Handling Imbalanced and Overlapping Classes in Smart Environments Prompting Dataset

Barnan Das[1], Narayanan C. Krishnan[2], Diane J. Cook[2]

School of Electrical Engineering and Computer Science, Washington State University

Pullman WA 99164

[1]barnandas@wsu.edu, [2]{ckn, cook}@eecs.wsu.edu

Abstract The area of supervised machine learning often encounters imbalanced class distribution problem where one class is under represented as compared to other classes. Additionally, in many real-life problem domains, data with an imbalanced class distribution contains ambiguous regions in the data space where the prior probability of two or more classes are approximately equal. This problem, known as overlapping classes, thus makes it difficult for the learners in classification task. In this chapter, intersection between the problems of imbalanced class and overlapping classes is explored from the perspective of Smart Environments as the application domain. In smart environments, the task of delivering in-home interventions to residents for timely reminders or brief instructions to ensure successful completion of daily activities, is an ideal scenario for the problem. As a solution to the aforementioned problem, a novel clustering-based under-sampling (ClusBUS) technique is proposed. Density-based clustering technique, DBSCAN, is used to identify "interesting" clusters in the instance space on which under-sampling is performed on the basis of a threshold value for degree of minority class dominance in the clusters. .

1 Introduction

Over the past two decades there has been tremendous development in the area of knowledge discovery and data engineering. While developing supervised machine learning techniques for several academic and industrial problem domains, researchers encountered *class imbalance* problem that appears in real-world domains such as text classification, detection of oil spills, and credit card fraud detection. The data in these domains are imbalanced or skewed towards one class and are thus under-represented. Thus far, a significant amount of literature has been dedicated to describing techniques that deal with the class imbalance problem. These techniques include pre-processing data, modifying classifier parameters to inherently suit the dataset, biasing the classifiers to make predictions in favor of the under-represented class, and grouping the dataset using clustering to understand the subdivisions of the datasets [1].

In spite of making significant progress in this area, 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 classes. The same situation can be present in problem domains like credit card fraud detection, character recognition or automated prompting in smart environments [2] where samples from different classes have very similar characteristics. The minor differences present between the samples of two different classes are usually difficult to capture in the feature vector proposed by the domain expert. Therefore, there is a growing algorithmic need to deal with this issue. In this chapter, automated prompting in smart environments has been considered as the application domain for the evaluation of the proposed approach. Although this chapter focuses on the automated prompting problem, our approach is easily extensible for other problem domains which have datasets of a similar nature.

Research in the area of smart environments has gained popularity over the last decade. Most of the attention has been directed towards health monitoring and activity recognition [3-6]. Recently, assistive health care systems have started making an impact in society, especially in countries where human care-giving facilities are expensive and a large population of adults prefers an independent lifestyle. According to the studies conducted by the US Census Bureau [7], the number of older adults in the US aged 65+ is expected to increase from approximately 35 million in 2000 to an estimated 71 million in 2030, and adults aged 80+ from 9.3 million in 2000 to 19.5 million in 2030. Moreover, there are currently 18 million people worldwide who are diagnosed with dementia and this number is predicted to reach 35 million by 2050 [8]. These older adults face problems completing both simple (e.g. eating, dressing) and complex (e.g. cooking, taking medicine) Activities of Daily Living (ADLs) [9].

Real-world caregivers do not perform all activities for the care recipient, nor do they prompt each step of a task. Instead, the caregiver recognizes when the care recipient is experiencing difficulty within an activity and at that time provides a prompt that helps in performing the activity completely. Therefore, an automated computerized system that would be able to provide some of the facilities of a human caregiver is the call of the hour and would help in alleviating the burden of many caregivers that are looking after a large section of the population.

A *prompt* in the context of a smart home environment and from a technical perspective can be defined [10] as any form of verbal or non-verbal intervention delivered to a user on the basis of time, context or acquired intelligence that helps in successful (in terms of time and purpose) completion of a task. Although the literature is flooded with similar terms such as reminders, alerts, and notifications, *prompt* is generically used to represent interventions that ensure accomplishment of certain activity goals. Prompts can provide a critical service in a smart home setting, especially for older adults and inhabitants with cognitive impairment. Prompts can remind individuals to initiate an activity or to complete incorrect or missing steps of an activity. The PUCK project [11], or Prompting Users and Control Kiosk, conducted at the Center for Advanced Studies in Adaptive Systems (CASAS) at Washington State University operates on the hypothesis that the timing of the prompt within an activity can be learned by identifying when an activity step has been missed or performed erroneously. As a result, PUCK's goal is to deliver an appropriate prompt when, and only when, one is required. The prompt granularity for this system is individual activity steps, unlike other projects which consider activities as a whole.

From a supervised learning perspective, the goal of the prompting system is to classify an activity step (data point) either as a *prompt* step or a *no-prompt* step. Thus, it is a binary classification problem. As, in a realistic setting, there are very few situations that would require a prompt as opposed to situations that wouldn't, 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 might be advocated that, if the features under consideration are not being able to capture the necessary properties that can help in proper distinction of examples from the two classes, why not propose new features that can add more information instead of letting the problem occur and propose a solution? The best answer to this question is that the lack of infrastructural requirements in a realistic setting restricts the addition of new features. More clarity on this issue would be achieved when the smart environment testbed is described in Section 4.

2 Problem Definition

The class imbalance problem occurs when a class in a dataset is under-represented as compared to other classes. This essentially means that the imbalanced distribution of instances can exist for multiple classes. However, the general consensus in the machine learning community is that most of the algorithms are designed for imbalanced binary class distribution. Therefore, the paper mainly evaluates methods for imbalanced binary class distribution.

Real-world imbalanced datasets usually consist of binary class instances where the number of training examples of one class is extremely low as compared to the other. The former class, which is under-represented is popularly termed as the *minority* class and the latter one as the *majority* class. Also, as in real life problem domains, the primary interest is in the minority class, thus it is also known as *positive* class and the other as the *negative* class. Most of the conventional supervised learning techniques try to optimize the accuracy or decrease the error rate on the data and therefore consider the performance on the dataset as a whole and not the two classes separately. For example, if a dataset consists of 1000 instances out of which 50 are positive, a random guessing mechanism that predicts all the instances as negative in the testing phase will give an accuracy of 95%. However, all the positive class instances were predicted as negative. As can be understood from the example, approaches with this kind of prediction can prove to be fatal in most of the real-world problem domains. For example, in the mammography data set where positive class represents *cancerous* and negative class as *healthy*, the aforementioned biased prediction would result in the prediction of cancerous patients being healthy. Credit card fraud detection and network security threat detection are some of the other examples where this kind of prediction is not desirable. Similarly, for the domain of automated prompting in smart environments, it is critically important to correctly classify *prompt* instances.

The class overlap problem [12] occurs when there are ambiguous regions in the data space where there are approximately the same number of training examples from both classes. Conceptually, ambiguous regions can be visualized as regions where the prior probability for both classes is approximately equal and thus makes it difficult or impossible to distinguish between the two classes. This is because it is difficult to make a principled choice of where to place the class boundary in this region since it is expected that the accuracy will be equal to the proportion of the volume assigned to each class. Figure 1 illustrates the difference between normal data with crisp class boundaries and data with class overlap.

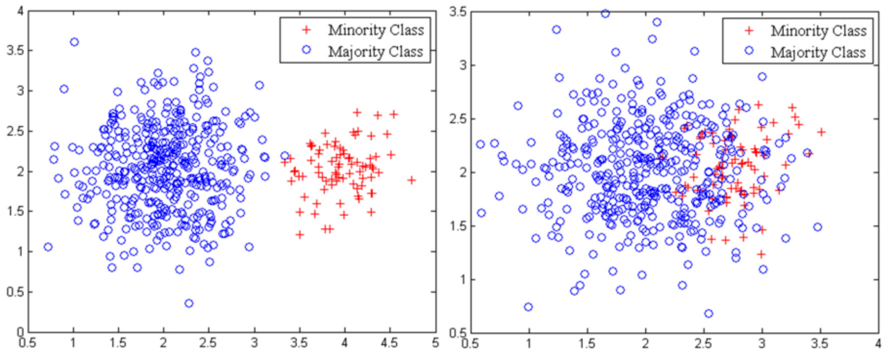


Fig. 1. (left) Data without class overlap, (right) Data with class overlap

The combination of the aforementioned problems(imbalanced class and overlapping class) makes the resultant problem to be much more difficult than solving them independently. It has been seen in some cases that identifying the overlapping region in the data space and dealing with those instances can make the data linearly separable. This idea is implemented in approaches like SMOTE+Tomek [13], Tomek+CNN [14] and few others. But, in some cases an additional problem of rare class instances exists which makes the class overlapping in imbalanced class distribution more difficult. As mentioned in the previous section, in a smart environment setting, class overlap occurs due to the fact that there are not sufficient number of features that can differentiate between the *prompt* class and the *no-prompt* class.

The prompting data has similar overlapping nature in between the two classes and that is confirmed by performing a dimensionality reduction on the attributes. Principal Component Analysis (PCA) [15] is considered for this purpose. The feature dimension is reduced to three and then plotted. Figure 2 shows a reduced three dimension plot of the prompting data. It can be easily seen from the figure that the positive (*prompt*) class instances are highly embedded in negative (*no-prompt*) class instances.

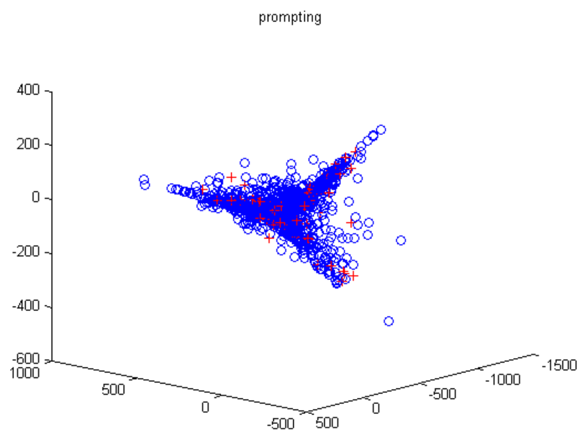


Fig. 2. 3D PCA plot for prompting data

The class overlap problem in imbalanced class data can be subdivided into a three-step sequential problem as shown in a schematic view in Figure 3. First it is important to identify the regions of overlap in the data space. However, there are major obstacles in studying overlap. Once the overlapping regions are successfully identified, the training examples in this region should be handled with a separating, merging or discarding scheme [16]. The next step is to perform the learning using different machine learning algorithms. Therefore, the approach taken in this paper is a preprocessing technique as it performs under-sampling that helps in achieving better learning models. Each of these steps are discussed in more detail in Section 6.

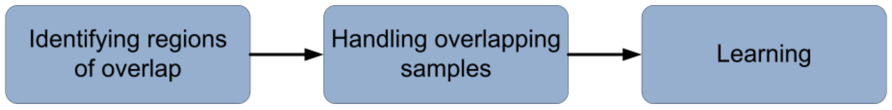


Fig. 3. Steps taken to address class overlap

3 Related Work

3.1 Imbalanced Class Distribution

Due to its common occurrence in real-life problem domains, the class imbalance problem has been studied extensively over the past decade. Numerous approaches have been proposed to deal with this issue.

One of the most common categories of approaches is a preprocessing technique known as Sampling that modifies the dataset by some means in order to provide a balanced class distribution. However, determining the ideal class distribution [17] is still an unanswered question and in most cases is done empirically. One of the naïve methods is to oversample the minority class by duplicating some of the available samples and/or under-sampling the majority class samples by throwing away randomly-chosen samples. However, these methods have the problem of either overfitting due to replication of data or losing potentially useful information, respectively. These drawbacks can be overcome by choosing a synthetic data generator like SMOTE [18] for oversampling instead of replicating data points, and informed under-sampling techniques like EasyEnsemble or BalancedCascade [19] which use strategic methods to carefully remove majority class instances. Again, synthetic data generation methods like SMOTE gives birth to the problem of over generalization due to the way it generates new data sample causing class overlaps. This could be overcome by an improved version of SMOTE known as Borderline-SMOTE [20], which generates synthetic minority class instances closer to the decision boundary. In addition to this, there are data cleansing techniques that can clean up unwanted overlapping between classes by removing pairs of minimally distanced nearest neighbors of opposite classes, popularly known as Tomek links [14]. SMOTE+ENN and SMOTE+Tomek [13] are two of the methods that utilize the capability of Tomek links to clean the data. However, the cleansing techniques might not be desirable for datasets which have inherent class overlaps or absolute rare class. Data cleansing would cause loss of highly informative data in these special cases.

Sampling techniques are data-level methods and can sometimes get complex and computationally expensive. Therefore, algorithm-level methods that can inherently take care of the imbalance are proposed. Cost sensitive learning methods (CSLs) take advantage of the underlying assumption of classical learning methods which consider the cost of all misclassification errors to be equal. Instead, CSLs use different cost factors that describe the costs for misclassifying any particular data example. The effectiveness of the application of theoretical foundations and algorithms of CSL methods to imbalanced learning problems can be verified by the works of Elkan [21] and Maloof [22]. Moreover, empirical studies have shown that in certain specific imbalanced learning domains [23,24], CSLs have performed better than sampling methods. The concepts of CSL have been coupled with existing learning methods to boost their performance. Cost Sensitive Dataspace Weighting with Adaptive Boosting [25] takes the advantage of iterative updating of weight distribution function of AdaBoost by introducing cost items in the weight updating strategy. Cost sensitive decision trees [26] use a cost-sensitive fitting approach to make adjustments that can be applied to decision threshold, split criteria at each node or pruning schemes. CSLs have also been used with neural networks [27] to combine the cost factor with the probabilistic estimate of the output values and the learning rate.

3.2 Overlapping Classes

While there has not been significant work in dealing with the class overlap problem in combination with an imbalanced class distribution, the problem of overlapping classes or ambiguous data has been widely studied in isolation [28-31], particularly in the areas of character recognition and document analysis [32], text classification, automated image annotation, credit card fraud detection, and drug design [33].

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. [34] 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 [35]. Garcia et al. [36] analyze the combined effects of class imbalance and class overlap on instance-based classification. This work is extended [37] by using several performance measures to see which one of them captures the degraded performance more accurately.

As mentioned before, a major hindrance to deal with overlapping class data is the 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. [30] proposed a k-Nearest Neighbor based approach to identify ambiguous regions in the data space. Trappenberg et al. [28] took a very similar approach to identify ambiguous regions. Visa et al.[38] perform a fuzzy set representation of the concept and thus incorporate overlap information in their fuzzy classifiers. In addition to this, Xiong et al. [16] use the one-class classification algorithm Support Vector Data Description (SVDD) to capture the overlapping regions in real-life datasets which have imbalanced class distribution too.

Once the overlapping region of the data space has been identified, the obvious next step is to handle the training examples that belong to this region. Xiong et al. [16] 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 that data in the overlapping region and just learns on the rest of the data that belongs to the non-overlapping region. SMOTE + Tomek Links [39] is such a discarding technique used to improve the performance of a 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 as 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. [28] proposes 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. [29] 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. [30] proposes 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. Thus, 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, attackers can disguise themselves as legitimate users. Due to high traffic of this kind of attackers, it is necessary to take a time critical decision on the authenticity of the user.

Therefore, in this chapter we take a preprocessing approach similar to the *discarding scheme* to deal with the overlapping data points. 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 Data Collection

The CASAS smart home infrastructure is used to replicate near ideal day to day lives of individuals in their homes. The facility is used for a wide spectrum of different goals which have both computer science and psychology focus. These goals include, activity recognition, activity discovery, activity prediction, functional assessment of participants based on demographic information, user study of machine learning driven solutions to some of the problems, testing a wide variety of sensor platforms, and at last but not the least, the automated prompting task (the PUCK project) that needs a real-life validation of the effectiveness of machine learning approaches from a technological perspective and effectiveness of prompts from psychological perspective. Although, the data that is used for automated prompting task is collected under a controlled environment, the best effort has been on maintaining a realistic daily life setting.

The test bed, or smart apartment, is located in an on-campus town house apartment. Undergraduate students, healthy older adults, and older adults with various levels of dementia are brought in to perform Activities of Daily Living, or ADLs. The data collected from these experiments are used to train classifiers in identifying these ADLs.

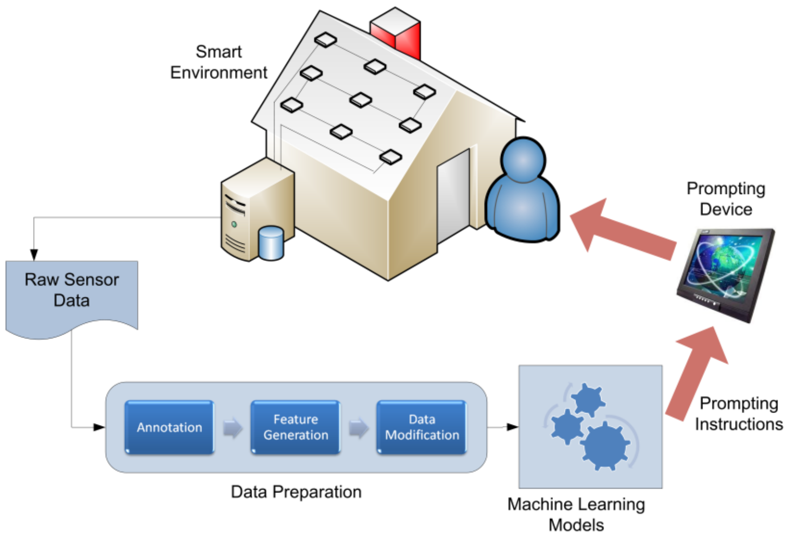


Fig. 4. PUCK system architecture

The current sensor system is composed of several different sensor types. There are sensors for motion, ambient light level, temperature, doors, light switches, items, objects, water flow, and power use. A majority of our sensors are now wireless, utilizing 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 so 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 where you are to help prevent tripping at night, or illuminating your work-space when enough natural light is no longer available. Temperature sensors are also useful for determining inhabitant behavior, such as thermal preferences, determining when the stove or oven is in use in the kitchen.

One of the bedrooms on the second floor is used as a control room where the experimenters monitor the activities performed by the participants (via web cameras) and deliver pre-defined prompts through an audio delivery system whenever necessary. The goal of PUCK is to learn from this collected data how to time the delivery of prompts and ultimately to automate the role of the experimenter in this setting. The following activities are used in our experiments: Sweep, Refill Medication, Birthday Card, Watch DVD, Water Plants, Make Phone Call, Cook and Select Outfit. These activities are subdivided into relevant steps by the psychologists in order to track their proper completion.

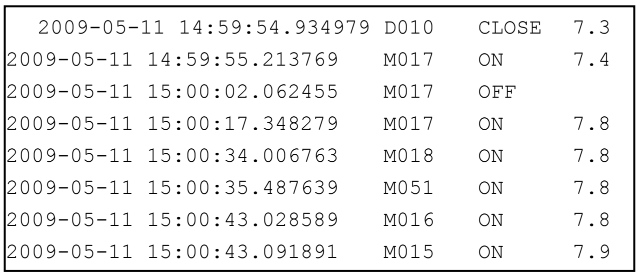
Volunteering participants are brought in the apartment and asked to perform the specific activities. While going through the steps of an activity, a prompt is given if he/she performs steps for other activities rather than the current one, if a step is skipped, if extra/erroneous steps are performed, if an inappropriate tool is used, or if too much time has elapsed since the beginning of the activity. Note that there is no ideal order of steps by which the activity can be completed. Therefore, a prompt is issued only when one of the conditions mentioned above occurs. Moreover, the goal is to deliver as few prompts as possible. The experimenters keep track of all the errors committed by the participants and the steps at which a prompt was issued, which are later extracted and used to train the machine learning algorithms.

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

After collecting data, sensor events are labeled with the specific activity and step within the activity, {activity#.step#}, that was being performed while the sensor events were generated, as shown in Figure 5.

Table 1. Sample of sensor events used for our study

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



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

Fig. 5. Annotation of activity steps. The sensor events belong to steps 3, 4, 8, and 9 of activity 7.

5 Dataset and Performance Metrics

5.1 Feature Generation

Relevant features are generated from the annotated data that is helpful in predicting whether a step is a *prompt* step or a *no-prompt* step. Each step of an activity is treated as a separate training example, and pertinent features are defined to describe the step based on sensor data. Each data instance is tagged with the class value. Specifically, a step at which a participant received a prompt is labeled as "1" indicating prompt, others are hence assumed to be no-prompt steps and labeled as "0". Table 2 provides a summary of all generated features. It should be noted that the machine learning models learn and predict class labels from this refined dataset. This way PUCK predicts if an instance (steps of activities in this context) constitutes a prompt instance. Thus, the problem of when a prompt should be delivered is addressed.

Table 2. Generated features.

Feature	Description
stepLength	Length of step in time (seconds)
numSensors	Number of unique sensors involves with the step
numEvents	Number of sensor events associated with the step
prevStep	Previous step ID
nextStep	Next step ID
timeActBegin	Time (seconds) elapsed since the beginning of the activity
timePrevAct	Time (seconds) difference between the last event of the previous step and first event of the current step
stepsActBegin	Number of steps visited since the beginning of the activity
activityID	Activity ID
stepID	Current step ID
location	A combination of features like, kitchen, kitchen sink, dining room, living room, hallway, etc. which represents motion sensor firing of those regions
Class	Binary class representing prompt and no-prompt

Sensor data collected from 128 participants is used to train the machine learning models. There are 53 steps in total for all the activities, out of which 38 are recognizable by the annotators. The rest of the steps are associated with specific object interactions that could not be tracked by the current sensor infrastructure. The participants were delivered prompts in 149 cases which involved any of the 38 recognizable steps. Therefore, approximately 3.74% of the total instances are positive (prompt steps) and the rest are negative (no-prompt steps). Essentially, this means that, predicting all the instances as negative, would give more than 96% accuracy even though all the predictions for positive instances were incorrect. This requires evaluation of the classifiers with performance metrics that can capture classification performance for both the classes. Section 5.2 highlights the performance metrics used for this study.

5.2 Performance Measures

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

Therefore, performance measures that directly measure the classification performance for positive and negative classes independently are considered. The True Positive (TP) Rate (the positive which is also in this case the minority class) here represents the percentage of activity steps that are correctly classified as requiring a prompt; the True Negative (TN) Rate here represents the percentage of steps that are accurately labeled as not requiring a prompt. The TP and TN Rates are thus capable of measuring the performance of the classifiers separately for the positive and negative classes. ROC curve analysis is used to evaluate overall classifier performance. An ROC curve plots the classifier's false positive rate [17] on the x-axis and the true positive rate on the y-axis. A ROC curve is generated by plotting the accuracy obtained by varying different parameters of the classifiers. The primary advantage of using these is that they illustrate the classifier's performance without taking into account class distribution or error cost. AUC, or the area under ROC curve [40], is reported in order to compute the performance over all costs and distributions. Also, the geometric mean of TP and TN rates denoted by G-mean is reported, which is commonly used as a performance metric in imbalanced class learning. G-mean is calculated as $\sqrt{TPRate \times TNRate}$ **.**

6 Current Approach

By performing a hypothesis testing, Denil et al. proved [41] that overlap and imbalance are not two independent factors. They have very strong coupling when it comes to the problem of imbalanced class distribution. Denil et al. showed that if overlap and imbalance levels are too high, good performance cannot be achieved regardless of amount of available training data. Therefore, employing a **Clus**ter-**B**ased **U**nder-**S**ampling (**ClusBUS**) technique, the purpose is to get rid of the overlapping class problem and the hypothesis is that achieving success with the overlap problem would also be helpful in getting rid of the detrimental effects of class imbalance problem to some extent, as the majority class is being under-sampled.

The idea of devising this technique is derived from the use of Tomek links [14] combined with other sampling methods like Condensed Nearest Neighbor [42] and SMOTE [13]. Tomek links are defined as: given two examples E_i and E_j belonging to different classes, and $d(E_i,E_j)$ being the distance between E_i and E_j, a (E_i,E_j) pair is called a Tomek link if there is not an example E_k such that $d(E_i,E_k) < d(E_i,E_j)$. If two examples form a Tomek link, then either one of these examples is noise or both examples are on or near the class boundary. Tomek links are used both as a data cleansing method and an under-sampling method. As a data cleansing method, examples of both classes are removed, and as an under-sampling method, only examples belonging to the majority class are eliminated.

One-sided selection (OSS) [43] 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 examples. As a small amount of noise can make the borderline examples fall on the wrong side of the decision boundary, borderline examples are considered as unsafe. CNN is used to remove examples from the majority class that are far away from the decision boundary. The rest of the majority and minority class examples are used for learning.

As opposed to the use of Tomek links in OSS to find minimally distanced nearest neighbor pairs of opposite classes and then remove majority class examples, ClusBUS finds *interesting* clusters with a *good* mix of minority and majority class examples. The definition of *good* mix is determined by a *degree of minority class dominance* explained in detail later in this section. The majority class examples from these clusters are then removed.

Table 3 summarizes the ClusBUS algorithm. First, the entire training data is clustered ignoring the class attribute using Euclidean distance as the distance measure. The degree of minority class dominance, denoted by r, for each of these clusters is calculated as the ratio of number of minority class examples to the size of the cluster. Therefore, $r=0$ indicates that all the examples of the cluster belong to the majority class, and $r=1$ indicate that all the examples belong to the minority class. The clusters whose r lies between 0 and 1 are of interest in this method as it indicates that the cluster has both minority and majority class examples. For this kind of cluster, the majority class examples are removed if r is equal to or greater that an empirically determined threshold value τ. Clearly, if the threshold τ is low more majority class examples would be removed as compared to when τ is high. This method creates a "*vacuum*" around the minority class examples in each cluster and thus helps the machine learning classifiers learn the decision boundary more efficiently.

Table 3. Algorithm of Cluster-Based Under-Sampling.

1. Let S be the original training set.
2. Use clustering to form clusters on S denoted by C_i where $1<i<|C|$.
3. Find the degree of minority class dominance for all C_i by:

$$r_i = \frac{\text{Number of Minority Class examples in } C_i}{|C_i|}$$

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

Figure 6 shows an illustration of ClusBUS on a synthetic dataset. The imbalanced and overlapping data is represented in 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 $r > \tau$. Note that, in the bottom diagram of Figure 6, 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 $r > \tau$.

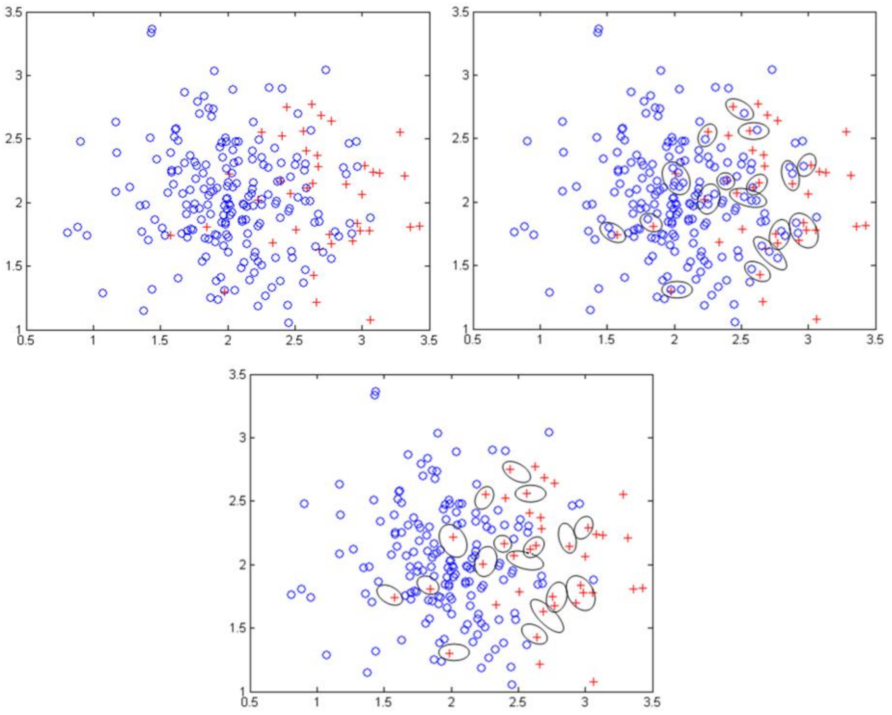


Fig. 6. Schematic representation of ClusBUS Algorithm

The clustering approach in order to identify interesting clusters can be any conventional clustering approach. In this experiment, partitioning based clustering methods are avoided due to two reasons: (1) requires user intervention in specifying the number of clusters that need to be constructed from the data, and (2) forms spherical-shaped clusters only. In this study, a density based clustering methods, namely, Density-Based Spatial Clustering of Applications with Noise or DBSCAN is used. The rationale behind using DBSCAN is that there can be arbitrary shapes of clusters in the data that are not necessarily spherical (Gaussian). As there is no prior knowledge of the distribution of the data, the notion of density, on which DBSCAN is based, is more meaningful rather than specifying the number of clusters and forcing the data to be partitioned accordingly.

DBSCAN [44,45] 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. Any 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 ***ε***. If the ε-neighborhood of a data point contains at least a minimum number of other points denoted by ***MinPts***, then the point is called a ***core point***, and the ε-neighboring points are ***directly density-reachable*** from the core point. A point *p* is ***density-reachable*** from point *q* with respect to *ε* and *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 *ε* and *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 *ε* and *MinPts*. The DBSCAN algorithm is summarized in Table 4.

Table 4. Algorithm of DBSCAN

```
1.  Search for clusters by checking ε-neighborhood of each point.
2.  If ε-neighborhood of a point p contains more than MinPts, a new cluster with p as
    core point is created.
3.  Iterative collection of directly density-reachable points from the core points.
4.  Terminate when no new point can be added to any cluster.
```

The threshold τ, on the basis of which the majority class points are under-sampled from interesting clusters, is empirically determined by considering statistical properties of the degree of minority class dominance *r*. Table 5 describes the algorithm to calculate τ.

Table 5. Algorithm to find τ

```
1.  Consider all r such that $0<r<1$.
2.  Find min and max r.
3.  On the basis of an empirical test, select a value of r in between min and median
    as τ.
```

It should also be kept in mind that the prompting data has an absolute rarity imbalance problem, that is, the minority class instances are not only relatively less as compared to majority class, but also rare in absolute number. Therefore, this serves as a rationale for not employing sampling methods that involve discarding minority class examples.

7 Experimentation and Results

For our experiments, learning algorithms are chosen from four widely accepted categories of learners, namely, decision trees, k nearest neighbors, Bayesian learner and support vector machines. The specific classifiers chosen from these areas are: C4.5 [46], IBk [47], Naïve Bayes Classifier [48] and SMO [49], respectively. Observing the results of the classifiers from four different categories gives the readers the opportunity to analyze the improvement that ClusBUS achieves.

In the domain of imbalanced class datasets, most of experiments are performed by k-fold cross validation. This method of testing is far away from reality for most real-life problems, especially methods which involve preprocessing techniques such as sampling. Most of the times sampling tampers with the original data distribution, and cross validation forces the classifiers to train as well as test on the same tampered data distribution and thus resulting in overly optimistic results which are tangential to reality. In order to avoid this inappropriate evaluation technique, the current experimental setup trains the classifiers on 80% data and considers the rest for testing. Also, the degree of imbalance in the original dataset is maintained in training and testing examples.

As the proposed approach is a preprocessing technique, under-sampling to be more specific, that is performed on the data before it could be fed to the classifiers, the comparison is done with a well known over-sampling technique, known as SMOTE [18]. SMOTE uses a combination of both under and over sampling, but without data replication. Over-sampling is performed by taking each minority class sample and synthesizing a new sample by randomly choosing any or all (depending upon the desired size of the class) of its *k* minority class nearest neighbors. Generation of the synthetic sample is accomplished by first computing the difference between the feature vector (sample) under consideration and its nearest neighbor. Next, this difference is multiplied by a random number between *0* and *1*. Finally, the product is added to the feature vector under consideration. As mentioned earlier, there are a number of downsides if the methods are evaluated with a cross validation technique. In case of SMOTE, as the training and testing were done on the same examples that are synthetically generated, the overfitting of the classifiers caused by an overwhelmed synthesis of artificial minority class examples, would never be detected. For the current experimentation, SMOTE is used to boost the number of training examples of minority class to be the same as that of majority class examples. Any further boosting to generate more artificial training examples would make is unrealistic in real-life problem domains due to high computation and sometimes monetary cost associated with synthesizing new data points.

As mentioned in the algorithm of ClusBUS, the threshold value of the degree of minority class dominance, on the basis of which *interesting* clusters are identified and the majority class examples are removed, is determined empirically. As mentioned in Table 5, τ is calculated by finding a value between min and median. This is done by varying the value of τ from median, 1st tercile, 1st quartile, 1st quintile, and so on, till 1st decile. In other words, the first *x%* of interesting clusters (where interesting clusters are increasingly ordered based on the *r*) is considered for under-sampling, where x is 50%, 33%, 25%, 20%, 16.67%, 14.28%, 12.5% 11.11% and 10%. TP Rate, FP Rate and AUC of C4.5 on the data after being pre-processed using ClusBUS are plotted in Figure 7. In the x axis, a number *k* indicates the first part of the data if it is divided into *k* equal parts.

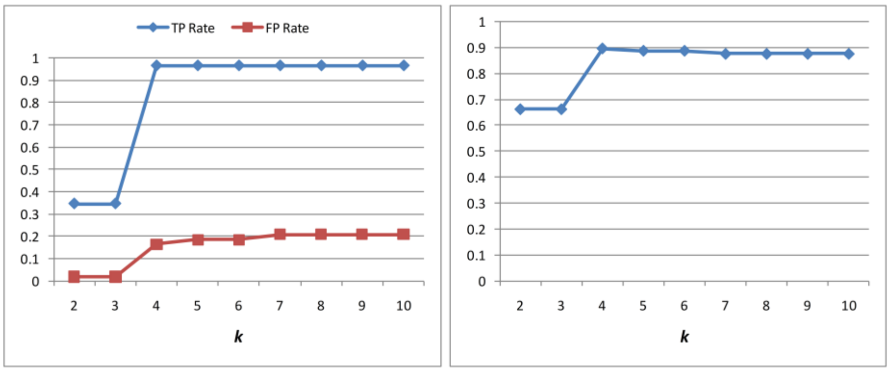


Fig. 7. Comparison of performance measures: (left) TP and FP Rate, (right) AUC, with respect to different values of *k*.

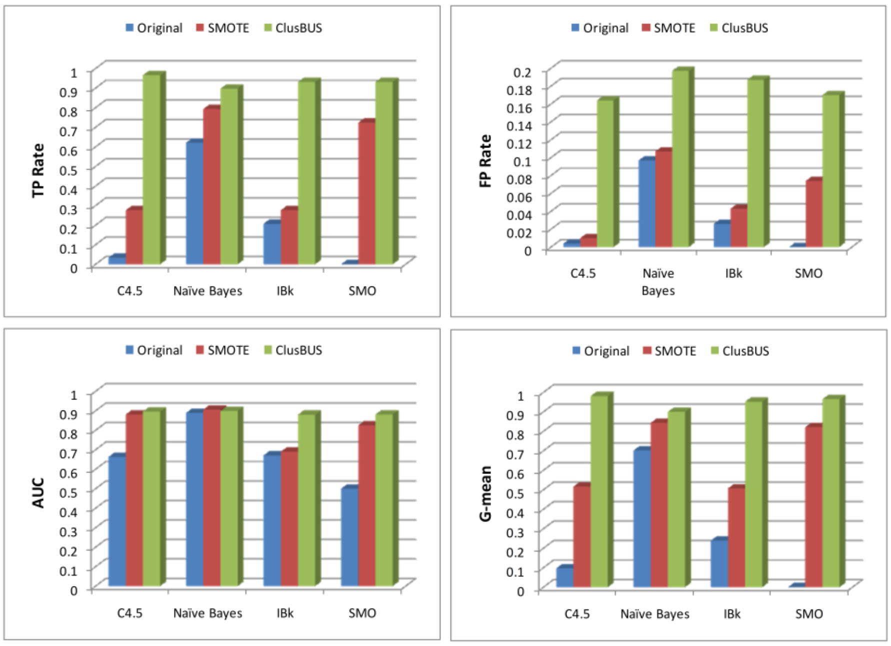


Fig. 8. Comparison of performance measures: (top-left) TP Rate, (top-right) FP Rate, (bottom-left) AUC, and (bottom-right) G-means

It can be seen that FP rate from *k=3* to *k=4*. Also, AUC is fairly high at *k=4*. Therefore, further experimentation with the rest of the classification algorithms is performed considering *k=4*, that is *τ=0.25*.

From Figure 8, it can be clearly stated that ClusBUS performs significantly better than SMOTE. ClusBUS does not entail any costly data synthesis technique like SMOTE, but performs under-sampling on majority class training examples that belong to the overlapping region in the data space. TP Rate and G-means has shown significant improvement over SMOTE, although the same trend is not followed for AUC. For C4.5 and Naïve Bayes classifiers AUC is marginally better than that achieved by SMOTE. However, ClusBUS has caused increase in FP Rate as compared to the performance of the classifiers on original data and after applying SMOTE. But, this increase is not significant enough to demean the advantages and improvement of ClusBUS on other performance metrics

8 Conclusions and Future Work

This chapter proposes a novel preprocessing technique, ClusBUS, to deal with the class overlap problem in the presence of imbalanced class distribution in the dataset. A density-based clustering technique is used to find interesting clusters that would be chosen for under-sampling. This problem occurs in many real-life problem domains like automated prompting task in smart environments, credit card fraud detection, network intrusion detection, etc. The effectiveness of the approach is validated by selecting smart environments as the target domain. The data of automated prompting task is therefore considered for experimentation. Experiments show that it is possible to achieve significant improvement over other well-known sampling techniques like SMOTE.

The plan for the future is to test the performance of ClusBUS on data from other real-life problem domains such as credit card fraud detection and network intrusion detection. Also, a comparative analysis would be performed with a wider spectrum of sampling techniques.

References

1. He H, Garcia EA (2008) Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering:1263-1284
2. Das B, Chen C, Dasgupta N, Cook DJ, Seelye AM (2010) Automated prompting in a smart home environment. Paper presented at the 2010 IEEE International Conference on Data Mining Workshops,
3. Singla G, Cook DJ, Schmitter-Edgecombe M (2010) Recognizing independent and joint activities among multiple residents in smart environments. Journal of ambient intelligence and humanized computing 1 (1):57-63

4. Singla G, Cook DJ, Schmitter-Edgecombe M (2009) Tracking activities in complex settings using smart environment technologies. International journal of biosciences, psychiatry, and technology (IJBSPT) 1 (1):25
5. Tapia EM, Intille SS, Larson K (2004) Activity recognition in the home using simple and ubiquitous sensors. Pervasive Computing:158-175
6. Maurer U, Smailagic A, Siewiorek DP, Deisher M Activity recognition and monitoring using multiple sensors on different body positions. In, 2006. IEEE, pp 4 pp.-116
7. Bureau UC (2011) US Population Projections. http://www.census.gov/population/www/projections/natdet-D1A.html. 2011
8. Bates J, Boote J, Beverley C (2004) Psychosocial interventions for people with a milder dementing illness: a systematic review. Journal of Advanced Nursing 45 (6):644-658
9. Wadley VG, Okonkwo O, Crowe M, Ross-Meadows LA (2008) Mild Cognitive Impairment and everyday function: Evidence of reduced speed in performing instrumental activities of daily living. American Journal of Geriatric Psych 16 (5):416
10. Das B, Chen C, Seelye AM, Cook DJ (2011) An Automated Prompting System for Smart Environments. Paper presented at the 9th International Conference on Smart Homes and Health Telematics,
11. Das B, Cook D, Schmitter-Edgecombe M, Seelye AM (2012) PUCK: An Automated Prompting System for Smart Environments. Personal and Ubiquitous Computing Theme Issue on Sensor-driven Computing and Applications for Ambient Intelligence (779)
12. Denil M (2010) The Effects of Overlap and Imbalance on SVM Classification. Master's, Dalhousie University,
13. Batista GE, Prati RC, Monard MC (2004) A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD Explorations Newsletter 6 (1):20-29
14. Tomek I (1976) Two modifications of CNN. IEEE Trans Syst Man Cybern 6:769-772
15. Jolliffe I (2002) Principal component analysis. Encyclopedia of Statistics in Behavioral Science
16. Xiong H, Wu J, Liu L Classification with Class Overlapping: A Systematic Study.
17. Provost F, Fawcett T, Kohavi R The case against accuracy estimation for comparing induction algorithms. In, 1998. Citeseer,
18. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) SMOTE: synthetic minority over-sampling technique. Journal of Artificial Intelligence Research 16 (1):321-357
19. Liu XY, Wu J, Zhou ZH (2006) Exploratory under-sampling for class-imbalance learning.
20. Han H, Wang WY, Mao BH (2005) Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning. Advances in Intelligent Computing:878-887
21. Elkan C The foundations of cost-sensitive learning. In, 2001. Citeseer, pp 973-978
22. Maloof M Learning when data sets are imbalanced and when costs are unequal and unknown. In, 2003. Citeseer,
23. McCarthy K, Zabar B, Weiss G Does cost-sensitive learning beat sampling for classifying rare classes? In, 2005. ACM, pp 69-77
24. Liu XY, Zhou ZH (2006) The influence of class imbalance on cost-sensitive learning: An empirical study.
25. Sun Y, Kamel MS, Wong AKC, Wang Y (2007) Cost-sensitive boosting for classification of imbalanced data. Pattern Recognition 40 (12):3358-3378
26. Drummond C, Holte RC Exploiting the cost (in) sensitivity of decision tree splitting criteria. In, 2000. pp 239-246
27. Kukar M, Kononenko I Cost-sensitive learning with neural networks. In, 1998. Citeseer, pp 445-449
28. Trappenberg TP, Back AD A classification scheme for applications with ambiguous data. In, 2000. IEEE, pp 296-301 vol. 296
29. Hashemi S, Trappenberg T (2002) Using SVM for Classification in Datasets with Ambiguous data. SCI 2002

30. Tang Y, Gao J (2007) Improved classification for problem involving overlapping patterns. IEICE TRANSACTIONS ON INFORMATION AND SYSTEMS E SERIES D 90 (11):1787
31. Lin YM, Wang X, Ng WWY, Chang Q, Yeung DS, Wang XL Sphere classification for ambiguous data. In, 2006. IEEE, pp 2571-2574
32. Liu CL (2008) Partial discriminative training for classification of overlapping classes in document analysis. International journal on document analysis and recognition 11 (2):53-65
33. Andrews SJD, Hofmann T, Van Hentenryck P, Black M (2007) Learning from ambiguous examples, vol 68. vol 07.
34. Prati RC, Batista GE, Monard MC (2004) Class imbalances versus class overlapping: an analysis of a learning system behavior. MICAI 2004: Advances in Artificial Intelligence:312-321
35. Batista GE, Prati RC, Monard MC (2005) Balancing strategies and class overlapping. Advances in Intelligent Data Analysis VI:24-35
36. García V, Alejo R, Sánchez J, Sotoca J, Mollineda R (2006) Combined effects of class imbalance and class overlap on instance-based classification. Intelligent Data Engineering and Automated Learning–IDEAL 2006:371-378
37. García V, Mollineda R, Sánchez J, Alejo R, Sotoca J (2007) When overlapping unexpectedly alters the class imbalance effects. Pattern Recognition and Image Analysis:499-506
38. Visa S, Ralescu A Learning imbalanced and overlapping classes using fuzzy sets. In.
39. Batista G, Bazan A, Monard MC Balancing training data for automated annotation of keywords: a case study. In, 2003. Citeseer, pp 35–43
40. Hand DJ (1997) Construction and assessment of classification rules, vol 15. Wiley New York,
41. Denil M, Trappenberg T (2010) Overlap versus Imbalance. Advances in Artificial Intelligence:220-231
42. Hart P (1968) The condensed nearest neighbor rule (corresp.). Information Theory, IEEE Transactions on 14 (3):515-516
43. Kubat M, Matwin S Addressing the curse of imbalanced training sets: one-sided selection. In, 1997. Citeseer, pp 179-186
44. Ester M, Kriegel HP, Sander J, Xu X A density-based algorithm for discovering clusters in large spatial databases with noise. In, 1996. Portland: AAAI Press, pp 226-231
45. Jiawei H, Kamber M (2001) Data mining: concepts and techniques. San Francisco, CA, itd: Morgan Kaufmann 5
46. Quinlan JR (1993) C4. 5: programs for machine learning. Morgan Kaufmann,
47. Aha DW, Kibler D, Albert MK (1991) Instance-based learning algorithms. Machine learning 6 (1):37-66
48. John GH, Langley P Estimating continuous distributions in Bayesian classifiers. In, 1995. Citeseer, pp 338–345
49. Platt J (1999) Sequential minimal optimization: A fast algorithm for training support vector machines. Advances in Kernel Methods-Support Vector Learning 208:98–112