A Comparison of Classification Models for Imbalanced Datasets

Thesis · June 2021

DOI: 10.13140/RG.2.2.26879.12966

2 authors:

Khdr Bashar
Eötvös Loránd University
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Zakarya Farou
Eötvös Loránd University
17 PUBLICATIONS **18** CITATIONS
SEE PROFILE

Eötvös Loránd University

Faculty of Informatics

Data Science and Engineering Department

A Comparison of Classification Models for Imbalanced Datasets

Supervisor:
Zakarya Farou
Data Science and Engineering Department

Author:
Khdr Bashar
Computer Science

Budapest, 2021

Topic Declaration

Many practical classification problems are difficult because of unbounded size and the imbalance nature of data. The class imbalance problem becomes a significant issue in data mining. An imbalance problem occurs where one of the classes has more samples than other classes. When a disproportionate ratio of samples per class is present, most machine learning classifiers (algorithms) are focusing on the classification of the majority class samples while ignoring or misclassifying minority class samples. The minority samples are those that rarely occur but very important. There exist different methods available for the classification of imbalanced data sets, mainly they are divided into three categories: the algorithmic approach, data preprocessing approach, and feature selection approach.

The goal of this thesis is to develop, implement, and compare existing approaches that handle imbalance problems. Imbalanced datasets can be obtained from the KEEL repository which contains many imbalanced datasets categorized in different IR factors. Also the UCI repository has so many imbalanced datasets for benchmarking.

The possibility of developing novel approach by combining the implemented ones based on the outcome of the comparison experiment will be considered. The thesis is expected to answer the following research question: which are the most efficient methods and the appropriate metrics that deal with imbalanced problems based on the state-of-the-art?

Contents

List of Figures

List of Tables

List of abbreviations

Abbreviation	**Meaning**
CIL	Class Imbalance Learning
IR	Imbalance Ratio
GAN	Generative Adversarial Network
GAMO	Generative Minority Oversampling
SMOTE	Synthetic Minority Over Sampling Technique
BFT	Belief Function Theory
Bel	Belief
pl	plausibility
GMM	Gaussian Mixture Model
LOF	Local Outlier Factor
KNOS	K Neighbors Oversampling
LR	Logistic Regression
DT	Decision Tree
SVM	Support Vector Machine
KNN	K Nearest Neighbors
FP	False Positive
FN	False Negative
TN	True Negative
TP	True Positive
FPR	False positive Rate
FNR	False Negative Rate
ROC-AUC	Area under the receiver operating characteristics
DNN	Deep Neural Network

G-mean	Geometric mean

Chapter 1

Introduction

1.1 Motivation

Class Imbalance Learning (CIL) [1, 2, 3] is one of the problems arises due to data source which provides unequal class where examples of one class in a training dataset vastly outnumber examples of the other class(es) [4]. CIL has been listed as the Top 10 challenging problems in data mining [5]. Traditional classification algorithms expect data samples to be distributed equally among all classes. However, in many real-world applications like fraud detection, pollution detection, risk management, and especially medical diagnosis [1], the distribution of data samples is imbalanced where one class, which we call the majority class, is represented by a large portion of data instances. The degree of imbalance (the ratio of the number of majority group samples to the size of the minority class) varies from one application to another [6]. On the other hand, the class of interest [7], or positive class, which in real-world applications might represent abnormal data flows in some network security application or infected patients in a medical diagnosis application, is underrepresented compared to the majority class. One of the significant difficulty that might arise from training a classifier on an unbalanced data set is the risk of misclassifying all minority samples. There exist many solutions in the literature to tackle the CIL problem. For instance, cost-sensitive methods modify the learning algorithm to increase the error of misclassifying minority instances, undersampling methods remove several instances from the majority class until some predefined imbalance rate (IR) is met; however, it increases the risk of losing important information upon

deleting. Furthermore, oversampling methods used to increase minority class size are the most popular and used techniques in practice. Random oversampling of the majority class by duplicating existing samples might lead to overfitting, so we tend to work with more advanced methods that generate new synthetic samples based on existing ones. Our research focuses on oversampling techniques that generate new instances for tackling CIL in binary classification applications. The main objective of our research is to address the problem of CIL, which does exist in many real-world applications; by comparing the state of the art data oversampling methods through extensive experiments, and based on the results of our selected metrics, we can determine the most efficient technique to be used with which classifier.

1.2 Thesis outline

The organization of the rest of the thesis is as follows:

- **Chapter 2** provides a theoretical overview on class imbalance learning and many of state of the art oversampling methods for addressing this problem;
- **Chapter 3** provides extensive experiments on many of the presented oversampling methods, in addition to the results of the conducted experiments and a discussion over it;
- **Chapter 4** provides a conclusion and some future directions.

Chapter 2

Literature Review

In this chapter, we will present many of the modern oversampling methods that have been recently used to tackle the class imbalance learning problem. To begin, we will start with a brief introduction to classification, then we will talk about class imbalance problem and how it emerges in many real world applications. Finally, we will widely go over the most recent methodologies for handling class imbalance problem.

2.1 Classification problems

Definition 1. *Classification [8] is a keystone in supervised machine learning [9, 10], and it is the process of finding a set of models (functions) that describe and distinguish data classes or concepts for the purpose of being able to use the model to predict the class of objects whose class label is unknown or unseen [1].*

These observations might represent a set of students in some university, and the target class can be the success or failure of some student in a specific course. We express each observation by a set of features or attributes. In the students' example, we can choose the interaction inside the class, course attendance percentage, etc ... to predict whether the student could pass the course or not 2.1. We represent the set of features that describes each observation as $x_1, x_2, \ldots, x_n$ where x represents the observation, and n is the number of features. The target label can be denoted by the symbol y, and for m observations we have $y_1, y_2, \ldots, y_k, \ldots, y_m$ labels where y_k is the target label associated with the k^{th} observation.

interaction inside class	attendance percentage	...	label
8.3	45%	...	success
2.1	95%	...	failed
...	...	...	...

Table 2.1: An example of a binary classification data set

The identified parameters form a model that can be used later to predict unknown/unlabeled instances. The model relies on a set of observations or instances together with the associated labels or classes. These observations, combined with their corresponding labels, form what is known as a data set. Each data set contains a set of m observations where the k^{th} observation in the given data set can be represented as $x_1^k, x_2^k, x_3^k, \ldots, x_n^k, y_k$ Depending on the number of classes that the data set contains, we can divide the classification into two approaches which are as the following:

2.1.1 Binary classification

In binary classification (Fig. 2.1), we use the target label y, where $y \in \{0, 1\}$ to indicate positive and negative classes. A positive class is the target class labelled with $y = 1$ while the negative class is the majority class labelled with $y = 0$.

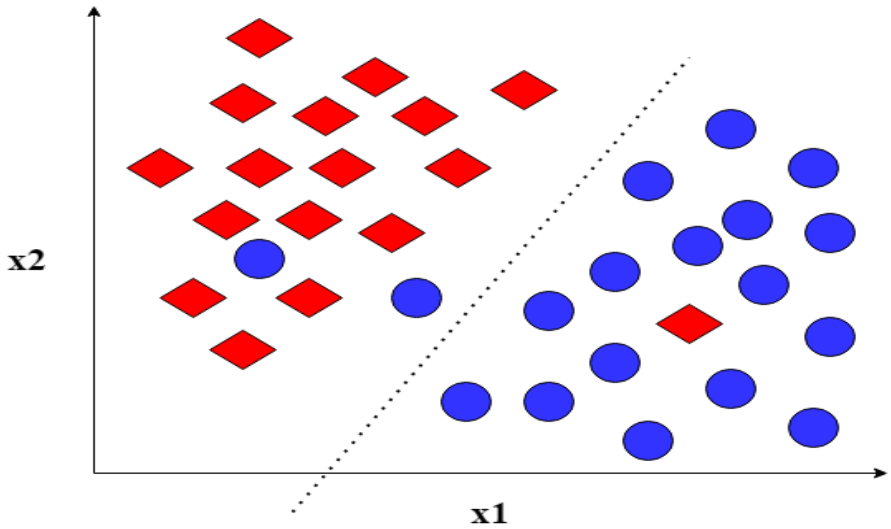

Figure 2.1: Example of a binary classification for a dummy dataset with two features.

2.1.2 Multi-class classification

For multi-class classification [11], the target label can take on multiple values $y \in \{0, 1, 2, \ldots, c-1\}$ where c is the number of classes. For example, given the features of a network flow, our task could be predicting the type of application this flow belongs to.

2.1.3 Imbalanced-class classification

Definition 2. *A dataset is said to be class imbalanced if the classification categories are not approximately equally represented [4].*

Class imbalance learning [12, 13, 14, 15, 16] arises due to many imbalanced data sets in nature. Traditional machine learning algorithms expect the number of instances that belong to each class to be roughly similar [17]. However, in many real-life situations, the distribution of observations is distorted, as the number of observations in some classes might appear more frequently, and machine learning algorithms will be biased towards the majority group. This situation implies difficulties identifying minority class samples. In all applications, the minority class is more important, and it is named the positive class. Imbalanced data sets can be found in numerous real-world classification problems. For example, in a medical diagnosis problem, the cases affected by the disease are usually relatively rare compared to the normal ones [18]. Other applications include fraud detection, churn prediction, identifying student's academic failures, unusual returns on the stock market, and disaster anticipation [19]. The amount of skewness in an imbalanced data set is measured by the imbalance ratio.

Definition 3. *Imbalance Ratio is defined as the ratio between the number of majority samples and the number of minority samples. Given a binary data set D, the number of instances belonging to majority class is represented by N^-, and the number of instances in minority group is represented as N^+.The Imbalance Ratio is defined as*

$$IR = \frac{N^-}{N^+} \tag{2.1}$$

When learning from imbalanced data sets, the majority class dominates the classifier decision boundary causing it to misclassify most of the minority samples.

2.2 Data balancing techniques

According to [17], many balancing techniques have been developed for tackling the class imbalance learning problem (CIL). They can be generally categorized into external or Data-level approaches, and internal or Algorithm-level approaches, and hybrid approaches.

2.2.1 External methods

External methods [20, 21] pre-process the training data either by adding more minority observations or reducing the size of the majority class. External methods, also called re-sampling methods, can be further categorized into under-sampling, oversampling methods.

Oversampling techniques

Oversampling methods work by synthesizing artificial instances, and adding them to the minority class [22, 23, 24]. In its simplest form, random oversampling is done on the minority class by duplicating samples either randomly or from areas close to the decision boundary. Duplicating existing instances might lead to over-fitting. [1, 5].

Under-sampling techniques

Undersampling methods remove instances from the majority class(es) based on a predefined set of rules [25, 26, 27]. In its simplest form, random under-sampling is performed on the majority class by removing random instances or instances distant from the decision boundary. Random under-sampling might remove significant patterns from the original data set. It has been verified that random oversampling is usually more accurate than random under-sampling [1, 5].

2.2.2 Internal methods

Internal methods adjust the classification algorithm to reduce its bias towards the majority class. The most popular approaches are cost-sensitive methods. Here, we modify the classification algorithm by assigning a higher cost to the less represented

set of observations. Such a method aims to boost the importance of minority group during the learning process. However, a difficulty that persists concerns how to set cost values for minority samples properly [17, 6].

2.2.3 Hybrid approaches

This type of approaches combines the advantages of internal and external methods to create more robust and efficient balancing techniques.

2.3 External CIL techniques

Data-level balancing techniques work by modifying the original training set by removing objects from the majority class (undersampling) or generating new objects from minority samples. In its simplest form, random oversampling can be applied by randomly duplicating existing minority objects. However, as previously mentioned, this might lead to an overfitting situation. On the other hand, random undersampling works by omitting randomly selected instances from the majority class. This approach might lead to the loss of critical samples that might have a strong influence while inferring classification rules such as support vectors used in the SVM algorithm [17].

2.3.1 Synthetic minority oversampling technique

Synthetic minority oversampling technique [7, 18, 28, 29, 30, 31], i.e., SMOTE, is the most popular and widely used data-level oversampling technique. It works by generating, and not duplicating, new objects in the vicinity of minority samples. The algorithm works by linear interpolation between two minority samples. Fig. 2.2 depicts the process of generating new instances by linear interpolation between two selected minority class instances [5, 32, 33, 34]

1. First, we randomly choose a minority sample. Let $x_i \in D^+$,where D^+ is the set of minority samples, be a randomly selected minority sample;

2. We find the k nearest neighbors of x_i. Extract the set $\{x_1, x_2, ..., x_j, ..., x_k\}$ such that the euclidean distance between x_i and x_j is minimal, and $x_j \in$ minority group.

3. Randomly select an instance from the set of k nearest neighbors of x_i.

4. Create a new minority synthetic sample using the following formula

$$x^{'} = x_i + \delta * (x_i - x_j) \tag{2.2}$$

where δ in the above equation is random fraction in the interval [0,1]

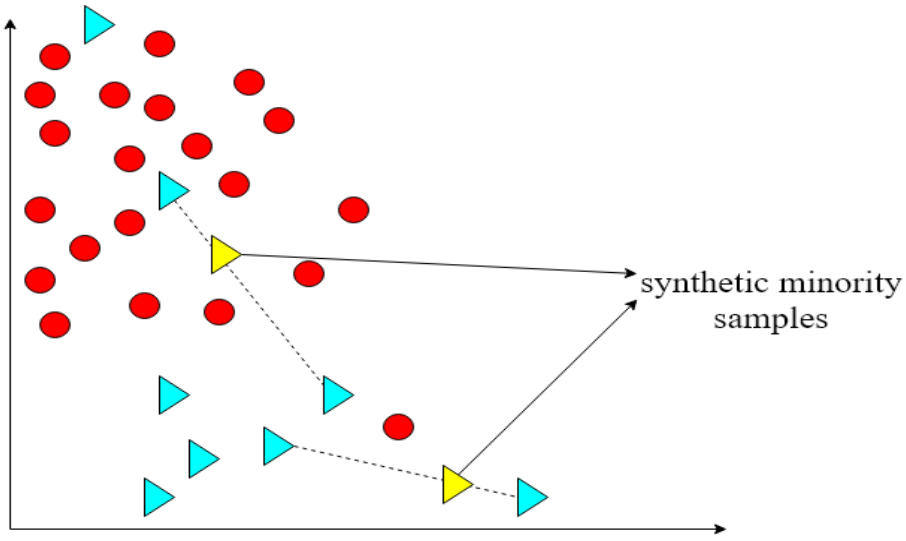

Figure 2.2: Generating artificial instances using SMOTE.

SMOTE Drawbacks

SMOTE has been proven to be a reliable technique to overcome the problem of over-fitting that might arise when duplicating existing minority samples in random oversampling. However, the algorithm still has some weaknesses. According to [35], SMOTE can effectively combat between-class imbalance, but it suffers from multiple issues:

- SMOTE chooses a minority sample to over-sample with uniform probability. Therefore, input areas counting many minority samples have a high probability of being inflated than sparsely populated minority areas, amplifying within-class imbalance and ignoring small disjuncts.

- SMOTE does not distinguish overlapping class regions from safe areas. It might linearly interpolate a minority sample located among majority class instances and its nearest minority neighbour, amplifying noise generation in data.

- Instances far from the class border are oversampled with the same probability as those close to the class boundary. SMOTE does not enforce the decision boundary.

2.3.2 Kmeans-SMOTE

Felix last et al. [35] proposed merging SMOTE with K-means clustering algorithm to combat within-class imbalance and inflating sparse minority samples areas. The algorithm consists of three main steps, clustering, filtering and oversampling.

1. In the clustering step, the original training data is clustered into k groups using kmeans [36, 37].

2. In the filtering step, clusters with a high proportion of minority samples are retained. The motivation for this step is to select clusters dominated by minority samples, targeting safe areas and alleviating noise amplification. Furthermore, more synthetic samples are assigned to clusters where minority samples are sparsely distributed to combat the within-class imbalance problem. By default, any cluster made up of at least 50% of minority samples is selected for oversampling. The default behaviour can be tuned by adjusting the imbalance ratio threshold (IRT). The imbalance ratio of a cluster c is defined in Eq 2.3

$$irt = \frac{majorityCount(c) + 1}{minorityCount(c) + 1} \tag{2.3}$$

3. In the final step, SMOTE is applied to each cluster independently to achieve the desired balance ratio. Filtered clusters are assigned sampling weights between zero and one. Clusters with high sampling weights correspond to low minority samples density areas yielding more synthetic samples for that cluster.

2.3.3 Noise removal using belief function theory

Since Vanilla SMOTE may create synthetic instances in noisy and overlapping areas, far from safe regions, F. Grina et al. [18] proposed a post-processing step for removing noisy generated instances after oversampling using SMOTE. The post-processing step is based on belief function theory (BFT). Using BFT, each generated instance can be represented by an evidential membership structure providing detailed information about class membership. Three rules based on BFT can be enforced to detect and remove noisy generated instances. The proposed method will only remove noisy generated instances.

Combining SMOTE and BFT

1. Generate synthetic minority samples using SMOTE.
2. For each synthetic minority sample, compute an evidential soft label using its nearest neighbors.
3. Identify synthetic minority samples that are generated in noisy areas using plausibility and belief functions.
4. Repeat steps two, and three until a user predefined minimum balance ratio Br_{min} is met.

Generating evidential soft labels

Let $\Omega = \{1,2,...,Q\}$, be the set of all classes, in binary classification $\Omega = \{1, 2\}$, x_i be a synthetic minority sample. We choose the k nearest neighbors of x_i. Each neighbor represents a piece of evidence for the class membership of x_i [38]. For each neighbor x_j, we calculate three mass functions using Eq 2.4, Eq 2.6, and Eq 2.7.

$$m_i^j(\{w_q\}|x_j) = \alpha\Phi_q(d_{ij}) \tag{2.4}$$

where:

- x_j is a nearest neighbor of the artificial instance x_i
- w_q is the class label of x_j

- d(x_i, x_j) is the euclidean distance between x_i, and its nearest neighbor x_j
- α is a fraction $\in$ [0..1]. It is recommended to use the value of $\alpha = 0.95$ in the experiments
- $\Phi_q(d_{ij})$ is a decreasing function verifying that $\Phi(0) = 1 =$ and $\Phi_{d\to\infty}(d) = 0$ [38].

$\Phi(d)$ is defined by Eq 2.5

$$\Phi = exp(-\gamma d^{\beta}) \tag{2.5}$$

In equation 2.5, γ is a constant > 0, a distinct value can be assigned for γ for each class q $\in \Omega$, and β can be chosen to be {1 or 2} [38]. Intuitively, equation 2.4 represents the belief that instances x_i, and x_j belong to the same class, or the degree of membership for minority class.

$$m_i^j(\Omega|x_j) = 1 - \alpha\Phi_q(d_{ij}) \tag{2.6}$$

Equation 2.6 represents the belief that the generated instance x_j is not a minority class instance.

$$m_i^j(A|x_j) = 0 \ \forall A \in 2^{\Omega} \setminus \{\{w_q\}, \Omega\} \tag{2.7}$$

Where

- 2^{Ω} is the power set of Ω. In binary classification where $\Omega = \{1, 2\}$. The power set 2^{Ω} would be equals to $\{\Phi, \{1\}, \{2\}, \{1, 2\}\}$.

We can clearly see that the belief for all subsets A of 2^{Ω} outside w_q, Ω is zero. We combine the evidential soft labels for each neighbor x_j. As a result, each synthetic instance has now three mass functions, which are

1. $m_i(w_1)$ degree of membership for majority class.
2. $m_i(w_2)$ degree of membership for minority class.
3. $m_i(\Omega)$ degree of membership for both classes.

Eliminating synthetic samples

For each synthetic minority sample, we calculate belief, and plausibility using Eq 2.8, and Eq 2.9.

$$Bel_i(\{w\}) = m_i(\{w\}) \tag{2.8}$$

$$pl_i(\{w\}) = Bel_i(\Omega) - Bel_i(\Omega \setminus \{w\}) \tag{2.9}$$

An elimination procedure follows the calculation of plausibility, and belief is based on one of the following rules:

- Overlap Threshold Rule: rules out synthetic samples located in strong overlap regions. A synthetic sample is rejected if the maximum plausibility pl_{max} (Eq 2.10) is less than a predefined threshold, $\beta_{pl} \in [0, 1]$. The higher this parameter is, the more synthetic instances will be rejected.

$$pl_{max} = max_{w \in \Omega} pl_i(\{w\}) \tag{2.10}$$

- Noise Threshold Rule: rules out synthetic samples that are suspected of belonging to a class that is not present in the training set. A synthetic sample will be rejected if the maximum belief Bel_{max} is less than a predefined threshold, $\beta_{Bel} \in [0, 1]$. β_{Bel} is set to be the minimum value of Bel_{Bel} across original minority samples.

$$Bel_{max} = max_{w \in \Omega} Bel_i(\{w\}) \tag{2.11}$$

- Misclassification Rule: rules out synthetic samples that are more likely to be misclassified after oversampling. Let w_1 be the majority class, and w_2 be the minority one. A synthetic minority sample x_i will be rejected if:

$$Pl_i(\{w_1\}) > pl_i(\{w_2\}) \tag{2.12}$$

The removal of noisy synthetic minority samples from the oversampled data set is an iterative process. Therefore, the process continues until we reach a predefined minimum Balance ratio Br_{min}.

2.3.4 Grouped SMOTE with noise filtering mechanism

Another variation for SMOTE proposed in [5]. The motivation for the proposed approach is to alleviate noise information propagation that could arise when applying vanilla SMOTE. The proposed approach consists of two main procedure, noise filtering and individual sampling.

Noise Filtering

Noise filtering is based on Gaussian Mixture Modeling (GMM). We use GMM to explore the distributions of minority and majority samples. We assume that several normal distributions or processes have produced the training data points; we can then employ Gaussian mixture to find the mean and standard deviation of each distribution or process. The number of distributions used to generate the observations is a crucial hyperparameter for GMM. Using a fitted GMM, we can calculate the probability density of each instance and rule it out based on simple criteria. The following enumeration provides the steps for noise filtering.

1. Split the data set into majority/minority subsets;
2. Construct a Gaussian Mixture for each subset, GMM^{-} (resp. GMM^{+}) for majority (resp. minority) subset;
3. For each instance we calculate the probability density in the same class p_i, and in the other class $p_i^{'}$ using GMM^{-}, and GMM^{+}.
4. If $p_i < \lambda \times p_i^{'}$, then remove this instance.
5. Repeat the procedure at 3, and 4 until all noisy instances have been detected.

Individual sampling

In this step, we divide the remaining minority class instances into three disjoint groups (i.e. safety, boundary, and outlier) depending on the data distribution. Each group will be allocated a different parameter k when oversampling using SMOTE [1]. In [5], the authors suggested to use the following values:

[1] k is the number of nearest neighbours considered for each minority instance

- $k = 5$ for the safety group;
- $k = 3$ for the boundary group;
- $k = 1$ for the outlier group, the reason for choosing that value is to avoid that any newly generated samples appearing in an inappropriate location.

2.3.5 SMOTE-LOF

SMOTE can produce artificial minority class samples considered as noise. Therefore, Asniar et al. [19] proposed a post-processing noise filtering procedure to identify such noisy artificial samples based on the Local Outlier Factor (LOF). Noise identification with LOF can be described as follows:

1. Select only artificial minority samples generated by SMOTE to work with;
2. Calculate the mean and standard deviation of each attribute in the filtered minority samples;
3. Perform normalization using Eq 2.13 on each attribute using the calculated mean and standard deviation;

$$Z = \frac{X - \mu}{\sigma} \tag{2.13}$$

   where, Z is the standard normal value, X is the original attribute value, and μ and σ are accordingly the distribution mean and standard deviation of the attribute X.

4. Compute the Euclidean distance between each pair of samples from the minority class;
5. Find the K nearest neighbors of each minority samples based on the calculated Euclidean distances;
6. Calculate the $k-distance$ for each minority sample using Eq 2.14. $k-distance$ is the maximum distance between a minority sample and one of its nearest neighbors;

$$k - distance(x) = max\{d(x, o)|o \in N_k(x)\} \tag{2.14}$$

where, $d(x, o)$ is the euclidean distance between an artificial instance x and one of its k-nearest neighbors o.

7. Determine the reach-ability_distance (Eq 2.15) between a minority sample and each of its nearest neighbors;

$$reach - ability_distance_k(x, o) = max\{k - distance(o), d(x, o)\} \quad (2.15)$$

where, o is a nearest neighbor to x, $k - distance(o)$ is described in Eq 2.14, and $d(x, o)$ is the Euclidean distance between x, and o.

8. The local reach-ability density for each minority sample is determined based on the reach-ability_distance, then, we calculate the LOF for x.

9. Finally, noisy samples are identified based on their LOF value.[2].

2.4 Hybrid-based approaches

2.4.1 K-neighbours oversampling with cleaning data

This section introduces an entirely new approach based on K nearest neighbours yet differs from the SMOTE policy while generating new samples. A data cleaning procedure is incorporated to enhance the quality of the generated data.

Budi Santoso et al. [39] proposed a new approach for handling the class imbalance problem. Unlike SMOTE, which employs only one minority sample when creating synthetic data, K-neighbours oversampling (KNOS) employs k minority samples in the generation of new instances. KNOS proceeds further with a data cleaning step that involves the removal of some majority samples. So far, it has only been applied for the Logistic Regression classifier. As one of our contributions, we extended KNOS by using several combinations with the classifiers described in Section 2.6 and experimented with several real-world data sets. Mainly, KNOS is performed using the following steps:

1. Set a value for k, the number of nearest neighbors involved in data generating procedure

[2]Only artificial minority samples generated by SMOTE have undergone the LOF procedure

2. Find the nearest neighbours for all original samples using Euclidean distance. Given two samples in n-dimensional space x_i and x_j, the Euclidean distance between data points in n-dimensional space is computed using the following formula:

$$d(x_i, x_j) = \sqrt{\Sigma_{k=1}^{k=n}(x_i^k - x_j^k)^2} \tag{2.16}$$

3. Keep only minority samples for which the k nearest neighbours are minority samples.

4. Generate new minority sample according to Eq 2.17

$$D_{new,k} = D_i + \Sigma_{j=1}^{j=k}(D_j - D_i) * \delta_j \tag{2.17}$$

where, $D_i \in S_{min}$ represents a minority class sample, k is the number of nearest neighbors employed in data generation procedure, and δ_j is .... [3]

5. Data cleaning: for each majority sample, identify its five nearest neighbours, then remove that sample if all its neighbours belongs to the minority class.

2.5 GAN-based approaches

In this section, we will present the core components of Generative Adversarial Networks, which are deep (have more than one hidden layer) neural networks capable of synthesizing new data, that is very consistent with the already existing ones.

Generative Adversarial Networks (GANs) [40, 41, 42] belongs to the generative models. Generative, because they can create new data based on its training data. As illustrated in Fig. 2.3, GANs have two principal components, a generator that generates new instances and a discriminator that validates the quality of the generated instances. Both components are Artificial Neural Networks (ANN) [43, 44].

[3] $\delta_1 \in [0, 1-\delta],\ \delta_2 \in [0, 1-\delta-\delta_1],\ \delta_3 \in [0, 1-\delta-\delta_1-\delta_2], \ldots,\ \delta_k \in [0, 1-\delta_1-\delta_2-\cdots-\delta_{k-1}]$

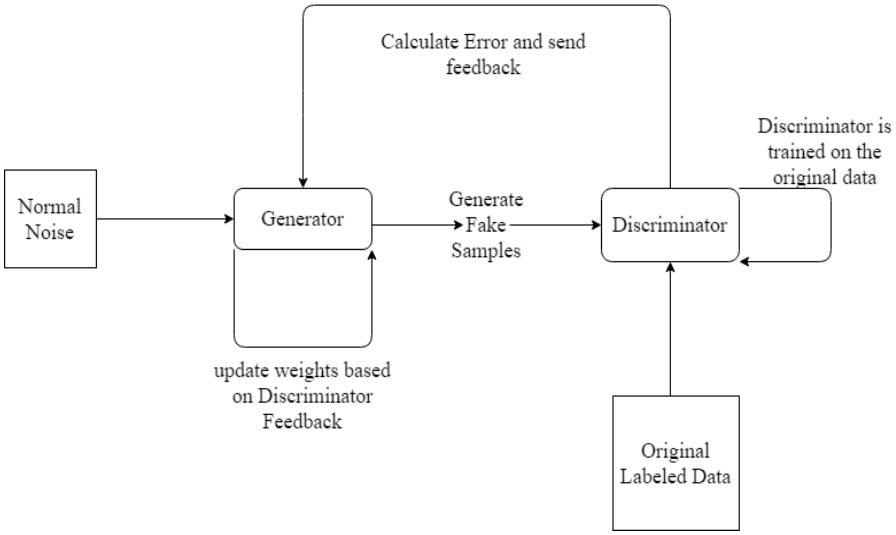

Figure 2.3: Vanilla GAN architecture.

2.5.1 The Generator

The generator receives as input a noise vector drawn from a normal distribution p and tries to map it to the original data distribution. The main task of the generator is to create samples that are similar to those in the original data [41, 44].

2.5.2 The Discriminator

The discriminator has two kinds of input data, original samples and fake samples created by the generator. The output of the discriminator is a single scalar representing the probability that the data created by the generator came from the original training set [40, 45]. When training both the networks simultaneously, they will get better by competing against each other, hence the name Generative Adversarial Network. The discriminator will get better at distinguishing fake samples from real ones, and the generator will output data that resembles the original [43, 44].

2.5.3 Cost function

On of the widely used loss functions when training a GAN [46] is the binary cross entropy, defined by Eq 2.18 as a cost function. This cost function is widely used in binary classification tasks when training a logistic regression or a neural network

classifier.

$$J(\theta) = -\frac{1}{m}\Sigma_{i=1}^{i=m}(y^{(i)}log(h_\theta(x^{(i)})) + (1 - y^{(i)})log(1 - h_\theta(x^{(i)}))) \tag{2.18}$$

Some remarks on binary cross entropy

- The cost is the average of all losses. The loss is the difference between the true label and the predicted one. We take the average by dividing by the size of the training data set.
- The binary cross entropy is an implementation of the maximum likelihood estimation, and since we want to minimize this estimation we have to multiply by (-1).
- $y^{(i)}$ is the true label for the i^{th} sample in the training data.
- $h_\theta(x^{(i)})$ is the predicted label for the i^{th} sample in the training data.

The cost for the generator is calculated as follows:

1. The generator creates fake data;
2. The discriminator evaluates the generated samples and outputs fake labels.
3. The generator updates its weights using the generator loss which is defined as the difference between fake labels and one.

The cost for the discriminator is calculated as follows:

1. The discriminator outputs fake labels for the generated samples and genuine labels for the original data;
2. the discriminator updates its weights using the fake and actual losses. The fake loss (resp. actual loss) is the difference between fake labels and zero (resp. genuine labels and one).

In a nutshell, we update the parameters for the generator to minimize log(1 - D(G(Z))), where Z is a noise input drawn from a prior normal distribution, G is the mapping function the generator is trying to learn, and D is the mapping function for the discriminator. On the other hand, the discriminator is trying to minimize

log(D(X)), where X is the original training data [45]. According to [43], the use of GANs has some disadvantages and complications. For instance, it is challenging to generate discrete data like text data. They might require considerable processing power depending on the number of trainable parameters. Also, their training can be unstable, or they can lead to an overfitting situation.

2.5.4 Oversampling minority classes using GANs

The efficiency of GANs has been widely explored in image data sets. They were some attempts to use GAN with gene expression datasets as well [47], but for the class imbalance problem where most real-world applications have standard numerical data sets, few attempts has been proceeded. In [43], the authors used GAN in two applications that require numerical databases, for oversampling the minority class, for generating synthetic data to keep the privacy of the original data sets. In our thesis, we are more interested in the use of GAN for oversampling the minority class. The efficiency of GAN for oversampling has been extended for more real-world data sets.

The GAN-based oversampling method proposed in [43] and described in Fig. 2.4, involves the following steps:

1. Separate the minority class from the rest of the training data;
2. Train the GAN using only the minority samples. Minority labels are included as an attribute in the training;
3. Use the GAN to generate new minority samples until the desired balance ratio is reached.

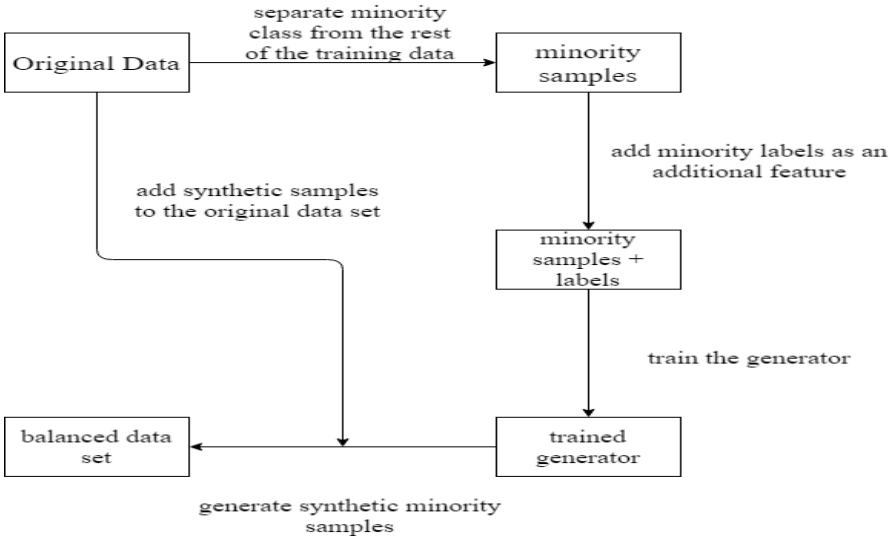

Figure 2.4: Oversampling using GAN.

2.5.5 Generative adversarial minority oversampling

In the previous section, we saw how Vanilla GAN had been successfully used as an oversampling technique to combat the CIL problem. This section introduces a new variation of GAN called Generative Adversarial Minority Oversampling (GAMO) for handling class imbalance within multi-class image data sets. The main problem with Vanilla GAN is that the data generated are likely to lie near the mode(s) of the minority class(es). The authors in [48] proposed a new variation named GAMO for tackling the CIL problem in multi-class image data sets. It is important to note that the GAMO consists of three main components:

- With the help of a classifier, a convex generator will eventually generate points that the classifier will misclassify. If left unchecked, the generated points will not coincide with the distribution of the intended minority class, which will lead to high misclassification of other class instances. To prevent that, we generate new points as a convex combination of existing points from the minority class under concern of Eq 2.19;

$$G(z|i) = g_i(t(z|i))^T X_i \tag{2.19}$$

  z is a latent variable drawn from a standard normal distribution, and t is a mapping from z to an intermediate space. X_i represents the existing points of i, i.e., the minority class in question. While g_i is the mapping from the

intermediate space $t(z|i)$ to a vector of n_i weights using softmax activation function.

- A real/fake discriminator to ensure that the generated samples do not fall outside the distribution of the intended minority class;
- A multi-class classifier that will guide the generator to produce new points at regions where the minority class in question is prone to misclassification.

2.6 Classification methods

Classification is a type of supervised machine learning. In supervised learning, both the data and labels are present, and this allows the learning algorithm to induce the hidden relationships between data and labels and produce a model represented by a set of parameters. In classification, the target label is discrete. The purpose of an oversampling technique is to enhance the performance of the existing classifiers regarding different evaluation metrics. We will describe in details the evaluation metrics used for CIL in Section 3.3.1.

This section will discuss different classification models that we will be using in our experiments—for instance, Decision Trees (DT), Logistic Regression (LR), Support Vector Machines (SVM), and K Nearest Neighbors (KNN).

2.6.1 Decision tree

A DT is a non-parametric model [1] where every leaf node is labelled with a class name, the root, and every internal node has at least two children and a specific label of a given attribute. It is essential to mention that root's branches and internal nodes are labelled with disjoint values or sets of values of the node's attribute such that the union of these values or partitions comprises the set of all possible values of that attribute.

Given a dataset, we can construct its decision tree by picking up the attribute that best separates the training set into two subsets (the possible split is when the samples in each subset belong to the same class) as a root. Iteratively redo the same

procedure for each child node until we meet one of the stopping criteria. Mainly, there three scenarios where the iterative procedure stops, which are:

- The training examples in each subset belong to one class;
- The depth of the tree reached a predefined threshold;
- The lower bound on the number of elements each subset must have to be partitioned further is reached.

2.6.2 Logistic regression

LR is a classification model that is based on linear regression. The two main differences between logistic and linear regression are that LR uses binary cross-entropy as a cost function (Eq 2.18), and since it should output a zero or one, it applies a sigmoid function σ given by Eq 2.21 that outputs a value in the interval [0, 1] [49]. The general form for logistic regression is:

$$h_\theta(x) = \sigma(\Sigma_{j=0}^{j=n}(\theta_j x_j)) \tag{2.20}$$

where x_0 is the intercept value, and $x_1, \ldots, x_n$ are the input features. Each feature is weighted by a parameter θ_j. Logistic regression parameters are learned using some optimization algorithm like gradient descent.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.21}$$

Logistic Regression parameters are learn-able using some optimization algorithm like gradient descent. Finally, to obtain a decision boundary that clearly separates between two classes, we define a simple rule to further compress the output of σ to one of two values, zero or one [49]

$$y_z = \begin{cases} 1, \ if \ \ \sigma(z) \geq \varphi \\ 0, \ if \ \ \sigma(z) < \varphi \end{cases}$$

A popular choice for φ that is widely used in practice is 0.5.

2.6.3 Support vector machine

SVM is used mainly for pattern recognition and classification tasks. A linear SVM tries to find a decision boundary that best separates the data points in their correct classes (Figure 2.5). The further away the data points are from the decision boundary, the more reliable and accurate the model will be. Support Vectors are the closest data points from both classes to the decision boundary. Excluding support vectors from the data set will reduce the chance of correctly classifying unseen data points. If the data points are not linearly separated, they are projected to a higher dimensional plane. The process of projecting data points to a higher dimensional plane is called kernelling [49].

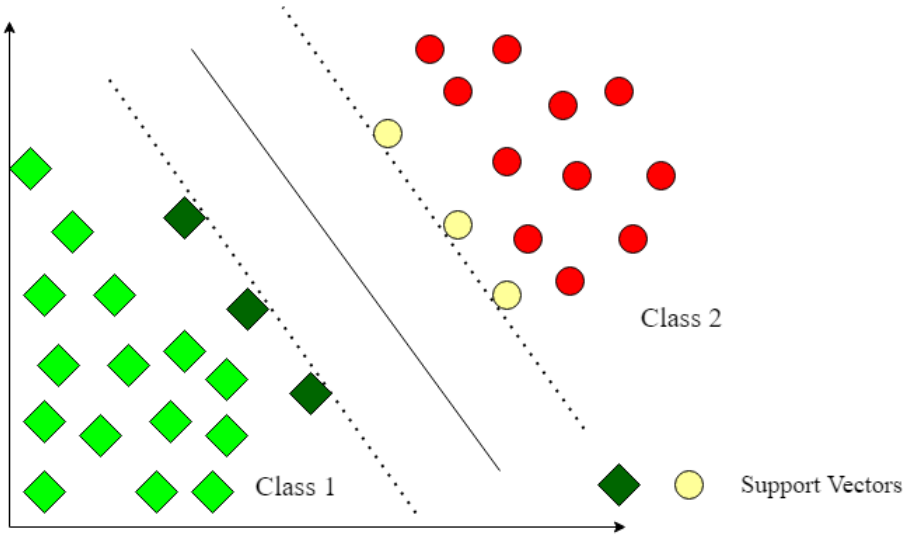

Figure 2.5: Example of a linear-SVM in 2-dimensions

2.6.4 K-nearest neighbours

KNN is a lazy learning algorithm. It is non-parametric, meaning it does not have any learnable parameters that can be updated using some optimizing algorithm like gradient descent. However, it is still a very effective classification method. KNN is based on similarity measurement between data points (usually, the similarity measurement used is Euclidean distance). The number of nearest neighbours K considered for classifying a new data point and is a crucial hyper-parameter for the algorithm [49].

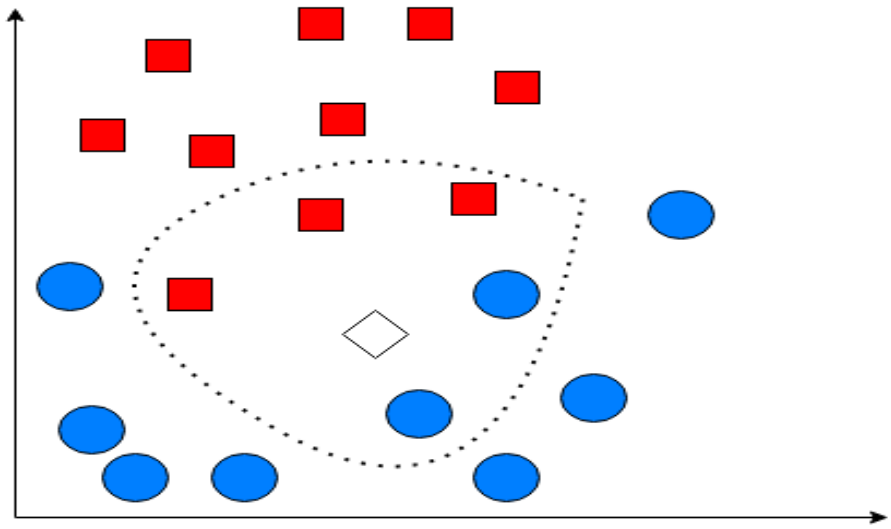

Figure 2.6: Example of 5-NN algorithm in 2-dimensions

Figure 2.6 describes an example using five nearest neighbors ($K = 5$). The diamond data point is an unseen example. The algorithm finds the first five nearest neighbours and uses majority voting to determine its class. From the depicted example, we can deduce that the resulting class will be a red square.

Chapter 3

Experiments and Results

This chapter contains the practical section of our work in which we describe the experimental environment, data sets, oversampling techniques settings and conclude with the results and detailed discussion concerning the pros and cons of each method used. It also introduces the experiments conducted for combating CIL problems. We used a variety of software packages to speed up the research process.

3.1 Software packages

In this section, we present the set of packages we used in our experiments. These packages are python-based and are pre-installed inside the development environment except for Kmeans-SMOTE, which we had to install manually.

3.1.1 Numpy

Numerical python is a standard package for performing vector and arrays operations [50]. Through all the experiments, we will mainly deal with 2d-arrays.

3.1.2 Pandas

Python data analysis is an open-source library for data manipulation and analysis. It offers two primary data structures, Series for dealing with time-series data and DataFrames for storing SQL-like tables. We used pandas to read data sets stored in CSV files (comma-separated values) and store them in data frames.

3.1.3 Sci-kit learn

A popular package that offers a wide range of supervised and unsupervised machine learning algorithms, as the main target of our experiments is to evaluate the performance of different classifiers before and after oversampling. We used the implementation of sci-kit learn for Logistic Regression (LR), K-Nearest Neighbours (KNN), Support Vector Machine (SVM), and Decision Tree (DT) [51].

3.1.4 TensorFlow

TensorFlow [52] is an end-to-end open-source platform for machine learning. One of its main use cases is to build deep neural networks. We will be using Keras, a subpackage of TensorFlow, for building and training generative adversarial networks. Keras provides the sequential API for efficiently stacking neural network layers and building sophisticated models in a few lines of code. It is also possible to write customized models.

3.1.5 Imblearn

A standard package in the CIL field offers an implementation of a variety of Data balancing algorithms.

3.1.6 Kmeans-SMOTE

This software package provides an implementation for Kmeans-SMOTE 2.3.2.

3.1.7 SMOTE-BFT

Provides an implementation for SMOTE with Belief Function Theory 2.3.3.

3.2 Used datasets

In this section, we present the data sets used in the experiments. Table 3.1 provides all the details for each data set, number of features, number of observations, positive samples ratio ($ratio^+$), negative samples ratio ($ratio^-$), imbalance ratio

(IR). All data sets have been retrieved from UCI and KEEL (Knowledge Extraction based on Evolutionary Learning) repositories.

Data set	No. features	No. samples	$ratio^+$	$ratio^-$	IR
Haberman	3	306	27.46	73.54	2.78
Vehicle1	18	846	25.65	74.35	2.9
default of credit card	23	30000	22.12	77.88	3.52
new-thyroid1	5	215	16.28	83.72	5.14
Bank Marketing	16	45211	11.7	88.3	7.55
page blocks	10	5472	10.21	89.79	8.79
yeast-0-2-5-7-9_vs_3-6-8	8	1004	9.86	90.14	9.14
Vowel0	13	988	9.11	90.89	9.98
car-good	6	1728	3.99	96.01	24.04
winequality-red-4	11	1599	3.31	96.69	29.17
winequality-white-9_vs_4	11	168	2.98	97.02	32.6
yeast5	8	1484	2.96	97.04	32.73
poker-8-9_vs_6	10	1485	1.68	98.32	58.4
shuttle-2_vs_5	9	3316	1.48	98.52	66.67
abalone91	8	4174	0.77	99.23	129.44

Table 3.1: Binary data sets used in the experiments

3.3 Experimental framework

We used a pipeline described in Fig. 3.1 for the conducted experiments.

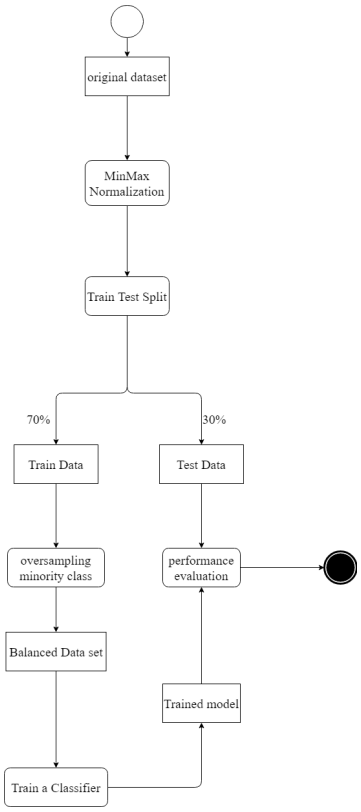

Figure 3.1: Experimental Framework

Below, We provide a brief discussion about each step in illustrated in Fig. 3.1.

1. The first step is to normalize the data by interpolating each attribute values to the range {0..1}. The theory behind normalization is that attributes might be on different scales, so that attributes with large ranges will cancel the importance of attributes with small ranges. Normalization aims at making all attributes at the same scale. Thus all attributes have the same influence. A widely used data normalization technique is the MinMax. Each attribute is transformed using equations Eq 3.1 and Eq 3.2.

$$X_{std} = \frac{X - min}{max - min} \tag{3.1}$$

$$X_{scaled} = X_{std} \times (max - min) + min \tag{3.2}$$

where X_{std} is the standard deviation for each attribute. min, and max are the attribute range. X_{scaled} is the transformed value.

2. In the next step, we split the original data set into train and test subsets. We will only add artificial samples to the train data set. We aim to investigate

how changing the original data distribution by applying oversampling affects the performance of a model on a unified and isolated test set derived from the original data. We have chosen to keep 30% of the original data as test data for all the experiments and apply oversampling for the remaining 70% data.

3. After that, we investigate the performance of different models on the test set without using any oversampling techniques. This step serves as a baseline for the evaluation metrics used in the experiments. We provide a detailed description of evaluation metrics in Section 3.3.1

4. Next, we apply an oversampling technique to the train set. We have chosen a variety of techniques for comparing the performance, they are mainly generative adversarial networks 2.5, Kmeans-SMOTE 2.3.2, SMOTE-BFT 2.3.3, and KNOS 2.4.1. We added some implementation details for each of the used techniques in subsequent sections.

3.3.1 Evaluation metrics

To evaluate the classification results, we count the number of True positives (samples that are actually positive and classified as positive), True negative (samples that are actually negative and classified as negative), False positive (samples that are actually negative but classified as positive), and False negative (samples that are actually negative but classified as negative). Accuracy is a widely used metric in classification 3.3, but it is not appropriate in the context of CIL because it does not distinguish between the number of correct labels of different classes [1]. For example, in a data set where $IR = 9$, for each minority or positive sample, there exists nine majority or negative samples in return, if the classifier detects all negative samples but recognizes all positive samples as negative, then the accuracy, in this case, would be 90% which does not reflect the correct number of detected labels in the minority group.

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{3.3}$$

where TP (True Positive), TN (True Negative), FP (False Positive), and FN (False Negative). A number of popular metrics that are commonly used in the context of CIL are [1, 53, 54]:

- True Positive Rate (TPR), also called sensitivity or recall, is the percentage of positive samples that were correctly classified, or the ability of a model to detect all positive examples, it is given by Eq 3.4

$$TruePositiveRate = \frac{TruePositive}{TruePositive + FalseNegative} \tag{3.4}$$

- True Negative Rate (TNR), also called specificity, is the percentage of negative samples that were correctly labeled, and it is given by Eq 3.5

$$TrueNegativeRate = \frac{TrueNegative}{TrueNegative + FalsePositive} \tag{3.5}$$

- Receiver Operating Characteristics (ROC) is a popular tool for visualizing, organizing and selecting classifiers based on their trade-offs between benefits (true positive) and true negative (false positive). A quantitative representation of ROC is the Area under it, and hence the name Area Under The Receiver Operating Characteristic. ROC AUC is the arithmetic mean of True Positive Rate, and True Negative Rate [1], and it is given by Eq3.6 or Eq3.7

$$AUC = \frac{1 - TP_{RATE} + FP_{RATE}}{2} \tag{3.6}$$

or

$$AUC = \frac{TP_{RATE} + TN_{RATE}}{2} \tag{3.7}$$

- Precision the percentage of samples that were correctly classified as positive, or the ability of a classifier not to classify as positive a negative sample, it is given by Eq 3.8

$$Precision = \frac{TP}{TP + FP} \tag{3.8}$$

Precision is an important metric when we want to investigate how reliable a classifier is in positive diagnosis of a rare disease [1].

- F-measure investigates the trade-off between precision and recall [5], and it is given by Eq 3.9

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{3.9}$$

- Geometric mean which is a trade-off between the accuracy of majority class and that of minority class [] and it is given by Eq 3.10

$$G - mean = \sqrt[n]{sensitivity * specificity} \tag{3.10}$$

where n is the number of classes. In binary classification, Eq 3.10 will translate to the square root.

Table 3.2 summarises advantages and disadvantages of mostly used evaluation measures for imbalance learning problems:

Table 3.2: Evaluation metrics comparison

Metric	Advantages	Disadvantages
ROC-AUC	• ROC analysis does not have any bias toward models that perform well on the minority class at the expense of the majority class—a property that is quite attractive when dealing with imbalanced data.	• Under class rarity, that is, when the problem of class imbalance is associated to the presence of a low sample size of minority instances, as the estimates can be unreliable.
Recall	• Tells us how good a classifier is in detecting all positive samples (anomalies in security-related data sets, or rare diseases in medical applications) • All oversampling methods aim at improving this metric	• Improving this metric might come at the expense of decreasing the precision. • Most classifiers have low value for the baseline recall.

Precision	• Tells us how much noisy data were generated during oversampling (indicated by false positives) • takes on high values for baseline classifiers for most cases.	• Negatively affected by most oversampling methods.
F-measure	• When the performance on both classes i.e. majority and minority classes is expected to be high, then, *F-score* is a good measure. • Tells us if increasing the recall with oversampling at the expense of decreasing the precision is better or not.	• Can take on low values for imbalanced data sets or may be ill-defined
G-mean	• Similar to *F-score*, when the performance on both classes i.e. majority and minority classes is expected to be high, then, *G-mean* is a good measure.	• Since it is directly related to recall metric, it can take on low values for imbalanced data sets or may be ill-defined.

3.3.2 Oversampling methods

GAN architecture

The used GAN has the following parameters:

- Since we are dealing with standard numerical data, we use Dense layers instead of convolution ones in both the generator and the discriminator.
- The use of Leaky Relu 3.3.2 as an activation function in hidden layers with a negative slope of 0.2.
- Dividing the input data set into batches of $size = 5$.
- Adam optimizer with $learning\ rate = 0.0002$;
- Using batch normalization in the generator. Batch normalization is an important technique for making the learning more stable especially for deep neural networks. It works by normalizing the input to each unit to have zero mean and unit variance [55].
- The use of dropout [56] in the generator with a probability of 0.3. It means that each epoch 30% of the hidden neurons will have zero activation.
- In the generator, if there is more than one dense layer, the number of hidden neurons are put in ascending order.
- In the discriminator, if there are more than one dense layer, the number of hidden neurons is put in descending order.

We have used one architecture for our GAN in which the generator is made up of three components of Dense $\rightarrow$ BatchNormalization $\rightarrow$ LeakyRelu $\rightarrow$ Dropout while the discriminator is made up of three components of Dense $\rightarrow$ LeakyRelu

Leaky Relu

Leaky Relu is a widely used non-linear activation function in hidden layers in deep neural networks. It defined as

$$a(z) = \begin{cases} z, \ \ if \ \ z \geq 0 \\ \frac{z}{\alpha}, \ \ if \ \ z < 0 \end{cases}$$

where α is a fixed parameter in the range {1,+∞}. Leaky Relu is an improvement over the standard Relu activation function **??**. The parameter α has been introduced to avoid having zero gradients when the input z is ≤ 0 [57].

$$a(z) = \begin{cases} z, & if \ \ z > 0 \\ 0, & if \ \ z \leq 0 \end{cases}$$

K-means SMOTE

For each data set, we ran k-means-smote with four different values of k (the number of clusters in k-means step). Table **??** contains the set of k values chosen for every data set in the experiments. However, in order to make the results more readable, we included only the Kmeans-SMOTE instance with the hyper-parameter k that corresponds to the best results obtained so far.

Data set	The set of k values
bank marketing	{1000, 1500, 3000, 5000}
haberman	{15, 20, 30, 40}
default of credit card	{20, 100, 250, 500}
page Blocks	{20, 50, 100, 250}
yeast5	{50, 75, 100, 125}
shuttle-2_vs_5	{20, 50, 100, 250}
poker-8-9_vs_6	{70, 100, 150, 250}
vehicle1	{15, 30, 50, 100}
abalone19	{20, 50, 100, 250}
vowel0	{30, 50, 100, 150}
car-good	{200, 250, 400, 500}
thyroid1	{5, 10, 20, 50}
yeast-0-2-5-7-9_vs_3-6-8	{10, 20, 50, 100}
winequality-red-4	{20, 50, 100, 250}
winequality-white-9_vs_4	{5, 10, 20, 50}

Table 3.3: The set of k clusters in k-means step - values for every data set

other hyper-parameters are described in the following list

- The imbalance threshold ratio (ITR) has been set to the default value, which is one. It means for any cluster resulting from k-means to be oversampled. It should have at least 50% of its samples belonging to the minority group;
- The number of nearest neighbours to consider when applying smote in the next step has been left to the default value, which is five;
- The sampling strategy for SMOTE is left to the default value, which is 1.0 meaning the algorithm will oversample the minority class till the IR becomes one.

SMOTE - Belief Function Theory

The hyper-parameters for SMOTE are left to the default (over-sampling strategy = 1.0, and the number of nearest neighbours is five). Other hyper-parameters are

- α a constant $\in [0..1]$ that is used in class membership calculation for an artificial minority sample. $\alpha = 0.95$ in all experiments;
- Minimum plausibility (pl_{min}) it is used to rule out synthetic minority samples in overlapping areas having maximum plausibility less than pl_{min} 2.3.3;
- Proportion over-sampling strategy for SMOTE, = 1.0 in all experiments;
- Minimum proportion The method will iteratively remove any minority sample that violates any of the rules described in 2.3.3 until the IR reaches a predefined threshold ($proportion_{min}$);
- Number of nearest neighbours to consider for membership calculation. we set it to seven in all experiments.

K-neighbour oversampling

By default, the algorithm will remove any minority sample in the filtering step with all the first five nearest neighbours belonging to the majority group. We have slightly modified this step in our experiments. In the modified version, we have considered the number of majority samples in the filtering step as a hyper-parameter, and for that, we introduce KNOS-3, where the algorithm will remove any minority sample with at least three majority samples among the first five nearest neighbours. Furthermore, we introduce KNOS-5, where the algorithm will remove any minority sample if all the first five nearest neighbours belong to the majority group.

Table 3.4 gives a an overview about the advantages and disadvantages of the studied oversampling methods:

Table 3.4: Comparison between the studied oversampling methods

Method	Advantages	Disadvantages
Kmeans-SMOTE	• Can significantly mitigate within-class imbalance problem when using with logistic regression. This is indicated by the significant improvement in terms of F-measure and Precision.	• The number of clusters indicated by k parameter needs to be manually set.
SMOTE-BFT	• Delivers optimal results with DT, SVM, and KNN. • A recommended method for extreme imbalanced data sets in order to improve Recall.	• Does not work well with data sets having few number of instances and small IR • Not applicable when the number of minority samples is less than the minimum required of nearest neighbors in the filtering step

GAN	• Since it is a neural network-based solution, we can easily test different architectures in our search for optimal solution using the available DNN software packages. • It can deliver good results if the generator successfully approximated the underlying minority class distribution.	• Needs a significant amount of data samples to train. • Training a GAN might be difficult and tricky.
KNOS	• A nearest-neighbor approach making it in essence similar to SMOTE. • It tries to alleviate noise by eliminating noisy minority instances from majority areas. • KNOS-5 is a proposed modification to mitigate the number of minority samples excluded during filtering step.	• Relying on all minority instances to generate artificial ones is impractical as SMOTE-BFT still outperforms it. • Inapplicable for highly overlapping data sets as all minority instances will be marked as noise in the filtering step.

The following section will present the results of our experiments and discuss the pros and cons of each oversampling method.

3.4 Results and discussion

From Table 3.5 to Table 3.8, the comparative results for the studied oversampling methods with four different classifiers on 15 data sets are given in detail. We only reported the Recall metric here; the remaining experimental results are attached in Appendix A. The mean has been recorded with the standard deviation below the mean and between brackets where it is required. The best result for each data set has been highlighted in boldface. For Kmeans-SMOTE, we have only given the best results that correspond to the optimal number of clusters. K parameter (next to the data set name) indicates the optimal number of clusters for Kmeans-SMOTE. By the results in these tables, we observe:

1. For well-separated data sets like shuttle-2_vs_5, the baseline performance, which uses only the original training data, was excellent for some classifiers (like decision trees), which achieved a maximum score all metrics. Adding artificial instances did not negatively affect any metrics, implying that the different oversampling techniques did not introduce any noisy data.

2. For some data sets KNOS, and KNOS-3 are not applicable (NA), meaning that the filtering step rejected all minority samples.

3. For some data sets, we observe the ill-defined (N/D) value for baseline performance and some oversampling techniques. It indicates that the classifier could not distinguish between a positive sample and a negative one (a roc-auc value of approximately 50 and ill-defined for other metrics).

4. For some data sets like (winequality-white-9_vs_4), SMOTE-BFT is not applicable since the number of minority samples is less than the required number of nearest neighbours in filtering step. Other oversampling methods are either not applicable (KNOS, KNOS-3) or generate bad instances (GAN). Surprisingly, the proposed approach KNOS-5 has given the best results demonstrating that KNOS-5 is a promising method.

5. Both SMOTE-BFT and KNOS-5 have given similar performance with logistic regression classifier when comparing recall metric indicating that KNOS-5 can perform well with logistic regression classifier.

6. The best preferable oversampling method for the decision tree, SVM, and KNN is SMOTE-BFT since it defeats other oversampling methods in recall and F-measure

scores. This fact indicates that the classifiers mentioned above are not affected by the within-class imbalance resulting from applying smote, unlike logistic regression, which is negatively affected by this phenomenon and can be solved using Kmeans-SMOTE.

7. The precision for baseline performance is better than all oversampling methods except for decision tree classifier where SMOTE-BFT has given the best precision results, and for logistic regression, the baseline and Kmeans-SMOTE performance were similar. For SVM and KNN, oversampling methods negatively affect precision, as we can observe from the results, which indicates the following:

    - The precision for classifiers like SVM and KNN decreased after oversampling because the number of false positives has increased (majority instances that were classified as minority ones).
    - Decision tree is more robust against misclassifying negative samples as positive ones when using SMOTE-BFT as an oversampling technique.
    - baseline and Kmeans-SMOTE introduce a less number of false positives when using with logistic regression.

8. If we consider extreme imbalanced data sets, the ones with an imbalance ratio higher than 100:1 like in the abalone19 data set, we can observe that SMOTE-BFT works deliver the best performance to all classifiers in terms of recall and G-mean. On the other hand, Kmeans-SMOTE can be a suitable method for such data sets to improve F-measure and precision.

9. Kmeans-SMOTE is the preferable method for logistic regression in terms of F-measure and precision. This is because inflating sparse minority areas will highly likely render the data linearly separable, which is a tremendous advantage for logistic regression.

10. Precision and F-measure are good metrics to investigate when tackling within-class imbalance.

11. For data sets with a small number of samples and an average imbalance ratio like inequality-white-9_vs_4, the modified version of KNOS, KNOS-5, and Kmeans-SMOTE are the preferable methods, as other oversampling techniques are not applicable due to a variety of restrictions on the minimum of required minority samples, which can be considered as a significant advantage for both oversampling techniques.

12. Two main drawbacks for SMOTE-BFT are worth mentioning is that it is not applicable when the number of minority instances should be greater than the minimum required number of nearest neighbours needed for class membership calculations in the noise filtering step. Also, it does not work well for data sets with a small number of instances, and a small imbalance ratio, like Haberman and vehicle1 data sets. Other than that, it can deliver an optimal solution in almost all scenarios.

13. The main drawback for original KNOS does not work well with highly overlapping data sets since it is highly likely that all the minority instances will be rejected. We overcome this drawback by modifying the number of majority instances that are part of the nearest neighbours required for a minority instance to be rejected. KNOS and its modification can be considered the best oversampling method when working with data sets with small instances and a small IR.

14. GANs are considerably a novel solution in balancing standard numerical data sets. Two main drawbacks for GANs are discussed here. The first one is what is known as mode collapse [58] when training vanilla-GAN on a data set with multiple classes or mode(s). We overcome this issue by training GAN on the minority class instances using only one mode in this case. The second drawback is that, like any other deep neural network, it needs a significant number of instances to train it well. Nevertheless, GAN is not recommended for oversampling minority classes, as we can observe from the results illustrated in the tables below.

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=15)	30.714 (2.857)	34.286 (5.838)	28.571	37.143 (2.857)	NA	**45.217 (2.13)**	32.174 (3.478)
vehicle1 (k=30)	42.154 (3.454)	47.815 (2.38)	56 (1.569)	53.538 (1.794)	**66.364 (3.09)**	46.552 (4.223)	27.586 (2.181)
default of credit card (k=2-)	42.229 (0.392)	42.27 (0.771)	**45.282 (0.518)**	41.924 (0.186)	41.016 (0.653)	35.741 (0.508)	17.763 (0.309)
new-thyroid1	88.889	88.889	88.889	88.889	**93.75**	77.778 (0.000)	75 (0.000)
bank marketing (k=1000)	47.283 (0.512)	48.39 (0.992)	**56.59 (0.23)**	46.662 (0.346)	47.464 (0.78)	41.09 (0.395)	27.678 (0.285)

page-blocks (k=20)	84.42 (1.071)	85.039 (0.548)	89.061 (1.071)	**92.338 (0.26)**	85.802 (0.676)	83.375 (1.879)	70.625 (0.559)
yeast-0-2-5-7-9_vs_3-6-8 (k=50)	79.13 (4.26)	89.576 (2.248)	**94.545 (2.969)**	85.455 (3.534)	77.037 (7.909)	58.621 (0.000)	46.897 (2.759)
vowel0 (k=100)	95.758 (1.485)	97.939 (1.74)	**99.394 (1.212)**	93.333 (4.454)	74.615 (3.077)	71.667 (1.667)	56.667 (3.333)
car-good (k=400)	**91.818 (1.818)**	86.182 (7.527)	72.727 (0.000)	77.273 (0.000)	NA (0.000)	77.273 (0.000)	N/D
winequality-red-4 (k=50)	23.333 (2.222)	27.333 (5.097)	12.222 (2.222)	**44.444 (3.514)**	NA	NA	4.615 (3.768)
winequality-white-9_vs_4	N/D	N/D	NA	N/D	NA	NA	**50**
yeast5 (k=125)	66.667 (0.000)	60 (15.061)	**76 (3.266)**	61.333 (7.775)	63.636 (0.000)	55.556 (0.000)	N/D
poker-8-9_vs_6 (k=70)	N/D	45.333 (17.588)	33.333 (0.000)	**100**	N/D	16.667 (0.000)	N/D
shuttle-2_vs_5	**100**	**100**	**100**	**100**	**100**	**100**	**100**
abalone19 (k=250)	N/D	16.923	**58.462 (0.000)**	N/D	NA	NA	5 (6.124)
winning times	2	1	**7**	4	3	2	2

Table 3.5: Recall results for decision tree

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=40)	3.571	42.857 (11.518)	46.429	42.857	NA	39.13	**47.826**
vehicle1 (k={50, 100})	16.923	44.615 (2.574)	63.077	44.615	60.606	72.414	**77.586**
default of credit card (k=20)	21.73	36.997 (6.729)	64.529	25.038	43.801	48.825	**65.832**
new-thyroid1 (k=5)	22.222	77.778 (14.055)	88.889	88.889	**100**	**100**	91.667

bank marketing (k=1000)	19.497	45.006 (6.468)	**78.365**	42.721	53.178	66.935	77.523
page-blocks (k=20)	53.704	57.127 (7.644)	**85.635**	65.584	75.309	82.5	83.75
yeast-0-2-5-7-9_vs_3-6-8 (k=10)	56.222	80 (17.313)	**96.97**	84.848	80.556	79.31	86.207
vowel0 (k=50)	60.606	83.03 (3.09)	**100**	87.879	88.462	87.5	95.833
car-good (k=500)	N/D	**61.818 (7.925)**	59.091	54.545	NA	27.273	59.091
winequality-red-4 (k=100)	N/D	**67.778 (2.222)**	66.667	33.333	NA	NA	53.846
winequality-white-9_vs_4	N/D	**100**	NA	N/D	NA	NA	**100**
yeast5 (k=50)	N/D	93.333 (8.433)	**100**	86.667	**100**	**100**	**100**
poker-8-9_vs_6 (k={70, 100})	N/D	13.333 (16.33)	N/D	**66.667**	16.667	50	**66.667**
shuttle-2_vs_5 (k=250)	8.333	86.667 (10.992)	**100**	70.833	**100**	**100**	**100**
abalone19	N/D	61.538	**90**	60	NA	NA	37.5
winning times	0	3	**7**	1	3	3	**7**

Table 3.6: Recall results for logistic regression

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=20)	7.143	27.857 (3.499)	**64.286**	39.286	NA	30.435	47.826
vehicle1 (k=50)	29.231	37.705 (2.933)	73.846	44.615	54.545	68.966	**82.759**
default of credit card (k=20)	31.298	36.56 (1.72)	56.997	31.807	39.787	49.081	**58.121**

new-thyroid1 (k={10,20,50})	77.778	86.667 (4.444)	**100**	88.889	93.75	**100**	**100**
bank marketing (k=3000)	15.472	23.409 (2.212)	**79.748**	22.695	17.118	53.994	74.056
page-blocks (k=50)	59.877	47.25 (1.611)	**92.818**	68.182	76.543	86.25	92.5
yeast-0-2-5-7-9_vs_3-6-8 (k=10)	82.609	93.333 (3.534)	**96.97**	90.909	80.556	75.862	75.862
vowel0 (k=30)	96.97	98.182 (1.485)	**100**	96.97	**100**	91.667	95.833
car-good (k=500)	N/D	86.364 (7.606)	**100**	72.727	NA	31.818	77.273
winequality-red-4 (k=50)	N/D	42.222 (2.722)	**44.444**	11.111	NA	NA	15.385
winequality-white-9_vs_4	N/D	N/D	NA	N/D	NA	NA	**50**
yeast5 (k=50)	6.667	70 (10)	**100**	80	81.818	88.889	88.889
poker-8-9_vs_6 (k={70,100,250})	66.667	60 (13.333)	**100**	**100**	16.667	33.333	N/D
shuttle-2_vs_5 (k=250)	70.833	95.294 (5.764)	**100**	79.167	**100**	**100**	**100**
abalone19 (k=250)	N/D	46.154 (0.000)	**80**	20	NA	NA	N/D
winning times	0	0	**12**	1	2	2	5

Table 3.7: Recall results for support vector machine

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=15)	17.857	24.286 (2.673)	**53.571**	46.429	NA	26.087	30.435
vehicle1 (k=100)	43.077	46.23 (4.199)	58.462	52.308	57.576	62.069	**81.034**

default of credit card (k=20)	34.249	36.346 (1.722)	**60.407**	34.249	33.994	41.062	53.422
new-thyroid1 (k=10)	77.778	93.333 (5.443)	**100**	88.889	93.75	88.889	91.667
bank marketing (k=1000)	17.799	19.862 (0.578)	**53.27**	18.245	18.188	20.681	31.517
page-blocks (k=50)	74.691	64 (0.637)	**90.608**	77.273	68.519	73.125	72.5
yeast-0-2-5-7-9_vs_3-6-8 (k=10)	86.957	86.061 (1.485)	**90.909**	84.848	80.556	75.862	75.862
vowel0 (k=150)	84.848	97.576 (2.268)	**100**	87.879	96.154	**100**	91.667
car-good (k=500)	18.182	55.455 (1.818)	**95.455**	77.273	NA	31.818	59.091
winequality-red-4 (k=50)	5.556	**30 (2.722)**	27.778	22.222	NA	NA	N/D
winequality-white-9_vs_4	N/D	**100**	NA	N/D	NA	NA	**100**
yeast5 (k={50,75,125})	66.667	95 (6.124)	**100**	86.667	45.555	66.667	66.667
poker-8-9_vs_6 (k=100)	66.667	80 (16.33)	**100**	**100**	33.333	**100**	50
shuttle-2_vs_5	**100**	**100**	**100**	**100**	**100**	**100**	**100**
abalone19 (k=50)	N/D	**37.778 (5.443)**	20	N/D	NA	NA	N/D
winning times	1	4	**11**	2	1	3	3

Table 3.8: Recall results for K Nearest Neighbors

Chapter 4

Conclusion and future directions

Our research has provided a detailed investigation of modern data-level minority oversampling techniques for tackling the CIL. We focused on binary classification for a variety of data sets retrieved from the KEEL repository. The oversampling methods chosen for this study were: Kmeans-SMOTE, SMOTE-BFT, KNOS, and GANs. We have proposed two additional methods derived from KNOS, namely are KNOS-3, and KNOS-5 depending on the number of majority samples among the nearest neighbours required to reject a minority sample. To evaluate our comparative results, we have chosen a collection of widely used metrics in class imbalance learning: ROC-AUC, recall, precision, F-measure, and G-mean.

Based on the results, SMOTE-BFT seems to be a reliable method delivering optimal results except for the LR classifier. It is recommended to use it for data sets with extreme IR (higher than 100:1). However, the results show that it is not a suitable method for data sets with a small number of instances and low IR. Its major drawback, alongside nearest-neighbour-based oversampling techniques, is that they are not applicable when the number of input nearest neighbours is greater than the total number of minority samples. Another interesting finding concerns the Kmeans-SMOTE that involves a pre-processing step based on Kmeans clustering before any oversampling occurs. Its advantage is that it can work with any data set and more precisely when the within-class imbalance problem is present. However, it needs manual tuning for the number of clusters (k parameter) in the pre-processing step.

Regarding GANs, which are novel deep-learning-based approaches for tackling CIL, they seem not to be relied on to deliver an optimal solution. Additionally, it needs a significant amount of training data. Furthermore, using a nearest-neighbour based oversampling approach such as KNOS relies on all nearest neighbours for generating artificial data rather

than randomly selecting a minority sample like SMOTE. Its main drawback is that it is not applicable for data sets with high overlapping, as this causes the rejection of all minority samples in the filtering step. KNOS-5 is a proposed modification that is less strict regarding filtering minority instances and works best with the LR classifier.

Some of the future directions related to our research will include focusing on highly imbalanced data sets for SMOTE-BFT, and KNOS, investigating some modern proposed cost-sensitive CIL methods and proposing a hybrid solution that combines some over-sampling solution with cost-sensitive ones. We have only experimented with one GAN architecture in this research, so we can focus on GAN and adapt some more advanced GAN-based techniques to the CIL problem in a future direction. As the last proposal, we see it is necessary to generalize the CIL problem to the multi-class case and search for the recent advancements to build on top of it or compare different approaches.

Appendix A

Remaining results

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=15)	54.107 (0.932)	54.299 (2.291)	50.692 (1.407)	54.509 (1.429)	NA	**60.435 (1.344)**	51.159 (1.977)
vehicle1 (k=15)	64.886 (2.326)	65.061 (1.403)	67.788 (1.372)	68.886 (1.348)	**73.129 (1.68)**	65.368 (1.939)	57.62 (0.849)
default of credit card (k=20)	**61.7 (0.264)**	61.506 (0.515)	60.934 (0.288)	61.565 (0.093)	61.37 (0.346)	58.488 (0.239)	52.733 (0.176)
new-thyroid1 (k=5)	93.552 (0.000)	94.444 (0.000)	94.444 (0.000)	93.373 (0.357)	**95.855 (0.000)**	88.532 (0.437)	87.5 (0.000)
bank marketing (k=1000)	69.91 (0.294)	70.286 (0.502)	**72.918 (0.118)**	69.619 (0.17)	70.214 (0.396)	66.76 (0.207)	61.141 (0.142)
page-blocks (k=20)	91.341 (0.526)	91.688 (0.259)	93.196 (0.603)	**94.979 (0.122)**	92.131 (0.388)	90.554 (0.949)	84.314 (0.261)
yeast-0-2-5-7-9_vs_3-6-8 (k=50)	87.63 (2.199)	92.476 (0.000)	**95.52 (1.512)**	90.757 (1.674)	87.028 (4.073)	77.296 (0.000)	72.02 (1.217)
vowel0 (k=100)	97.955 (1.382)	98.129 (0.863)	**98.826 (0.649)**	95.265 (2.143)	87.234 (1.578)	85.614 (0.947)	78.187 (1.593)
car-good (k=400)	**95.627 (0.949)**	92.688 (3.863)	85.961 (0.000)	87.993 (0.08)	NA	88.033 (0.000)	49.899 (0.000)
winequality-red-4 (k=50)	60.476 (1.178)	60.29 (2.175)	53.341 (0.998)	**66.898 (1.622)**	NA	NA	50.402 (1.6)

winequality-white-9_vs_4	47.6 (0.000)	49 (0.000)	NA	45.2 (0.4)	NA	NA	**69.694 (0.764)**
yeast5 (k=125)	82.869 (0.000)	79.652 (7.57)	**87.652 (1.633)**	80.133 (3.84)	81.519 (0.056)	77.343 (0.112)	50 (0.000)
poker-8-9_vs_6 (k=70)	49.323 (0.226)	71.746 (8,802)	65.651 (0.175)	**100 (0.000)**	48.841 (0.196)	57.265 (0.31)	49.727 (0.091)
shuttle-2_vs_5	**100 (0.000)**	**100 (0.000)**	**100 (0.000)**	**100 (0.000)**	**100 (0.000)**	**100 (0.000)**	**100 (0.000)**
abalone19 (k = 250)	49.453 (0.055)	56.403 (1.993)	**63.622 (0.000)**	49.743 (0.041)	NA	NA	51.994 (3.131)
winning times	3	1	**6**	4	3	2	2

Table A.1: roc-auc results for decision tree

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=40)	51.004	62.991 (3.302)	**64.621**	62.054	NA	50.725	47.826
vehicle1 (k=50)	56.61	65.218 (1.699)	68.84	67.017	68.335	71.666	**72.211**
default of credit card (k=20)	59.572	64.779 (2.119)	66.941	60.707	67.158	**68.846**	67.09
new-thyroid1 (k=5)	61.111	88.889 (7.027)	94.444	92.659	97.959	**100**	95.833
bank marketing (k=1000)	58.888	68.077 (2.721)	**79.992**	67.495	71.144	75.14	79.48
page-blocks (k=20)	75.94	77.277 (3.498)	87.239	80.709	84.411	**87.269**	87.253
yeast-0-2-5-7-9_vs_3-6-8 (k=10)	78.261	87.77 (7.716)	**91.608**	88.707	88.586	88.373	91.272
vowel0 (k=50)	80.303	89.356 (1.7)	**94.886**	90.909	90.725	90.453	94.803

car-good (k=200)	50	66.09 (9.034)	60.129	59.868	NA	50.658	58.721
winequality-red-4 (k=100)	50	**73.326 (1.165)**	73.16	58.874	NA	NA	70.713
winequality-white-9_vs_4 (k=50)	50	**98.4 (0.49)**	NA	50	NA	NA	94.898
yeast5 (k=50)	50	95.019 (4.089)	96.897	90.897	**98.621**	97.025	96.911
poker-8-9_vs_6 (k=70)	50	49.985 (1.253)	30.361	**64.259**	58.333	55.114	62.311
shuttle-2_vs_5 (k=250)	54.162	93.21 (5.35)	**100**	85.417	**100**	**100**	**100**
abalone19 (k=20)	50	71.326 (0.121)	**83.254**	72.88	NA	NA	60.959
winning times	0	2	**6**	1	2	4	2

Table A.2: roc-auc results for logistic regression

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=20)	52.79	60.96 (0.985)	**68.862**	64.174	NA	55.072	55.072
vehicle1 (k=50)	62.499	64.345 (1.006)	72.902	68.339	71.688	73.003	**78.879**
default of credit card (k=20)	63.609	65.38 (0.599)	70.027	63.75	66.409	69.4	**70.554**
new-thyroid1 (k={10,20,50})	88.889	93.333 (2.222)	**100**	94.444	96.875	**100**	**100**
bank marketing (k=3000)	57.214	60.676 (0.971)	**81.318**	60.451	57.857	73.264	79.705
page-blocks (k=50)	79.567	73.261 (0.762)	**92.85**	83.486	86.245	90.358	92.707

yeast-0-2-5-7-9_vs_3-6-8 (k=10)	90.946	95.18 (1.217)	**96.068**	93.224	89.902	87.748	87.565
vowel0 (k=30)	98.485	99.091 (0.742)	**100**	98.485	**100**	95.833	97.917
car-good (k=500)	50	92.397 (3.743)	**98.793**	84.553	NA	65.305	87.429
winequality-red-4 (k=50)	50	64.358 (1.359)	**65.404**	52.958	NA	NA	55.979
winequality-white-9_vs_4	50	50	NA	49	NA	NA	**73.98**
yeast5 (k=50)	53.333	84.224 (4.671)	**97.912**	88.724	90.104	92.843	92.957
poker-8-9_vs_6 (k={70,100})	83.333	80 (6.667)	**100**	**100**	58.333	66.667	50
shuttle-2_vs_5 (k=250)	85.417	97.647 (2.882)	**100**	89.532	**100**	**100**	**100**
abalone19 (k=250)	50	65.779 (0.099)	**78.013**	57.265	NA	NA	49.116
winning times	0	0	**12**	1	2	2	5

Table A.3: roc-auc results for support vector machine

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=15)	57.366	59.174 (1.297)	58.415 (0.9)	**63.839**	NA	52.174	47.101
vehicle1 (k=100)	64.66	65.861 (2.092)	63.622	69.011	72.139	69.045	**75.466**
default of credit card (k=20)	63.265	64.076 (0.664)	63.857	63.258	62.901	**65.092**	64.513
new-thyroid1 (k=10)	88.889	96.667 (2.722)	**100**	94.444	96.875	94.444	95.833

bank marketing (k=1000)	57.985	58.863 (0.251)	**68.154**	58.047	58.154	59.022	62.185
page-blocks (k=50)	86.535	81.278 (0.304)	**93.114**	87.864	83.178	85.753	85.17
yeast-0-2-5-7-9_vs_3-6-8 (k=10)	**93.12**	91.543 (0.084)	90.25	90.937	90.09	87.565	87.382
vowel0 (k=150)	92.424	98.788 (1.134)	**100**	93.939	97.892	**100**	95.833
car-good (k=500)	59.091	76.621 (0.975)	**94.609**	85.015	NA	65.607	78.439
winequality-red-4 (k=50)	52.561	**58.268 (1.26)**	56.854	57.215	NA	NA	49.572
winequality-white-9_vs_4 (k=50)	50	**99**	NA	50	NA	NA	97.959
yeast5 (k={75})	82.985	96.655 (2.979)	**98.84**	92.405	72.497	82.761	82.876
poker-8-9_vs_6 (k=100)	83.333	89.819 (8.039)	98.871	98.307	66.667	**99.886**	75
shuttle-2_vs_5	**100**	**100**	**100**	99.949	**100**	**100**	**100**
abalone19 (k=50)	50	**63.977 (2.811)**	53.966	49.276	NA	NA	49.92
winning times	2	4	**7**	1	1	4	2

Table A.4: roc-auc results for K Nearest Neighbors

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=15)	37.326 (1.323)	36.544 (3.322)	31.477 (2.144)	36.568 (1.831)	NA	**38.248 (1.745)**	26.417 (2.505)
vehicle1 (k=15)	**53.954 (4.484)**	50.412 (3.768)	48.578 (2.209)	53.9 (2.364)	53.657 (1.61)	46.489 (2.084)	39.847 (1.77)

default of credit card (k=500)	38.518 (0.432)	38.091 (0.7)	35.072 (0.319)	38.388 (0.142)	**38.577 (0.454)**	34.622 (0.299)	28.654 (0.439)
new-thyroid1 (k=5)	88.889 (0.000)	**100 (0.000)**	**100 (0.000)**	87.111 (3.556)	93.75 (0.000)	95 (6.124)	**100 (0.000)**
bank marketing (k=1500)	45.693 (0.566)	45.237 (0.535)	41.133 (0.205)	45.192 (0.24)	**47.231 (0.000)**	42.321 (0.321)	40.945 (0.476)
page-blocks (k=50)	85.753 (0.765)	**86.443 (0.243)**	80.52 (0.991)	80.069 (0.348)	85.912 (0.939)	79.872 (0.537)	79.252 (0.79)
yeast-0-2-5-7-9_vs_3-6-8 (k=10)	62.747 (2.444)	71.762 (1.769)	**76.881 (2.061)**	72.761 (2.245)	71.569 (4.238)	60.714 (0.000)	63.689 (1.834)
vowel0 (k=50)	90.299 (1.288)	89.001 (1.551)	87.71 (1.215)	80.665 (1.453)	**98 (2.449)**	93.567 (3.91)	94.667 (2.667)
car-good (k=400)	**87.826 (1.739)**	82.626 (5.873)	80 (0.000)	72.73 (2.365)	NA	73.913 (0.000)	N/D
winequality-red-4 (k=250)	**27.932 (3.822)**	19.881 (4.902)	7.892 (1.017)	13.967 (0.669)	NA	NA	2.87 (2.355)
winequality-white-9_vs_4	N/D	N/D	NA	N/D	NA	NA	**16.381 (2.1)**
yeast5 (k=125)	71.429 (0.000)	74.699 (7.806)	**79.143 (0.7)**	66.607 (1.414)	73.111 (3.81)	57.5 (6.124)	N/D
poker-8-9_vs_6 (k=100)	N/D	19.569 (9.525)	10.205 (0.000)	**100 (0.000)**	N/D	10.324 (2.742)	N/D
shuttle-2_vs_5	**100**	**100**	**100**	**100**	**100**	**100**	**100**
abalone19 (k=250)	N/D	4.187 (1.053)	**39.175 (0.000)**	N/D	NA	NA	3.548 (4.395)
winning times	4	3	**5**	2	4	2	3

Table A.5: Precision results for decision tree

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=15)	50	**63.54 (2.575)**	54.167	50	NA	25.714	23.404

vehicle1 (k=50)	**61.111**	52.017 (2.606)	46.067	59.184	47.059	42.424	40.909
default of credit card (k=100)	**70.115**	65.831 (0.229)	37.033	65.863	56.377	54.943	36.64
new-thyroid1 (k={5, 10, 20, 50})	**100**	**100**	**100**	80	88.889	**100**	**100**
bank marketing (k=1500)	**60.078**	44.223 (5.198)	36.147	42.026	39.321	35.2	36.081
page-blocks (k=20)	**76.316**	73.763	48.742	61.963	55.963	52.8	49.446
yeast-0-2-5-7-9_vs_3-6-8 (k=100)	**100**	89.915 (1.779)	46.377	58.333	76.316	76.667	71.429
vowel0 (k=100)	**100**	73.778 (6.27)	55	64.444	54.762	53.846	57.5
car-good (k=200)	N/D	**13.689 (6.427)**	6.311	6.486	NA	4.444	5.909
winequality-red-4 (k=250)	N/D	**17.285 (4.844)**	11.321	7.692	NA	NA	10.769
winequality-white-9_vs_4 (k=50)	N/D	**40 (8.165)**	NA	N/D	NA	NA	28.571
yeast5 (k=100)	N/D	**52.864 (5.054)**	34.884	38.235	47.826	25.714	25
poker-8-9_vs_6 (k=70)	N/D	0.283 (0.35)	N/D	1.17	**100**	1.685	2.116
shuttle-2_vs_5 (k= {20, 50, 100})	**100**	**100**	**100**	**100**	**100**	**100**	**100**
abalone19 (k=20)	N/D	**3.304 (0.041)**	2.99	3.279	NA	NA	1.523
winning times	**8**	**8**	2	1	2	2	2

Table A.6: Precision results for logistic regression

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=40)	66.667	**75.482 (9.867)**	51.429	61.111	NA	33.333	29.73
vehicle1 (k=100)	**70.37**	60.289 (2.278)	47.525	65.909	63.158	47.059	49.485
default of credit card (k=500)	**68.182**	66.096 (0.828)	48.443	67.349	61.508	57.033	48.716
new-thyroid1	**100**	**100**	**100**	**100**	**100**	**100**	**100**
bank marketing (k=1000)	**66.307**	62.04 (2.331)	38.227	62.413	61.818	49.433	40.597
page-blocks (k=100)	**89.815**	87.915 (0.726)	61.765	85.366	67.391	62.727	58.498
yeast-0-2-5-7-9_vs_3-6-8	90.476	87.227 (1.235)	71.111	71.429	93.548	**95.652**	91.667
vowel0	**100**	**100**	**100**	**100**	**100**	**100**	**100**
car-good (250)	N/D	73.789 (10.952)	64.706	47.059	NA	53.846	58.621
winequality-red-4 (k=250)	N/D	21.297 (9.76)	11.268	7.692	NA	NA	11.111
winequality-white-9_vs_4	N/D	N/D	NA	N/D	NA	NA	**50**
yeast5 (k=100)	**100**	49.956 (7.919)	45.555	52.174	56.25	36.364	38.095
poker-8-9_vs_6 (k={70,100,150})	**100**	**100**	**100**	**100**	**100**	**100**	N/D
shuttle-2_vs_5	**100**	**100**	**100**	95	**100**	**100**	**100**
abalone19 (k=250)	N/D	**3.209 (0.042)**	2.614	2.857	NA	NA	N/D
winning times	**9**	6	4	3	4	3	4

Table A.7: Precision results for support vector machine

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=40)	**71.429**	65.303 (6.577)	38.462	52	NA	28.571	21.875
vehicle1 (k=100)	51.852	50.089 (2.347)	39.175	55.738	**60.137**	43.373	44.34
default of credit card (k=250)	55.345	**55.463 (0.184)**	34.041	55.3	53.735	51.21	37.844
new-thyroid1	**100**	**100**	**100**	**100**	**100**	**100**	**100**
bank marketing (k=1000)	**56.375**	55.257 (0.404)	29.43	52.661	56.226	51.464	37.344
page-blocks (k=100)	83.448	83.388 (0.293)	71.93	**83.803**	77.622	82.979	78.378
yeast-0-2-5-7-9_vs_3-6-8 (k=100)	90.909	84.848	51.724	77.778	**96.667**	91.667	88
vowel0	**100**	**100**	**100**	**100**	96.154	**100**	**100**
car-good (k=200)	**100**	57.238 (14.025)	40.385	32.075	NA	70	54.167
winequality-red-4 (k=250)	**33.333**	16.127 (5.117)	7.143	10	NA	NA	N/D
winequality-white-9_vs_4 (k=50)	N/D	**50**	NA	N/D	NA	NA	**50**
yeast5 (k=100)	**76.923**	52.967 (2.705)	60	61.905	71.429	54.545	60
poker-8-9_vs_6 (k=70)	**100**	86.667 (16.33)	23.077	16.667	**100**	85.714	**100**
shuttle-2_vs_5	**100**	**100**	**100**	95	**100**	**100**	**100**
abalone19 (k=50)	N/D	2.713 (0.43)	1.316	N/D	NA	NA	N/D

winning times	**9**	5	3	3	5	3	5

Table A.8: Precision results for K Nearest Neighbors

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=15)	33.661 (2.263)	35.284 (4.77)	29.951 (2.198)	36.846 (2.343)	NA	**41.434 (1.835)**	28.995 (2.853)
vehicle1 (k=15)	47.325 (3.876)	47.846 (2.146)	52.019 (1.903)	53.174 (2.034)	59.33 (2.161)	46.477 (3.008)	32.551 (1.742)
default of credit card (k=20)	**40.288 (0.39)**	40.027 (0.752)	39.529 (0.399)	40.078 (0.141)	39.758 (0.533)	35.172 (0.39)	21.93 (0.356)
new-thyroid1 (k=5)	88.889	**94.118 (0.000)**	**94.118 (0.000)**	87.953 (1.871)	93.75 (0.000)	85.441 (2.522)	85.714 (0.000)
bank marketing (k=1000)	46.474 (0.525)	46.691 (0.776)	**47.639 (0.185)**	45.915 (0.262)	47.346 (0.627)	41.696 (0.35)	33.028 (0.294)
page-blocks (k=20)	85.076 (0.652)	85.7 (0.259)	84.575 (1.026)	85.766 (0.201)	**85.857 (0.793)**	81.581 (1.171)	74.686 (0.389)
yeast-0-2-5-7-9_vs_3-6-8 (k=50)	69.976 (3.043)	78.787 (1.608)	**84.779 (1.994)**	78.541 (1.928)	74.143 (5.879)	59.649 (0.000)	53.923 (1.487)
vowel0 (k=150)	92.941 (1.103)	93.045 (1.733)	**93.183 (1.059)**	86.476 (1.936)	84.694 (0.000)	81.152 (2.432)	70.769 (2.051)
car-good (k=400)	**89.778 (1.778)**	84.308 (6.462)	76.19 (0.000)	74.913 (1.286)	NA	75.556 (0.000)	N/D
winequality-red-4 (k=250)	25.363 (2.719)	20.179 (3.315)	9.573 (1.379)	21.25 (1.167)	NA	NA	3.535 (2.892)
winequality-white-9_vs_4	N/D	N/D	NA	N/D	NA	NA	**24.603 (2.341)**
yeast5 (k=125)	68.966 (0.000)	65.913 (12.376)	**77.517 (2.027)**	63.644 (4.554)	68 (1.633)	56.347 (3.033)	N/D
poker-8-9_vs_6 (k=100)	N/D	24.337 (9.486)	15.577 (0.000)	**100**	N/D	12.539 (2.081)	N/D
shuttle-2_vs_5	**100**	**100**	**100**	**100**	**100**	**100**	**100**

abalone19 (k=250)	N/D	6.704 (1.649)	46.914 (0.000)	N/D	NA	NA	4.127 (5.079)
winning times	3	2	**6**	2	2	2	2

Table A.9: F-measure results for decision tree

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=40)	6.667	46.673 (6.428)	**50**	46.154	NA	31.034	31.429
vehicle1 (k=50)	26.506	47.97 (2.76)	53.247	50.877	52.98	53.503	**53.571**
default of credit card (k=20)	33.178	44.859 (3.931)	47.059	36.283	49.299	**51.704**	47.078
new-thyroid1 (k=5)	36.364	86.756 (9.414)	94.118	84.211	94.118	**100**	95.652
bank marketing (k=1000)	29.44	42.327 (3.465)	**49.474**	42.371	45.211	46.137	49.243
page-blocks (k=20)	63.043	63.955 (4.11)	62.124	63.722	64.211	**64.39**	62.181
yeast-0-2-5-7-9_vs_3-6-8 (k=10)	72.222	73.159 (7.374)	62.745	69.136	**78.378**	77.966	78.125
vowel0 (k=50)	75.472	**76.429 (3.862)**	70.968	74.359	67.647	66.667	71.875
car-good (k=200)	N/D	**20.733 (8.284)**	11.404	11.594	NA	7.643	10.744
winequality-red-4 (k=250)	N/D	**20.937 (4.872)**	19.355	12.5	NA	NA	17.949
winequality-white-9_vs_4 (k=50)	N/D	**56.667 (8.165)**	NA	N/D	NA	NA	44.444
yeast5 (k=50)	N/D	64.702 (2.61)	51.724	53.061	**64.706**	40.909	40

poker-8-9_vs_6 (k=70)	N/D	0.554 (0.684)	N/D	2.299	**28.571**	3.261	4.103
shuttle-2_vs_5 (k=250)	15.385	88.492 (6.516)	**100**	82.927	**100**	**100**	**100**
abalone19 (k=20)	N/D	**6.27 (0.073)**	5.788	6.218	NA	NA	2.927
winning times	0	**5**	3	0	4	4	2

Table A.10: F-measure results for logistic regression

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=15)	12.903	39.233 (2.787)	**57.143**	47.826	NA	31.818	36.667
vehicle1 (k=50)	41.304	45.306 (1.9)	57.831	53.211	58.537	55.944	**61.935**
default of credit card (k=20)	42.902	46.452 (1.152)	52.373	43.208	48.318	52.759	**53.004**
new-thyroid1 (k={10,20,50})	87.5	92.794 (2.647)	**100**	94.118	96.774	**100**	**100**
bank marketing (k=3000)	25.089	33.643 (2.132)	51.681	33.287	26.811	51.613	**52.445**
page-blocks (k=50)	71.852	61.346 (1.122)	74.172	**75.812**	71.676	72.632	71.671
yeast-0-2-5-7-9_vs_3-6-8 (k=50)	86.364	**89.025 (0.645)**	82.051	80	86.567	84.615	83.019
vowel0 (k=30)	98.462	99.077 (0.754)	**100**	98.462	**100**	95.652	97.872
car-good (k=500)	N/D	77.722 (3.351)	**78.571**	57.143	NA	40	66.667
winequality-red-4	N/D	17.281 (1.102)	**17.978**	9.091	NA	NA	12.903

winequality-white-9_vs_4	N/D	N/D	NA	N/D	NA	NA	**50**
yeast5 (k=50)	12.5	55.56 (4.5)	62.5	63.158	**66.667**	51.613	53.333
poker-8-9_vs_6 (k={70,100})	80	74 (12)	**100**	**100**	28.571	50	N/D
shuttle-2_vs_5 (k=250)	82.927	97.5 (3.062)	**100**	86.364	**100**	**100**	**100**
abalone19 (k=250)	N/D	**6.001 (0.073)**	5.063	4	NA	NA	N/D
winning times	0	2	**7**	2	3	2	6

Table A.11: F-measure results for support vector machine

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=15)	28.571	35.171 (3.051)	44.776	**49.057**	NA	27.273	25.455
vehicle1 (k=100)	47.059	48.052	46.914	53.968	**58.915**	51.064	57.317
default of credit card (k=500)	42.314	43.851 (1.256)	43.544	42.3	41.543	**45.578**	44.303
new-thyroid1 (k=10)	87.5	96.471	**100**	94.118	96.774	94.118	95.652
bank marketing (k=1000)	27.055	29.214 (0.606)	**37.914**	27.101	27.485	29.505	34.184
page-blocks (k=50)	78.827	72.162 (0.354)	80.196	**80.405**	72.787	77.741	75.325
yeast-0-2-5-7-9_vs_3-6-8 (k=100)	**88.889**	84.848	65.934	81.159	87.879	83.019	81.481
vowel0 (k=150)	91.803	98.76 (1.167)	**100**	93.548	96.154	**100**	95.652
car-good (k=500)	30.769	54.062	**56.757**	45.333	NA	43.75	56.522

winequality-red-4 (k=50)	9.524	12.605 (0.943)	11.364	**13.793**	NA	NA	N/D
winequality-white-9_vs_4 (k=50)	N/D	**66.667**	NA	N/D	NA	NA	**66.667**
yeast5 (k=100)	71.429	67.273 (2.969)	**75**	72.222	55.556	60	63.158
poker-8-9_vs_6 (k=70)	80	74.667 (6.532)	37.5	28.571	50	**92.308**	66.667
shuttle-2_vs_5	**100**	**100**	**100**	97.959	**100**	**100**	**100**
abalone19 (k=50)	N/D	5.062 (0.797)	2.469	N/D	not-aplicable	NA	N/D
winning times	2	2	6	3	2	4	2

Table A.12: F-measure results for k Nearest Neighbors

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=15)	48.714 (2.049)	50.162 (4.271)	45.581 (1.969)	51.629 (0.000)	NA	**58.473 (1.561)**	47.43 (2.73)
vehicle1 (k=30)	60.738 (2.9)	62.373 (1.871)	66.753 (1.418)	67.149 (1.48)	**72.797 (1.818)**	62.527 (2.731)	49.127 (1.732)
default of credit card (k=20)	58.547 (0.314)	58.417 (0.604)	**58.889 (0.359)**	58.347 (0.125)	57.895 (0.472)	53.882 (0.369)	39.468 (0.35)
new-thyroid1 (k=5)	93.435 (0.000)	94.281 (0.000)	94.281 (0.000)	93.265 (0.000)	**95.831 (0.000)**	87.875 (0.387)	86.603 (0.000)
bank marketing (k=1000)	66.146 (0.384)	66.784 (0.686)	**71.067 (0.145)**	65.725 (0.241)	66.424 (0.55)	61.627 (0.302)	51.17 (0.26)
page-blocks (k=20)	91.076 (0.569)	91.445 (0.28)	93.103 (0.625)	**94.942 (0.126)**	91.913 (0.409)	90.263 (1.031)	83.194 (0.312)
yeast-0-2-5-7-9_vs_3-6-8 (k=50)	87.187 (2.386)	92.384 (1.22)	**95.509 (0.000)**	90.58 (1.809)	86.338 (4.651)	75.006 (0.000)	67.462 (1.923)

vowel0 (k=100)	97.221 (0.775)	98.119 (0.869)	**98.822 (0.649)**	95.216 (2.181)	86.299 (1.788)	84.465 (1.071)	75.131 (2.21)
car-good (k=400)	**95.547 (0.978)**	92.355 (4.207)	84.936 (0.000)	87.337 (0.071)	NA	87.373 (0.000)	N/D
winequality-red-4 (k=50)	47.677 (2.229)	50.146 (4.315)	33.849 (2.401)	**62.964 (2.401)**	NA	NA	16.281 (13.293)
winequality-white-9_vs_4	N/D	N/D	NA	N/D	NA	NA	**66.851 (0.57)**
yeast5 (k=125)	81.27 (0.000)	76.531 (10.024)	**86.854 (1.859)**	77.74 (4.859)	79.534 (0.045)	74.211 (0.084)	N/D
poker-8-9_vs_6 (k=70)	N/D	64.389 (14.112)	57.145 (0.000)	**100**	N/D	40.386 (0.128)	N/D
shuttle-2_vs_5	**100**	**100**	**100**	**100**	**100**	**100**	**100**
abalone19 (k=250)	N/D	39.61 (4.732)	**63.413 (0.000)**	N/D	NA	NA	14.082 (17.247)
winning times	2	1	**7**	4	3	2	2

Table A.13: G-mean results for decision tree

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=40)	18.75	58.523 (5.806)	**62.007**	59.01	NA	49.382	47.826
vehicle1 (k=50)	40.369	61.822 (2.346)	68.598	63.162	67.896	71.662	**72.011**
default of credit card (k=20)	46.009	58.191 (4.559)	66.898	49.123	62.965	65.871	**67.078**
new-thyroid1 (k=5)	47.14	87.806 (8.244)	94.281	92.582	97.938	**100**	95.743
bank marketing (k=1000)	43.774	63.848 (4.325)	**79.975**	62.784	68.838	74.691	79.456
page-blocks (k=20)	72.611	74.419 (4.743)	**87.225**	79.279	83.919	87.138	87.183

yeast-0-2-5-7-9_vs_3-6-8 (k=10)	75.181	86.685 (9.765)	**91.45**	88.623	88.221	87.907	91.131
vowel0 (k=50)	77.85	89.116 (1.789)	94.748	90.859	90.697	90.405	**90.798**
car-good (k=200)	N/D	**63.753 (9.768)**	60.12	59.631	NA	44.938	58.719
winequality-red-4 (k=100)	N/D	**73.105 (1.225)**	72.871	53.046	NA	NA	68.672
winequality-white-9_vs_4 (k=50)	N/D	**98.386 (0.497)**	NA	N/D	NA	NA	94.761
yeast5 (k=50)	N/D	94.898 (4.236)	96.697	90.799	**98.611**	96.98	96.862
poker-8-9_vs_6 (k=70)	N/D	19.03 (23.319)	N/D	**64.214**	40.825	54.876	62.158
shuttle-2_vs_5 (k=250)	28.868	92.787 (5.706)	**100**	84.163	**100**	**100**	**100**
abalone19 (k=20)	N/D	70.651 (0.105)	**82.98**	71.733	NA	NA	56.264
winning times	0	3	**6**	1	2	2	4

Table A.14: G-mean results for logistic regression

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=20)	26.517	51.039 (2.784)	**68.709**	59.151	NA	49.254	54.594
vehicle1 (k=50)	52.909	58.513 (1.92)	72.896	64.089	69.608	72.891	**78.784**
default of credit card (k=20)	54.791	58.664 (1.236)	68.804	55.169	60.839	66.359	**69.45**
new-thyroid1 (k={10,20,50})	88.192	93.063 (2.436)	**100**	94.281	96.825	**100**	**100**

bank marketing (k=3000)	39.128	47.817 (2.176)	**81.303**	47.211	41.082	70.685	79.505
page-blocks (k=50)	77.092	68.477 (1.311)	**92.85**	82.071	85.697	90.265	92.707
yeast-0-2-5-7-9_vs_3-6-8 (k=10)	90.563	95.132 (1.272)	**96.064**	93.195	89.415	86.939	86.779
vowel0 (k=30)	98.473	99.084 (0.748)	**100**	98.473	**100**	95.743	97.895
car-good (k=500)	N/D	92.107 (4.033)	**98.785**	83.722	NA	56.066	86.837
winequality-red-4 (k=50)	N/D	60.398 (1.949)	**61.955**	32.456	NA	NA	38.545
winequality-white-9_vs_4	N/D	N/D	NA	N/D	NA	NA	**69.985**
yeast5 (k=50)	25.82	82.792 (5.488)	**97.89**	88.294	89.723	92.758	92.868
poker-8-9_vs_6 (l={70,100})	81.65	76.867 (9.566)	**100**	**100**	40.825	57.735	N/D
shuttle-2_vs_5 (k=250)	84.163	97.573 (2.972)	**100**	88.93	**100**	**100**	**100**
abalone19 (k=250)	N/D	62.783 (0.073)	**77.988**	43.481	NA	NA	N/D
winning times	0	0	**12**	1	2	2	5

Table A.15: G-mean results for Support vector machine

data set	Baseline	Kmeans-SMOTE	SMOTE-BFT	GAN	KNOS	KNOS-3	KNOS-5
haberman (k=15)	41.592	47.718 (2.544)	57.864	**61.419**	NA	45.184	44.054
vehicle1 (k=100)	60.952	62.802 (2.835)	63.413	66.959	70.654	68.691	**75.261**

default of credit card (k=20)	56.219	57.746 (1.25)	**63.763**	56.215	55.865	60.494	63.552
new-thyroid1 (k=10)	88.192	96.569 (2.802)	**100**	94.281	96.825	94.281	95.743
bank marketing (k=1000)	41.801	44.083 (0.626)	**66.509**	42.253	42.244	44.873	54.097
page-blocks (k=50)	85.721	79.419 (0.384)	**93.08**	87.223	81.876	84.818	84.223
yeast-0-2-5-7-9_vs_3-6-8 (k=10)	**92.916**	91.368 (0.151)	90.248	90.733	89.584	86.779	86.619
vowel0 (k=150)	92.113	98.774 (1.15)	**100**	93.744	97.877	**100**	95.743
car-good (k=500)	42.64	73.63 (1.238)	**93.605**	84.661	NA	56.237	76.015
winequality-red-4 (k=50)	23.519	**50.897 (2.226)**	48.857	45.267	NA	NA	N/D
winequality-white-9_vs_4 (k=50)	N/D	**98.995**	NA	N/D	NA	NA	97.938
yeast5 (k=125)	81.365	96.59 (3.147)	**98.833**	92.227	67.265	81.181	81.275
poker-8-9_vs_6 (k=100)	81.65	88.817 (8.856)	98.865	98.292	57.735	**99.886**	70.711
shuttle-2_vs_5	**100**	**100**	**100**	99.948	**100**	**100**	**100**
abalone19 (k=50)	N/D	58.223 (4.214)	41.936	N/D	NA	NA	N/D
winning times	2	3	**8**	1	1	3	2

Table A.16: G-mean results for K Nearest Neighbors

Bibliography

[1] Dr.J.V.R.MURTHY K.P.N.V.SATYASREE. "An Exhaustive Literature Review on Class Imbalance Problem". In: *International Journal of Emerging Trends of Technology in Computer Science (IJETTCS)* 2.3 (2013), pp. 109–118. ISSN: 2278-6856.

[2] Ajith Abraham Shaza M. Abd Elrahman. "A Review of Class Imbalance Problem". In: *Journal of Network and Innovative Computing* 1 (2013), pp. 332–340. ISSN: 2160-2174.

[3] Dr. Latesh Malik Mr. Longadge Ms. Snehlata S. Dongre. "Class Imbalance Problem in Data Mining:Review". In: *International Journal of Computer Science and Network (IJCSN)* 2.1 (2013). ISSN: 2277-5420.

[4] Kuppa M.R Naganjaneyulu S. "A novel framework for class imbalance learning using intelligent under-sampling". In: *Prog Artif Intell 2* (2013), 73–84. DOI: https://doi.org/10.1007/s13748-012-0038-2.

[5] Ke Cheng et al. "Grouped SMOTE With Noise Filtering Mechanism for Classifying Imbalanced Data". In: *IEEE Access* 7 (2019), pp. 170668–170681. DOI: 10.1109/ACCESS.2019.2955086.

[6] Qiu C. Li C. Jiang L. In: *International Journal of Pattern Recognition and Artificial Intelligence* (2015), pp. 1551004–1 –1551004–18. DOI: https://10.1142/S0218001415510040.

[7] L. O. Hall W. P. Kegelmeyer N. V. Chawla K. W. Bowyer. "SMOTE: Synthetic Minority Over-sampling Technique". In: *Journal of Artificial Intelligence Research 16* 16 (2002), 321–357.

[8] Charu C. Aggarwal. "Data Classification". In: *Data Mining: The Textbook*. Cham: Springer International Publishing, 2015, pp. 285–344. ISBN: 978-3-319-14142-8. DOI: 10.1007/978-3-319-14142-8_10. URL: https://doi.org/10.1007/978-3-319-14142-8_10.

[9] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. "Supervised Learning". In: *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*. Ed. by Matthieu Cord and Pádraig Cunningham. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 21–49. ISBN: 978-3-540-75171-7. DOI: 10.1007/978-3-540-75171-7_2. URL: https://doi.org/10.1007/978-3-540-75171-7_2.

[10] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "Overview of Supervised Learning". In: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY: Springer New York, 2009, pp. 9–41. ISBN: 978-0-387-84858-7. DOI: 10.1007/978-0-387-84858-7_2. URL: https://doi.org/10.1007/978-0-387-84858-7_2.

[11] Francisco Herrera et al. "Multilabel Classification". In: *Multilabel Classification : Problem Analysis, Metrics and Techniques*. Cham: Springer International Publishing, 2016, pp. 17–31. ISBN: 978-3-319-41111-8. DOI: 10.1007/978-3-319-41111-8_2. URL: https://doi.org/10.1007/978-3-319-41111-8_2.

[12] Anjana Gosain and Saanchi Sardana. "Handling class imbalance problem using oversampling techniques: A review". In: *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2017, pp. 79–85. DOI: 10.1109/ICACCI.2017.8125820.

[13] Xinjian Guo et al. "On the Class Imbalance Problem". In: *2008 Fourth International Conference on Natural Computation*. Vol. 4. 2008, pp. 192–201. DOI: 10.1109/ICNC.2008.871.

[14] Rushi Longadge and Snehalata Dongre. *Class Imbalance Problem in Data Mining Review*. 2013. arXiv: 1305.1707 [cs.LG].

[15] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. "A systematic study of the class imbalance problem in convolutional neural networks". In: *Neural Networks* 106 (2018), pp. 249–259. ISSN: 0893-6080. DOI: https://doi.org/

10.1016/j.neunet.2018.07.011. URL: https://www.sciencedirect.com/science/article/pii/S0893608018302107.

[16] Nathalie Japkowicz and Shaju Stephen. "The Class Imbalance Problem: A Systematic Study". In: *Intelligent Data Analysis* 6.5 (2002), 429 – 449. DOI: https://10.3233/IDA-2002-6504.

[17] B. Krawczyk. "Learning from imbalanced data: open challenges and future directions". In: *Prog Artif Intell 5* (2016), 221–232. DOI: https://doi.org/10.1007/s13748-016-0094-0.

[18] Fares Grina, Zied Elouedi, and Eric Lefevre. "A Preprocessing Approach for Class-Imbalanced Data Using SMOTE and Belief Function Theory". In: *Intelligent Data Engineering and Automated Learning – IDEAL 2020*. Ed. by Cesar Analide et al. Cham: Springer International Publishing, 2020, pp. 3–11. ISBN: 978-3-030-62365-4.

[19] N. Surendro K. Asniar Ulfa Maulidevi. "SMOTE-LOF for Noise Identification in Imbalanced Data Classification". In: *Journal of King Saud University - Computer and Information Sciences* (2021), pp. 1–22.

[20] Aida Ali, Siti Mariyam Shamsuddin, and Anca Ralescu. "Classification with class imbalance problem: A review". In: 7 (Jan. 2015), pp. 176–204.

[21] Nathalie Japkowicz. "The Class Imbalance Problem: Significance and Strategies". In: *In Proceedings of the 2000 International Conference on Artificial Intelligence (ICAI.* 2000, pp. 111–117.

[22] Shiven Sharma et al. "Synthetic Oversampling with the Majority Class: A New Perspective on Handling Extreme Imbalance". In: *2018 IEEE International Conference on Data Mining (ICDM)*. 2018, pp. 447–456. DOI: 10.1109/ICDM.2018.00060.

[23] Adnan Amin et al. "Comparing Oversampling Techniques to Handle the Class Imbalance Problem: A Customer Churn Prediction Case Study". In: *IEEE Access* 4 (2016), pp. 7940–7957. DOI: 10.1109/ACCESS.2016.2619719.

[24] Jiawen Kong et al. "On the Performance of Oversampling Techniques for Class Imbalance Problems". In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Hady W. Lauw et al. Cham: Springer International Publishing, 2020, pp. 84–96. ISBN: 978-3-030-47436-2.

[25] Tawfiq Hasanin and Taghi Khoshgoftaar. "The Effects of Random Undersampling with Simulated Class Imbalance for Big Data". In: *2018 IEEE International Conference on Information Reuse and Integration (IRI)*. 2018, pp. 70–79. DOI: 10.1109/IRI.2018.00018.

[26] Debashree Devi, Saroj K. Biswas, and Biswajit Purkayastha. "Learning in presence of class imbalance and class overlapping by using one-class SVM and undersampling technique". In: *Connection Science* 31.2 (2019), pp. 105–142. DOI: 10.1080/09540091.2018.1560394. eprint: https://doi.org/10.1080/09540091.2018.1560394. URL: https://doi.org/10.1080/09540091.2018.1560394.

[27] Godfrey A. Mills Koudjo M. Koumadi Robert A. Sowah Moses A. Agebure and Seth Y. Fiawoo. "New Cluster Undersampling Technique for Class Imbalance Learning". In: *International Journal of Machine Learning and Computing* 6.3 (2016).

[28] Juanjuan Wang et al. "Classification of Imbalanced Data by Using the SMOTE Algorithm and Locally Linear Embedding". In: *2006 8th international Conference on Signal Processing*. Vol. 3. 2006. DOI: 10.1109/ICOSP.2006.345752.

[29] Liliya Demidova and Irina Klyueva. "SVM classification: Optimization with the SMOTE algorithm for the class imbalance problem". In: *2017 6th Mediterranean Conference on Embedded Computing (MECO)*. 2017, pp. 1–4. DOI: 10.1109/MECO.2017.7977136.

[30] Reshma C. Bhagat and Sachin S. Patil. "Enhanced SMOTE algorithm for classification of imbalanced big-data using Random Forest". In: *2015 IEEE International Advance Computing Conference (IACC)*. 2015, pp. 403–408. DOI: 10.1109/IADCC.2015.7154739.

[31] Liu Y. Chen R. et al Guo S. "Improved SMOTE Algorithm to Deal with Imbalanced Activity Classes in Smart Homes". In: *Neural Process Lett 50* (2019), 1503–1526. DOI: https://doi.org/10.1007/s11063-018-9940-3.

[32] György Kovács. "smote-variants: a Python Implementation of 85 Minority Oversampling Techniques". In: *Neurocomputing* (June 2019). DOI: 10.1016/j.neucom.2019.06.100.

[33] Baiyun Chen et al. "RSMOTE: A self-adaptive robust SMOTE for imbalanced problems with label noise". In: *Information Sciences* 553 (Oct. 2020). DOI: 10.1016/j.ins.2020.10.013.

[34] Enislay Ramentol et al. "SMOTE-RSB *: A hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using SMOTE and rough sets theory". In: *Knowledge and Information Systems* 33 (Nov. 2011). DOI: 10.1007/s10115-011-0465-6.

[35] Georgios Douzas, Fernando Bacao, and Felix Last. "Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE". In: *Information Sciences* 465 (2018), 1–20. ISSN: 0020-0255. DOI: 10.1016/j.ins.2018.06.056. URL: http://dx.doi.org/10.1016/j.ins.2018.06.056.

[36] Chunhui Yuan and Haitao Yang. "Research on K-Value Selection Method of K-Means Clustering Algorithm". In: *J* 2.2 (2019), pp. 226–235. ISSN: 2571-8800. DOI: 10.3390/j2020016. URL: https://www.mdpi.com/2571-8800/2/2/16.

[37] Kristina P. Sinaga and Miin-Shen Yang. "Unsupervised K-Means Clustering Algorithm". In: *IEEE Access* 8 (2020), pp. 80716–80727. DOI: 10.1109/ACCESS.2020.2988796.

[38] Zoulficar Younes, Fahed Abdallah, and Thierry Denœux. "An Evidence-Theoretic k-Nearest Neighbor Rule for Multi-label Classification". In: *Scalable Uncertainty Management*. Ed. by Lluís Godo and Andrea Pugliese. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 297–308. ISBN: 978-3-642-04388-8.

[39] Khairil Anwar Notodiputro Bagus Sartono Budi Santoso Hari Wijayanto. "K-Neighbor over-sampling with cleaning data: a new approach to improve classification performance in data sets with class imbalance". In: *Applied Mathematical Sciences* 12.10 (2018), pp. 449–460.

[40] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[41] Antonia Creswell et al. "Generative Adversarial Networks: An Overview". In: *IEEE Signal Processing Magazine* 35.1 (2018), pp. 53–65. DOI: 10.1109/MSP.2017.2765202.

[42] Xudong Mao et al. "Least Squares Generative Adversarial Networks". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017.

[43] Fabio Henrique Kiyoiti dos Santos Tanaka and Claus Aranha. *Data Augmentation Using GANs*. 2019. arXiv: 1904.09135 [cs.LG].

[44] Agrawal H. Kotecha K Chaudhari P. "Data augmentation using MG-GAN for improved cancer classification on gene expression data". In: *Soft Comput 24* (2020), 11381–11391. DOI: https://doi.org/10.1007/s00500-019-04602-2.

[45] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784 [cs.LG].

[46] Zhaoqing Pan et al. "Loss Functions of Generative Adversarial Networks (GANs): Opportunities and Challenges". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 4.4 (2020), pp. 500–522. DOI: 10.1109/TETCI.2020.2991774.

[47] Zakarya Farou, Noureddine Mouhoub, and Tomáš Horváth. "Data Generation Using Gene Expression Generator". In: *International Conference on Intelligent Data Engineering and Automated Learning*. Springer. 2020, pp. 54–65.

[48] Sankha Subhra Mullick, Shounak Datta, and Swagatam Das. *Generative Adversarial Minority Oversampling*. 2020. arXiv: 1903.09730 [cs.CV].

[49] TAAN Ahmad and Zakarya FAROU. "Supervised Learning Methods for Skin Segmentation Based on Pixel Color Classification". In: *Central-European Journal of New Technologies in Research, Education and Practice* (2021).

[50] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[51] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[52] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[53] Mollineda R.A. Sánchez J.S García V. "On the k-NN performance in a challenging scenario of imbalance and overlapping". In: *Pattern Anal Applic 11* (2008), 269–280. DOI: https://doi.org/10.1007/s10044-007-0087-5.

[54] Fadi Thabtah et al. "Data imbalance in classification: Experimental evaluation". In: *Information Sciences* 513 (2020), pp. 429–441. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2019.11.004. URL: https://www.sciencedirect.com/science/article/pii/S0020025519310497.

[55] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[56] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[57] Bing Xu et al. *Empirical Evaluation of Rectified Activations in Convolutional Network*. 2015. arXiv: 1505.00853 [cs.LG].

[58] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].