is an on-campus town house apartment (shown in Figure 2.1), modified to become a smart home. The current sensor system is composed of several different sensor types for motion, ambient light level, temperature, doors, light switches, items, objects, water flow, and power use.

Figure 2.1: On-campus smart home.
A majority of our sensors are wireless and use a Control4 ZigBee wireless mesh network. There are two types of motion detectors, ceiling mounted and wall mounted. The ceiling mounted motion detectors sense directly below them and have their viewing aperture confined in a way that they can only sense approximately a four feet diameter area below them. The wall mounted motion detectors are mounted so that they can look out into an area, such as an entire room, and detect any motion within that space. Integrated into the motion detector enclosure are ambient light level sensors. This can be useful for allowing the home to automatically turn on lights to help prevent the residents tripping at night, or illuminating the workspace when enough natural light is no longer available. Temperature sensors are also useful for determining inhabitant behavior, such as thermal preferences and determining when the stove or oven is in use in the kitchen. Figure 2.2 shows the sensor layout of the smart home test bed.

Every sensor in the smart home logs time stamp, sensor ID and current state, whenever there is a change of its state. For example, when a cabinet door is opened, the door sensor associated with it logs an “OPEN” state into the database. The next log from this sensor appears when the cabinet door is closed. This protocol of “change-of-event”-based sensor logging helps in achieving longer battery lives of the sensors. The sensor logs are stored in a database server in real time. Table 2.1 shows an example of raw sensor data collected in the smart home.
Figure 2.2: Sensor layout of on-campus smart apartment.
<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Sensor ID</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-05-11</td>
<td>14:59:54.934979</td>
<td>D010</td>
<td>CLOSE</td>
</tr>
<tr>
<td>2009-05-11</td>
<td>14:59:55.213769</td>
<td>M017</td>
<td>ON</td>
</tr>
<tr>
<td>2009-05-11</td>
<td>15:00:02.062455</td>
<td>M017</td>
<td>OFF</td>
</tr>
<tr>
<td>2009-05-11</td>
<td>15:00:17.348279</td>
<td>M017</td>
<td>ON</td>
</tr>
<tr>
<td>2009-05-11</td>
<td>15:00:34.006763</td>
<td>M018</td>
<td>ON</td>
</tr>
<tr>
<td>2009-05-11</td>
<td>15:00:35.487639</td>
<td>M051</td>
<td>ON</td>
</tr>
<tr>
<td>2009-05-11</td>
<td>15:00:43.028589</td>
<td>M016</td>
<td>ON</td>
</tr>
<tr>
<td>2009-05-11</td>
<td>15:00:43.091891</td>
<td>M015</td>
<td>ON</td>
</tr>
<tr>
<td>2009-05-11</td>
<td>15:00:45.008148</td>
<td>M014</td>
<td>ON</td>
</tr>
</tbody>
</table>

**Table 2.1:** Raw-sensor data stored in the database.

### 2.2 Smart Home Sensors

In our experiments, we mainly use the data from four kinds of sensors: motion detectors to collect the location information of the participant in the apartment; door sensors to monitor opening and closing of closets, cabinets, microwave oven, refrigerator, and entryway; pressure-based item sensors to monitor use of household objects, such as water cup, cup noodles and medication dispenser; and, vibration
sensors to track the use of household objects such as broom, dustpan and brush. In
the following, we describe these four kinds of sensors.

2.2.1 Passive Infrared Motion Detector

The passive infrared motion detectors detect the presence or absence of motion in
the smart home. Two types of motion sensors, ceiling-mounted downward facing
sensors and wall-mounted area sensors, are used in our smart home. The area sensors
are installed in a room such that their field of view is an area, for example, an entire
room. While these sensors can detect when a resident enters a room, they cannot
track the location of the resident in the room. In contrast, the ceiling-mounted sensors
have the detector facing downwards, such that their field of view is restricted to a four
feet diameter directly below them. This arrangement allows achieving more focused
view of the space. To cover the maximum possible space, multiple ceiling-mounted
motion detectors are placed in the rooms. Thus, when they are triggered, they can
provide information about the resident’s location. These motion detectors send “ON”
and “OFF” status messages to the middleware. Figure 2.3 shows wireless PIR motion
detectors used in our smart home.
2.2.2 Magnetic Door Sensor

A magnetic door sensor is a simple magnet-driven reed switch that is used to detect the opening and closing of doors in bedrooms and bathrooms, kitchen cabinets, closets, microwave oven and refrigerator. Whenever the magnet moves away from the reed switch, it sends an “OPEN” status to the middleware. When the magnet moves back in place, a “CLOSED” event is created. Figure 2.4 shows a magnetic door sensor attached to a kitchen cabinet. This simple system installed at entrances to rooms and buildings provides a stronger source of evidence for entrances and exits than motion detectors alone.
2.2.3 Item Pressure Sensor

An item sensor is a simple contact switch and plate that detects the presence or absence of important items in the smart home such as, medicine dispenser, cookware and toiletries. When an item is removed from the plate, the switch is depressed and sends an “ABSENT” event to the middleware. Similarly, it sends a “PRESENT” event when the object is placed back on the plate. Figure 2.5 shows an item sensor used in our smart home.
2.2.4 Vibration Sensor

Vibration object sensors are based on accelerometers that are capable of tracking when an object is moved. They are attached to household objects that residents need to carry around in order to use them, such as, broom, dustpan, brush, and water pitcher. We use the basic accelerometer-based movement detector manufactured by Shimmer\textsuperscript{TM} [Burns et al., 2010] and reprogram them to detect object interaction by setting a threshold on the tri-axial accelerometer values. Whenever an object is moved, the object sensor sends out a “MOVED” event to the middleware. When the object is in resting condition, a “STILL” event is sent out. Figure 2.6 shows the objects sensors used in our studies.
2.3 CASAS Lightweight Middleware (CLM)

The CASAS Lightweight Middleware [Kusznir, 2010] or CLM is a lightweight, flexible, fast and easily extensible middleware for smart homes. CLM uses a publish-subscribe paradigm with the power and simplicity of eXtensible Messaging and Presence Protocol or XMPP [Saint-Andre, 2011]. In this architecture, a smart home is broken down into modules, called agents and a middleware hub. The agents may
be publishers, subscribers or both. The middleware allows low coupling amongst the agents by ensuring independence of each agent from the knowledge of other agents, and high cohesion by having each agent associated to a specific task with a standardized communication protocol amongst them.

The CASAS smart home infrastructure consists of several data sources (e.g. sensors) and several applications as data sink (e.g. activity recognition algorithms, prompting systems, and graphical visualizers.). This makes the infrastructure highly suitable for an event based model, where the event consists of a data originator (such as a sensor), a timestamp and a message. For the purpose of simplicity and human readability, XML is used for data formatting. A single clock time stamps all messages. Every device, entity or agent has a globally-unique ID. The location for each entity maps the physical location of the entity to a logical position. Also, the spec includes the message itself. CLM uses the instant messaging protocol XMPP to communicate between the agents.

The CLM implementation consists of a Manager daemon that listens on a known address for XML commands and data messages. It manages the event channels, accepting events from various agents and adding a UNIX timestamp to the event (this is done to prevent issues of clock skew across different computers). It also maintains a list of the subscribers for different channels. When a publisher publishes information to a channel, it simply sends the message to the manager. The manager
then forwards the message to all subscribers on that channel.

The Manager daemon dynamically creates three channels: Raw Events, Log and Entities. However, there is one additional channel Debug that is mainly used for debugging purpose. The agents could be hardware devices, software services and applications. As mentioned earlier, these agents could be publishers, subscribers or both. The following gives a brief description of the different components in Figure 2.7.

**Figure 2.7:** CLM middleware architecture.
2.3.1 Channels

- **Raw Event**: This channel is meant for all kinds of raw events that comes from the hardware devices such as the sensors. The software agents can subscribe to this channel to listen to the recent events published by the sensors. However, there are certain software agents like the Prompting System that publish to this channel.

- **Log**: The log messages of both hardware and software agents are published in this channel. This helps in getting an idea of current state of functionality of the agents and helps in troubleshooting when something goes wrong and the cause is hard to determine.

- **Entities**: An application that can track multiple inhabitants in a smart environment [Crandall, 2011], represented as EntityTracker, mainly uses this channel to publish events that represents the presence of multiple inhabitants. The graphical visualizer PyViz also subscribes to this channel to perform a color coded representation of multiple inhabitants.
2.3.2 Hardware Agents

- **1-Wire**: This is a publish-only agent that manages the primary sensor platform, the OneWire bus. The sensor readings from motion detectors, door/cabinet detectors, temperature sensors, and other basic analog values are handled by this agent.

- **TED**: The Energy Detective (TED) power meter agent publishes to both Raw Events and Log channels. It takes a reading from the meter and puts it out on a serial port once a second.

- **Control4**: The Control4 agent communicates with the Control4 HC300 controller and publishes events to the raw events channel as they occur. The agent can also receive standardized commands for controlling relays and actuators, formatting those to be sent to the HC300, which will in turn send the command through the Control4 ZigBee mesh network to the requested device.

- **Shimmer**: Shimmer is a wireless sensor platform that allows one to capture and communicate wide range of sensed data in real-time. The hardware consists of a tri-axial accelerometer (used for object interaction) and gyroscope (used for gesture monitoring). The shimmer agent publishes accelerometer (as MOVE/STILL) and gyroscope data to the Raw Events channel with a sampling
rate of 5Hz. It also publishes to the Log channel.

2.3.3 Software Agents

- **Scribe**: The Scribe agent subscribes to the system Log and Raw Events channels recording the events to disk before they are securely transferred to the main CASAS database every 15 minutes.

- **EntityTracker**: Multiple inhabitant tracking in a smart environment [Crandall, 2011] is done with the help of EntityTracker that publishes information about where it believes entities to be on the Entities channel. PyViz can subscribe to this channel and perform a color coded representation of multiple entities in a smart environment.

- **PromptingSystem**: A context-aware prompting system is used at CASAS that watches the events from Raw Events and waits for an appropriate set of events to trigger a prompt. On detecting a prompt trigger pattern, it publishes the fact to the same channel and issues the prompt. Occasionally two separate agents are used instead of the PromptingSystem agent. A logic agent does the work of determining the prompt triggering pattern and the other agent is used to issue the prompt on a laptop or a smart phone.

- **PyViz**: PyViz [Thomas and Crandall, 2011] is a graphical visualizer application
that represents smart home inhabitant motion in real-time or from an archived raw sensor data file. It uses a Scalable Vector Graphics (SVG) file to represent the floor plan of the smart environment. PyViz subscribes to Raw Events and Log for real-time representation, or to the master database for simulating past inhabitant motion in the environment.

- **LightControl**: This is an automated ambient light adaptation agent that subscribes to Raw Events to detect motion in certain area of the smart space and adjusts light according to the requirements, by publishing events to the hardware Control4 agent which adapts the lighting control.

- **Observer**: This agent is capable of showing the current state of the smart environment by subscribing to the Raw Events channel. It has filters that can be used to display information as per the requirements and interests of the user.

### 2.4 Smart Home in a Box (SHiB)

Smart home technology has found a long lasting impact in applications such as health monitoring and energy-efficient automation. However, most implementations of this technology to date are somewhat narrow and are performed in controlled laboratory settings. These limitations are primarily due to the difficulty of creating a fully functional smart home infrastructure. Even a single smart home implementation
entails a significant investment in equipment, engineering time and maintenance. Therefore, we are designing Smart Home in a Box (SHiB), a smart home kit that has a small form factor, lightweight in infrastructure, extendable with minimal effort, and ready to perform key capabilities out of the box.

Each kit consists of a standard set of parts, plus a few optional ones. It includes 30–40 sensors that are pre-labeled with the intended locations. These sensors are Control4 [Control4] and Card Access [Access] ZigBee-based wireless sensors that have a proven battery life of 6 months to a year and gives reliable performance. The sensor package provides motion/lighting detectors, door open/close, and temperature sensors. Each kit optionally includes a power meter, object sensors and a data collection software. The backbone of the SHiB infrastructure are the computing and communication components, as shown in Figure 2.8. This box contains the power sources, networking components and a barebone computer to handle the computation, data collection and storage for the home. While the SHiB infrastructure is designed to allow each smart home to run independently and locally, smart homes can also securely upload events to be stored in a relational database or in the cloud.

We have made the SHiB kits available for volunteer participants to install in their own homes. The SHiB kits run in the volunteer’s home for at least six months before being shipped back to be re-deployed somewhere else. The net result of CASAS’ SHiB effort is a massive data collection framework. A smart home that runs on SHiB
components, typically generates an average of 5000 sensor events per day. When smart phones and power meter readings are incorporated, the volume increases drastically. The data collected from these smart homes offer unprecedented opportunities for researchers to study human health and behavior. It is also a big leap towards the goal of scaling pervasive computing to larger population groups and settings. SHiB provides a solid platform with significant tools to take the next steps towards making smart home technologies available to a large number of people.

Figure 2.8: Smart Home in a Box kit.
2.5 Studies

To date we have conducted two large scale studies in our on-campus smart home with individuals who are either healthy or have been diagnosed with mild cognitive impairments. The first study (Study 1) [Dawadi et al., 2013, Seelye et al., 2013, Schmitter-Edgecombe et al., 2011] was conducted on 400 participants over a period of three years. Diagnostic information was available for 216 participants, out of which, 53 were younger adults, 88 were cognitively healthy older adults, 55 individuals had mild cognitive impairments, and 18 individuals with possible or probable Alzheimer’s disease. The latter study (Study 2) has been conducted with 180 participants to date. Diagnostic information was available for 150 participants, out of which, 112 were health older adults, 23 individuals had mild cognitive impairments, 1 individual had Alzheimer’s disease, 6 had traumatic brain injury, and 8 individuals has Parkinson’s disease.

In these studies the participants are requested to perform a pre-selected set of activities of daily living (ADLs) as naturally as they would do in their own homes. These ADLs include activities such as Sweeping and Dusting, in which the participants are instructed to retrieve a broom, dustpan/brush, and duster from a specified closet. Then the participants are to sweep the kitchen floor and dust the furnitures in the
living and dining rooms. Table 2.2 lists the activities from the two studies whose data are used in this dissertation. A detailed description of all the activities conducted in these studies is given in Appendix A.

<table>
<thead>
<tr>
<th>Study 1</th>
<th>Study 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sweeping and Dusting</td>
<td>Sweeping and Dusting</td>
</tr>
<tr>
<td>Taking Medication</td>
<td>Cleaning Kitchen Countertops</td>
</tr>
<tr>
<td>Writing Birthday Card</td>
<td>Taking Medication</td>
</tr>
<tr>
<td>Watching DVD</td>
<td>Watering Plants</td>
</tr>
<tr>
<td>Watering Plants</td>
<td>Washing Hands</td>
</tr>
<tr>
<td>Answering Phone Call</td>
<td>Cooking</td>
</tr>
<tr>
<td>Cooking</td>
<td>Selection Outfit</td>
</tr>
</tbody>
</table>

**Table 2.2:** Activities of daily living for two different studies.

As the participants perform these activities, clinically-trained psychologists watch over a web camera. As most of the participants were older adults, some of them committed errors that could lead to improper completion of activities. The psychology experimenters remotely issue a prompt via a tablet or a smart phone.

---

1The data of these studies are available at [http://ailab.wsu.edu/casas/datasets.html](http://ailab.wsu.edu/casas/datasets.html)
when the individual faces difficulty initiating or completing an activity. The experimenters report activity-specific errors and approximate time of occurrence for all the participants. Sensor events, denoted by the event date, time, sensor identifier, and message, are collected continuously and archived in a relational database.

After an experiment is conducted, the sensor data is passed to human annotators who determine the start and end of an activity from the raw sensor events and tag them with appropriate labels. The sensor data are also annotated for activity errors reported by the experimenters. The annotator determines the errors based on the recorded time and/or context of occurrence. Table 2.3 shows a snippet of annotated sensor data. On the basis of these annotations, temporal, spatial and contextual features are extracted to address problems such as activity recognition, activity discovery, and automated prompting.

In the following chapters, we discuss and propose solutions to the machine learning challenges that occur in the sensor data for the automated prompting task. Although our approaches are motivated from pervasive computing, they can be used in other domains that face similar challenges in the data.
<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>SensorID</th>
<th>Message</th>
<th>Activity</th>
<th>Error</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-05-11</td>
<td>14:59:54.934979</td>
<td>D010</td>
<td>CLOSE</td>
<td>cooking</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009-05-11</td>
<td>14:59:55.213769</td>
<td>M017</td>
<td>ON</td>
<td>cooking</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009-05-11</td>
<td>15:00:02.062455</td>
<td>M017</td>
<td>OFF</td>
<td>none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009-05-11</td>
<td>15:00:17.348279</td>
<td>M017</td>
<td>ON</td>
<td>cooking</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009-05-11</td>
<td>15:00:34.006763</td>
<td>M018</td>
<td>ON</td>
<td>cooking</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009-05-11</td>
<td>15:00:38.01839</td>
<td>M051</td>
<td>ON</td>
<td>cooking</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009-05-11</td>
<td>15:00:43.028589</td>
<td>M016</td>
<td>ON</td>
<td>cooking</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009-05-11</td>
<td>15:00:43.091891</td>
<td>M015</td>
<td>ON</td>
<td>cooking</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009-05-11</td>
<td>15:00:45.008148</td>
<td>M014</td>
<td>ON</td>
<td>cooking</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 2.3:** Human annotated sensor events.
3.1 Introduction

We hypothesize that machine learning algorithms, trained on smart home sensor data, can predict when an individual faces difficulty while performing everyday activities. Our first goal is to emulate natural interventions, provided by a caregiver to individuals with memory impairments, by using a supervised machine learning approach to classify pre-segmented activity steps into “prompt” or “no-prompt” classes.

In the smart home studies (discussed in Section 2.5) conducted with older adults, the psychology experimenters issue a prompt when a participant faces difficulty in activity completion. However, in most of the cases, there are very few situations when the participants need prompts, as compared to situations when they do not. For example, in Study 1, there were only 149 out of 3980 cases during which the participants were prompted. As we use the sensor data from this study to build a classification model on training examples that represent prompt and no-prompt activity steps, the distribution of the class labels in the data is highly skewed.

In a classification scenario, imbalanced class distribution occurs in the data
when one or more of the target classes is under-represented in comparison with the other classes. The class(es) that are under-represented are usually known as minority class(es), and the class(es) for which training examples are in abundance are known as majority class(es). According to a widely accepted rule of thumb, any dataset that contains less than $\sim 10\%$ minority class training examples is considered to be imbalanced. Thus, imbalanced class distribution exists in our smart home prompting data as well. In our case, the prompt class, consisting of 3.94% of total training examples, is in minority and the no-prompt class is in majority.

Because the goal of machine learning classifiers is to optimize prediction accuracy for the entire data set, most approaches ignore performance on the individual class labels. Therefore, a random classifier that labels all data samples from an imbalanced class dataset as members of the majority class would become the highest performing algorithm despite incorrectly classifying all minority class samples. However, in addition to automated prompting, many other application domains such as, cancerous cell identification [Woods et al., 1993], oil-spill detection [Kubat et al., 1998], fraud detection [Phua et al., 2004], keyword extraction [Turney, 2000], and text classification [Lewis and Gale, 1994], identifying members of the minority class is critical, sometimes more so than achieving optimal overall accuracy for the majority class.

The imbalanced class distribution problem has vexed researchers for over a
decade and has thus received focused attention. The common techniques that have been investigated for addressing this problem include: resampling, which balances class priors of training data by either increasing the number of minority class data samples (oversampling) or decreasing the number of majority class data samples (undersampling); cost-sensitive learning, which assigns higher misclassification cost for minority class samples than majority class; and kernel-based learning methods, which make the minority class samples more separable from the majority class by mapping the data to a high dimensional feature space.

Among these techniques, resampling methods remain at the forefront due to their ease of implementation. Liu et al. [Liu et al., 2007] articulated number of reasons to prefer resampling to other methods. First, because resampling occurs during preprocessing, the approach can be combined with others such as cost-sensitive learning, without changing the algorithmic anatomy [Zadrozny et al., 2003]. Second, theoretical connections between resampling and cost-sensitive learning indicate that resampling can alter the misclassification costs of data points [Elkan, 2001]. Third, empirical evidence demonstrates nearly identical performance between resampling and cost-sensitive learning techniques [McCarthy et al., 2005, Maloof, 2003]. Although both under- and over-sampling techniques have been improving over the years, we focus our attention on oversampling because it is well suited for the application that motivates our investigation. Because many class imbalance problems
involve an absolute rarity of minority class data samples [Weiss, 2004], undersampling of majority class examples is not advisable.

Most of the oversampling approaches [Chawla et al., 2002, 2003] that add synthetic data to alleviate class imbalance problem, typically rely on spatial location of minority class samples in the Euclidean space. These approaches harvest local information of minority class samples to generate synthetic samples that are also assumed to belong to the minority class. Although this approach may be acceptable for data sets where a crisp decision boundary exists between the classes, spatial location-based synthetic oversampling is not suitable for data sets that have overlap between the minority and majority classes. Therefore, a better idea is to exploit the global information of minority class samples, which can be done by considering the probability distribution of the minority class while synthetically generating new minority class samples.

In this chapter, we utilize the probability distribution of the minority class to introduce two oversampling approaches, namely, RACOG and wRACOG, to generating and strategically selecting new minority class data points. Specifically, the proposed algorithms optimally approximate $n$-dimensional discrete probability distributions of the minority class using Chow-Liu’s dependence tree algorithm, and generate samples from this distribution using Gibbs sampling. Gibbs sampling generates a Markov chain of new minority class samples which undergo a strategic selection procedure.
While RACOG selects samples from the Markov chain generated by the Gibbs sampler using a predefined lag, wRACOG selects those samples that have the highest probability of being misclassified by the existing learning model.

We validate our approach on the prompting dataset\(^2\) and nine publicly available datasets obtained from the UCI machine learning repository which are carefully modified to exhibit class imbalance. The proposed methods, RACOG and wRACOG, are compared with two existing oversampling techniques, SMOTE [Chawla et al., 2002] and SMOTEBoost [Chawla et al., 2003], as well as an undersampling technique, RUSBoost [Seiffert et al., 2010]. To validate the effectiveness of our proposed approaches in comparison with non-classification based prompting prediction techniques, we compare their performance with a baseline prompting algorithm that is only based on the probability distribution of the prompt and no-prompt classes. These approaches are compared on the basis of the following performance measures: Sensitivity, G-mean, and Area Under the ROC Curve (AUC-ROC).

\(^2\)Available at http://ailab.wsu.edu/casas/datasets/promoting.zip
3.2 Related Work

Learning from class imbalanced datasets is a niche, yet critical area in supervised machine learning due to its increased prevalence in real world problem applications. Over the last decade, a number of different review articles [Chawla et al., 2004, He and Garcia, 2009, Sun et al., 2009, Chawla, 2010] have appeared in leading journals supporting the fact that this area continues to be of importance among machine learning researchers.

Due to the pervasive nature of the imbalanced class problem, a wide spectrum of related techniques has been proposed. One of the most common solutions that has been investigated is cost sensitive learning (CSL). CSL methods counter the underlying assumption that all errors are equal by introducing customized costs for misclassifying data points. By assigning a sufficiently high cost to minority sample points, the algorithm may devote sufficient attention to these points to learn an effective class boundary. The effectiveness of CSL methods has been validated theoretically [Elkan, 2001, Maloof, 2003] and empirically [McCarthy et al., 2005, Liu and Zhou, 2006], although other studies indicate that there is no clear winner between CSL and other methods such as resampling [Weiss et al., 2007]. In addition, CSL concepts have been coupled with existing learning methods to boost their performance [Provost et al., 1998, Drummond and Holte, 2000, Kukar and Kononenko, 1998]. However, CSL
approaches have drawbacks that limit their application. First, the misclassification costs are often unknown or need to be painstakingly determined for each application. Second, not all learning algorithms have cost sensitive implementation.

A second direction is to adapt the underlying classification algorithm to consider imbalanced classes, typically using kernel-based learning methods. Since kernel-based methods provide state-of-the-art techniques for many machine learning applications, using them to understand the imbalanced learning problem has attracted increased attention. The kernel classifier construction algorithm proposed by Hong et al. [Hong et al., 2007] is based on orthogonal forward selection and a regularized orthogonal weighted least squares (ROWLSs) estimator. Wu et al. [Wu and Chang, 2005] proposed a kernel-boundary alignment (KBA) algorithm for adjusting the SVM class boundary. KBA is based on the idea of modifying the kernel matrix generated by a kernel function according to the imbalanced data distribution. Another interesting kernel modification technique is the $k$-category proximal support vector machine (PSVM) [Fung and Mangasarian, 2005] proposed by Fung et al. This method transforms the soft-margin maximization paradigm into a simple system of $k$-linear equations for either linear or non-linear classifiers.

Probably the most common approach, however, is to resample, or modify the dataset in a way that balances the class distribution. Determining the ideal class distribution is an open problem [Provost et al., 1998] and in most cases it is handled
empirically. Naive resampling methods include oversampling the minority class by duplicating existing data points and under-sampling the majority class by removing chosen data points. However, random over-sampling and under-sampling increases the possibility of overfitting and discarding useful information from the data, respectively.

An intelligent way of oversampling is to synthetically generate new minority class samples. Synthetic minority class oversampling technique, or SMOTE [Chawla et al., 2002], has shown a great deal of success in various application domains. SMOTE oversamples the minority class by taking each minority class data point and introducing synthetic examples along the line segments joining any or all of the $k$-minority class nearest neighbors. In addition, adaptive synthetic sampling techniques have been introduced that take a more strategic approach to selecting the set of those minority class samples on which synthetic oversampling should be performed. For example, Borderline-SMOTE [Han et al., 2005] generates synthetic samples only for those minority class examples that are “closer” to the decision boundary between the two classes. ADASYN [He et al., 2008], on the other hand, uses the density distribution of the minority class samples as a criterion to automatically decide the number of synthetic samples that need to be generated for each minority example by adaptively changing the weights of different minority examples to compensate for the skewed distribution. Furthermore, class imbalance problems associated with intra-class imbalanced distribution of data in addition to inter-class imbalance can be handled by
cluster-based oversampling (CBO) proposed by Jo et al. [Jo and Japkowicz, 2004].

The information loss incurred by random under-sampling can be overcome by utilizing methods that are designed to strategically remove minority class samples. Liu et al. [Liu et al., 2009] proposed two informed under-sampling techniques that use ensembles to learn from majority class subsets. For data sets that have overlapping minority and majority classes [García et al., 2008], data cleansing is used to minimize unwanted overlapping between classes by removing pairs of minimally distanced nearest neighbors of opposite classes, popularly known as Tomek links [Tomek, 1976, Kubat and Matwin, 1997, Batista et al., 2004]. Because removing Tomek links changes the distribution of the data, it might not be beneficial in situations when there is a dense overlap between the classes, because this approach might lead to overfitting. There are additional ensemble learning approaches [Galar et al., 2012] that combine the powers of resampling and ensemble learning. These approaches include, but are not restricted to, SMOTEBoost [Chawla et al., 2003], RUSBoost [Seiffert et al., 2010], IVotes [Błaszczyński et al., 2010] and SMOTE-Bagging [Wang and Yao, 2009].

Empirical studies [Weiss et al., 2007] have shown that approaches such as cost sensitive learning or kernel-based learning are not quite suitable for class imbalanced data sets that have “rare” minority class samples. Also, any form of under-sampling witnesses the same problem. We are interested in applying supervised learning tech-
niques to a dataset that includes rare minority class samples. While oversampling is a natural solution to this problem, existing oversampling techniques such as SMOTE, Borderline-SMOTE and SMOTEBoost generate new data samples that are spatially close to existing minority class examples in the Euclidean space. Ideally new data samples should be representative of the entire minority class and not just be drawn from local information. Therefore, in this chapter we focus on satisfying the criteria of global representation of the minority class by generating multivariate samples from the joint probability distribution of the underlying random variables or attributes.

Although not very popular, exploiting the probability distribution of the minority class for oversampling has been used in the past. Liu et. al. [Liu et al., 2007] proposed a generative oversampling technique that assumes a probability distribution for the minority class whose parameters are learned from the training samples. Artificial data points are added to the resampled data set by generating points from the learned probability distribution until the desired number of minority class points in the training set has been reached. The application domain under consideration is text classification and therefore the authors assume a multinomial distribution of the minority class. This approach would probably work well when the probability distribution is known or can at least be guessed from the data. However our approach is effective when the user is agnostic of the underlying distribution. Moreover, the authors report experimental results for SVM alone and thus it does not give a clear idea
of whether their approach is effective for other classifiers or not. On the other hand, Gao et al. [Gao et al., 2012] proposed an oversampling approach based on Parzen Windows or kernel density estimation of the minority class data samples. According to the estimated probability density function, synthetic instances are generated as the additional training data. Oversampling is done by treating a randomly-drawn minority class sample as the mean of a Gaussian distribution, and adding the product of a pseudo-random vector, smoothing parameter of the kernel function and Cholesky decomposition of the covariance matrix of the minority class, to the randomly-drawn minority class sample. Like Liu et al.’s approach, this oversampling method is also highly tuned to a specific classifier, Radial Basis Function-based classifier in this case.

Thus, for most of the probabilistic oversampling techniques, we observe that specific classifiers are tuned to perform best with the oversampling approach. In contrast, we propose two purely pre-processing approaches for probabilistic oversampling. With extensive experiments we validate that our approaches work well with four commonly-used classifiers.

3.3 The Prompting Dataset

We use annotated sensor data to generate relevant and distinguishable temporal and spatial features of the activity steps. Thus, each step of an activity is treated
as a separate training sample and pertinent features are defined to describe the step. The annotations are also used to describe if an activity step received a prompt by tagging the data sample corresponding to the step with a “prompt” or “no-prompt” class label. This dataset has 3980 examples with 17 features. Out of the 17 features, 4 are categorical and the rest are numeric. A detailed description of attribute types and measurement units is given in Appendix B. The description of the features is listed in Table 3.1. The difficulty that is faced for the prompting problem is that the majority of activity steps are “no-prompt” cases and standard machine learning algorithms will likely map all data points to this class, which defeats the purpose of the intervention. Our goal is to design solutions to the class imbalance problem that improve sensitivity for this prompting application.

3.4 Probabilistic Approaches for Oversampling

We propose pre-processing techniques that can be used to oversample the minority class in class imbalanced datasets. These techniques are based on the idea of using the probability distribution of the minority class to generate new minority class training samples. This way, to some extent, we can avoid the possibility of the synthetically generated training samples actually belonging to any other class in case of class overlap. In order to sample from the probability distribution of the minor-
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stepLength</td>
<td>Length of activity step in seconds</td>
</tr>
<tr>
<td>numSensors</td>
<td>Number of unique sensors involves with the step</td>
</tr>
<tr>
<td>numEvents</td>
<td>Number of sensor events associated with the step</td>
</tr>
<tr>
<td>prevStep</td>
<td>Previous step ID</td>
</tr>
<tr>
<td>nextStep</td>
<td>Next step ID</td>
</tr>
<tr>
<td>timeActBegin</td>
<td>Time (seconds) elapsed since the beginning of the activity</td>
</tr>
<tr>
<td>timePrevAct</td>
<td>Time (seconds) difference between the last event of the previous step and</td>
</tr>
<tr>
<td></td>
<td>first event of the current step</td>
</tr>
<tr>
<td>stepsActBegin</td>
<td>Number of steps visited since the beginning of the activity</td>
</tr>
<tr>
<td>activityID</td>
<td>Activity ID</td>
</tr>
<tr>
<td>stepID</td>
<td>Current step ID</td>
</tr>
<tr>
<td>location</td>
<td>A set of features representing sensor frequencies in various regions (such</td>
</tr>
<tr>
<td></td>
<td>as kitchen, dining room, living room, etc.) of a smart home when the current</td>
</tr>
<tr>
<td></td>
<td>activity step was performed</td>
</tr>
<tr>
<td>Class</td>
<td>Binary class attribute representing prompt and no-prompt</td>
</tr>
</tbody>
</table>

**Table 3.1:** Description of generated features.
it class, we first need to estimate the distribution and then employ a mechanism to generate samples from that distribution. In the following, we describe how the probability distribution is estimated and new samples are generated.

3.4.1 Approximating Discrete Multivariate Probability Distribution Using a Dependence Tree

A central problem in many application domains, including ours, is to estimate the underlying \( n \)-dimensional probability distribution from a finite number of samples. Limitations on the available samples often require the distribution to be approximated by some simple assumptions, such as mutual independence or normality of the \( n \)-random variables under consideration. Chow-Liu proposed a notion of tree dependence to approximate the probability distribution [Chow and Liu, 1968]. Specifically, the proposed dependence tree approach optimally approximates an \( n \)-dimensional discrete probability distribution by a product of second order distributions, or the distribution of the first-order tree dependence. In other words, the objective is to find an optimal set of \( n - 1 \) first order dependence relationship among the \( n \) variables.

If \( P(x) \) is a joint probability distribution of \( n \) discrete variables \( x_1, x_2, \ldots, x_n \), \( x \) denoting the \( n \) vector \( (x_1, x_2, \ldots, x_n) \), a product approximation in which only the
second order distributions are used can be represented as follows:

\[ P(x) = \prod_{i}^{n} P(x_i|x_{j(i)}), 0 \leq j(i) < 1 \]  

(3.1)

Each variable in Equation 3.1 may be conditioned upon one of the other variables and thus \( P(x_i|x_0) \) is by definition equal to \( P(x_i) \). A probability distribution that can be represented as in Equation 3.1 is called a probability distribution of first-order tree dependence.

Thus, the objective of Chow-Liu’s approximation method is as follows: given an \( n^{th}\)-order probability distribution \( P(x_1, x_2, \ldots, x_n) \), \( x_i \) being discrete, find a distribution of tree dependence \( P_\tau(x_1, x_2, \ldots, x_n) \) such that \( I(P,P_\tau) \leq I(P,P_t) \) for all \( t \in T_n \) where \( T_n \) is the set of all possible first-order dependence trees. The solution \( \tau \) is called the optimal first-order dependence tree. \( I(P,P') \) is a metric to measure the closeness of two probability distributions \( P(x) \) and \( P'(x) \) of \( n \) discrete variables, popularly known as the Kullback-Leibler divergence, and is represented as follows:

\[ I(P,P') = \sum_{x} P(x) log \frac{P(x)}{P'(x)} \]  

(3.2)

As there could be \( n^{n-2} \) trees with \( n \) vertices, there will be an enormous number of dependence trees from which to identify the optimal tree structure. As a solution to this optimization problem, the optimal dependence tree is constructed using mutual information and a maximum-weighted tree. Specifically, the mutual information \( I(x_i, x_j) \) between two variables \( x_i \) and \( x_j \) is assigned as the weight of the \( (x_i \leftarrow x_j) \)
Algorithm 1: Chow-Liu Dependence Tree Construction

1: Take a set of samples from the distribution that is being approximated as input.

2: On the basis of the input samples, compute all $n(n - 1)/2$ pairwise mutual information $I(x_i, x_j), i = 1, 2, 3, \ldots, (n - 1), j = 2, 3, \ldots, n$, and $i < j$. $I(x_i, x_j)$ is the weight of the branch $(x_i \leftarrow x_j)$ in the dependence tree.

3: Use Kruskal’s algorithm to construct a maximum weight dependence tree.

Figure 3.1: Chow-Liu dependence tree construction.

A probability distribution of tree dependence $P_t(x)$ is an optimal approximation to $P(x)$ if and only if its dependence tree $t$ has maximum weight. The mathematical proof behind this hypothesis is avoided due to space constraints. Thus the problem of finding the optimal first-order dependence tree is transformed to that of maximizing the total branch weight of a dependence tree. The algorithm for Chow-Liu’s dependence tree construction is given in Figure 3.1.

In this chapter, we use Chow-Liu’s algorithm to approximate the joint probability distribution of the minority class. This algorithm runs in $O(n^2 \log(n))$ time where...
is the dimension of the data. The dependence tree for the prompting dataset is shown in Figure 3.2.

![Dependence trees for the prompting dataset.](image)

**Figure 3.2:** Dependence trees for the *prompting* dataset.

In order to generate new minority class samples from the approximated probability distribution, we need to use a sampling technique. In the current approach, we use Gibbs sampling.

### 3.4.2 Standard Gibbs Sampling

Gibbs sampling is rooted in image processing and was introduced by Geman and Geman (1984). The family of Markov chain Monte Carlo (MCMC) methods, of which Gibbs sampling is a descendant, originated with the Metropolis algorithm...
The Gibbs sampler generates a sequence of data samples from the joint probability distribution of two or more random variables that forms a Markov chain. A Markov chain, as the name suggests, is a collection of random variables with the property that, given the present, the future is conditionally independent of the past. A first-order Markov chain is defined as a series of random variables $x^{(1)}, ..., x^{(M)}$, such that the conditional independence property given in Equation 3.4 holds true for $m \in \{1, ..., M - 1\}$.

$$P(x^{(m+1)}|x^{(1)}, ..., x^{(m)}) = P(x^{(m+1)}|x^{(m)})$$  \hfill (3.4)

The equation can be represented as a directed graph that forms a chain as shown in Figure 3.3. A Markov chain is thus specified by the probability distribution of the initial variable $P(x^{(0)})$ and the conditional probabilities of the subsequent variables (also known as transition probabilities).

The marginal probability of the variable of interest can be expressed in terms of the marginal probability of the previous variable and the transition probability from the previous variable to the current variable (see Equation 3.5).
\[ P(x^{(m+1)}) = \sum_{x^{(m)}} P(x^{(m+1)}|x^{(m)}) P(x^{(m)}) \] (3.5)

The goal of a Gibbs sampler is to create a Markov chain of random variables that converge to a target probability distribution. The approach is applicable in situations where a random variable \( X \) has at least two dimensions \( (x =< x_1, ..., x_n >, n > 1) \).

At each sampling step, the algorithm considers univariate conditional distributions where each of the dimensions but one is assigned a fixed value. Rather than picking the entire collection of attribute values at once, a separate probabilistic choice is made for each of the \( n \) dimensions, where each choice depends on the values of the other \( n - 1 \) dimensions and the previous value of the same dimension. Such conditional distributions are easier to model than the full joint distribution. Figure 3.4 shows the algorithm for the standard Gibbs sampler. The univariate conditional distribution of each dimension represented by \( P \left( X_i | x_1^{(t+1)}, ..., x_{i-1}^{(t+1)}, x_{i+1}^{(t)}, ..., x_n^{(t)} \right) \) is used to choose attribute values that form a new synthetically generated sample.

The conditional distribution in Step 4 of the Gibbs sampling algorithm (Figure 3.4) is essentially sampling from the joint probability distribution given in Equation 3.6.

\[ P \left( X_i | x_1^{(t+1)}, ..., x_{i-1}^{(t+1)}, x_i^{(t)}, x_{i+1}^{(t)}, ..., x_n^{(t)} \right) = \frac{P \left( x_1^{(t+1)}, ..., x_{i-1}^{(t+1)}, x_i^{(t)}, x_{i+1}^{(t)}, ..., x_n^{(t)} \right)}{P \left( x_1^{(t+1)}, ..., x_{i-1}^{(t+1)}, x_{i+1}^{(t)}, ..., x_n^{(t)} \right)} \] (3.6)

As discussed in Section 3.4.1, we approximate the joint probability distribution
Algorithm 2: Gibbs Sampler

1: $X^{(0)} = < x_1^{(0)}, ..., x_n^{(0)} >$

2: for $t = 1$ to $T$

3: for $i = 1$ to $n$

4: $x_i^{(t+1)} \sim P\left(X_i|x_1^{(t+1)}, ..., x_{i-1}^{(t+1)}, x_{i+1}^{(t)}, ..., x_n^{(t)}\right)$

5: end for

6: end for

Figure 3.4: Gibbs sampling.

of the minority class using Chow-Liu’s dependence tree represented in Equation 3.1. To generate a new minority class sample, the value of an attribute $x_i$ represented by $x_i^{(t+1)}$ is determined by randomly sampling from the distribution of the state space (all possible values) of attribute $x_i$.

Implementation of Gibbs samplers are traditionally dependent on two factors. The first factor is the number of sample generation iterations that are needed for the samples to reach a stationary distribution, that is, when the marginal distribution of $X^{(m)}$ is independent of $m$. In order to avoid the estimates being contaminated by values at iterations before that point (referred to as the burn-in), earlier samples are discarded. The second factor is based on the fact that a sample generated during one iteration is highly dependent on the previous sample. This correlation between
successive values, or *autocorrelation*, is avoided by defining a suitable *lag*, or number of consecutive samples to be discarded from the Markov chain following each accepted generated sample.

### 3.4.3 The RACOG Algorithm

We first describe the RACOG algorithm and eventually show what improvements have been incorporated in wRACOG. The RApidy COnterging Gibbs sampler (RACOG) uses Gibbs sampling to generate new minority class samples from the probability distribution of the minority class approximated using Chow-Liu algorithm. RACOG enhances standard Gibbs sampling by offering an alternative mechanism for choosing the initial values of the random variable $X$ (denoted by $X^{(0)}$).

Conventionally, the initial values of the random variable to “ignite” the Gibbs sampler are randomly chosen from the state space of the attributes. On the other hand, in RACOG, we choose the minority class data points as the set of initial samples and run the Gibbs sampler for every minority class sample. The total number of iterations for the Gibbs sampler is restricted by the desired *minority:majority* class distribution. Thus, RACOG produces multiple Markov chains, each starting with a different minority class sample, instead of one very long chain as done in conventional Gibbs sampling. As the initial samples of RACOG are chosen directly from the mi-
nority class samples, it helps in achieving faster convergence of the generated samples with the minority class distribution.

There are arguments in the literature about the pros and cons of a single long Markov chain versus multiple shorter chain approaches. Geyer [Geyer, 1992] argues that a single long chain is a better approach because if long burn-in periods are required or if the chains have high autocorrelations, using a number of shorter chains may result in chains that are too short to adequately represent the minority class. However, a single long chain requires a very large number of iterations to converge with the target distribution. Our experiments show that the argument made by Geyer does not hold when multiple chains are generated using the minority class samples as the initial samples of the Gibbs sampler.

Figure 3.5 summarizes the RACOG algorithm that runs in $O(n^2\log(n) + D.T.n)$ time where $n =$ dimension of data, $D =$ cardinality of minority class, and $T =$ predetermined number of iterations.

While the RACOG algorithm enhances the traditional Gibbs sampler by making it suitable for class imbalanced data, this approach does not take into account the usefulness of the generated samples. As a result, RACOG might add minority class samples that are redundant and have no contribution towards constructing a better hypothesis. In order to address this issue, we propose wRACOG, an enhancement to the RACOG algorithm.
Algorithm 3: RACOG

1: function RACOG \((\text{minority}, D, n, \beta, \alpha, T)\)

Input: \(\text{minority} = \) minority class data points; \(D = \) size of \(\text{minimum}\); \(n = \) minority dimensions; \(\beta = \) burn-in period; \(\alpha = \) lag; \(T = \) total number of iterations

Output: \(\text{new} \_ \text{samples} = \) new minority class samples

2: Construct Dependence tree \(DT\) using Chow-Liu algorithm.

3: for \(d = 1\) to \(D\) do

4: \(X^{(0)} = \text{minority}(d)\)

5: for \(t = 1\) to \(T\) do

6: for \(i = 1\) to \(n\) do

7: Simplify \(P\left( X_i \parallel x_1^{(t+1)}, \ldots, x_{i-1}^{(t+1)}, x_{i+1}^{(t)}, \ldots, x_n^{(t)} \right)\)

using \(DT\)

8: \(x_i^{(t+1)} \sim P(S_i)\) where \(S_i\) is the state space

of attribute \(x_i\)

9: if \(t > \beta\ \text{AND} \ t \mod (\alpha) = 0\)

10: \(\text{new} \_ \text{samples} = \text{new} \_ \text{samples} + X^{(t)}\)

11: return \(\text{new} \_ \text{samples}\)

Figure 3.5: RACOG algorithm.
3.4.4 The wRACOG algorithm

The enhanced RACOG algorithm, named wRACOG, is a wrapper-based technique over RACOG utilizing Gibbs sampling as the core data sampler. The purpose of introducing wRACOG is to get rid of burn-in, lag and predefined number of iterations associated with sample selection in Gibbs sampling. By performing iterative training on the dataset with newly generated samples, wRACOG selects those samples from the Markov chain that have the highest probability of being misclassified by the learning model generated from the previous version of the dataset.

While the RACOG algorithm generates minority class samples for a fixed (pre-defined) number of iterations, the wRACOG algorithm keeps on fine tuning its hypothesis at every iteration by adding new samples until there is no further improvement with respect to a chosen performance measure. As our goal is to improve the performance of classifiers on the minority class, wRACOG keeps on adding new samples until there is no further improvement in sensitivity (true positive rate) of the wrapper classifier (the core classifier that retrains at every iteration) of wRACOG. This process acts as the “stopping criterion” for the wRACOG algorithm. Please note that sensitivity is not an invariant choice for the stopping criteria. Other performance measures, such as precision and F-measure, can also be used and the choice is entirely application dependent.
At each iteration of wRACOG, new minority class samples are generated by the Gibbs sampler. The model learned by the wrapper classifier on the enhanced set of samples produced in the previous iteration is used to make predictions on the newly generated set of samples. During classification, the class labels of the new samples are assumed to be the same as the minority class because the Gibbs sampler produces new samples from the probability distribution of the minority class. Those samples that are misclassified by the model are added to the existing set of data samples and a new model is trained using the wrapper classifier. At each iteration, the trained model performs prediction on a held out validation set and the sensitivity of the model is recorded. Generation of new samples stops once the standard deviation of sensitivities over the past iterations falls below a threshold. As the wRACOG algorithm might end up running many iterations, the standard deviation of sensitivities is calculated over a fixed number of the most recent iterations (slide_win). We use the values slide_win=10 and threshold=0.02 for our experiments, determined by performing an empirical study on the datasets described in this paper.

The wRACOG algorithm is summarized in Figure 3.6. wRACOG runs in $O(n^2 \log(n) + B.D.n + B.W)$ time where, $n$ = dimension of data, $B$ = number of sample batches generated before stopping criteria is satisfied, $D$ = cardinality of minority class, and $W$ = time complexity of wrapper classifier.
Algorithm 4: wRACOG

1: function wRACOG (train, validation, wrapper, slide_win, threshold, slide_win)

    Input: train = training dataset enhanced at each iteration with new samples; validation = validation set on which trained model is tested at every iteration to track improvements; wrapper = classifier that is retrained on the enhanced dataset at every iteration; slide_win = sensitivities of previous iterations; threshold = threshold of standard deviation of sensitivities over slide_win

    Output: new_train = final hypothesis encoded in the oversampled training set

2: Build model by training wrapper on train

3: Run Gibbs sampler on all minority class samples simultaneously

4: do

5: Perform prediction on newly generated samples

   using model

6: Add misclassified samples to form new_train

7: Train model on new_train using wrapper

8: Perform prediction on validation set using trained

   model and add sensitivity to slide_win

9: while (σ(slide_win) ≥ threshold)

10: return new_train

Figure 3.6: wRACOG algorithm.
Although wRACOG is similar to existing boosting techniques (such as AdaBoost) in the way misclassified samples are assigned higher weights to ensure their selection during random sampling in the next boosting iteration, there are a number of major differences. Firstly, while in traditional boosting, both training and prediction are performed on the same set of data samples, wRACOG trains the current hypothesis on the data samples from the previous iteration and performs prediction only on the newly generated samples from the Gibbs sampler. Secondly, there is no concept of updating weights of the samples before resampling, as the newly generated samples are directly added to the existing set of samples. Thirdly, wRACOG does not use weighted voting of multiple hypotheses learned at every iteration. Instead, it employs multiple iterations to fine tune a single hypothesis. We hypothesize that by applying this approach we can reduce the generation of redundant samples to converge more closely to the true distribution of the minority class, and also reduce the overhead of generating multiple hypotheses as is employed by traditional boosting techniques.

### 3.5 Experimental Setup

We hypothesize that approximating the joint probability distribution of the minority class and using Gibbs sampling to synthetically generate minority class
samples will yield improved results over existing resampling methods for problems that exhibit class imbalance. We compare the performance of the classifiers on the datasets (summarized in Table 3.2) preprocessed by wRACOG, RACOG, and three well-known sampling techniques (SMOTE, SMOTEBoost, RUSBoost), against the datasets with no preprocessing (henceforth named Baseline). In the following, we provide brief description of three well-known sampling techniques that have been used in our experiments.

3.5.1 Alternative Sampling Approaches

SMOTE

SMOTE [Chawla et al., 2002] is an oversampling approach in which the minority class is oversampled by creating “synthetic” examples based on spatial location of the data points in the Euclidean space. Oversampling is performed by considering each minority class data point and introducing synthetic examples along the line segments joining any or all of the $k$-minority class nearest neighbors. The $k$-nearest neighbors are randomly chosen depending upon the amount of oversampling required.

Synthetic data points are generated in the following way. First, the difference between the data point under consideration and its nearest neighbor is computed. This difference is multiplied by a random number between 0 and 1, and it is added to the data point under consideration. This results in the selection of a random point
in the Euclidean space, along the line segment between two specific data points. Consequently, by adding diversity to the minority class, this approach forces the decision boundary between the two regions to be crisper. However, as SMOTE does not rely on the probability distribution of the minority class as a whole, there is no guarantee that the generated samples will belong to the minority class, particularly, when the samples from the majority and minority classes overlap [García et al., 2008].

In our experiments, we have used the SMOTE implementation available with the Weka API [Hall et al., 2009]. Although there is no ideal class distribution for effective classification of examples from all classes, we use SMOTE to oversample the minority class and attain a 50:50 class distribution, which is considered near optimal [Weiss and Provost, 2003].

**SMOTEBoost**

By using a combination of SMOTE and a standard boosting procedure, SMOTEBoost [Chawla et al., 2003] models the minority class by providing the learner with misclassified minority class samples from the previous boosting iteration and a broader representation of those samples achieved by SMOTE. The inherent skewness in the updated distribution is rectified by introducing SMOTE to increase the number of minority class samples according to the new distribution. SMOTEBoost maximizes the margin of the skewed class dataset and increases the diversity among the classifiers in the ensemble by producing a unique set of synthetic samples at every iteration.
No implementation of SMOTEBoost is currently available. As a result, we implemented the algorithm in MATLAB and it is available for public use at the MATLAB CENTRAL File Exchange website\(^3\). This implementation considers 10 boosting iterations as the experiments performed by Seiffert et al. [Seiffert et al., 2010] suggest that there is no significant improvement between 10 and 50 iterations of AdaBoost. However, if the *error rate* or *pseudo loss* of the learning hypothesis in the boosting procedure exceeds 0.5 for more than a specified number of consecutive iterations, the boosting iteration is terminated and rolled back to the state before *pseudo loss* $> 0.5$ was encountered.

Although iterative learning of the weak learner by the boosting procedure attempts to form a stronger hypothesis which better classifies minority class data points, the quality of the generated samples is still dependent on the spatial location of minority class samples in the Euclidean space, as is done by SMOTE. Moreover, SMOTEBoost executes SMOTE ten times (for ten boosting iterations), generating $10 \times (\# \text{majority class samples} - \# \text{minority class samples})$ samples and thus making it highly computationally expensive.

\(^3\)http://www.mathworks.com/matlabcentral/fileexchange/37311
RUSBoost

RUSBoost [Seiffert et al., 2010] is very similar to SMOTEBoost, but claims to achieve better classification performance on the minority class data points by random under-sampling (RUS) of majority class examples. Although this method results in a simpler algorithm with a faster model training time, it is not able to achieve favorable performance (explained later in Section 3.6) as claimed by Seiffert et al., especially when the datasets have an absolute rarity of minority class examples. RUSBoost is used as an example of an under-sampling technique for comparing the performance of under-sampling approaches with the oversampling techniques, which is the primary focus of this work.

We also implemented RUSBoost in MATLAB and it is available for public use at the MATLAB CENTRAL File Exchange website\(^4\). The number of boosting iterations and types of weak learners used for the boosting procedure are same as that of the SMOTEBoost implementation. However, as most of the data sets under consideration have an absolute rarity of minority class examples, the class imbalance ratio has been set to 35:65 (minority:majority). The choice of class distribution is based on the empirical investigations performed by Khoshgoftaar et al. [Khoshgoftaar et al., 2007] on datasets with rare minority class samples. Khoshgoftaar et al. empirically verified

that a 35:65 class distribution would result in better classification performance than a 50:50 class distribution when examples from one class are extremely rare, but the examples of the other class(es) are plentiful.

The proposed approaches, RACOG and wRACOG, are compared with the aforementioned alternative sampling techniques. RACOG oversamples the minority class to achieve a 50:50 class distribution. Therefore, the total number of iterations is fixed and is determined on the basis of $(\#majority\ class\ samples - \#minority\ class\ samples)$, burn-in and lag. A burn-in period of 100 and a lag of 20 iterations is chosen as the convention in the literature [Raftery and Lewis, 1992] to avoid autocorrelation among the samples generated by the Gibbs sampler. As mentioned earlier, wRACOG’s sample selection strategy continues to add samples to the minority class until the standard deviation of sensitivity over the 10 recent iterations fall below an empirically determined threshold. The classifiers presented in Section 3.5.3 are used as wrapper classifiers and are trained on an enhanced dataset at every iteration of wRACOG.

### 3.5.2 Other Datasets

In order to evaluate the generalizability of RACOG and wRACOG on the datasets from other application domains, we choose nine additional real-world datasets
obtained from the UCI repository that exhibit the class imbalance problem: abalone, car, nursery, letter, connect-4, ecoli1, haberman, yeast4 and yeast6. Characteristics of these datasets are summarized in Table 3.2. The real-valued attributes are transformed into discrete sets using equal-width binning. The multi-class datasets are converted into binary class by choosing a particular class label as minority class and the rest of the class labels together as majority class.

3.5.3 Classifiers Used for Performance Evaluation

We choose four most common classifiers in machine learning to evaluate the performance of the proposed methods and other sampling approaches. The Weka implementations of these classifiers found with the Weka Java API are integrated into our implementations in MATLAB. The parameter values that we use for the classifiers in these experiments are listed in Table 3.3. All the experiments are performed using 5-fold cross validation.

3.5.4 Performance Measures

Performance measures are critically important to assess the effectiveness of classifiers. By convention, in a binary classifier, the minority and majority classes are referred to positive and negative classes, respectively. Traditionally, the most fre-
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th>Dim</th>
<th>% Min. Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prompting</td>
<td>3,980</td>
<td>17</td>
<td>3.7437</td>
<td>Description in Section 3.3.</td>
</tr>
<tr>
<td>abalone</td>
<td>4,177</td>
<td>8</td>
<td>6.2006</td>
<td>Predicting age of large sea snails, abalone, from its physical measurements.</td>
</tr>
<tr>
<td>car</td>
<td>1,728</td>
<td>6</td>
<td>3.9931</td>
<td>Evaluating car acceptability based on price, technology and comfort.</td>
</tr>
<tr>
<td>nursery</td>
<td>12,960</td>
<td>8</td>
<td>2.5463</td>
<td>Nursery school application evaluation based on financial standing, parents’ education and social health.</td>
</tr>
<tr>
<td>letter</td>
<td>20,000</td>
<td>16</td>
<td>3.7902</td>
<td>Classifying English alphabets using image frame features.</td>
</tr>
<tr>
<td>connect-4</td>
<td>5000</td>
<td>42</td>
<td>10.00</td>
<td>Predicting first player’s outcome in connect-4 game given 8-ply positions information.</td>
</tr>
<tr>
<td>ecoli1</td>
<td>336</td>
<td>7</td>
<td>22.92</td>
<td>Classification of E.coli from other bacteria.</td>
</tr>
<tr>
<td>haberman</td>
<td>306</td>
<td>3</td>
<td>26.46</td>
<td>Survival of patients who had undergone surgery for breast cancer.</td>
</tr>
<tr>
<td>yeast4</td>
<td>1,484</td>
<td>8</td>
<td>3.44</td>
<td>Modified yeast dataset with ME2 as minority class and combination of other classes a majority class.</td>
</tr>
<tr>
<td>yeast6</td>
<td>1,484</td>
<td>8</td>
<td>2.36</td>
<td>Modified yeast dataset with EXC as minority class and combination of other classes a majority class.</td>
</tr>
</tbody>
</table>

Table 3.2: Description of selected datasets.

...sequently used performance measures are Accuracy and Error Rate. Following this
### Table 3.3: Classifiers and parameter values.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Parameter Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>C4.5 Decision Tree</td>
<td>Confidence factor = 2,</td>
</tr>
<tr>
<td></td>
<td>Minimum # instances per leaf = 2</td>
</tr>
<tr>
<td>SVM</td>
<td>Kernel = RBF, RBF kernel $\gamma = 0.01$</td>
</tr>
<tr>
<td>$k$-Nearest Neighbor</td>
<td>$k = 5$, Distance measure = Euclidean</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>Log likelihood ridge value = $1 \times 10^{-8}$</td>
</tr>
</tbody>
</table>

convention, **Accuracy** and **Error Rate** can be defined as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}; \quad \text{Error Rate} = 1 - \text{Accuracy} \quad (3.7)$$

These conventional performance measures give a general idea about the classifier’s performance on a given data set. However, **Accuracy** and **Error Rate** are ineffective for evaluating classifier performance in a class imbalanced dataset as they consider different types of classification errors as equally important. For example, in an imbalanced dataset with 5% minority class, a random prediction of all the test instances being negative will give an accuracy of 95%, although in this case the classifier could not correctly predict any of the minority class samples. Therefore, in order to provide comprehensive assessment of the imbalanced class learning problem, we need to consider metrics that can report the performance of the classifier on two classes
separately. In the following, performance metrics that are capable of measuring the effectiveness of the classifiers in the presence of class imbalance are described.

**Sensitivity**: Sensitivity is a measure of completeness, i.e., the fraction of the positive class examples that were predicted correctly. It is not sensitive to data distribution and does not give any insight on the number of examples that have been incorrectly predicted as positive. Sensitivity is represented as:

\[
\text{Sensitivity} = \frac{TP}{TP + FN}
\]  

**G-mean**: G-mean [Chan et al., 1999] evaluates the degree of inductive bias in terms of a ratio of positive accuracy (i.e., Sensitivity) and negative accuracy (i.e., Specificity). It is given by:

\[
G \text{- mean} = \sqrt{\frac{TP}{TP + FN} \times \frac{TN}{TN + FP}} = \sqrt{\text{Sensitivity} \times \text{Specificity}}
\]  

**ROC Curves and AUC-ROC** The graphical representation of ROC assessment techniques [Fawcett, 2004, 2006] helps in visualizing the benefits (true positive rate (TPR)) and costs (false positive rate (FPR) of the classifier, defined as:

\[
TPR = \frac{TP}{TP + FN}; \quad FPR = \frac{FP}{FP + TN}
\]  

On the other hand, the area under the ROC curve (AUC-ROC) [Huang and Ling, 2005] gives a single measure of a classifier’s performance for evaluating which model is better on average.
3.6 Results and Discussion

The primary limitation of classifiers that model imbalanced class datasets is in achieving desirable prediction accuracy for minority class instances. That is, the sensitivity is typically low assuming that the minority class is represented as the positive class. The proposed approaches place emphasis on boosting the sensitivity of the classifiers while maintaining a strong prediction performance for both of the classes, which is measured by G-mean. However, we understand that the choice of performance measure that needs to be boosted when dealing with class imbalanced dataset is tightly coupled with the application domain. Moreover, finding a trade-off between improving classifier performance on the minority class in isolation and on the overall dataset should ideally be left at the discretion of the domain expert.

3.6.1 Comparative Assessment

We compare the sensitivity of the C4.5 decision tree on all six approaches in Figure 3.7. Performance of other classifiers are represented in Tables 3.5, 3.6 and 3.7. From the figure it is quite evident that both RACOG and wRACOG perform better than the other methods. Additionally, there is not much performance variability for RACOG and wRACOG over the 5-fold cross validation, except for the connect-4 and haberman datasets.
RUSBoost fails by performing nowhere close to the oversampling techniques. The poor performance of RUSBoost can be attributed to the rarity of minority class samples in the datasets under consideration. When the minority class samples are inherently rare, random under-sampling of the majority class to achieve a 35:65 class distribution, as done by RUSBoost at every iteration, makes the majority class samples rare as well. The poor performance of RUSBoost shows that it adds minor to no value over Baseline.

Both SMOTE and SMOTEBost are good contenders, although there is no clear winner between them. This might be because of our implementation decision to keep the number of boosting iterations of SMOTEBost to be constant at 10. Seiffert et. al. suggest [Seiffert et al., 2010] that there is no significant improvement between 10 and 50 boosting iterations. We might get a better picture if the performance of SMOTEBost is compared against variable number of boosting iterations. This analysis is not included in the current discussion, but can be part of our future work.

From a sensitivity standpoint, wRACOG performs better than RACOG on a majority of the datasets. For the rest of the datasets, the performance of wRACOG is at par with RACOG. We verify the statistical significance of the improvements using Student’s t-test. RACOG and wRACOG exhibit significant ($p < 0.05$) performance improvement over SMOTE and SMOTEBost. These improvements have been reported in **bold** in Tables 3.5, 3.6 and 3.7.
Because the sampling techniques boost the minority class, there is always a tendency for the false positive rate to increase. This trend is observed for most of the sampling techniques. However, the increase in false positive rate is not very significant and therefore it does not affect the G-mean scores. Figure 3.8 reports the G-mean scores of C4.5 decision tree on all the methods when tested with the ten datasets. Clearly, RACOG and wRACOG result in a superior performance over SMOTE and SMOTEBoost. An interesting thing to note is that the G-means of the decision tree due to wRACOG is 0 for the haberman dataset, implying that the false positive rate is increased to 1 when haberman is preprocessed with wRACOG. This is
likely due to the fact that *haberman* has only three attributes, which prevents correct approximation of the joint probability distribution.

In Figure 3.9, we plot the ROC curves produced by the different approaches on each of the 10 datasets when evaluated with a C4.5 decision tree. For the *prompting*, *abalone*, *car*, *yeast4* and *yeast6* datasets, Baseline and RUSBoost do not perform any better than random prediction. The performance of RACOG and wRACOG are clearly better than SMOTE and SMOTEBoost for the *prompting*, *yeast4* and *yeast6* datasets. However, there is no clear winner among other datasets. The AUC-ROCs for the C4.5 decision tree are reported in Table 3.4. For the *letter* and *connect-4*
datasets, the AUC-ROC for SMOTEBoost is higher than RACOG and wRACOG. However, no statistically significant improvement of RACOG and wRACOG was found over SMOTEBoost based on AUC-ROC. As there is no clear winner between RACOG and wRACOG on any of the performance measures, we do not conduct any statistical significance test between them.
Datasets | Baseline | SMOTE | SMOTEBoost | RUSBoost | RACOG | wRACOG  
--- | --- | --- | --- | --- | --- | ---  
prompting | 0.5000 ± 0.00 | 0.8511 ± 0.05 | 0.8676 ± 0.02 | 0.5000 ± 0.00 | 0.8707 ± 0.04 | 0.8680 ± 0.03  
abalone | 0.9282 ± 0.05 | 0.9068 ± 0.05 | 0.9230 ± 0.03 | 0.8802 ± 0.05 | 0.9057 ± 0.06 | 0.8610 ± 0.03  
car | 0.5000 ± 0.00 | 0.9526 ± 0.06 | 0.9944 ± 0.01 | 0.5000 ± 0.00 | 0.9884 ± 0.01 | 0.9885 ± 0.01  
nursery | 0.9908 ± 0.01 | 0.9773 ± 0.02 | 0.9999 ± 0.00 | 0.8070 ± 0.04 | 0.9976 ± 0.00 | 0.9967 ± 0.00  
letter | 0.8894 ± 0.03 | 0.9535 ± 0.01 | 0.9883 ± 0.01 | 0.8179 ± 0.03 | 0.9714 ± 0.00 | 0.9708 ± 0.01  
connect-4 | 0.6651 ± 0.04 | 0.6836 ± 0.03 | 0.8196 ± 0.03 | 0.5801 ± 0.02 | 0.7544 ± 0.02 | 0.7104 ± 0.03  
ecoli1 | 0.9337 ± 0.03 | 0.9318 ± 0.03 | 0.9384 ± 0.02 | 0.8677 ± 0.03 | 0.8983 ± 0.05 | 0.8798 ± 0.03  
haberman | 0.5211 ± 0.04 | 0.5330 ± 0.10 | 0.5994 ± 0.06 | 0.5402 ± 0.03 | 0.5641 ± 0.09 | 0.5000 ± 0.00  
yeast4 | 0.5000 ± 0.00 | 0.7309 ± 0.11 | 0.7962 ± 0.06 | 0.5000 ± 0.00 | 0.7598 ± 0.18 | 0.7636 ± 0.13  
yeast6 | 0.5000 ± 0.00 | 0.8124 ± 0.11 | 0.8932 ± 0.06 | 0.5000 ± 0.00 | 0.8519 ± 0.08 | 0.8741 ± 0.08  

Table 3.4: AUC-ROC for C4.5 decision tree.

3.6.2 Generated Sample Size

The number of samples added to the baseline datasets by the different oversampling algorithms for achieving the reported performance is also an important parameter to analyze. Ideally, we would want to obtain a high performance by adding the least number of samples possible. Figure 3.10 illustrates the number of samples that are added by the different approaches. As the number of samples added by SMOTEBoost is far higher than other methods, we present \( \log_{10} \) values of the \textit{number of added samples} so that the comparison could be better represented in the plot. SMOTE and RACOG try to achieve a 50:50 class distribution and thus add samples accordingly.
### Table 3.5: Results for SVM.

#### Sensitivity

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Baseline</th>
<th>SMOTE</th>
<th>SMOTEBoost</th>
<th>RUSBoost</th>
<th>RACOG</th>
<th>wRACOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>prompting</td>
<td>0.0268 ± 0.00</td>
<td>0.5971 ± 0.10</td>
<td>0.5196 ± 0.17</td>
<td>0.0268 ± 0.00</td>
<td>0.6930 ± 0.08</td>
<td>0.6826 ± 0.08</td>
</tr>
<tr>
<td>abalone</td>
<td>0.0000 ± 0.00</td>
<td>0.9482 ± 0.06</td>
<td>0.7812 ± 0.13</td>
<td>0.00 ± 0.00</td>
<td>0.9231 ± 0.07</td>
<td>0.9773 ± 0.06</td>
</tr>
<tr>
<td>car</td>
<td>0.0000 ± 0.00</td>
<td>1.0000 ± 0.00</td>
<td>0.9258 ± 0.13</td>
<td>0.0000 ± 0.00</td>
<td>1.0000 ± 0.00</td>
<td>1.0000 ± 0.00</td>
</tr>
<tr>
<td>nursery</td>
<td>0.0000 ± 0.00</td>
<td>1.0000 ± 0.00</td>
<td>0.9818 ± 0.03</td>
<td>0.0000 ± 0.00</td>
<td>1.0000 ± 0.00</td>
<td>1.0000 ± 0.00</td>
</tr>
<tr>
<td>letter</td>
<td>0.7336 ± 0.06</td>
<td>0.9248 ± 0.04</td>
<td>0.8747 ± 0.02</td>
<td>0.7336 ± 0.06</td>
<td>0.9221 ± 0.01</td>
<td>0.9143 ± 0.01</td>
</tr>
<tr>
<td>connect-4</td>
<td>0.0000 ± 0.00</td>
<td>0.62 ± 0.07</td>
<td>0.562 ± 0.07</td>
<td>0.0000 ± 0.00</td>
<td>0.762 ± 0.04</td>
<td>0.986 ± 0.02</td>
</tr>
<tr>
<td>ecoli1</td>
<td>0.0000 ± 0.00</td>
<td>0.9365 ± 0.06</td>
<td>0.7529 ± 0.13</td>
<td>0.0000 ± 0.00</td>
<td>0.9098 ± 0.06</td>
<td>0.9882 ± 0.03</td>
</tr>
<tr>
<td>haberman</td>
<td>0.5211 ± 0.00</td>
<td>0.5330 ± 0.08</td>
<td>0.5994 ± 0.04</td>
<td>0.5402 ± 0.00</td>
<td>0.5641 ± 0.23</td>
<td>0.5003 ± 0.00</td>
</tr>
<tr>
<td>yeast4</td>
<td>0.0000 ± 0.00</td>
<td>0.5873 ± 0.18</td>
<td>0.4909 ± 0.19</td>
<td>0.0000 ± 0.00</td>
<td>0.7655 ± 0.11</td>
<td>0.7655 ± 0.11</td>
</tr>
<tr>
<td>yeast6</td>
<td>0.0000 ± 0.00</td>
<td>0.8000 ± 0.16</td>
<td>0.6000 ± 0.19</td>
<td>0.0000 ± 0.00</td>
<td>0.8571 ± 0.14</td>
<td>0.8571 ± 0.14</td>
</tr>
</tbody>
</table>

#### G-mean

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Baseline</th>
<th>SMOTE</th>
<th>SMOTEBoost</th>
<th>RUSBoost</th>
<th>RACOG</th>
<th>wRACOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>prompting</td>
<td>0.1461 ± 0.00</td>
<td>0.7317 ± 0.05</td>
<td>0.6860 ± 0.14</td>
<td>0.1452 ± 0.00</td>
<td>0.7919 ± 0.05</td>
<td>0.7913 ± 0.05</td>
</tr>
<tr>
<td>abalone</td>
<td>0.0000 ± 0.00</td>
<td>0.8968 ± 0.04</td>
<td>0.8425 ± 0.06</td>
<td>0.0000 ± 0.00</td>
<td>0.8844 ± 0.04</td>
<td>0.8621 ± 0.05</td>
</tr>
<tr>
<td>car</td>
<td>0.0000 ± 0.00</td>
<td>0.9622 ± 0.01</td>
<td>0.9422 ± 0.07</td>
<td>0.0000 ± 0.00</td>
<td>0.9622 ± 0.01</td>
<td>0.9622 ± 0.01</td>
</tr>
<tr>
<td>nursery</td>
<td>0.0000 ± 0.00</td>
<td>0.976 ± 0.00</td>
<td>0.9788 ± 0.02</td>
<td>0.0000 ± 0.00</td>
<td>0.9672 ± 0.00</td>
<td>0.9775 ± 0.00</td>
</tr>
<tr>
<td>letter</td>
<td>0.8553 ± 0.04</td>
<td>0.9424 ± 0.02</td>
<td>0.9241 ± 0.01</td>
<td>0.8553 ± 0.04</td>
<td>0.9364 ± 0.01</td>
<td>0.9394 ± 0.01</td>
</tr>
<tr>
<td>connect-4</td>
<td>0.0000 ± 0.00</td>
<td>0.7372 ± 0.04</td>
<td>0.7063 ± 0.05</td>
<td>0.0000 ± 0.00</td>
<td>0.7877 ± 0.01</td>
<td>0.5850 ± 0.03</td>
</tr>
<tr>
<td>ecoli1</td>
<td>0.0000 ± 0.00</td>
<td>0.8888 ± 0.04</td>
<td>0.8211 ± 0.07</td>
<td>0.0000 ± 0.00</td>
<td>0.8750 ± 0.05</td>
<td>0.8378 ± 0.10</td>
</tr>
<tr>
<td>haberman</td>
<td>0.0000 ± 0.00</td>
<td>0.5032 ± 0.07</td>
<td>0.471 ± 0.03</td>
<td>0.0000 ± 0.00</td>
<td>0.5586 ± 0.11</td>
<td>0.0000 ± 0.00</td>
</tr>
<tr>
<td>yeast4</td>
<td>0.0000 ± 0.00</td>
<td>0.7082 ± 0.13</td>
<td>0.6672 ± 0.12</td>
<td>0.0000 ± 0.0000</td>
<td>0.8082 ± 0.07</td>
<td>0.8106 ± 0.06</td>
</tr>
<tr>
<td>yeast6</td>
<td>0.0000 ± 0.00</td>
<td>0.8721 ± 0.09</td>
<td>0.7568 ± 0.12</td>
<td>0.0000 ± 0.0000</td>
<td>0.8853 ± 0.07</td>
<td>0.8999 ± 0.08</td>
</tr>
</tbody>
</table>

#### AUC-ROC

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Baseline</th>
<th>SMOTE</th>
<th>SMOTEBoost</th>
<th>RUSBoost</th>
<th>RACOG</th>
<th>wRACOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>prompting</td>
<td>0.8651 ± 0.06</td>
<td>0.8585 ± 0.06</td>
<td>0.8323 ± 0.00</td>
<td>0.5126 ± 0.04</td>
<td>0.8896 ± 0.04</td>
<td>0.8857 ± 0.04</td>
</tr>
<tr>
<td>abalone</td>
<td>0.5000 ± 0.00</td>
<td>0.8989 ± 0.04</td>
<td>0.9307 ± 0.05</td>
<td>0.5000 ± 0.00</td>
<td>0.8864 ± 0.04</td>
<td>0.8707 ± 0.05</td>
</tr>
<tr>
<td>car</td>
<td>0.5000 ± 0.00</td>
<td>0.9630 ± 0.01</td>
<td>0.9862 ± 0.01</td>
<td>0.5000 ± 0.00</td>
<td>0.9630 ± 0.01</td>
<td>0.9630 ± 0.01</td>
</tr>
<tr>
<td>nursery</td>
<td>0.5000 ± 0.00</td>
<td>0.9763 ± 0.00</td>
<td>0.9948 ± 0.00</td>
<td>0.5000 ± 0.00</td>
<td>0.9677 ± 0.00</td>
<td>0.9777 ± 0.00</td>
</tr>
<tr>
<td>letter</td>
<td>0.9878 ± 0.03</td>
<td>0.9861 ± 0.01</td>
<td>0.9825 ± 0.01</td>
<td>0.8654 ± 0.03</td>
<td>0.9856 ± 0.00</td>
<td>0.9861 ± 0.01</td>
</tr>
<tr>
<td>connect-4</td>
<td>0.5000 ± 0.00</td>
<td>0.7497 ± 0.03</td>
<td>0.8253 ± 0.03</td>
<td>0.5000 ± 0.00</td>
<td>0.7884 ± 0.01</td>
<td>0.6672 ± 0.02</td>
</tr>
<tr>
<td>ecoli1</td>
<td>0.5000 ± 0.00</td>
<td>0.8913 ± 0.03</td>
<td>0.9153 ± 0.02</td>
<td>0.5000 ± 0.00</td>
<td>0.8759 ± 0.05</td>
<td>0.8534 ± 0.08</td>
</tr>
<tr>
<td>haberman</td>
<td>0.5000 ± 0.00</td>
<td>0.5852 ± 0.04</td>
<td>0.5684 ± 0.10</td>
<td>0.5000 ± 0.00</td>
<td>0.5927 ± 0.06</td>
<td>0.5000 ± 0.00</td>
</tr>
<tr>
<td>yeast4</td>
<td>0.5000 ± 0.00</td>
<td>0.7291 ± 0.10</td>
<td>0.7937 ± 0.07</td>
<td>0.5 ± 0.00</td>
<td>0.8109 ± 0.06</td>
<td>0.8140 ± 0.06</td>
</tr>
<tr>
<td>yeast6</td>
<td>0.5000 ± 0.00</td>
<td>0.8796 ± 0.08</td>
<td>0.9272 ± 0.05</td>
<td>0.5 ± 0.00</td>
<td>0.8885 ± 0.07</td>
<td>0.9037 ± 0.07</td>
</tr>
</tbody>
</table>
## Sensitivity

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Baseline</th>
<th>SMOTE</th>
<th>SMOTEBoost</th>
<th>RUSBoost</th>
<th>RACOG</th>
<th>wRACOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>prompting</td>
<td>0.0268 ± 0.02</td>
<td>0.5971 ± 0.07</td>
<td>0.5256 ± 0.12</td>
<td>0.0268 ± 0.02</td>
<td>0.6930 ± 0.14</td>
<td>0.6826 ± 0.11</td>
</tr>
<tr>
<td>abalone</td>
<td>0.7631 ± 0.11</td>
<td>0.9365 ± 0.04</td>
<td>0.8447 ± 0.07</td>
<td>0.7631 ± 0.11</td>
<td>0.9482 ± 0.06</td>
<td>0.9482 ± 0.06</td>
</tr>
<tr>
<td>car</td>
<td>0.0579 ± 0.06</td>
<td>0.7919 ± 0.09</td>
<td>0.8724 ± 0.07</td>
<td>0.0579 ± 0.06</td>
<td>1.00 ± 0.00</td>
<td>1.0000 ± 0.00</td>
</tr>
<tr>
<td>nursery</td>
<td>0.3030 ± 0.02</td>
<td>0.9061 ± 0.02</td>
<td>0.9576 ± 0.02</td>
<td>0.3030 ± 0.02</td>
<td>1.0000 ± 0.00</td>
<td>1.0000 ± 0.00</td>
</tr>
<tr>
<td>letter</td>
<td>0.7336 ± 0.02</td>
<td>0.9248 ± 0.01</td>
<td>0.8747 ± 0.03</td>
<td>0.7336 ± 0.02</td>
<td>0.9221 ± 0.02</td>
<td>0.9143 ± 0.02</td>
</tr>
<tr>
<td>connect-4</td>
<td>0.0360 ± 0.01</td>
<td>0.7220 ± 0.03</td>
<td>0.8480 ± 0.07</td>
<td>0.0360 ± 0.01</td>
<td>0.622 ± 0.05</td>
<td>0.9040 ± 0.04</td>
</tr>
<tr>
<td>ecoli</td>
<td>0.6847 ± 0.12</td>
<td>0.8965 ± 0.06</td>
<td>0.7945 ± 0.14</td>
<td>0.6847 ± 0.12</td>
<td>0.9216 ± 0.06</td>
<td>0.9482 ± 0.06</td>
</tr>
<tr>
<td>haberman</td>
<td>0.0610 ± 0.04</td>
<td>0.4199 ± 0.17</td>
<td>0.2956 ± 0.08</td>
<td>0.0610 ± 0.04</td>
<td>0.3699 ± 0.08</td>
<td>1.0000 ± 0.00</td>
</tr>
<tr>
<td>yeast4</td>
<td>0.0382 ± 0.05</td>
<td>0.5091 ± 0.12</td>
<td>0.5885 ± 0.12</td>
<td>0.0382 ± 0.05</td>
<td>0.7255 ± 0.11</td>
<td>0.7618 ± 0.14</td>
</tr>
<tr>
<td>yeast6</td>
<td>0.0857 ± 0.08</td>
<td>0.8286 ± 0.12</td>
<td>0.6 ± 0.19</td>
<td>0.0857 ± 0.08</td>
<td>0.9143 ± 0.13</td>
<td>0.8571 ± 0.10</td>
</tr>
</tbody>
</table>

## G-mean

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Baseline</th>
<th>SMOTE</th>
<th>SMOTEBoost</th>
<th>RUSBoost</th>
<th>RACOG</th>
<th>wRACOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>prompting</td>
<td>0.1461 ± 0.08</td>
<td>0.7317 ± 0.04</td>
<td>0.6895 ± 0.08</td>
<td>0.1461 ± 0.08</td>
<td>0.7919 ± 0.08</td>
<td>0.7907 ± 0.07</td>
</tr>
<tr>
<td>abalone</td>
<td>0.8389 ± 0.05</td>
<td>0.8957 ± 0.03</td>
<td>0.8592 ± 0.04</td>
<td>0.8389 ± 0.05</td>
<td>0.8955 ± 0.05</td>
<td>0.8536 ± 0.06</td>
</tr>
<tr>
<td>car</td>
<td>0.1824 ± 0.18</td>
<td>0.8237 ± 0.05</td>
<td>0.898 ± 0.04</td>
<td>0.1824 ± 0.18</td>
<td>0.8742 ± 0.01</td>
<td>0.9458 ± 0.01</td>
</tr>
<tr>
<td>nursery</td>
<td>0.5503 ± 0.02</td>
<td>0.9157 ± 0.01</td>
<td>0.9592 ± 0.01</td>
<td>0.5503 ± 0.02</td>
<td>0.9178 ± 0.00</td>
<td>0.9828 ± 0.00</td>
</tr>
<tr>
<td>letter</td>
<td>0.8553 ± 0.01</td>
<td>0.9424 ± 0.01</td>
<td>0.9241 ± 0.02</td>
<td>0.8553 ± 0.01</td>
<td>0.9364 ± 0.01</td>
<td>0.9394 ± 0.01</td>
</tr>
<tr>
<td>connect-4</td>
<td>0.1885 ± 0.02</td>
<td>0.7118 ± 0.01</td>
<td>0.6534 ± 0.04</td>
<td>0.1885 ± 0.02</td>
<td>0.7209 ± 0.03</td>
<td>0.6739 ± 0.03</td>
</tr>
<tr>
<td>ecoli</td>
<td>0.7926 ± 0.07</td>
<td>0.8719 ± 0.03</td>
<td>0.8251 ± 0.08</td>
<td>0.7926 ± 0.07</td>
<td>0.8803 ± 0.05</td>
<td>0.8684 ± 0.05</td>
</tr>
<tr>
<td>haberman</td>
<td>0.2178 ± 0.13</td>
<td>0.5289 ± 0.09</td>
<td>0.4820 ± 0.07</td>
<td>0.2178 ± 0.13</td>
<td>0.4831 ± 0.07</td>
<td>0.1671 ± 0.22</td>
</tr>
<tr>
<td>yeast4</td>
<td>0.1232 ± 0.17</td>
<td>0.6514 ± 0.08</td>
<td>0.7370 ± 0.08</td>
<td>0.1232 ± 0.17</td>
<td>0.7653 ± 0.05</td>
<td>0.807 ± 0.07</td>
</tr>
<tr>
<td>yeast6</td>
<td>0.2265 ± 0.21</td>
<td>0.879 ± 0.06</td>
<td>0.7434 ± 0.12</td>
<td>0.2265 ± 0.21</td>
<td>0.8932 ± 0.06</td>
<td>0.8923 ± 0.06</td>
</tr>
</tbody>
</table>

## AUC-ROC

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Baseline</th>
<th>SMOTE</th>
<th>SMOTEBoost</th>
<th>RUSBoost</th>
<th>RACOG</th>
<th>wRACOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>prompting</td>
<td>0.8651 ± 0.04</td>
<td>0.8585 ± 0.04</td>
<td>0.8323 ± 0.05</td>
<td>0.5126 ± 0.01</td>
<td>0.8854 ± 0.04</td>
<td>0.8843 ± 0.05</td>
</tr>
<tr>
<td>abalone</td>
<td>0.9361 ± 0.03</td>
<td>0.9280 ± 0.03</td>
<td>0.9030 ± 0.04</td>
<td>0.8449 ± 0.05</td>
<td>0.9295 ± 0.04</td>
<td>0.9188 ± 0.04</td>
</tr>
<tr>
<td>car</td>
<td>0.9859 ± 0.01</td>
<td>0.9523 ± 0.02</td>
<td>0.9698 ± 0.01</td>
<td>0.5290 ± 0.03</td>
<td>0.9951 ± 0.00</td>
<td>0.9930 ± 0.01</td>
</tr>
<tr>
<td>nursery</td>
<td>0.9999 ± 0.00</td>
<td>0.9848 ± 0.00</td>
<td>0.9944 ± 0.00</td>
<td>0.6515 ± 0.01</td>
<td>0.9995 ± 0.00</td>
<td>0.9992 ± 0.00</td>
</tr>
<tr>
<td>letter</td>
<td>0.9878 ± 0.01</td>
<td>0.9861 ± 0.00</td>
<td>0.9882 ± 0.01</td>
<td>0.8654 ± 0.01</td>
<td>0.9856 ± 0.00</td>
<td>0.9861 ± 0.00</td>
</tr>
<tr>
<td>connect-4</td>
<td>0.7895 ± 0.02</td>
<td>0.7893 ± 0.01</td>
<td>0.7849 ± 0.04</td>
<td>0.5173 ± 0.00</td>
<td>0.8187 ± 0.02</td>
<td>0.8098 ± 0.04</td>
</tr>
<tr>
<td>ecoli</td>
<td>0.9236 ± 0.02</td>
<td>0.9206 ± 0.02</td>
<td>0.9063 ± 0.03</td>
<td>0.8037 ± 0.06</td>
<td>0.9128 ± 0.03</td>
<td>0.9151 ± 0.05</td>
</tr>
<tr>
<td>haberman</td>
<td>0.5634 ± 0.09</td>
<td>0.5774 ± 0.07</td>
<td>0.5720 ± 0.08</td>
<td>0.5261 ± 0.02</td>
<td>0.5678 ± 0.09</td>
<td>0.6046 ± 0.11</td>
</tr>
<tr>
<td>yeast4</td>
<td>0.8287 ± 0.08</td>
<td>0.8038 ± 0.07</td>
<td>0.8286 ± 0.07</td>
<td>0.5174 ± 0.03</td>
<td>0.8517 ± 0.06</td>
<td>0.8658 ± 0.06</td>
</tr>
<tr>
<td>yeast6</td>
<td>0.8932 ± 0.09</td>
<td>0.8931 ± 0.09</td>
<td>0.8838 ± 0.08</td>
<td>0.5422 ± 0.04</td>
<td>0.9517 ± 0.03</td>
<td>0.9540 ± 0.03</td>
</tr>
</tbody>
</table>

Table 3.6: Results for k-nearest neighbor.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Baseline</th>
<th>SMOTE</th>
<th>SMOTEBoost</th>
<th>RUSBoost</th>
<th>RACOG</th>
<th>wRACOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>prompting</td>
<td>0.0880 ± 0.04</td>
<td>0.5488 ± 0.10</td>
<td>0.3183 ± 0.15</td>
<td>0.0880 ± 0.04</td>
<td>0.4738 ± 0.09</td>
<td>0.4867 ± 0.10</td>
</tr>
<tr>
<td>abalone</td>
<td>0.8314 ± 0.06</td>
<td>0.8565 ± 0.09</td>
<td>0.7898 ± 0.09</td>
<td>0.8314 ± 0.06</td>
<td>0.8565 ± 0.12</td>
<td>0.9082 ± 0.08</td>
</tr>
<tr>
<td>car</td>
<td>0.2896 ± 0.06</td>
<td>0.9692 ± 0.07</td>
<td>0.8335 ± 0.10</td>
<td>0.2896 ± 0.06</td>
<td>1.0000 ± 0.00</td>
<td>1.0000 ± 0.00</td>
</tr>
<tr>
<td>nursery</td>
<td>0.7576 ± 0.07</td>
<td>0.9606 ± 0.02</td>
<td>0.9849 ± 0.02</td>
<td>0.7576 ± 0.07</td>
<td>1.0000 ± 0.00</td>
<td>1.0000 ± 0.00</td>
</tr>
<tr>
<td>letter</td>
<td>0.6437 ± 0.03</td>
<td>0.8943 ± 0.03</td>
<td>0.8983 ± 0.02</td>
<td>0.6437 ± 0.03</td>
<td>0.9220 ± 0.03</td>
<td>0.9485 ± 0.02</td>
</tr>
<tr>
<td>connect-4</td>
<td>0.3200 ± 0.06</td>
<td>0.6200 ± 0.05</td>
<td>0.5860 ± 0.05</td>
<td>0.3200 ± 0.06</td>
<td>0.8100 ± 0.02</td>
<td>0.9880 ± 0.01</td>
</tr>
<tr>
<td>ecoli</td>
<td>0.7882 ± 0.13</td>
<td>0.8267 ± 0.15</td>
<td>0.7529 ± 0.17</td>
<td>0.7882 ± 0.13</td>
<td>0.8400 ± 0.12</td>
<td>0.9333 ± 0.05</td>
</tr>
<tr>
<td>haberman</td>
<td>0.2221 ± 0.07</td>
<td>0.4081 ± 0.07</td>
<td>0.3963 ± 0.08</td>
<td>0.2221 ± 0.07</td>
<td>0.4706 ± 0.08</td>
<td>0.9375 ± 0.06</td>
</tr>
<tr>
<td>yeast4</td>
<td>0.1764 ± 0.15</td>
<td>0.5291 ± 0.16</td>
<td>0.4109 ± 0.12</td>
<td>0.1764 ± 0.15</td>
<td>0.7273 ± 0.19</td>
<td>0.7473 ± 0.20</td>
</tr>
<tr>
<td>yeast6</td>
<td>0.4000 ± 0.19</td>
<td>0.5714 ± 0.23</td>
<td>0.5714 ± 0.14</td>
<td>0.4000 ± 0.19</td>
<td>0.7714 ± 0.16</td>
<td>0.7714 ± 0.19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Baseline</th>
<th>SMOTE</th>
<th>SMOTEBoost</th>
<th>RUSBoost</th>
<th>RACOG</th>
<th>wRACOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>prompting</td>
<td>0.2880 ± 0.07</td>
<td>0.7062 ± 0.07</td>
<td>0.5393 ± 0.13</td>
<td>0.2880 ± 0.07</td>
<td>0.6649 ± 0.06</td>
<td>0.6714 ± 0.06</td>
</tr>
<tr>
<td>abalone</td>
<td>0.8659 ± 0.03</td>
<td>0.8691 ± 0.05</td>
<td>0.849 ± 0.05</td>
<td>0.8659 ± 0.03</td>
<td>0.8638 ± 0.06</td>
<td>0.8404 ± 0.05</td>
</tr>
<tr>
<td>car</td>
<td>0.5314 ± 0.06</td>
<td>0.9566 ± 0.03</td>
<td>0.8898 ± 0.05</td>
<td>0.5314 ± 0.06</td>
<td>0.9713 ± 0.00</td>
<td>0.9710 ± 0.00</td>
</tr>
<tr>
<td>nursery</td>
<td>0.8677 ± 0.04</td>
<td>0.9696 ± 0.01</td>
<td>0.9814 ± 0.01</td>
<td>0.8677 ± 0.04</td>
<td>0.9872 ± 0.00</td>
<td>0.9890 ± 0.00</td>
</tr>
<tr>
<td>letter</td>
<td>0.7996 ± 0.02</td>
<td>0.9219 ± 0.01</td>
<td>0.9245 ± 0.01</td>
<td>0.7996 ± 0.02</td>
<td>0.9287 ± 0.01</td>
<td>0.933 ± 0.01</td>
</tr>
<tr>
<td>connect-4</td>
<td>0.5571 ± 0.05</td>
<td>0.7363 ± 0.04</td>
<td>0.7202 ± 0.03</td>
<td>0.5571 ± 0.05</td>
<td>0.8014 ± 0.02</td>
<td>0.6316 ± 0.06</td>
</tr>
<tr>
<td>ecoli</td>
<td>0.8521 ± 0.08</td>
<td>0.8578 ± 0.09</td>
<td>0.8176 ± 0.10</td>
<td>0.8521 ± 0.08</td>
<td>0.8658 ± 0.08</td>
<td>0.8672 ± 0.03</td>
</tr>
<tr>
<td>haberman</td>
<td>0.4446 ± 0.07</td>
<td>0.5300 ± 0.06</td>
<td>0.5306 ± 0.07</td>
<td>0.4446 ± 0.07</td>
<td>0.5470 ± 0.06</td>
<td>0.6725 ± 0.17</td>
</tr>
<tr>
<td>yeast4</td>
<td>0.3633 ± 0.23</td>
<td>0.6879 ± 0.10</td>
<td>0.6178 ± 0.09</td>
<td>0.3633 ± 0.23</td>
<td>0.7798 ± 0.10</td>
<td>0.7969 ± 0.09</td>
</tr>
<tr>
<td>yeast6</td>
<td>0.6185 ± 0.13</td>
<td>0.7364 ± 0.15</td>
<td>0.7431 ± 0.09</td>
<td>0.6185 ± 0.13</td>
<td>0.8421 ± 0.08</td>
<td>0.8453 ± 0.10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Baseline</th>
<th>SMOTE</th>
<th>SMOTEBoost</th>
<th>RUSBoost</th>
<th>RACOG</th>
<th>wRACOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>prompting</td>
<td>0.8460 ± 0.01</td>
<td>0.8413 ± 0.03</td>
<td>0.8479 ± 0.01</td>
<td>0.5396 ± 0.02</td>
<td>0.8264 ± 0.02</td>
<td>0.8350 ± 0.02</td>
</tr>
<tr>
<td>abalone</td>
<td>0.9213 ± 0.03</td>
<td>0.9303 ± 0.03</td>
<td>0.9252 ± 0.03</td>
<td>0.8672 ± 0.03</td>
<td>0.9198 ± 0.04</td>
<td>0.8924 ± 0.04</td>
</tr>
<tr>
<td>car</td>
<td>0.9746 ± 0.01</td>
<td>0.9759 ± 0.00</td>
<td>0.9705 ± 0.01</td>
<td>0.6370 ± 0.03</td>
<td>0.9751 ± 0.01</td>
<td>0.9778 ± 0.00</td>
</tr>
<tr>
<td>nursery</td>
<td>0.9962 ± 0.00</td>
<td>0.9955 ± 0.00</td>
<td>0.9942 ± 0.00</td>
<td>0.8765 ± 0.03</td>
<td>0.9959 ± 0.00</td>
<td>0.9958 ± 0.00</td>
</tr>
<tr>
<td>letter</td>
<td>0.9814 ± 0.00</td>
<td>0.9772 ± 0.00</td>
<td>0.9726 ± 0.00</td>
<td>0.8187 ± 0.02</td>
<td>0.9804 ± 0.01</td>
<td>0.9792 ± 0.00</td>
</tr>
<tr>
<td>connect-4</td>
<td>0.8829 ± 0.02</td>
<td>0.802 ± 0.01</td>
<td>0.8293 ± 0.01</td>
<td>0.6482 ± 0.03</td>
<td>0.8799 ± 0.02</td>
<td>0.8778 ± 0.02</td>
</tr>
<tr>
<td>ecoli</td>
<td>0.9310 ± 0.04</td>
<td>0.9072 ± 0.05</td>
<td>0.9221 ± 0.03</td>
<td>0.8570 ± 0.08</td>
<td>0.9227 ± 0.05</td>
<td>0.9201 ± 0.03</td>
</tr>
<tr>
<td>haberman</td>
<td>0.6045 ± 0.10</td>
<td>0.5801 ± 0.08</td>
<td>0.5640 ± 0.10</td>
<td>0.5666 ± 0.04</td>
<td>0.5961 ± 0.07</td>
<td>0.5507 ± 0.06</td>
</tr>
<tr>
<td>yeast4</td>
<td>0.7984 ± 0.12</td>
<td>0.7384 ± 0.09</td>
<td>0.7818 ± 0.07</td>
<td>0.5851 ± 0.07</td>
<td>0.8047 ± 0.12</td>
<td>0.8053 ± 0.12</td>
</tr>
<tr>
<td>yeast6</td>
<td>0.9097 ± 0.04</td>
<td>0.9007 ± 0.05</td>
<td>0.8349 ± 0.09</td>
<td>0.6969 ± 0.09</td>
<td>0.9037 ± 0.06</td>
<td>0.8969 ± 0.07</td>
</tr>
</tbody>
</table>

Table 3.7: Results for logistic regression.
Figure 3.10: Comparison of $\log$(number of instances added) by different methods.

SMOTEBost requires ten boosting iterations to produce ten hypotheses on which weighted voting is performed while predicting. The hypothesis learned at each iteration of SMOTEBost has the form: $h_t : X \times Y \rightarrow [0, 1]$, and therefore stores the instances generated by SMOTE at every iteration. Hence, SMOTEBost adds ten times the number of samples added by SMOTE. On the other hand, wRACOG requires a fraction ($\sim 56\%$) of the number of samples generated by SMOTE and RACOG, and ($\sim 5.6\%$) of the number of samples generated by SMOTEBost, to obtain superior performance. We attribute this behavior to wRACOG’s sample selection methodology which ensures diversity in the samples that are added.
3.6.3 Comparison with a Baseline Prompting Algorithm

Our results show that RACOG and wRACOG perform better than the state-of-the-art resampling techniques to address imbalance class distribution. However, all the techniques that are considered in our experiments are fundamentally classification-based approaches. We opt for this type of approach based on our hypothesis that classifying activity steps into prompt and no-prompt classes, based on the corresponding smart home sensor data, is the best way to predict prompting situations.

Therefore, in this section, we compare the performance of RACOG and wRACOG against a baseline algorithm that also determines prompting situation from activity steps, but is not a supervised classification approach. This algorithm, named PromptingBaseline, evaluates an activity step based on its “closeness” to the probability distributions of activity steps belonging to the prompt and no-prompt classes. We utilize the sensor trigger counts associated with the activity steps to fit multivariate Poisson distributions for both the prompt and the no-prompt classes. For a test sample, represented as a feature vector of sensor trigger counts, the probability that it belongs to a particular class is calculated using the Poisson probability mass function given by \( \frac{\lambda^k}{k!} e^{-\lambda} \). The class for which the test sample obtains the highest probability, is assigned as its class label.

We compare the performance of PromptingBaseline with C4.5 decision tree,
Figure 3.11: Comparison of sensitivity (left) and g-mean (right) of PromptingBaseline versus the C4.5 preprocessed with RACOG and wRACOG.

pre-processed using RACOG and wRACOG, in Figure 3.11. The results show that both RACOG and wRACOG perform significantly ($p < 0.01$) better than PromptingBaseline in terms of sensitivity and g-mean. For other classifiers, whose results are report in Tables 3.5, 3.6, and 3.7, PromptingBaseline only performs better than Logistic Regression when it is pre-processing with RACOG (sensitivity = 0.4738, g-mean = 0.6649) and wRACOG (sensitivity = 0.4867, g-mean = 0.6714). The AUC-ROC for PromptingBaseline is 0.6857 (std. dev. = 0.0612) which is lower than RACOG and wRACOG for all classifiers. These results validate our hypothesis that classification is the right choice of algorithms for the automated prompting task.
3.7 Summary

In this chapter, we propose two probabilistic oversampling algorithms for generating new minority class samples for class imbalanced datasets. The unknown probability distribution of the minority class is approximated using Chow-Liu’s dependence tree algorithm. Although learning a Bayesian network on the minority class training examples is an alternative approach to approximate the probability distribution, we choose the dependence tree approach as it is less computationally expensive than learning an arbitrary Bayesian network. Gibbs sampling is used as a sampling technique to synthetically generate new minority class samples from the approximated probability distribution. The proposed approaches, RACOG and wRACOG, differ in the sample selection strategy that is used to select samples from the Markov chain generated by the Gibbs sampler. RACOG runs the Gibbs sampler for a predetermined number of iterations and selects samples from the Markov chain using predefined burn-in and lag. On the other hand, wRACOG selects samples that have the highest probability of being misclassified by the existing learning model. This approach keeps on adding samples unless there is no further improvement in sensitivity over a predefined number of most recent iterations.

Experiments with RACOG and wRACOG on a wide variety of datasets and classifiers indicate that the algorithms are able to attain higher sensitivity than other
methods, while maintaining higher G-mean. This supports our hypotheses that generating new samples by considering the global distribution of minority class samples is a good approach for dealing with class imbalance. The motivation to focus mainly on improving sensitivity comes from the automated prompting application. However, we ensure that the performance of the classifiers on both the classes is not hampered.

An open question that still remains is to determine how the proposed approaches and alternative sampling techniques compare with random oversampling and undersampling. Counter-intuitive to the widely accepted notion that random oversampling and under-sampling perform worse than sophisticated sampling methods for imbalanced class distribution, our experiments indicate that these approaches perform at par with, and sometimes even better than other sampling approaches discussed in this chapter. However, random oversampling and under-sampling are inherently different from the sampling approaches discussed in this chapter because they duplicate training samples rather than creating synthetic samples, which makes the comparison difficult. This issue can be investigated as part of future work and will need a more extensive experimental setup that includes more datasets with high dimensions and very high cardinality. In addition, sampling approaches for handling continuous-valued variables can also be examined.
In the prompting dataset, the features that represent each activity step are sometimes insufficient to draw crisp boundaries between the prompt and the no-prompt classes. This characteristic of the data creates some regions in the data space that are “ambiguous”. Ambiguous regions of data gives rise to a well-known problem in machine learning, namely, overlapping classes, which is discussed in the next chapter.
CHAPTER 4. CLASS OVERLAP PROBLEM

4.1 Introduction

In spite of making a significant progress in the area of the class imbalance problem, researchers are facing new emerging class imbalance challenges that make the problem harder to solve with existing techniques. Consider a network intrusion detection system. Network intruders these days have become smart enough to disguise their identity as legitimate users. This is a binary class learning problem where data points may appear as valid examples of both positive and negative classes. This situation occurs widely in character recognition [Liu, 2008], drug design [Andrews, 2007] and also automated prompting in smart environments [Das et al., 2010] where samples from different classes have very similar characteristics. The minor differences present between the samples of two different classes are usually difficult to be captured in the feature vector proposed by the domain expert. Therefore, there is a growing algorithmic need to deal with this issue. Although this chapter focuses on addressing overlapping classes in automated prompting problem, our approach is easily extensible for other problem domains which have datasets of a similar nature.
As discussed in Chapter 3, our first goal is to classify an activity step (represented as a training example) either as a prompt step or a no-prompt step. As, in a realistic setting, there are very few situations that would require a prompt as opposed to situations that would not, the number of training examples for prompt class is extremely low as compared to no-prompt class. This makes the data inherently class imbalanced. Moreover, the features that represent each activity step are insufficient to draw crisp boundaries between these classes for some regions in the data space that is ambiguous. This causes the occurrence of overlapping classes in addition to the inherent presence of imbalance class distribution. It can be argued that, if the proposed features are incapable of capturing the representative properties of the training examples that can help in proper distinction of the two classes, why not engineer new features that can add more distinctive representation of the classes? The best answer to this question is that the lack of infrastructural requirements in a realistic setting restricts the addition of new features.

As the accurate classification of prompt samples, which is in the minority, is critically more important than identification of no-prompt samples, which is in the majority, we propose a solution that addresses both class imbalance and overlap. Our solution ClusBUS, is a clustering-based under-sampling technique, that identifies data regions where minority class (prompt) samples are embedded deep inside majority class samples. By removing majority class samples from these regions, ClusBUS
preprocesses the data in order to give the prompt class more importance during classification.

### 4.2 Problem Definition

Although, both the problems of class imbalance and overlap can exist in data with multiple classes, due to a general consensus in the literature [Woods et al., 1993, Kubat et al., 1998, Phua et al., 2004] and our current application objective, in this chapter we deal with data that has one minority and one majority class. The class overlap problem [Denil, 2010] 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 probabilities 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 4.1 illustrates the difference between normal data with crisp class boundaries and data with overlapping classes.

Overlapping classes when combined with class imbalance causes the resultant problem to be harder to solve than solving them independently. It has been seen in
some cases that identifying the overlapping region in the data space and getting rid of those instances makes the data linearly separable. This idea is implemented in approaches such as SMOTE+Tomek [Batista et al., 2004] and Tomek+CNN [Tomek, 1976], among others. However, in some cases an additional problem of rare training examples of the positive class makes dealing with class overlap in imbalanced class distribution even more difficult.

Experiments with human participants in our smart home testbed show that there are certain prompt situations where features of the smart home sensor data are quite similar to the situations when the participant would probably need no prompt. This kind of situation is prevalent in daily activities that involve object interactions. As the sensor data used in this work were gathered during the study which did not
have dedicated vibration-based object sensors to track household object use, it is difficult to gauge from the raw sensor data if the participant actually committed an error in such activities. Thus, the absence of sufficient data attributes to differentiate between prompt and no-prompt classes causes class overlap. Our claim is verified when the prompting data is plotted in a 3-dimensional space using Principal Component Analysis for dimensionality reduction, which is a well-known data visualization technique. Figure 4.2 shows that the minority class samples are highly embedded in the majority class, thus making them inseparable if fed as they are to the classifiers. Moreover, due to infrastructural limitations, it is not possible to add new differentiating features to the current dataset, leaving us with the only option of addressing the problem algorithmically.

**Figure 4.2:** 3D PCA plot for prompting data.
As a solution to the overlap problem, the literature primarily talks about preprocessing the data before using it to learn a classification model. Preprocessing data to address class overlap is usually a two step process shown in Figure 4.3. The first and most crucial step is to identify regions of overlap in the data space. This is followed by handling samples in the overlapping region by using methods that mainly fall into three categories [Xiong et al., 2010]: separating, merging and discarding. We discuss these methods in the following section. The approach discussed in this chapter preprocesses the data by performing a clustering-based under-sampling of the overlapping region to build a better learning model.

![Figure 4.3: Steps taken to address class overlap.](image)

### 4.3 Related Work

While there has not been a significant work in dealing with the class overlap problem in combination with imbalanced class distribution, the problem of overlapping classes or ambiguous data has been widely studied in isolation [Trappenberg and

There have been several systematic and extensive investigations to study the nature of classifiers when they are faced with the class overlap problem in addition to an imbalanced class problem. Prati et al. [Prati et al., 2004] give a vivid illustration of the cause of imbalanced class distribution posing a problem in the presence of high degree of class overlap. They show that overlap aggravates the problem of imbalance and is sufficient to degrade the performance of the classifier on its own. The same authors report the performance of different balancing strategies on artificial datasets in [Batista et al., 2005]. García et al. [García et al., 2006] analyze the combined effects of class imbalance and class overlap on instance-based classification. This work is extended [García et al., 2007] by using several performance measures to see which one of them captures the degraded performance more accurately.

A major obstacle in solving class overlap problem in data is identification of ambiguous or overlapping regions. However, this issue has been addressed to some extent by the approaches that deal with class overlap problem in isolation. Tang et al. [Yaohua and Jinghuai, 2007] proposed a k-Nearest Neighbor based approach to identify ambiguous regions in the data space. Trappenberg et al. [Trappenberg
and Back, 2000] take a very similar approach to identify ambiguous regions. Visa et al. [Visa and Ralescu, 2003] perform a fuzzy set representation of the concept and thus incorporate overlap information in their fuzzy classifiers. In addition, Xiong et al. [Xiong et al., 2010] use a one-class classification algorithm Support Vector Data Description (SVDD) to capture the overlapping regions in real-life datasets which have imbalanced class distribution as well.

Once the overlapping region of the data space is identified, the obvious next step is to handle the training examples that belong to this region. Xiong et al. [Xiong et al., 2010] propose that the data with the presence of class overlapping can be modeled with three different schemes: discarding, merging and separating. The discarding scheme ignores the data in the overlapping region and just learns on the rest of the data that belongs to the non-overlapping region. SMOTE + Tomek Links [Batista et al., 2003] uses the discarding scheme to improve classification performance of protein annotations in bioinformatics. While the discarding scheme works satisfactorily for datasets that have ample number of training examples from both classes, it would perform drastically when applied to datasets which have absolute rarity in data.

The merging scheme merges the data in the overlapping region into a new class. A two-tier classification model is built on the data. The upper tier classifier focuses on the whole data with an additional class which represents the overlapping region. The lower tier classifier on the other hand focuses on the data that belongs to the
overlapping region. Trappenberg et al. [Trappenberg and Back, 2000] proposed a scheme that refers to the overlapping region class as IDK (I don’t know) and do not attempt to predict the original class of this data. The authors argue that, although this scheme loses some prediction of data, a drastic increase of confidence can be gained on the classification of the remaining data. Hashemi et al. [Hashemi and Trappenberg, 2002] take a very similar approach to address the issue.

In the *separating scheme*, the data from overlapping and non-overlapping regions are treated separately to build the learning models. Tang et al. [Yaohua and Jinghuai, 2007] proposed a multi-model classifier named Dual Rough Support Vector Machine (DR-SVM) which combines SVM and kNN under rough set technique. kNN is used to extract boundary patterns or overlapping regions. Two different SVMs are then trained for the overlapping and non-overlapping regions. But, the classification result will show whether a pattern lies in overlapping region. Although, the classification of a test example as belonging to overlapping and non-overlapping region depends on the goal of the application problem, this methodology would involve an additional domain expert knowledge to determine the class of the test example. As a result, this scheme is not suitable for applications where it is a requirement of the system to determine the class of the test example and has no room for additional domain expert intervention.

All of the aforementioned schemes either consider the overlapping data as noise
or just avoid making a decision on their original classes so that the confidence of prediction on the remaining data could be increased. This approach of partially “avoiding the problem” rather than proposing a solution is not appropriate for many real-life problem domains where it is absolutely necessary for the system to take a decision with certainty (often due to a time-critical nature) rather than waiting for the domain expert intervention. For example, in the problem domain of intrusion detection where attackers can disguise themselves as legitimate users, high traffic of attackers necessitates to take a time-critical decision on the authenticity of the user.

In this chapter we take a preprocessing approach, described in the following section, similar to the discarding scheme to deal with the overlapping training examples. Instead of designating the boundary points as noises, our approach considers them as crucial for decision making in the overlapping region. The minority class points in the overlapping region are retained and the majority class points are discarded to make a clear distinction between the minority class points in the overlapping region and the rest of the dataset.

4.4 Proposed Approach

Class imbalance and overlap are strongly coupled with each other when it comes to learning from class imbalanced data. Denil et al. proved [Denil and Trappenberg,
2010] this by performing hypothesis tests. They also showed that if overlap and imbalance levels are too high, good performance cannot be achieved regardless of the amount of available training data. We hypothesize that by addressing the overlap problem, we would be able to get rid of the detrimental effects of class imbalance to some extent. Therefore, we employ a Clustering-Based Under-Sampling (ClusBUS) technique to get rid of the overlapping classes.

The idea of devising this technique is derived from the use of Tomek links [Tomek, 1976] combined with other sampling methods such as Condensed Nearest Neighbor [Hart, 1968] and SMOTE [Batista et al., 2004]. Given two data samples $E_i$ and $E_j$ belonging to different classes, and $d(E_i, E_j)$ representing the distance between $E_i$ and $E_j$, a $(E_i, E_j)$ pair is called a Tomek link if there is no sample $E_k$ such that $d(E_i, E_k) < d(E_i, E_j)$. Quite naturally, if two samples form a Tomek link, then either one of these samples is noise or both samples are on or near the class boundary. Tomek links are used both as a data cleansing method by removing samples of both classes, and as an under-sampling method by eliminating only majority class samples from the links. For instance, One-sided selection (OSS) [Kubat and Matwin, 1997] is an under-sampling method that applies Tomek links followed by the application of Condensed Nearest Neighbor (CNN). In this method, Tomek links are used to remove noisy and borderline majority class samples. As a small amount of noise can make the borderline samples fall on the wrong side of the decision boundary, borderline
samples are considered unsafe. CNN is used to remove samples from the majority class that are far away from the decision boundary. The rest of the majority and minority class samples are used for the purpose of training the classifiers.

4.4.1 Cluster-Based Under-Sampling (ClusBUS)

As opposed to using Tomek links in OSS to find minimally-distant nearest neighbor pairs of opposite class samples and then removing majority class samples, we find clusters which have a good mix of minority and majority class samples. A good mix is determined by the degree of minority class dominance in each cluster. The majority class samples from these clusters are then removed.

First, the entire training data is clustered ignoring the class attribute using Euclidean distance as the distance measure. The degree of minority class dominance of each cluster, denoted by $r$, is calculated as the ratio of number of minority class samples to the size of the cluster. Therefore, $r = 0$ indicates that all the samples of the cluster belong to the majority class, and $r = 1$ indicate that all the samples belong to the minority class. For clusters with $0 \leq r \leq 1$, the majority class samples are removed if $r$ is equal to or greater that an empirically determined threshold $\tau$. Clearly, if $\tau$ is low, more majority class examples would be removed as compared to when $\tau$ is high. This method creates a “vacuum” around the minority class samples in each
Algorithm 1: ClusBUS

1: Let $S$ be the original training set.

2: Form clusters on $S$ denoted by $C_i$ such that $1 < i < |C|$.

3: Find the degree of minority class dominance for all $C_i$ by:
   $$r_i = \frac{\# \text{ minority class samples in } C_i}{|C_i|}$$

4: For clusters that satisfy $0 < r_i < 1$ and $r \geq \tau$ (where, $\tau = f(r)$ is an empirically determined threshold for $r$ and is uniform over all the clusters), remove all the majority class samples and retain the minority class samples.

**Figure 4.4:** ClusBUS algorithm.

cluster and thus helps the machine learning classifiers learn the decision boundary more efficiently. The ClusBUS algorithm is summarized in Figure 4.4.

Figure 4.5 shows an illustration of ClusBUS on a synthetic dataset. The imbalanced and overlapping data is represented in the figure at top-left. Identification of overlapping regions in the data space is performed using clustering as shown in the top-right diagram. The majority class points are removed from the clusters for which $\delta > \tau$. Note that, in the bottom diagram of Figure 4.5, the majority class points have been removed from all the clusters in order to make the visual representation of the step explanatory. In the actual algorithm, removal of majority class points is done only on the basis of $\delta > \tau$.  

Figure 4.5: Schematic representation of ClusBUS algorithm.

4.4.2 Choice of Clustering Algorithm

Theoretically, there is no restriction on the choice of clustering algorithm that should be used to identify clusters containing samples from both classes. However, we avoid any form of partitioning-based clustering method in our experiments for a
couple of reasons. Firstly, partitioning-based clustering requires user intervention to specify the number of clusters that need to be formed from the data. Secondly, these methods form spherical clusters only, which might not be the ideal cluster shape in many datasets. Therefore, in this study, we use a density-based clustering algorithm, namely, Density-Based Spatial Clustering of Applications with Noise or DBSCAN [Ester et al., 1996, Han and Kamber, 2006]. DBSCAN forms arbitrary shapes of clusters in the data that are not necessarily spherical (Gaussian). Moreover, as we do not make any assumption on the distribution of the data, the notion of density on which DBSCAN is based is more meaningful than specifying the number of clusters and forcing the data to be partitioned accordingly. In the following, we provide a brief background description of DBSCAN.

DBSCAN is a density based clustering technique that treats clusters as dense regions of objects in the data space that are separated by regions of low density, mostly representing noise. An object that is not contained in any cluster is considered as noise. In other words, DBSCAN defines a cluster as a maximal set of density-connected points. The neighborhood of an object or data point is defined by a parameter $\epsilon$. If the $\epsilon$-neighborhood of a data point contains at least a minimum number of other points denoted by $\text{MinPts}$, then the point is called a core point, and the $\epsilon$-neighboring points are directly density-reachable from the core point. A point $p$ is density-reachable from point $q$ with respect to $\epsilon$ and $\text{MinPts}$, if there is
a chain of objects \( p_1, \ldots, p_n \), where \( p_1 = q \) and \( p_n = p \) such that \( p_{i+1} \) is directly density-reachable from \( p_i \) with respect to \( \epsilon \) and \textbf{MinPts}. In a similar way, a point \( p \) is density-connected to \( q \) if there is a point \( o \) in the same data space such that both \( p \) and \( q \) are density-reachable from \( o \) with respect to \( \epsilon \) and \textbf{MinPts}. The algorithm is summarized in Figure 4.6.

**Algorithm 2: DBSCAN**

1: Search for clusters by checking \( \epsilon \)-neighborhood of each point.

2: If \( \epsilon \)-neighborhood of a point \( p \) contains more than MinPts, a new cluster with \( p \) as core point is created.

3: Iterate collection of directly density-reachable points from the core points.

4: Terminate when no new point can be added to any cluster.

**Figure 4.6:** DBSCAN algorithm.

The degree of minority class dominance is also a very crucial factor for the selection of candidate clusters to perform under-sampling. An appropriate choice of threshold \( \tau \) could directly affect the performance of the classifiers. We take an empirical approach (described in Section 4.5) to determine \( \tau \) by choosing a value between min and median of \( r \).
4.4.3 Ensemble of Activity-Specific Classifiers

In the ClusBUS approach, we preprocess the prompting dataset that contains training examples from all activities of daily living that were part of Study 1 (discussed in Section 2.5). This results in a generic solution for all activities that does not take advantage of any activity-specific behavior that can be utilized for better prediction of prompts. In order to take the advantage of activity-specific information, we propose an ensemble of classifiers trained on the activities separately and preprocessed by ClusBUS. A test sample is evaluated by all the classifiers of the activities. If any of the classifiers predicts the test sample as “prompt”, the ensemble labels the test sample as prompt as well. Otherwise, it is labeled as “no-prompt”. The ensemble approach is illustrated in Figure 4.7.

4.5 Experiments and Results

4.5.1 Setup

We use four commonly known machine learning classifiers to perform our experiments on the prompting data and validate the effectiveness of our proposed data preprocessing approach. These classifiers are: C4.5 decision tree (C4.5), k-nearest neighbor (IBk), naive Bayes classifier and a Sequential Minimal Optimization (SMO)
version of support vector machines (SVM). We perform a 5-fold cross validation on the data where the training and test data in each fold retains the ratio of class imbalance and overlap, and the training data is preprocessed using ClusBUS.

We report the True Positive rate (TPR) (prompt class being the positive class) that represents the fraction of activity steps that are correctly classified as requiring a prompt, and the False Positive rate (FPR) that represents the fraction of no-prompt steps that are classified as prompt steps. The TPR is thus capable of measuring the performance of the classifiers separately for the prompt class. The area under the ROC

**Figure 4.7:** Ensemble created from the classifiers of all activities.
curve (AUC) is used to evaluate the performance over the costs and distributions. We also report the geometric mean of true positive and true negative rates (G-mean = $\sqrt{TPR \times TNR}$) to measure the effectiveness of a classifier on both of the classes together. In the current application, 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 individuals with dementia. However, considering that the purpose of this research is to assist people by delivering fewer 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.

Due to the unavailability of any implementation of approaches that provide a unified solution to address class imbalance and overlap, we compare the performance of ClusBUS with a well-known oversampling technique, known as SMOTE [Chawla et al., 2002]. SMOTE oversamples the minority class by creating “synthetic” samples based on spatial location of the samples in the Euclidean space. A detailed description of SMOTE is given in Section 3.5.1. However, SMOTE does not guarantee that the generated samples would belong to the minority class, especially when the samples from the majority and minority classes overlap. We use SMOTE to produce a 50:50 distribution of minority and majority class samples, which is considered near optimum [Weiss and Provost, 2003].
In addition, we evaluate the performance of ClusBUS when used in combination with RACOG and wRACOG, proposed for imbalanced class distribution in Chapter 3. In these approaches, the majority class is first under-sampled using ClusBUS, and then RACOG and wRACOG are used to oversample the minority class. Hence, the resultant techniques are a combination of oversampling and under-sampling-based preprocessing techniques to address imbalanced class distribution and class overlap.

4.5.2 Determining the Threshold on Minority Class Dominance

As mentioned in Section 4.4, candidate clusters for under-sampling are chosen on the basis of an empirically determined threshold $\tau$ on the degree of minority class dominance $r$. In order to determine $\tau$, we assume real values of $r$ as $q$-quantile ordered values. Quantiles are points taken at regular intervals from the cumulative distribution function of a random variable, which in our case is $r$. Dividing ordered values into $q$ essentially equal-sized value subsets is the motivation for $q$-quantiles. We vary $\tau$ from the boundary values of first quantiles for $q$-quantile values of $r$ where $2 < q < 10$. Based on these threshold values, we preprocess the prompting data using ClusBUS and feed it to a C4.5 decision tree classifier.

The plots for TPR, FPR and AUC obtained by applying C4.5 on prompting data using different values of $\tau$ are shown in Figure 4.8. A uniform trend in the
Figure 4.8: TPR, FPR, and AUC obtained from C4.5 for different 1st quantile boundary values of $q$-quantile $r$ values.

Performance measures indicate that there is no further improvement of a classifier after 1st quantile boundary value of a 4-quantile model of ordered $r$ values. Therefore, further experiments with other classifiers are performed with $\tau = 0.25$.

4.5.3 Comparative Assessment

In this section, we compare the performance of six different approaches: (i) Baseline, which does not use any preprocessing, (ii) SMOTE, (iii) ClusBUS, (iv) ClusBUS Ensemble, (v) ClusBUS+RACOG, and (vi) ClusBUS+wRACOG. As mentioned before, ClusBUS+RACOG and ClusBUS+wRACOG are a combination of oversampling and under-sampling techniques.
The performance of the approaches in terms of TPR and FPR are plotted in Figures 4.9 and 4.10. Classifiers C4.5, IBk, and SMO, preprocessed with ClusBUS, yield statistically significant improvement \( (p < 0.05) \) over SMOTE in terms of TPR. An interesting thing to note is that the improvement varies across different classifiers which can be attributed to the underlying principle of the classifiers and how SMOTE and ClusBUS affect those principles. For example, the performance of SMOTE is worse when used with C4.5 and IBk than when used with SMO. This is because SMO uses a kernel function that helps in separating overlapping classes to some extent. The consistent improvement of ClusBUS across all four classifiers is a strong evidence that it can be used as a reliable preprocessing techniques irrespective of the classifier.

Although the ClusBUS Ensemble gives improvement over Baseline and SMOTE in terms of TPR, the improvement over ClusBUS is only significant in case of Naive Bayes and SMO classifiers. ClusBUS+RACOG and ClusBUS+wRACOG on the other hand, perform marginally better than ClusBUS. However, no statistically significant improvement was achieved in terms of TPR.

ClusBUS causes the classifiers to yield higher FPR than Baseline and SMOTE. However, as the FPR is < 20\% for all the classifiers, it does not affect their overall performance. ClusBUS Ensemble, on the other hand, yields > 90\% FPR for most of the cases. This indicates that the classifiers trained on individual activity data, which
Figure 4.9: Comparison based on True Positive Rate.

Figure 4.10: Comparison based on False Positive Rate.
are part of ClusBUS Ensemble, do not learn any common pattern for the prompt class of that specific activity. The poor performance of the classifiers in ClusBUS Ensemble can be attributed to extreme class imbalance and rarity of prompt class sample for an activity. ClusBUS+RACOG and ClusBUS+wRACOG yield significantly better \((p < 0.02)\) performance over ClusBUS in terms of FPR. However, FPR is still high as compared to Baseline and SMOTE.

Figures 4.11 and 4.12 shows the performance of all the approaches in terms of AUC and G-mean. ClusBUS results in better AUC over Baseline \((p < 0.05)\) for all classifiers except Naives Bayes. However, there is no notable improvement in AUC when compared with SMOTE. This is because ClusBUS induces minor increase in FPR. ROC curves are plotted as FPR versus TPR. Therefore, a minor increase in FPR can cause the curve to deviate away from the top-left corner which is the ideal target for any classifier. Although this can be considered as the only limitation of ClusBUS from a theoretical perspective, it is found in the literature that most sampling techniques often cause increase in FPR. However, the minor increase in FPR does not demean the advantages and improvement of ClusBUS on other methods. This is because ClusBUS does not entail any costly data synthesis technique like SMOTE, but performs under-sampling of majority class training samples that belong to the overlapping region. Moreover, from our application’s perspective, achieving a low FPR does not justify classifying prompt situations as no-prompt.
**Figure 4.11:** Comparison based on AUC.

**Figure 4.12:** Comparison based on G-mean.
The very high FPR obtained by ClusBUS Ensemble causes poor overall performance of the classifiers. This is reflected by the the low AUC and G-mean values of ClusBUS Ensemble. Apart from SMO, all the other classifiers used in the ensemble results in worse performance than Baseline in terms of AUC. ClusBUS+RACOG and ClusBUS+wRACOG do not yield significant improvement over ClusBUS in terms of AUC. However, we can see statistically significant improvement ($p < 0.05$) of G-means for Naive Bayes, IBk and SMO classifiers.

### 4.6 Summary

This chapter proposes ClusBUS, a preprocessing technique to deal with the class overlap problem in the presence of imbalanced classes in the data. The proposed method helps us in better identification of potential prompt situations in daily human activities performed in smart homes. The effectiveness of ClusBUS is established by an improved performance over a widely known technique to handle class imbalance, SMOTE. However, creating an ensemble of classifiers, trained on various activities separately, does not give promising results. This can be attributed to the extreme class imbalance and rarity of prompt class samples during training when the data is split activity-wise.

As part of the future work, the generalizability of ClusBUS can be evaluated
by applying it on datasets from other problem domains, including network intrusion
detection and credit card fraud detection. Comparison of ClusBUS with other uni-
fied solutions that address class imbalance and overlap such as SMOTE+ENN and
SMOTE+Tomek, can also be performed.

Our second methodology to automated prompting is to automatically detect
activity errors in real time, while an individual performs an activity. An activity
error can be treated a potential situation for prompt delivery. In the next chapter,
we discuss a one-class classification based activity error detection approach that does
not need training data for the activity errors.
5.1 Introduction

In this chapter, we introduce an alternative approach to automating prompting-based interventions. We propose an approach that automatically detects activity errors in real time, while an individual performs an activity, using no training data for the activity errors. Activity error occurrences are the potential situations when human caregivers would issue prompts to an individual with cognitive impairment. Thus, we hypothesize that if activity errors are detected correctly, information, such as error occurrence time and type of error, can be used to deliver effective instructions for activity completion.

In the previous chapters, we discussed emulating caregiver interventions by using human prompt timings to train machine learning algorithms. This approach makes a number of assumptions. Firstly, we assume that there is an availability of training data that indicates when caregivers would detect activity errors and deliver corresponding prompts. Secondly, we also assume that each sensor event is labeled
with corresponding activity steps in order to effectively learn the prompt model. These assumptions introduce challenges, listed below, that limit the practicality of an automated prompting system modeled in this fashion.

(i) It takes several hundred man-hours to annotate sensor data with activity and activity step labels that provide ground truth for the machine learning algorithms.

(ii) Annotating smart home sensor data is a tedious task and there is a high probability for the annotations to be erroneous. The typical inter-annotator consistency is approximately 80%.

(iii) Real-time detection of activity errors from streaming sensor data, which could be potential prompt situations, needs some form of recognition model for the activity steps. Recognition of activity steps is a hard problem.

In order to address the limitations of the previous approach, we propose one-class classification-based algorithms that do not need training data for the activity errors. Our approach, DERT, or Detecting Activity Errors in Real Time, is trained only on the activity data of the participants for whom the psychology experimenters reported no errors. Data that does not contain activity errors, comprise the target class for a particular activity. The remainder of the data for the same activity from the participants who committed errors, also called the test samples, are used to validate
the efficacy of DERT in detecting activity errors in real time. As with most outlier
detection techniques, DERT detects the outliers in the test samples. In the automated
prompting application, these outliers are activity errors. We use the sensor data from
580 participants who were part of our two smart home studies discussed in Section
2.5. Statistical features that are capable of capturing activity errors are extracted
from the raw sensor data before training our algorithms.

DERT is a collection of outlier detection algorithms to detect activity errors. We start with a basic technique that fits a multivariate Poisson distribution on the
training samples of the target class. The training samples are represented as a vector
of the number of events generated by each sensor since the beginning of the activity.
A test sample, represented in the same way a training sample, is evaluated on the
basis of its “closeness” to the target class samples.

Most traditional outlier detection algorithms assume a specific probability dis-
tribution type for the available data. In our case, the multiple features (described
in Section 5.4.1) that are used in detecting activity errors, do not follow the same
distribution as each other. For example, while sensor trigger count-based features are
likely to follow a Poisson distribution, temporal features that measure elapsed time
are more likely to follow a Gaussian distribution. Therefore, our second approach uses
a one-class support vector machine-based technique proposed by Schölkopf [Schölkopf
et al., 2001]. This algorithm uses the available training data to find a hyper-plane
that separates all target class samples from the origin with maximum margin. Test samples that are found to be close to the origin are classified as outliers. Like most of the other one-class classification approaches, DERT also requires a significant number of “normal” training examples in order to accurately detect activity errors. This is because, in one-class SVM, a boundary should be defined in all dimensions around the data [Tax, 2001]. This topic is further discussed in Section 5.3.2.

As mentioned earlier, DERT does not require training data for the activity errors in order to be effective. However, we postulate that including available information about typical types of activity errors could improve this method. Currently, there is limited understanding of the changes in daily activities that occur as individuals transition from normal aging to dementia. Characterization of this change can be done by evaluating the types of activity errors that are made by the participants. This analysis was performed by Schmitter-Edgecombe et al. [Schmitter-Edgecombe, 2014] in our smart home studies conducted with older adults. We utilize the sensor data collected from the studies to classify the error types. We also utilize knowledge gathered from the identified error classification to build ensembles that may improve error detection by our algorithm.

DERT assumes that activity labels are available in real time. In our current experiments, we use the activity labels collected from human annotators. In the future, human annotation of activities will be replaced by the predictions of a real-
time activity recognition algorithm proposed by Krishnan and Cook [Krishnan and
Cook, 2014]. Krishnan and Cook achieved 98% accuracy in predicting twelve activities
of daily living in real time. As most of the features used in the one-class classification
requires a notion of the activity start time, we assume that beginning of the activities
can be successfully identified using the activity labels obtained from the real-time
activity recognition algorithm. In addition, we also assume that the activities are
performed when the smart home has one resident and the activities are not interleaved
or concurrent.

In the following sections, we analyze the problem on the basis of available an-
notations, discuss the algorithms proposed under DERT and their limitations, and
discuss the results of our experiments.

5.2 Problem Analysis

In this chapter, we use activity data from the two studies conducted in our on-
campus smart home. These studies are discussed in Section 2.5. The activity errors
reported by psychology experimenters in Study 1 were labeled by human annotators.
An error label is assigned to a single sensor event and indicates the beginning of an
error. Once an error occurs, the activity is in an error state until it is interrupted
by the experimenter or the participant realizes the mistake and corrects the error.
However, only the error initiation is annotated, not the entire duration of the error state. Note that a number of errors that were identified by experimenters were not annotated in the sensor data because of a lack of sensor-based error evidence. Moreover, the annotator did some guess work on the basis of reported error timing. In Figure 5.1 we show the distribution of error annotations across all eight activities of daily living that were part of Study 1. The *Sensor-Based* data series represents those errors for which the annotator could manually observe abnormality in the sensor data stream. The *Time-Based* data series represents the annotations that were done by guessing the error occurrence time (and no sensor-based evidence).

In this chapter, we experiment with errors that could be identified by a human...
annotator from the sensor data, as only this category of errors has the potential to be
recognized by a computational model. We focus our experiments and optimize our
results for the top four activities based on the ratio of the number of errors annotated
in the sensor data to the total number of reported errors. These activities are as
follows: *Sweeping and Dusting, Taking Medication, Watering Plants,* and *Cooking.*

Annotations in Study 2 were made by the psychology experimenters in real-time
using our Real-Time Annotation Tool [Feuz and Cook, 2013]. We do not categorize
these errors into time-based and sensor-based, as done in Study 1. Thus, data from
all activity errors are used in our experiments.

Our goal is to design an automated prompting system that utilizes sensor data
to learn the “normal patterns” in which an activity is completed successfully. When
this prompting system is deployed in a smart home, it should be able to detect errors,
if they occur when the resident performs the activity on which the system is trained.

One-class classification is a core component of DERT. In the following section,
we offer a background description of one-class classification before diving into the
details of DERT.
5.3 One-Class Classification: A Background

5.3.1 Introduction

The idea of detecting abnormalities in data exists from the early days of data analysis. Researchers have always been fascinated with data samples that are not normal. This is because normal cases are routine and abnormal cases open up questions that drive ongoing research. In fact, data abnormality is found in a variety of application domains, including very large astronomical data, banking, credit card use, diagnosis of diseases and fault detection in engines and devices. In practice, the normal case usually has a good representation, however, the abnormal cases are rare and ill-defined. In such cases the abnormality of the data needs to be judged from the normal class only, although one must also be aware that abnormalities may be introduced due to the presence of noise in the data.

The presence of abnormalities in data has been considered and applied under many research themes, such as outlier/anomaly detection [Ritter and Gallegos, 1997], novelty detection [Bishop, 1994], concept learning [Japkowicz, 1999], and most commonly as one-class classification, a term first coined by Moya et. al. [Moya et al., 1993]. However, this unique situation where the outlier (henceforth used synonymously with positive) class is either absent, poorly sampled or not well defined in the training data, constrains the learning of efficient classifiers by defining class
boundaries just with the knowledge of the target (henceforth used synonymously with negative\textsuperscript{5}) class. Table 5.1 summarizes the characteristic differences between datasets with and without outliers. These characteristics can be used by researchers to make a choice between one-class classification and supervised learning on a given dataset.

<table>
<thead>
<tr>
<th>One-Class Classification</th>
<th>Supervised Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very small number of positive examples (sometimes even 0-20) and large number of negative examples.</td>
<td>Large number of both positive and negative examples.</td>
</tr>
<tr>
<td>Highly diverse outlier “types”. Hard for any algorithm to learn common characteristics of outliers from the positive examples. Future outliers may look nothing like any of the outliers seen before.</td>
<td>Enough positive examples for algorithms to get a sense of what the positive examples are like. Future positive examples are likely to be similar to the ones in the training set.</td>
</tr>
</tbody>
</table>

\textbf{Table 5.1:} One-class classification vs supervised learning.

In one-class classification, we have a target class containing training examples

\textsuperscript{5}The target class is sometimes referred to positive class and outlier class as negative in the literature.
that represent normal data. Each training example \( x \) is characterized in a multivariate way and is given a multivariate data vector \( \{ x_1, x_2, ..., x_n \} \). The examples constitute a cluster of multivariate data samples in the data space. The objective is to establish some kind of bound for the normal examples that describe the target class. The bound on the target class can be defined in terms of the following parameters:

- distance \( d(x) \) or probability \( p(x) \) of an example \( x \) from the target class represented by the training set \( X_{train} \)

- threshold \( \theta_d \) or \( \theta_p \) applied to the distance or probability, respectively

New examples are accepted when the distance to the target class is smaller than the threshold \( \theta_d \):

\[
    f(x) = I(d(x) < \theta_d)
\]

or when the probability is larger than the threshold \( \theta_p \):

\[
    f(x) = I(p(x) > \theta_p)
\]

where \( I(.) \) is the indicator function. Various one-class classification techniques differ in their definition of \( d(x) \) or \( p(x) \), in their optimization and in their thresholds with respect to the training set \( X_{train} \).

The most important feature of a one-class classifier is the trade-off between that fraction of the target class that is accepted, and the fraction of the outlier that
is rejected. The fraction of examples belonging to the target class and classified as outliers, is called the error of the first kind, $\mathcal{E}_I$, represented as false positive rate in this chapter. This error can be evaluated using an independent test set sampled from the same target class.

5.3.2 One-Class vs Binary Classification

The problems that are encountered in conventional classification, such as the curse of dimensionality, measuring the complexity of the solution, generalization of the method and definition of the error, also appear in one-class classification. Moreover, some problems become more prominent in one-class classification. In binary classification the decision boundary is supported from both sides due to the presence of training examples from both the classes. Most conventional classifiers assume a fairly balanced class distribution and do not work well when one class is severely under-sampled or even completely absent. Because in one-class classification only training examples from one class are available, only one side of the boundary can be determined. It is hard to decide on the basis of just one class how tightly the boundary should fit in each of the dimensions around the data. It is even harder to decide which features should be used to find the best separation between the target and the outlier classes [Jeong et al., 2012].
The computation of true error in conventional classification requires the knowledge of the probability distribution for both of the classes. In case of one-class classification only the probability distribution of the target class is known. This means that from the available data only the number of false positives or error of the first type, $E_I$, can be minimized. Without examples of outliers, or an estimate of the outlier class distribution, it is not possible to estimate the number of outlier examples that will be accepted by the one-class learner. This is known as the error of the second kind, $E_{II}$, or the false negative rate. Moreover, in one-class classification, a boundary should be defined in all dimensions around the data. In particular, when the boundary of the data is long and non-convex, the required number of training examples might be very high [Tax, 2001]. So it is expected that one-class classification will require a larger sample size in comparison with conventional classification [Yu, 2005].

5.3.3 Categories of Algorithms in One-Class Classification

Outlier detection has been a core component of data analysis for quite some time. An extensive survey on outlier detection in the context of one-class classification and novelty detection is done by Markou and Singh [Markou and Singh, 2003b,a]. Patcha and Park [Patcha and Park, 2007] provide a more recent survey on the trends in anomaly detection. Khan and Madden [Khan and Madden, 2010] proposed a
taxonomy for the study of one-class classification problems. The taxonomy is broadly divided into the following three categories:

(i) **Availability of Training Data**: Learning with target class data only or learning with target and unlabeled data and/or some amount of outlier training examples.

(ii) **Algorithm Used**: Algorithms based on One-Class Support Vector Machines (OC-SVMs) or methodologies based on algorithms other than OC-SVMs.

(iii) **Application Domain**: One-class classification applied in the field of text classification or in other application domains.

We are addressing a problem in the application domain of pervasive computing with data only from the target class. In the following, we provide a brief review of algorithms used in one-class classification.

**One-Class Support Vector Machines**

Tax and Duin [Tax and Duin, 2004] solve the problem of one-class classification by distinguishing the target class from all other possible patterns in the pattern space. Instead of using a hyper-plane to distinguish between two classes, this approach fits a hyper-sphere around the target class data that encompasses almost all points in the dataset with minimum radius. This method is called the Support Vector Data Description (SVDD). Training this model on a dataset has the risk of rejecting some
fraction of the target class examples when the algorithm sufficiently decreases the volume of the hyper-sphere. However, the hypersphere model can be made more flexible by introducing kernel functions. In his PhD thesis [Tax, 2001], Tax considered a polynomial and a Gaussian kernel and found that the Gaussian kernel works best for most datasets. A limitation of this approach is that it requires a very large sample size and becomes inefficient in high dimensional feature spaces. Also, problems may arise when large differences in densities in the target class exists. Examples in low-density areas will be rejected although they are legitimate target class examples. Tax and his group developed and still maintain a MATLAB toolbox [Tax, 2013] devoted to data domain description and outlier detection.

Schölkopf et al. [Schölkopf et al., 2001] proposed an alternative approach to using SVMs for one-class classification using a separating hyper-plane. Instead of trying to find a hyper-sphere with minimum radius to fit the data, Schölkopf’s approach tries to separate the surface region containing the data from the region that contains no data. This is accomplished by constructing a hyper-plane which is maximally distant from the origin, with all the data points lying on the opposite side from the origin and such that the margin is positive. The proposed algorithm computes a binary function that returns $+1$ in small regions (subspaces) that contain data and $-1$ elsewhere. The data is mapped onto the feature space corresponding to the kernel and is separated from the origin with maximum margin. The efficacy of this approach
is evaluated on USPS handwritten digits dataset.

Manevitz and Yousef [Manevitz and Yousef, 2002] proposed a different flavor of one-class SVM based on identifying outlier data as representative of the second class. The basic idea behind this approach is to first work in the feature space and assume that not only the origin is in the second class, but also the data points that are “close enough” to the origin are to be considered as noise or outliers. The outliers are identified by counting the features of an example with non-zero value; and if this is less than a threshold, then the feature is labeled as an outlier example. The threshold is determined by experimenting with different global values on a validation set. After having determined the threshold, the approach then continues with the the two-class SVM. Thus, one has to choose how far from the origin a point can be before being classified as an outlier.

Non-One-Class Support Vector Machines

The category of one-class classification algorithms that are not based on SVMs is quite broad and includes methods such as density estimation, one-class ensemble and neural-network.

Density Estimation: The most straight forward method to obtain a one-class classifier is to estimate the density of the target class and set a threshold on this density. Several distributions can be assumed, such as Gaussian, mixture of Gaussians, Poisson or a Parzen density [Parzen, 1962]. Numerous tests, called dicordancy tests
[Barnett and Lewis, 1994] are available to test new objects. When the sample size is sufficiently high and a flexible density model (such as a Parzen density estimation) is used, density-based approaches work really well for one-class classification. However, they require a large number of training examples to overcome the curse of dimensionality. Finding the right model to describe the target class distribution and the sample size could also prove to be a difficult decision to make.

**Neural Networks:** Skabar [Skabar, 2003] describes how to learn a classifier based on a feed-forward neural network using target class examples and a corpus of unlabeled data containing both target and outlier class examples. Since unlabeled data can contain some unlabeled target class examples, the output of the trained neural network may be less than or equal to the actual probability that an example belongs to the target class. If it is assumed that the labeled target examples adequately represent the target concept, it can be hypothesized that the neural network will be able to draw a class boundary between the outlier and the target examples.

Manevitz and Yousef [Manevitz and Yousef, 2000] show that a simple neural network can be trained to filter documents when data from only target class is available. They design a bottleneck filter that uses a basic feed-forward neural network that can incorporate the restriction of availability of only target examples. A three-level network is chosen with $m$ input neurons, $m$ output neurons and $k$ hidden neurons, where $k < m$. The network is trained using a standard back-propagation algorithm to
learn the identity function on the target class. The idea is that while the bottleneck prevents learning the full identity function on \( m \)-space, the identity on the small set of examples is in fact learnable. The set of vectors for which the network acts as the identity function is more like a sub-space which is similar to the trained set. For testing a given vector, it is shown to the network and if the result is the identity, the vector is deemed interesting (i.e. target class) otherwise it is deemed an outlier.

**Nearest Neighbors:** Tax [Tax, 2001] proposes a one-class nearest neighbor method, named Nearest Neighbor Description (NN-d), where a test data sample is accepted as a member of the target class provided that its local density is greater than or equal to the local density of its nearest neighbor in the training set. The first nearest neighbor is used for the local density estimation. This algorithm can be tuned to various parameters. For example, different numbers of nearest neighbors can be considered. However, increasing the number of neighbors will decrease the local sensitivity of the method, but it will make the method less sensitive to noise.

Datta [Datta, 1997] modifies the standard nearest neighbor algorithm to make it appropriate for learning a single class. The algorithm takes examples from only one class as input and learns a constant \( \delta \). This \( \delta \) is the maximum distance a test example can be from any learned example and still be considered a member of the target class. Any test sample that has a distance greater than \( \delta \) from any training example will not be considered a member of the target class.
Cabral et al. [Cabral et al., 2007] proposes a one-class nearest neighbor data description using the concept of structural risk minimization. k-Nearest Neighbour (kNN) algorithms suffer from the limitation of having to store all the training samples as prototypes that would be used to classify an unseen sample. The approach is based on the idea of removing redundant samples from the training set, thereby obtaining a compact representation which in turn improves the generalization performance of the classifier. The results on artificial and UCI datasets show improved performance over the NN-d classifiers and also achieve considerable reduction in number of stored prototypes.

**One-Class Classifier Ensemble:** Classifiers are commonly ensembled to provide a combined decision by averaging the estimated posterior probability. However, one-class classifiers cannot directly provide posterior probability for target class data points as the distribution of the outlier class is unknown. Thus, in most cases the posterior probability is estimated by assuming that the outliers are uniformly distributed. On doing so, the same types of combination rules found in conventional classification can be used.

Tax [Tax, 2001] shows how to use a combination of classifiers trained on different feature sets. In the experiments, it was found that the product combination rule gives the best results, while the mean combination rule suffers from the fact that the area covered by the target class tends to be overestimated. Juszczak and Duin [Juszczak
and Duin, 2004] extend the one-class classifier problem to classify missing data. The approach forms an ensemble of one-class classifiers trained on each feature or each pre-selected group of features. The ensemble was able to predict missing feature values based on the remaining classifiers.

Shieh and Kamm [Shieh and Kamm, 2009] propose an ensemble method using bagging. This method uses kernel density estimation to assign weights to the training examples, such that the outliers get the least weights and target class members get the higher weights for creating bootstrap samples. This approach achieves higher true positive rate and is also useful in applications where low false positive rates are required, such as disease diagnosis.

Among all the one-class classification techniques proposed to date, Schölkopf’s one-class support vector machine [Schölkopf et al., 2001] yields best results under appropriate estimation of parameters. Therefore, this technique has become a widely-accepted approach for one class classification [Manevitz and Yousef, 2002]. We choose one-class support vector machine as the primary building block of DERT.

5.4 Detecting Activity Errors in Real-Time (DERT)

We formulate the problem of detecting activity errors in real-time as an outlier detection problem. The primary motivation behind using an outlier detection-based
approach is that the probability distribution of activity errors is unknown. There can be a variety of ways in which errors in daily activities occur. Modeling each separate class of errors is impractical.

For a specific activity, the sensor data from participants for whom no errors were reported by the psychology experimenters, are used to train the target class. Sensor data for the same activity from participants who made errors, is used as the test data to evaluate the efficacy of the proposed approach in accurately detecting activity errors. Figure 5.2 illustrates the way the data from our participant pool are used in our experiments. While $n$ denotes the total number of participants who did not commit any errors for a particular activity, $m$ denotes the number of participants for whom errors were reported by the experimenters for that activity.

The distribution of normal and erroneous activity data among the participant pool of our studies is given in Figure 5.3. In Study 1, more than 80% of the total number of participants performed the activities correctly. However, only 35% of the participants performed Cooking correctly. In Study 2, an average of 44% of the participants committed errors in the activities.

One of the primary objectives of our approach is to design error detection algorithms that work in real-time on streaming sensor data. When a participant performs an activity, the algorithms should label every sensor event occurring in the smart home either as “normal” or “error”. Therefore, a data sample in our methodology
corresponds to a single sensor event and is represented by a corresponding feature vector. We extract temporal and contextual features, described in detail in Section 5.4.1, from the sensor events.

All data samples belonging to the target class are labeled as “normal”. However, the test samples obtained from the sensor data of the participants who committed errors, have both “normal” and “error” labels. Specifically, a test sample correspond-
Figure 5.3: Distribution of normal and erroneous activity data in Studies 1 and 2.

...ing to a sensor event that is labeled as the beginning of an error state, is labeled as “error”. In Table 5.2 we report the distribution of data samples from the target and the outlier classes during the training and test phases.

In the following sections, we provide details of our proposed feature extraction and error detection algorithms. Our error detection methods start with a simple outlier detection algorithm and then evolve into more complex algorithms by adding additional features and creating ensembles.

5.4.1 Feature Extraction

We extract statistical features from sensor events that capture contextual information while an activity is performed. The features can be broadly classified into longitudinal and cross-sectional features. Longitudinal features are based on whether
<table>
<thead>
<tr>
<th>Activity</th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>target</td>
<td>outlier</td>
</tr>
<tr>
<td>Sweeping and Dusting</td>
<td>7887</td>
<td>0</td>
</tr>
<tr>
<td>Taking Medication</td>
<td>2363</td>
<td>0</td>
</tr>
<tr>
<td>Watering Plants</td>
<td>5757</td>
<td>0</td>
</tr>
<tr>
<td>Cooking</td>
<td>2261</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Study 1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sweeping and Dusting</td>
<td>10038</td>
<td>0</td>
</tr>
<tr>
<td>Cleaning Kitchen Countertops</td>
<td>3438</td>
<td>0</td>
</tr>
<tr>
<td>Taking Medication</td>
<td>4716</td>
<td>0</td>
</tr>
<tr>
<td>Watering Plants</td>
<td>5668</td>
<td>0</td>
</tr>
<tr>
<td>Washing Hands</td>
<td>1268</td>
<td>0</td>
</tr>
<tr>
<td>Cooking</td>
<td>1075</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Study 2</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sweeping and Dusting</td>
<td>10038</td>
<td>0</td>
</tr>
<tr>
<td>Cleaning Kitchen Countertops</td>
<td>3438</td>
<td>0</td>
</tr>
<tr>
<td>Taking Medication</td>
<td>4716</td>
<td>0</td>
</tr>
<tr>
<td>Watering Plants</td>
<td>5668</td>
<td>0</td>
</tr>
<tr>
<td>Washing Hands</td>
<td>1268</td>
<td>0</td>
</tr>
<tr>
<td>Cooking</td>
<td>1075</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 5.2:** Distribution of data samples from Studies 1 and 2.

they are calculated from the sensor data of an individual participant collected from
the beginning of the activity. On the other hand, cross-sectional features are derived
from data across a set of participants.

As mentioned earlier, a feature vector, comprising the features listed below,
are generated for every sensor event of an activity. An event, $e_j$, where $1 < j < z$
and \( j \in \mathbb{Z}_{+} \), represents a time step from the beginning of an activity performed by a participant. That is, while \( e_1 \) corresponds to the first event of an activity, \( e_z \) corresponds to the last event.

**Longitudinal Features**

- **Sensor Identifier**: A nominal feature that uniquely identifies the sensor associated with the current sensor event in the smart home. This feature gives an idea the sensors that are commonly triggered for a particular activity, as indicated by the target class data instances.

- **Event Pause**: A temporal feature that represents the time elapsed (in milliseconds) between the previous and current sensor events.

- **Sensor Pause**: A temporal feature that represents the time elapsed (in milliseconds) since the last event that was generated by the same sensor and the current event.

- **Sensor Count**: The contextual information of a participant’s progress in an activity is obtained by keeping a count of the number of times each sensor generated an event, up through the current sensor event. This information is stored in the multi-dimensional feature *Sensor Count*, where every dimension corresponds to a specific sensor. Thus, for an event \( e_j \) in a particular activity, if \( l \) represents the total number of sensors associated with the activity, *Sensor
Count is given as follows:

\[ \text{Count}(s_i) = \text{Number of times sensor } s_i \text{ (where } i = 1..l) \]  

was triggered until the current event

Cross-sectional Features

- **Support**: Not all sensors in a smart home are triggered frequently for a particular activity. If a sensor, that appears infrequently in normal occurrences of an activity, appears when a participant performs that activity, it could possibly indicate an activity error. This property is captured by the feature support. Thus, for a specific activity, support is the ratio between the number of participants that triggered a specific sensor, and the total number of participants.

\[ \text{support}(s_i) = \frac{\text{Number of participants that triggered sensor } s_i}{\text{Total number of participants}} \]  

- **Probability of Event Number**: The length of time it takes an individual to complete an activity is an indication of whether the activity was completed in a timely manner or if the participant faced any difficulty that took extra time. As the number of sensor events associated with an activity is directly proportional to the time typically spent on the activity, we consider the fraction of participants reaching a certain event number to be a good feature that indicates an activity error.

\[ P(e_j) = \frac{\text{Number of participants who reached event } e_j}{\text{Total number of participants}} \]
• **Probability of Event Time:** This feature represents the probability associated with the time elapsed from the beginning of the activity until the current event. We assume that the amount of time participants spend from the beginning of an activity to a certain event number \( e_j \), represented as \( time_{e_j} \), is distributed normally across all the participants. Therefore, *Probability of Event Time* can be represented as follows:

\[
P(time_{e_j}; \mu_{e_j}, \sigma^2_{e_j}) = \frac{1}{\sigma_{e_j} \sqrt{2\pi}} exp \left( - \frac{(time_{e_j} - \mu_{e_j})^2}{2\sigma^2_{e_j}} \right)
\]  \hspace{1cm} (5.6)

• **Poisson Probability of Sensor Count:** The longitudinal feature *Sensor Count* captures the count of events for all sensors through the current event. However, as the training data representing the target class samples consists of feature vectors for all sensor events of the participants, there is a very high variance in the Sensor Count feature values. This is because the feature vectors representing sensor events at the beginning of an activity with low sensor event count are put together in the same target class with feature vectors corresponding to sensor events towards the end of the activity that have high sensor event count.

Therefore, we need additional features that take the event number into consideration, as the event number indicates the progress in the activity. As the sensor counts are discrete, we assume that the counts of events of the sensors
until a certain event across all the participants follows a Poisson distribution.

**Poisson Probability of Sensor Count** represents the probabilities of the counts of events for all sensors for a specific event number calculated using the Poisson probability mass function.

\[
P(k_{s_i,e_j}; \lambda_{s_i,e_j}) = \frac{(\lambda_{s_i,e_j})^{k_{s_i,e_j}}}{(k_{s_i,e_j})!} \exp(-\lambda_{s_i,e_j})
\]  

(5.7)

where, \(k_{s_i,e_j}\) is the count for sensor \(s_i\) at event \(e_j\), \(\lambda_{s_i,e_j}\) is the mean of counts of events for sensor \(s_i\) at event \(e_j\) for all participants, and \(i\) belongs to the set of all relevant sensors for an activity.

5.4.2 Baseline

DERT detects activity errors on streaming sensor data where a data sample at time \(t\) has a strong correlation with the sample at time \((t - 1)\). As we take an outlier detection based approach to predicting errors, the streaming aspect of the sensor data should be taken into consideration. No standard outlier detection algorithm fits this criterion. Therefore, we design a simple error detection algorithm that we refer to as **Baseline**, to provide a basis for comparison with alternative approaches.

Baseline uses only the **Sensor Counts** features corresponding to sensor events, discussed in the previous section, to detect activity errors. While training the target class, using the activity data of the participants who did not commit any errors, the
data samples corresponding to an event, $e_j$, are assumed to follow a multivariate Poisson distribution across all participants. We estimate the Poisson distribution parameter, $\lambda$, corresponding to all $l$ sensors of an activity, represented as $\lambda_{e_j} = \{\lambda_1, ..., \lambda_l\}$. A test sample, $x$, obtained from the activity data of those participants who committed errors, is evaluated on the basis of the probability derived from the Poisson probability mass function:

$$P(x_i; \lambda_i) = \frac{\lambda_i^{x_i}}{(x_i)!} exp(-\lambda_i), \text{ where } 1 < i < l \quad (5.8)$$

If the product of all probabilities, $\prod_{i=1}^{l} P(x_i; \lambda_i)$ is less than a certain threshold $\tau$, then the test sample $x$ is classified as an error; otherwise, it belongs to the target class.

The threshold for outlier detection algorithms are usually determined empirically by optimizing a performance metric. In our case, we optimize $F_1$-score (defined in Section 5.4.5), which is a balance between recall and precision, on a validation set that consists of both target class and outlier class samples. The threshold value, $\tau$, corresponding to the maximum $F_1$-score is chosen to evaluate the performance of Baseline on the test samples. The Baseline approach is illustrated algorithmically in Figure 5.4.
Algorithm 1: Baseline

Train:

1: Assume that the sensor counts follow a Poisson distribution, i.e., \( x_i \sim \text{Pois}(\lambda_i) \) and are conditioned over event \( e_j \)

2: Estimate \( \lambda_i \) for \( i = 1 \) to \( l \) and all events \( e_j \)

Test:

3: for every data sample \( x = \{ x_1, ..., x_l \} \) do

4: \hspace{1em} for \( i = 1 \) to \( l \) do

5: \hspace{2em} Find the probability of a sensor count using the Poisson probability mass function \( P(x_i; \lambda_i) = \frac{\lambda_i^{x_i}}{(x_i)!} \exp(-\lambda_i) \)

6: \hspace{1em} end for

7: Find the log of the product of \( P(x_i; \lambda_i) \) for all \( i \):

\[
\log(P(x)) = \log \left( \prod_{i=1}^{l} P(x_i; \lambda_i) \right) = \log \left( \prod_{i=1}^{l} \frac{\lambda_i^{x_i}}{(x_i)!} \exp(-\lambda_i) \right) = \sum_{i=1}^{l} \frac{\lambda_i^{x_i}}{(x_i)!} \exp(-\lambda_i)
\]

8: if \( P(x) > \tau \) then

9: \hspace{1em} \( x \in \text{target class} \)

10: else

11: \hspace{1em} \( x \in \text{outlier class} \)

12: end if

13: end for

Figure 5.4: Baseline algorithm for outlier detection.
5.4.3 One-Class Support Vector Machine (OCSVM)

The Baseline approach considers only the counts of events of all sensors to build the outlier detection model. We hypothesize that incorporating additional temporal and contextual features described in Section 5.4.1 will help in predicting those activity errors that cannot be identified on the basis of sensor event counts alone. However, these features are usually represented in real or categorical values which are not suitable for Baseline. Therefore, we use a one-class support vector machine-based approach for our second method. The One-Class SVM (OCSVM) proposed by Schölkopf [Schölkopf et al., 2001] is the core component of this approach to predict activity errors. It is a special type of support vector machine where the training phase intends to find a hyper-plane that separates all target class samples from the origin with maximum margin. Given $n$ target class samples $\mathbf{x} = \{x_1, x_2, ..., x_n\}$, the objective is to separate the dataset from the origin by solving the following quadratic program:

$$\min_{w \in \mathbb{F}, \xi \in \mathbb{R}^n, \rho \in \mathbb{R}} \frac{1}{2} \| w \|^2 + \frac{1}{\nu n} \sum_i \xi_i - \rho$$

subject to $(w \cdot \Phi(x_i)) \geq \rho - \xi_i, \xi_i \geq 0$

(5.9)

Here, $\nu \in (0, 1]$ is a parameter which is an upper bound on the fraction of outliers and a lower bound on the fraction of support vectors (SV). More explanation on the role of $\nu$ is offered later. $\Phi : \mathcal{X} \rightarrow \mathbb{F}$ is a kernel map which transforms the
training samples to a high-dimensional feature space and \((w, \rho)\) represents the kernel’s weight vectors and offset parameterizing a hyperplane in the feature space. Non-zero slack variables \(\xi_i\) are penalized in the objective function. Therefore, if \(w\) and \(\rho\) solve this problem, then the decision function

\[
f(x) = \text{sgn}((w \cdot \Phi(x) - \rho)
\]

will be positive for most examples \(x_i\) contained in the training set. Because many non-linear problems may be linearly separable after proper transformations, kernel transformations \(\Phi(\cdot)\) are normally employed to transfer an input sample from one feature space to another. \(\Phi\) is a feature map \(\mathcal{X} \rightarrow F\) into a high-dimensional feature space \(F\) such that the inner product of the transformed samples in \(F\) can be computed by evaluating a simple kernel function

\[
k(x, y) = (\Phi(x) \cdot \Phi(y)),
\]

such as the Gaussian kernel

\[
k(x, y) = \exp \left(-\frac{||x - y||^2}{c}\right)
\]

For new sample \(x_t\), if \(f(x) > 0\), it is classified as a target class sample, otherwise it is regarded as an outlier.

According to Schölkopf’s proposition [Schölkopf et al., 2000], if the solution of Equation 5.9 satisfies \(\rho \neq 0\), the following statements hold true for the parameter \(\nu\):
(i) \( \nu \) is an upper bound on the fraction of outliers

(ii) \( \nu \) is a lower bound on the fraction of SVs

(iii) If the data \( \{x_1, ..., x_n\} \) is generated independently from a distribution \( P(x) \) which does not contain discrete components, and if the kernel is analytic and non-constant, with probability 1, asymptotically, \( \nu \) equals both the fraction of SVs and the fraction of outliers.

The parameter \( \nu \) plays an important role in determining what fraction of the test data, with both target class and outlier samples, will be classified as outliers. In our application domain, the fraction of outliers defines the false positive rate of OCSVM in predicting errors or prompt situations in an activity. A larger fraction of outlier would mean more frequent interventions, which limits the practicality of the system. Therefore, a balance between outlier and target sample prediction is desired. The fraction of outliers on the test data is also dependent on the kernel parameter \( \gamma = \frac{1}{c} \). In Section 5.5.1, we perform experiments on all activities separately to choose appropriate values for \( \nu \) and \( \gamma \).

5.4.4 Activity Error Classification and Ensembles

As mentioned in Section 5.1, currently, there is a limited understanding of the course of functional change that occurs between normal aging and dementia [Tam
et al., 2007]. To better characterize the nature of this change, it is important to evaluate the error types. Schmitter-Edgecombe coded [Schmitter-Edgecombe, 2014] the activity errors for Studies 1 and 2 and categorized them into four and nine different types, respectively. For instance, error type *Substitution* is coded when an alternate object, or a correct object but an incorrect gesture, is used and disrupts accurate completion of the activity. Dusting the kitchen instead of the living room is an example of a *Substitution* error for the *Sweeping and Dusting* activity. Descriptions of the error types are given in Appendix C.

We use sensor data obtained from the smart home to classify the errors into different error types. The distribution of error data samples among all the error types is given in Table 5.3. A selection of different classifiers, such as decision tree, naive Bayes, nearest neighbors, logistic regression and support vector machine are used in our experiments. The results are described in Section 5.5.3.

We also explore if the knowledge of error types can be harnessed to boost the performance of OCSVM in accurately detecting activity errors. Thus, knowledge of errors and error types in addition to the normal data samples, is used to learn an ensemble that consists of OCSVM and the error classification model. We use two different methodologies to build this error classification model. The first method considers all error samples from different activities to belong to a single class and a one-class SVM is trained on this error class. The second method considers the error
<table>
<thead>
<tr>
<th>Study 1</th>
<th>Study 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Type</td>
<td># samples</td>
</tr>
<tr>
<td>Omission</td>
<td>71</td>
</tr>
<tr>
<td>Substitution</td>
<td>7</td>
</tr>
<tr>
<td>Irrelevant action</td>
<td>9</td>
</tr>
<tr>
<td>Inefficient action</td>
<td>125</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* task related activity, # non-task related activity

**Table 5.3:** Distribution of error data samples on various activity error types.

samples to have error-type class labels and a multi-class classifier is trained on all the error samples. Thus, we come up with two ensemble techniques described below.

**OCSVM + OCEM**

This technique is an ensemble of OCSVM trained only on normal activity data and a One-Class Error Model (OCEM), which is a second OCSVM trained on only
error data samples. The model selection for OCEM is done in the same way as OCSVM. That is, the $\nu$ and $\gamma$ parameters are determined empirically.

**OCSVM + MCEM**

This is an ensemble of OCSVM trained on normal activity data and a classifier trained on a Multi-Class Error Model (MCEM). Conventional multi-class classifiers output a distribution of membership probability of a test sample on all the classes on which the classifier is trained. We use this property of the classifiers to devise a technique that determines if a test sample belongs to any of the error classes, or if it is “out-of-vocabulary”, that is, if the test sample is actually a normal data sample. This is done by assigning a threshold value on the membership probabilities. Specifically, if the membership probability of the test sample on any of the error classes is greater than the threshold, then the test sample is considered to belong to that class; else, the test sample is not an error. The threshold value of the membership probability is determined empirically from a randomly selected validation set. $F_1$-scores obtained by the ensemble is plotted against the threshold values between 0 and 1. The threshold value that gives the highest $F_1$-score of OCSVM+MCEM is chosen as the final threshold value.

The final class label of a test sample is decided as a logical AND between the OCSVM and either of the classifiers for the error model. A pictorial representation of the ensemble is given in Figure 5.5.
Figure 5.5: Ensemble of OCSVM and error classification modeled as one-class and multi-class.

5.4.5 Performance Measures for Evaluation

The performance of the proposed approach is evaluated in two ways. First, we use the performance measures commonly used for one class classification to evaluate the efficiency of the algorithm from the perspective of classification. Secondly, as this approach is intended to work on streaming sensor data in real-time, we need an idea of the temporal accuracy. That is, how far (in time) before or beyond an actual activity error, did the algorithm detect the error.
Like conventional binary classification, one class classifiers can be evaluated on the basis of a confusion matrix given in Table 5.4. The following metrics are used to evaluate the performance of the proposed approach from the perspective of a classifier.

<table>
<thead>
<tr>
<th>Actual</th>
<th></th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>outlier</td>
<td>True Positive (TP)</td>
<td>False Positive (FP)</td>
</tr>
<tr>
<td>target</td>
<td>False Negative (FN)</td>
<td>True Negative (TN)</td>
</tr>
</tbody>
</table>

Table 5.4: Confusion matrix for one-class classification.

• **Recall**: Recall represents the performance of the algorithm on the outlier data samples. It is the fraction of actual outliers that were correctly predicted as outliers. Recall is calculated as follows:

\[
Recall = \frac{TP}{TP + FN} \tag{5.13}
\]

• **Precision**: Precision is the fraction of actual outliers among all predicted outliers. It is calculated as follows:

\[
Precision = \frac{TP}{TP + FP} \tag{5.14}
\]

• **F₁-score**: F₁-score is the harmonic mean of precision and recall. Therefore, it gives an overall idea of the performance of the classifier on both target and
outlier class data samples. \( F_1 \)-score is calculated as follows:

\[
F_1 \text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]  

(5.15)

- **Matthews Correlation Coefficient (MCC):** MCC [Matthews, 1975] is regarded as a balanced performance measure that can be used under extreme class imbalance. It is a correlation coefficient between the actual class labels and the predicted labels, which can have a value between \(-1\) and \(+1\). A coefficient of \(+1\) represents a perfect prediction, 0 no better than random prediction and \(-1\) indicates total disagreement between predicted and actual labels. It is calculated as follows:

\[
MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}
\]  

(5.16)

On the other hand, temporal accuracy determines how close to the actual error (and subsequent need for a prompt) in the sensor data the proposed approach detected errors. Typically there could be multiple error detections in the temporal vicinity of the actual error as shown in Figure 5.6. For every actual prompt, we consider temporal accuracy to be the minimum between (a) the time difference between the actual prompt and the beginning of a series of prompt predictions before that, and (b) the time difference between the actual prompt and the beginning of a series of prompt predictions after that. That is, in Figure 5.6, the minimum between \( t_{prev} \) and \( t_{next} \) is considered as the temporal accuracy of the proposed algorithm. If the
algorithm fails to identify an existing activity error for a particular participant and activity, temporal accuracy is considered as the maximum between: (a) the time difference between the beginning of the activity and the actual prompt, and (b) the time difference between the actual prompt and the end of the activity.

![Figure 5.6: Temporal accuracy.](image)

The temporal accuracy of the algorithm on all test participant data can be calculated by the root mean squared errors given below:

- **RMSE(seconds):** Root mean square error of prompt prediction in seconds:

\[
RMSE(\text{seconds}) = \sqrt{\frac{\sum_{t=1}^{n} (\text{Predicted clock time} - \text{Actual clock time})^2}{n}}
\]  

(5.17)

- **RMSE(events):** Root mean squared error of prompt prediction in number of events:

\[
RMSE(\text{events}) = \sqrt{\frac{\sum_{t=1}^{n} (\text{Predicted event#} - \text{Actual event#})^2}{n}}
\]  

(5.18)

Here, \( n \) represents the total number of test participants.
5.5 Results

5.5.1 Model Selection for OCSVM

As mentioned in Section 5.4.3, the sensitivity of OCSVM on the outlier class is dependent on the parameters $\nu$ and $\gamma$. While $\nu$ is an upper bound on the fraction of outliers $\gamma$ is the inverse of kernel width $c$. Traditionally, the appropriate values for $\nu$ and $\gamma$ are obtained by performing an empirical test on the training data with various values of $\nu$ and $\gamma$ and monitoring the fraction of SVs chosen by the OCSVM. The values for $\nu$ are usually varied from 1.0 to 0.01, as done in the experiments by Schölkopf et al. [Schölkopf et al., 2001]. The $\gamma$ values are varied exponentially and chosen from the set $\{2^5, 2^3, ..., 2^{-1}, 2^{-3}, ..., 2^{-59}\}$ as suggested by Hsu et al. [Hsu et al., 2003].

Figure 5.7 shows a surface plot of the fraction of SVs for different pairs of $\nu$ and $\gamma$ values for the Sweeping and Dusting activities from Study 1. It is quite evident from the figure that the fraction of SVs steadily decrease with decreasing values of $\nu$ and $\gamma$. However, after a particular point there is no further decrease in the fraction of SVs. This point in the $\nu$ and $\gamma$ plane, beyond which there is no significant decrease in the fraction of SVs, is chosen as the values of $\nu$ and $\gamma$ for the OCSVM model of Sweeping and Dusting activity. Similarly, model selection for other activities is done independently.
Table 5.5 lists the $\nu$ and $\gamma$ values for all activities. Our experiments for model selection show that the $\nu$ values for a majority of the activities is 0.3. On the other hand, the mean $\gamma$ values is $2^{-19}$. Therefore, as a general rule of thumb, 0.3 and $2^{-19}$ can be used as the values of $\nu$ and $\gamma$ for a one-class SVM-based activity error detection algorithm, if the feature set similar to that described in Section 5.4.1 are extracted from the smart home sensor data.

**Figure 5.7:** Surface plot of $\nu$ and $\gamma$ values for OCSVM of *Sweeping and Dusting* activity.
### Table 5.5: \( \nu \) and \( \gamma \) values for all activities.

<table>
<thead>
<tr>
<th>Activity</th>
<th>( \nu )</th>
<th>( \gamma )</th>
<th>Activity</th>
<th>( \nu )</th>
<th>( \gamma )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sweeping and Dusting</td>
<td>0.3</td>
<td>( 2^{-19} )</td>
<td>Sweeping and Dusting</td>
<td>0.3</td>
<td>( 2^{-17} )</td>
</tr>
<tr>
<td>Taking Medication</td>
<td>0.2</td>
<td>( 2^{-21} )</td>
<td>Cleaning Kitchen Countertops</td>
<td>0.3</td>
<td>( 2^{-17} )</td>
</tr>
<tr>
<td>Watering Plants</td>
<td>0.3</td>
<td>( 2^{-19} )</td>
<td>Taking Medication</td>
<td>0.3</td>
<td>( 2^{-15} )</td>
</tr>
<tr>
<td>Cooking</td>
<td>0.2</td>
<td>( 2^{-21} )</td>
<td>Watering Plants</td>
<td>0.3</td>
<td>( 2^{-18} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Washing Hands</td>
<td>0.3</td>
<td>( 2^{-17} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cooking</td>
<td>0.3</td>
<td>( 2^{-19} )</td>
</tr>
</tbody>
</table>

5.5.2 Baseline vs OCSVM

We first compare the performance of OCSVM trained on “normal” activity data with the performance of Baseline in correctly predicting the activity errors. Figure 5.8 compares the performance of Baseline and OCSVM in terms of recall and precision on datasets available from the two studies.

The results obtained from Study 1 indicate that OCSVM performs significantly better \((p < 0.05)\) than Baseline in terms of recall on the Sweeping and Dusting and Cooking activities. The recall of OCSVM for the Taking Medication activity is at par
Study 1

![Graph showing Recall and Precision values for all activities.]

**Figure 5.8:** Recall (left) and precision (right) values for all activities.

with Baseline. This means that the temporal features such as Event Pause and Sensor Pause do not play any role in predicting *Taking Medication* activity errors. However, these features do play an important role in reducing the number of false positives, and thus precision of OCSVM on *Taking Medication* is better than Baseline. In fact, in terms of precision, OCSVM performs significantly better ($p < 0.05$) than Baseline for all activities except *Cooking.*
In Study 2, OCSVM achieves statistically significant improvement over Baseline in terms of recall only for Washing Hands and Cleaning Kitchen Countertops activities. For the rest of the activities, the performance of OCSVM is almost at par with Baseline. The precision of OCSVM achieves significant improvement over Baseline for all classes except Sweeping and Dusting and Cooking.

From these observations we can conclude that, although Baseline does a fairly good job of predicting activity errors just on the basis of Poisson probability values for various sensors, low precision on all activities implies that Baseline gives a high false positive rate. That is, a significant number of “normal” samples are predicted as outliers by the Baseline algorithm. The high false positive rate in our application domain implies a higher rate of intervention for the smart home residents, which might be unnecessary. The additional set of features that include temporal features and importance of a specific sensor in an activity (represented by support), plays a very important role in reducing the number of false positives. Also, for some of the activities, these features help in predicting the activity errors better than Baseline.

In spite of the better performance of OCSVM over Baseline in terms of precision, the precision values are still quite low. This can be attributed to the extreme imbalance of class distribution between normal and outlier data samples. For example, only 0.67% of all test samples in the Sweeping and Dusting task from Study 2 are actual outliers. (Table 5.2 reports the distribution of target and outlier samples
during training and test.) Therefore, the number of false positives is much higher than the number of true positives. As precision is calculated as \( \frac{TP}{TP + FP} \), a high value of FP causes the precision to be low.

We also evaluate the proposed approaches in terms of temporal accuracy. This represents how close in time the OCSVM error prediction is to the actual error occurrence. Temporal accuracies, represented in RMSE values in seconds and number of sensor events, obtained by OCSVM and Baseline are compared in Figure 5.9. The results show that most of the errors predicted by OCSVM are in the window of 4 sensor events before or after the actual error. In terms of time, this is \(< 20\) seconds before or after the actual error occurrence for most of the activities. In addition, in our experiments we found that in many cases, our approach predicted errors before they actually occurred. This implies that the features extracted from the sensor data are capable of tracking a smart home resident’s deviation from the “common” activity pattern in real time.

5.5.3 Error Classification

As mentioned earlier, successful classification of activity errors into predefined error types can provide better insight into the changes dementia brings into one’s daily life. Therefore, we first treat classification of errors as an independent problem
Figure 5.9: RMSE values for all activities in seconds (left) and number of events (right).

to see how well these errors can be classified based on the sensor data obtained from
the smart home when the everyday activities were performed by the participants. We
perform a 5-fold cross validation of five commonly used classifiers on error samples
labeled with error type classes. The classifiers that are used in this experiment are
as follows: C4.5 decision tree, naive Bayes, k-nearest neighbor, logistic regression
and SMO support vector machine. Tables 5.6 and 5.7 report the performance of the
classifiers on all error types separately.

We find that most of the classifiers produce fairly good results on the Study 1 dataset. In spite of the fact that Substitution and Irrelevant Action error types have very few samples which results in an imbalanced class distribution, the weighted average $F_1$-score is more than 0.73 for C4.5, kNN and SMO classifiers.

Study 2 dataset has nine different error types which makes the classification problem harder. Out of the nine classes, Help, Wandering, Microslip and Error Detection are in the minority. Therefore, most of the classifiers do not learn anything from the training samples of these classes. Thus, the highest $F_1$-score is 0.53 and is achieved by kNN.

It should be noted that Microslip is not a critical activity error and can be ignored from the error classes. It is coded when a participant initiates and terminates an incorrect action before the error is completed. For example, in the Cooking activity, when the participant begins to make action toward the sink as if she is going to fill water with faucet water but then corrects behavior and obtains the pitcher from the refrigerator. The total number of training examples available for the Microslip class is only 14. As this class is highly under-represented in the dataset, the classifiers do not learn any common pattern. Thus, C4.5 decision tree yields 0% true positive rate, 0.011% false positive rate, 100% false negative rate, and 98.94% true negative rate for the Microslip class.
<table>
<thead>
<tr>
<th>Error Type</th>
<th>C4.5</th>
<th>Naive Bayes</th>
<th>kNN</th>
<th>Logistic Regression</th>
<th>SMO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precision</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Omission</td>
<td>0.671</td>
<td>0.667</td>
<td>0.658</td>
<td>0.539</td>
<td>0.781</td>
</tr>
<tr>
<td>Substitution</td>
<td>0.600</td>
<td>0.107</td>
<td>0.250</td>
<td>0.091</td>
<td>0.500</td>
</tr>
<tr>
<td>Irrelevant Action</td>
<td>0.600</td>
<td>0.119</td>
<td>0.800</td>
<td>0.500</td>
<td>0.667</td>
</tr>
<tr>
<td>Inefficient Action</td>
<td>0.790</td>
<td>0.766</td>
<td>0.795</td>
<td>0.765</td>
<td>0.815</td>
</tr>
<tr>
<td><strong>Weighted Avg.</strong></td>
<td>0.736</td>
<td>0.684</td>
<td>0.731</td>
<td>0.656</td>
<td>0.787</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Omission</td>
<td>0.690</td>
<td>0.169</td>
<td>0.704</td>
<td>0.577</td>
<td>0.704</td>
</tr>
<tr>
<td>Substitution</td>
<td>0.429</td>
<td>0.429</td>
<td>0.143</td>
<td>0.143</td>
<td>0.286</td>
</tr>
<tr>
<td>Irrelevant Action</td>
<td>0.667</td>
<td>0.556</td>
<td>0.444</td>
<td>0.556</td>
<td>0.667</td>
</tr>
<tr>
<td>Inefficient Action</td>
<td>0.784</td>
<td>0.760</td>
<td>0.808</td>
<td>0.704</td>
<td>0.880</td>
</tr>
<tr>
<td><strong>Weighted Avg.</strong></td>
<td>0.736</td>
<td>0.542</td>
<td>0.736</td>
<td>0.637</td>
<td>0.792</td>
</tr>
<tr>
<td><strong>F1-score</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Omission</td>
<td>0.681</td>
<td>0.270</td>
<td>0.680</td>
<td>0.558</td>
<td>0.741</td>
</tr>
<tr>
<td>Substitution</td>
<td>0.500</td>
<td>0.171</td>
<td>0.182</td>
<td>0.111</td>
<td>0.364</td>
</tr>
<tr>
<td>Irrelevant Action</td>
<td>0.632</td>
<td>0.196</td>
<td>0.571</td>
<td>0.526</td>
<td>0.667</td>
</tr>
<tr>
<td>Inefficient Action</td>
<td>0.787</td>
<td>0.763</td>
<td>0.802</td>
<td>0.733</td>
<td>0.846</td>
</tr>
<tr>
<td><strong>Weighted Avg.</strong></td>
<td>0.735</td>
<td>0.554</td>
<td>0.731</td>
<td>0.645</td>
<td>0.787</td>
</tr>
</tbody>
</table>

Table 5.6: Error classification results for Study 1 dataset.
<table>
<thead>
<tr>
<th>Error Type</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C4.5</td>
<td>Naive Bayes</td>
<td>kNN</td>
</tr>
<tr>
<td>Substitution</td>
<td>0.537</td>
<td>0.643</td>
<td>0.455</td>
</tr>
<tr>
<td>Additional TRA</td>
<td>0.532</td>
<td>0.548</td>
<td>0.518</td>
</tr>
<tr>
<td>Additional NTRA</td>
<td>0.491</td>
<td>0.548</td>
<td>0.632</td>
</tr>
<tr>
<td>Perseveration</td>
<td>0.455</td>
<td>0.026</td>
<td>0.000</td>
</tr>
<tr>
<td>Searching</td>
<td>0.426</td>
<td>0.308</td>
<td>0.597</td>
</tr>
<tr>
<td>Help</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Wandering</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Microslip</td>
<td>0.000</td>
<td>0.028</td>
<td>0.000</td>
</tr>
<tr>
<td>Error Detection</td>
<td>0.000</td>
<td>0.120</td>
<td>0.000</td>
</tr>
<tr>
<td><strong>Weighted Avg.</strong></td>
<td><strong>0.505</strong></td>
<td><strong>0.423</strong></td>
<td><strong>0.506</strong></td>
</tr>
<tr>
<td>Substitution</td>
<td>0.615</td>
<td>0.307</td>
<td>0.721</td>
</tr>
<tr>
<td>Additional TRA</td>
<td>0.706</td>
<td>0.107</td>
<td>0.738</td>
</tr>
<tr>
<td>Additional NTRA</td>
<td>0.333</td>
<td>0.067</td>
<td>0.000</td>
</tr>
<tr>
<td>Perseveration</td>
<td>0.105</td>
<td>0.421</td>
<td>0.000</td>
</tr>
<tr>
<td>Searching</td>
<td>0.418</td>
<td>0.445</td>
<td>0.418</td>
</tr>
<tr>
<td>Help</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Wandering</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Microslip</td>
<td>0.000</td>
<td>0.286</td>
<td>0.000</td>
</tr>
<tr>
<td>Error Detection</td>
<td>0.000</td>
<td>0.136</td>
<td>0.000</td>
</tr>
<tr>
<td><strong>Weighted Avg.</strong></td>
<td><strong>0.540</strong></td>
<td><strong>0.246</strong></td>
<td><strong>0.573</strong></td>
</tr>
<tr>
<td>Substitution</td>
<td>0.573</td>
<td>0.378</td>
<td>0.603</td>
</tr>
<tr>
<td>Additional TRA</td>
<td>0.673</td>
<td>0.180</td>
<td>0.681</td>
</tr>
<tr>
<td>Additional NTRA</td>
<td>0.385</td>
<td>0.038</td>
<td>0.000</td>
</tr>
<tr>
<td>Perseveration</td>
<td>0.160</td>
<td>0.242</td>
<td>0.000</td>
</tr>
<tr>
<td>Searching</td>
<td>0.422</td>
<td>0.364</td>
<td>0.492</td>
</tr>
<tr>
<td>Help</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Wandering</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Microslip</td>
<td>0.000</td>
<td>0.050</td>
<td>0.000</td>
</tr>
<tr>
<td>Error Detection</td>
<td>0.000</td>
<td>0.128</td>
<td>0.000</td>
</tr>
<tr>
<td><strong>Weighted Avg.</strong></td>
<td><strong>0.519</strong></td>
<td><strong>0.267</strong></td>
<td><strong>0.530</strong></td>
</tr>
</tbody>
</table>

Table 5.7: Error classification results for Study 2 dataset.
Among all classifiers, C4.5 seems to perform consistently well and is computationally less expensive than classifiers such as logistic regression or SMO. Therefore, we use a C4.5 decision tree as the base classifier for the multi-class error model in the ensemble OCSVM + MCEM. We report the performance of the ensembles in the following section.

5.5.4 Ensembles

In this section, we explore the knowledge of error types to boost the performance of OCSVM by building an ensemble of OCSVM and the error type classification model. The primary motivation of using an ensemble is to improve the precision of error prediction on all activities. As mentioned in Section 5.4.4, we use two ensembles that differ in the error classification model. The first ensemble, OCSVM+OCEM, is an ensemble of the primary OCSVM and a second OCSVM trained on all error samples. The second ensemble, OCSVM+MCEM, is an ensemble of the primary OCSVM and a C4.5 decision tree trained on multi-class error samples. A test sample is evaluated by both the primary OCSVM and the classifier designated for the error classification model. If both classifiers predict the test sample as an “error”, then the final outcome of the ensemble is “error” as well; otherwise, it is “normal”. This approach helps in restricting the number of false positives.
Figure 5.10 compares the performance of the ensembles with OCSVM and Baseline in terms of recall and precision. The ensembles achieve similar precision scores as that of OCSVM, apart from a couple of exceptions. OCSVM+MCEM and OCSVM+OCEM perform better ($p < 0.01$) than OCSVM for the *Taking Medication* and *Watering Plants* activities, respectively, in both datasets. However, there is no improvement in recall. In fact, the ensembles either perform worse or are at par with OCSVM. In some cases, such as *Taking Medication* in Study 1 and *Sweeping and Dusting* in Study 2, the ensembles yield lower recall than Baseline. The $F_1$-score and MCC for all the techniques are reported in Tables 5.8 and 5.9.

Between OCSVM+OCEM and OCSVM+MCEM, there is no clear winner. It appears that the performances are more activity dependent. For example, OCSVM+MCEM achieves higher precision than OCSVM+OCEM for *Taking Medication* in both datasets. However, the situation is just the opposite for *Watering Plants* where OCSVM+OCEM performs better than OCSVM+MCEM. These observations reinforce the fact that due to the differences in the ways daily activities are usually performed, there is no one-size-fits-all solution for all activities.

The temporal accuracy of the ensembles is not an improvement over OCSVM either. RMSE values in terms of seconds and number of events is given in Figure 5.11. In most of the cases, the RMSE of ensembles is at par with OCSVM and worse than OCSVM in some cases. However, the performance is definitely better than Baseline.
5.6 Discussion

5.6.1 High False Positive Rate

The results obtained by the proposed approaches validate our hypothesis that machine learning techniques, particularly, outlier detection algorithms, are capable of recognizing activity errors from sensor data in smart homes. We get an average of
60% recall in predicting activity errors. However, the average false positive rate of our approach on all activities is about 45%. We believe, the reason behind a fairly high false positive rate is rooted in the way activity errors were annotated in the sensor data. As mentioned in Section 5.2, annotations of activity errors only give an idea of when the errors start. However, the duration of the activity error varies vastly. For example, opening the wrong cabinet during Taking Medication is an error.
<table>
<thead>
<tr>
<th>Activity</th>
<th>Baseline</th>
<th>OCSVM + OCEM</th>
<th>OCSVM + MCEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sweeping and Dusting</td>
<td>0.014</td>
<td>0.016</td>
<td>0.015</td>
</tr>
<tr>
<td>Cleaning Countertops</td>
<td>0.033</td>
<td>0.046</td>
<td>0.051</td>
</tr>
<tr>
<td>Taking Medication</td>
<td>0.015</td>
<td>0.027</td>
<td>0.029</td>
</tr>
<tr>
<td>Watering Plants</td>
<td>0.044</td>
<td>0.064</td>
<td>0.094</td>
</tr>
<tr>
<td>Washing Hands</td>
<td>0.015</td>
<td>0.038</td>
<td>0.041</td>
</tr>
<tr>
<td>Cooking</td>
<td>0.056</td>
<td>0.06</td>
<td>0.056</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activity</th>
<th>MCC</th>
<th>MCC</th>
<th>MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sweeping and Dusting</td>
<td>0.015</td>
<td>0.017</td>
<td>0.008</td>
</tr>
<tr>
<td>Cleaning Countertops</td>
<td>0.01</td>
<td>0.025</td>
<td>0.034</td>
</tr>
<tr>
<td>Taking Medication</td>
<td>−0.015</td>
<td>−0.025</td>
<td>−0.013</td>
</tr>
<tr>
<td>Watering Plants</td>
<td>−0.029</td>
<td>0.01</td>
<td>0.072</td>
</tr>
<tr>
<td>Washing Hands</td>
<td>0.002</td>
<td>0.02</td>
<td>0.031</td>
</tr>
<tr>
<td>Cooking</td>
<td>0.0</td>
<td>0.01</td>
<td>0.001</td>
</tr>
</tbody>
</table>

**Table 5.9**: $F_1$-score and MCC for Study 2 dataset.

that corresponds to a single event of the door sensor attached to the wrong cabinet.

On the other hand, errors such as wandering in the apartment or failing to return
Figure 5.11: RMSE values of all methods in seconds (left) and number of events (right).

items used in Cooking, can last for quite a while. In our approaches, every sensor event is considered as a data sample, thus the events corresponding to an error after it has already started, are labeled as “normal”. However, in reality these sensor events are errors as well. Moreover, annotation of sensor data also depends on the perspective of the annotators. The inter-annotator consistency that we found for activity annotations was roughly 80%. Therefore, if we are able to overcome all of
these limitations, we can achieve a much lower false positive rate.

5.6.2 Clinical Evaluation of False Positives

In order to further analyze the false positives among the detected activity errors, we enlisted the help of a psychology clinician who is involved in the smart home studies conducted with older adults. We randomly chose five participants each for the \textit{Sweeping and Dusting} and \textit{Cooking} activities, for whom activity errors were reported by the psychology experimenters. All these participants were part of Study 1. The sensor data from these participants were evaluated by OCSVM described in Section 5.4.3. The false positives that were predicted on this dataset were ranked on the basis of $\nu$ and $\gamma$ values. The top 33\% of the false positives are used for the purpose of clinical evaluation. The purpose of this task was to better understand the possible reasons for DERT generating false positives in its error detection task.

The clinician watched the smart home video recordings for the chosen participants to analyze the false positives predicted by our algorithm. Out of a total of 45 false positives, 15 were found to be continuation of errors that had already occurred, but were not annotated. Note that only the beginning of errors in the activities were annotated in the sensor data. In 8 additional cases, the clinician thought that they should have been reported as errors by the psychology experimenters. Most of these
unreported errors were believed to be either irrelevant action or inefficient action type of errors. The remainder of the 22 false positives were part of regular activity steps.

From this clinical evaluation we can conclude that a precise annotation of the error start and end times, in the sensor data corresponding to an activity performed by a participant, is crucial for evaluating the performance of our proposed algorithms. This will help in getting rid of the false positives that correspond to the continuation of previously occurred errors. Moreover, an experimenter’s perspective in reporting errors also plays an important role in justifying the effectiveness of our algorithm. We find it interesting that some of the “false positive” cases were actually true positives that were not initially caught by the experimenters. This indicates that automated approaches to error detection could potentially improve upon human-only methods for detecting errors and prompting individuals with memory limitations to correctly complete critical daily activities.

5.6.3 Future Work

In our future work, we will improve the precision of our error annotation. Specifically, we want ground truth information for both error start and end times. Just like the inconsistency amongst annotators, it is also possible for the psychology experimenters to overlook some activity errors while conducting the studies.
Currently we use a standard set of extracted features for all activities. However, a well-selected set of features, according to the characteristics of a particular activity, might help in defining a tighter bound to the target class. Therefore, we will also explore feature selection techniques for the proposed one-class classification approach.

5.7 Summary

In this chapter, we use one-class classification-based algorithms to propose techniques for detecting errors in daily activities. Our activity error detection approach, DERT, automatically detects activity errors in real time, while an individual performs an activity. Successful detection of activity errors can help us identify potential prompt situations which could be used to provide help to older adults with their daily activity completion. Sensor-based daily activity data from two studies that were conducted on 580 human participants are used to train the one-class models for the activities. DERT learns only from the normal activity patterns of the participants and does not require any training examples for the activity errors. Activity data from participants who committed errors are used to evaluate the efficiency of DERT. Test samples that are classified as outliers, are potential prompt situations.

We also use smart home sensor data to classify various error types. This information is important for psychologists and gerontologists to characterize the nature of
functional changes among older adults due to dementia.

In the next chapter, we summarize the contributions in this dissertation and discuss future directions for research in automated prompting.
CHAPTER 6. CONCLUSION AND FUTURE WORK

In this dissertation, we propose the foundation of an automated prompting system that can aid older adults with their everyday activities in a smart home. We take a two-step methodology to validate our hypothesis that machine learning algorithms, trained on smart home sensor data, can predict when an individual faces difficulty while performing everyday activities. First, we emulate natural interventions, provided by a caregiver to individuals with memory impairments, by using a supervised machine learning approach to classify pre-segmented activity steps into prompt or no-prompt classes. Second, we automatically detect activity errors in real time, while an individual performs an activity, using no training data for the activity errors.

The lack of prompt-class training examples, and an ambiguity between prompt and no-prompt instances, raises two machine learning challenges: imbalanced class distribution and overlapping classes, in the first methodology. We propose two novel probabilistic oversampling techniques, RACOG and wRACOG, to address imbalanced class distribution. Although the proposed algorithms are primarily developed for the automated prompting problem, they are generalizable to datasets from other domains that exhibit class imbalance. Our claim is validated by the improvements achieved by RACOG and wRACOG over other state-of-the-art sampling techniques.
We address the overlapping classes problem by proposing ClusBUS, a clustering-based under-sampling technique that identifies data regions where minority class samples are embedded deep inside majority class. Experiments show that ClusBUS achieves improved performance over an existing sampling technique and helps in better learning of the prompt class.

Our second methodology is motivated by the fact that there can be a variety of ways in which errors in everyday activities occur. Modeling each separate class of errors is impractical. We propose a collection of one-class classification-based algorithms, known as DERT, that learns only from the normal activity patterns and without any training examples for the activity errors. When evaluated on unseen activity data, DERT achieves an average of 60% accuracy on correctly predicting activity errors in real time. In a clinical evaluation, some of the false positives predicted by DERT turned out to be true positives that were not caught by the psychology experimenters when the smart home studies were conducted.

The effectiveness of the algorithms in predicting potential prompt situations are validated on the sensor data of ten activities of daily living, collected from 580 participants, who were part of two smart home studies.
Future Research Directions

As mentioned in Section 3.7, there is still an open question to determine how RACOG and wRACOG compare with random oversampling and under-sampling. In the future, more extensive experimental setup should be tested, that includes a large number of datasets, with high dimensions and very high cardinality. This setup will raise some interesting research questions because random over and under-sampling usually have linear time complexity and are thus far less computationally expensive than other state-of-the-art sampling techniques. Computational complexity is more important now, than ever before, as most of the real-world application domains today deal with “big” data.

Our algorithm to handle class overlap in the presence of imbalanced classes, ClusBUS, works well for the prompting dataset. As part of future work, the generalizability of ClusBUS can be evaluated by applying it on datasets from other problem domains, including network intrusion detection and credit card fraud detection. Comparison of ClusBUS with other unified solutions that address class imbalance and overlap such as SMOTE+ENN and SMOTE+Tomek [Batista et al., 2004] can be performed.

While detecting activity errors in real time, the primary limitation of our DERT algorithms is their high false positive rates. A clinical evaluation of the algorithm-
predicted false positives, performed by a psychology clinician, shows that about 1/3 of
the false positives were actually continuations of previously occurred errors. There-
fore, in the future, the precision of our error annotations should be improved. In
addition, currently we use a standard set of extracted features for all activities. How-
ever, a well-selected set of features, according to the characteristics of a particular
activity, might help in defining a tighter bound to the target class. Therefore, feature
selection techniques for our proposed algorithms should also be explored.
APPENDIX

A Description of activities used in the smart home studies

• Sweeping and Dusting: Participants were instructed to retrieve a broom, dust-pan/brush, and duster from a specified closest. Then they were to sweep the kitchen floor and dust the furniture in the living and dining rooms.

• Taking Medication: Participants were instructed to fill a 7-day pill holder using three bottles of different medications. Written directions were provided for the weekly medication.

• Writing Birthday Card: Participants were instructed to retrieve a birthday card, envelope, and blank check. They were to fill out these items as if they were sending a monetary birthday gift to a close friend or relative.

• Watching DVD: Participants were instructed to retrieve a DVD containing a 5-minute video clip from “Good Morning America”. This task required operating the DVD player and watching the entirety of the news clip.

• Watering Plants: Participants were instructed to retrieve a watering can and water the three houseplants located in the living room and kitchen.

• Taking Phone Call: Participants were to answer a ringing telephone and answer
questions about the news clip from the DVD task.

- **Cooking**: Participants were instructed to retrieve a microwaveable cup of noodle soup and follow the microwave cooking directions on the packaging. In addition, they were to pour a glass of water and bring the soup and water glass to the dining room table.

- **Selecting Outfit**: Participants were instructed to retrieve an appropriate job interview outfit for a male friend. Items were to be chosen among distractor items in a closet and, when selected, laid out on the couch.

- **Cleaning Kitchen Countertops**: Participants were instructed to use dish washing soap and a sponge to wash kitchen countertops.

- **Washing Hands**: Participants were instructed to wash hands in the kitchen sink choosing correct soap and using towel to dry.

**B Attribute information for prompting dataset**

Information related to the data types and measurement units of the attributes that are used in the prompting dataset is given in Table 6.1.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Data Type</th>
<th>Measurement Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>stepLength</td>
<td>real</td>
<td>seconds</td>
</tr>
<tr>
<td>numSensors</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>numEvents</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>prevStep</td>
<td>nominal</td>
<td>*</td>
</tr>
<tr>
<td>nextStep</td>
<td>nominal</td>
<td>*</td>
</tr>
<tr>
<td>timeActBegin</td>
<td>real</td>
<td></td>
</tr>
<tr>
<td>timePrevStep</td>
<td>real</td>
<td>seconds</td>
</tr>
<tr>
<td>stepsActBegin</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>livingRoom</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>diningRoom</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>kitchen</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>hallway</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>kitchenDoor</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>closet</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>kitchenItems</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>activityID</td>
<td>nominal</td>
<td>1, 2, 3, 4, 5, 6, 7, 8</td>
</tr>
<tr>
<td>stepID</td>
<td>nominal</td>
<td>*</td>
</tr>
<tr>
<td>class</td>
<td>nominal</td>
<td>0, 1</td>
</tr>
</tbody>
</table>

* Attributes prevStep, nextStep and stepID have the following nominal values: 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 2.1, 2.2, 2.3, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 8.1, 8.2, 8.3

Table 6.1: Attribute information for prompting dataset.
C Description of error types coded in our smart home studies

Study 1 was coded for four different error types described below:

- **Omission**: Coded when a step or subtask necessary for accurate task completion is not performed (i.e., critical omission; e.g., failure to retrieve broom for sweeping and dusting task); or when a step or subtask is not performed but the activity is still completed accurately (i.e., noncritical omission; e.g., failure to turn off the television for DVD task).

- **Substitution**: Coded when an alternate object, or a correct object but an incorrect gesture, is used and disrupts accurate completion of the activity (i.e., critical substitution; e.g., dusting the kitchen instead of the living room); or when an alternate object, or a correct object but an incorrect gesture, is used but the activity is still completed accurately (i.e., non-critical substitution; e.g., uses container other than watering can to water plants).

- **Irrelevant Action**: Coded when an action that is unrelated to the activity, and completely unnecessary for activity completion, is performed (e.g., opens the refrigerator when completing the sweeping and dusting task).

- **Inefficient Action**: Coded when an action that slows down or compromises the efficiency of task completion is performed (e.g., making multiple trips to
the dining room table, opening and closing extraneous cupboards and drawers unnecessary for task completion).

Study 2 was coded for nine different error types described below:

- **Substitution:** Coded when an alternate object, or a correct object but an incorrect gesture, is used and disrupts accurate completion of the activity. For example, when a participant dusts the kitchen instead of the living room; uses a container other than watering can to water plants.

- **Additional TRA (additional task related activity):** Coded when a participant is engaging in a task that appears to be task or goal related, but might not be necessary. For example, drying the watering can after watering the plants.

- **Additional NTRA (additional non-task related activities):** Coded when participant engages in an activity that is not related to the goal of the overall task. For example, retrieved brown sugar during task that did not involve brown sugar.

- **Perseveration:** Coded when participant engages in task after it has been completed. For example, watering the same plant more than once.

- **Searching:** Coded when the participant is actively searching through cupboards/drawers/closets for an item. For example, looking through all the cupboards after all of the items have been retrieved.
• **Help:** Coded when participant asks the experimenter a question or for help on a task. For example, asking where the broom is located.

• **Wandering:** Coded when participant is wandering around apartment and appears directionless. For example, walking around kitchen looking around but searching for an item.

• **Microslip:** Coded when the participant initiates and terminates an incorrect action before the error is completed. For example, when participant begins to make action toward the sink as if she was going to fill water with faucet water but then corrects behavior and obtains the pitcher from the refrigerator.

• **Error Detection:** Coded when a participant completes an error and then recognizes that she made an error and rectifies it before the end of the task.
Bibliography


Gustavo E. Batista, Ronaldo C. Prati, and Maria C. Monard. Balancing strategies


George G. Cabral, Adriano L. I. Oliveira, and Carlos B. G. Cahu. A novel method for one-class classification based on the nearest neighbor data description and struc-


Nitesh V. Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Editorial: special issue


Diane J. Cook, Narayanan C. Krishnan, and Parisa Rashidi. Activity discovery and


George Demiris and BK Hensel. Technologies for an aging society: a systematic

Misha Denil. *The Effects of Overlap and Imbalance on SVM Classification*. PhD

Misha Denil and Thomas Trappenberg. Overlap versus imbalance. *Advances in

Anind K. Dey and Gregory D. Abowd. CybreMinder: A context-aware system for

Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework
and a toolkit for supporting the rapid prototyping of context-aware applications.

Faiyaz Doctor, Hani Hagras, and Victor Callaghan. A fuzzy embedded agent-based
approach for realizing ambient intelligence in intelligent inhabited environments.
*IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Hu-

Chris Drummond and Robert C. Holte. Exploiting the cost (in) sensitivity of de-


Mohamed Ali Feki, Jit Biswas, and Andrei Tolstikov. Model and algorithmic frame-


Jiawei Han and Micheline Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann, 2006.


Haibo He and Edwardo A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.


Jesse Hoey, Pascal Poupart, Axel von Bertoldi, Tammy Craig, Craig Boutilier, and


Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification, 2003.


Nathalie Japkowicz. *Concept-learning in the absence of counter-examples: an*


Henry Kautz, Oren Etzioni, Dieter. Fox, Dan Weld, and Lokendra Shastri. Founda-


Jay Lundell, Tamara L. Hayes, Sengul Vurgun, Umut Ozertem, Janna Kimel, Jeffrey Kaye, Farzin Guilak, and Michael Pavel. Continuous activity monitoring and intelligent contextual prompting to improve medication adherence. In Engineering

Marcus A. Maloof. Learning when data sets are imbalanced and when costs are unequal and unknown. In ICML-2003 Workshop on Learning from Imbalanced Data Sets II, 2003.


Alex Mihailidis, Jennifer Boger, Tammy Craig, and Jesse Hoey. The coach prompting...


Davis M.J. Tax. DDtools, the data description toolbox for matlab, Oct 2013. version 2.0.2.


Gary M. Weiss, Kate McCarthy, and Bibi Zabar. Cost-sensitive learning vs. sam-
pling: Which is best for handling unbalanced classes with unequal error costs. In


